

The Journal

Volume 7/2, December 2015

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

Contents

Editorial	4
---------------------	---

Contributed Research Articles

Fitting Conditional and Simultaneous Autoregressive Spatial Models in hglm	5
VSURF : An R Package for Variable Selection Using Random Forests	19
zoib : An R package for Bayesian Inference for Beta Regression and Zero/One Inflated Beta Regression	34
apc : An R Package for Age-Period-Cohort Analysis.	52
QuantifQuantile : An R Package for Performing Quantile Regression Through Optimal Quantization	65
Numerical Evaluation of the Gauss Hypergeometric Function with the hypergeo Package.	81
SRCS : Statistical Ranking Color Scheme for Visualizing Parameterized Multiple Pairwise Comparisons with R	89
An R Package for the Panel Approach Method for Program Evaluation: pampe	105
BSGS : Bayesian Sparse Group Selection.	122
ClustVarLV : An R Package for the Clustering of Variables Around Latent Variables. .	134
Working with Multilabel Datasets in R: The mldr Package	149
PracTools : Computations for Design of Finite Population Samples	163
ALTopt : An R Package for Optimal Experimental Design of Accelerated Life Testing .	177
abctools : An R Package for Tuning Approximate Bayesian Computation Analyses .	189
mtk : A General-Purpose and Extensible R Environment for Uncertainty and Sensitivity Analyses of Numerical Experiments	206
treeClust : An R Package for Tree-Based Clustering Dissimilarities	227
mmp : A Package for Calculating Similarity and Distance Metrics for Simple and Marked Temporal Point Processes	237
Open-Channel Computation with R	249
Generalized Hermite Distribution Modelling with the R Package hermite	263
Code Profiling in R: A Review of Existing Methods and an Introduction to Package GUIProfiler	275

News and Notes

The R Consortium and the R Foundation	288
Conference Report: useR! 2015	289
News from the Bioconductor Project	291
Changes in R	293
Changes on CRAN	298

The R Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are licensed under the Creative Commons Attribution 3.0 Unported license (CC BY 3.0, <http://creativecommons.org/licenses/by/3.0/>).

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

Editor-in-Chief:
Bettina Grün

Editorial Board:
Deepayan Sarkar, Michael Lawrence and Roger Bivand

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

Email of editors and editorial board:
firstname.lastname@R-project.org

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

Editorial

by Bettina Grün

On behalf of the editorial board, I am pleased to publish Volume 7, Issue 2 of the R Journal. This issue contains 20 contributed research articles and several contributions to the News and Notes section.

In the Contributed Research Articles section each of the articles presents an R package or new features of an existing R package which was extended. The articles cover different specialized statistical modeling tools, e.g., for regression with different distributions for the dependent variable, such as implemented by packages **zoib** for zero/one inflated beta regression and **hermite** for the generalized Hermite distribution, or different data structures as provided by packages **hglm** for conditional and simultaneous autoregressive spatial models and **apc** for age-period-cohort analysis. In addition tools for program evaluation are provided by package **pampe** and quantification-based quantile regression by package **QuantifQuantile**. Other aspects of statistical modeling are to perform variable selection or variable grouping and are covered for example by packages **VSURF**, **BSGS** and **ClustVarLV** or to determine dissimilarities between observations with special data structures as provided by packages **treeClust** and **mmpp**.

Other areas in applied statistics, e.g., to determine optimal designs, suitable sample sizes or perform uncertainty and sensitivity analysis are covered by packages **ALTopt**, **PracTools** and **mtk**. Diagnostic tools for approximate Bayesian computation are provided by package **abctools**; tools for dealing with multilabel datasets are contained in package **mldr**. Visualization methods for pairwise comparisons, which are for example used when comparing algorithms in machine learning, are given in package **SRCS**.

In addition new areas of application of R are seized, e.g., by package **rivr** which provides the tools for undergraduate and graduate courses in open-channel hydraulics. Infrastructure in the area of numerical mathematics for evaluating the hypergeometric function is given in package **hypergeo** and tools to help with coding in R are contained in package **GUIProfiler** for profiling R code.

Overall this collection of contributed research articles indicates the wide range of statistical methods from different areas currently covered by extension packages for R and the increasing possible use of R in new areas of application.

In addition the News and Notes section contains news by the R Consortium and the R Foundation providing some background information to the setting up of the Consortium and its interaction with the R Foundation. In addition a conference report on useR! 2015, the annual international R user conference, which took place from June 30 until July 3, 2015 in Aalborg, Denmark and attracted 660 participants from 42 countries is given. Furthermore this section contains the usual updates on the Bioconductor project, changes in R itself and CRAN.

I hope you enjoy the issue.

Bettina Grün
Bettina.Gruen@jku.at

Fitting Conditional and Simultaneous Autoregressive Spatial Models in **hglm**

by Moudud Alam, Lars Rönnergård, and Xia Shen

Abstract We present a new version (≥ 2.0) of the **hglm** package for fitting hierarchical generalized linear models (HGLMs) with spatially correlated random effects. CAR() and SAR() families for conditional and simultaneous autoregressive random effects were implemented. Eigen decomposition of the matrix describing the spatial structure (e.g., the neighborhood matrix) was used to transform the CAR/SAR random effects into an independent, but heteroscedastic, Gaussian random effect. A linear predictor is fitted for the random effect variance to estimate the parameters in the CAR and SAR models. This gives a computationally efficient algorithm for moderately sized problems.

Introduction

We present an algorithm for fitting spatial generalized linear models with conditional and simultaneous autoregressive (CAR & SAR; Besag, 1974; Cressie, 1993) random effects. The algorithm completely avoids the need to differentiate the spatial correlation matrix by transforming the model using an eigen decomposition of the precision matrix. This enables the use of already existing methods for hierarchical generalized linear models (HGLMs; Lee and Nelder, 1996) and this algorithm is implemented in the latest version (≥ 2.0) of the **hglm** package (Rönnergård et al., 2010).

The **hglm** package, up to version 1.2-8, provides the functionality for fitting HGLMs with uncorrelated random effects using an extended quasi likelihood method (EQL; Lee et al., 2006). The new version includes a first order correction of the fixed effects based on the current EQL fitting algorithm, which is more precise than EQL for models having non-normal outcomes (Lee and Lee, 2012). Similar to the h-likelihood correction of Noh and Lee (2007), it corrects the estimates of the fixed effects and thereby also reduces potential bias in the estimates of the dispersion parameters for the random effects (variance components). The improvement in terms of reduced bias is substantial for models with a large number of levels in the random effect (Noh and Lee, 2007), which is often the case for spatial generalized linear mixed models (GLMMs).

Earlier versions of the package allow modeling of the dispersion parameter of the conditional mean model, with fixed effects, but not the dispersion parameter(s) of the random effects. The current implementation, however, enables the user to specify a linear predictor for each dispersion parameter of the random effects. Adding this option was a natural extension to the package because it allowed implementation of our proposed algorithm that fits CAR and SAR random effects.

Though the CAR/SAR models are widely used for spatial data analysis there are not many software packages which can be used for their model fitting. GLMMs with CAR/SAR random effects are often fitted in a Bayesian way using BUGS software (e.g., WinBUGS; Lunn et al., 2000, or alike), which leads to extremely slow computation due to its dependence on Markov chain Monte Carlo simulations. The R package INLA (Martins et al., 2013) provides a relatively fast Bayesian computation of spatial HGLMs via Laplace approximation of the posterior distribution where the model development has focused on continuous domain spatial modeling (Lindgren and Rue, 2015) whereas less focus has been on discrete models including CAR and SAR. Recently, package spaMM (Rousset and Ferdy, 2014) was developed to fit spatial HGLMs but is rather slow even for moderately sized data. Here, we extend package **hglm** to also provide fast computation of HGLMs with CAR and SAR random effects and at the same time an attempt has been made to improve the accuracy of the estimates by including corrections for the fixed effects.

The rest of the paper is organized as follows. First we give an introduction to CAR and SAR structures and present the h-likelihood theory together with the eigen decomposition of the covariance matrix of the CAR random effects and show how it simplifies the computation of the model. Then we present the R code implementation, show the use of the implementation for two real data examples and evaluate the package using simulations. The last section concludes the article.

CAR and SAR structures in HGLMs

HGLMs with spatially correlated random effects are commonly used in spatial data analysis (Cressie, 1993; Wall, 2004). A spatial HGLM with Gaussian CAR random effects is given by

$$\begin{aligned} E(z_s|u_s) &= \mu_s, & s = 1, 2, \dots, n, \\ g(\mu_s) &= \eta_s = \mathbf{X}_s \boldsymbol{\beta} + \mathbf{Z}_s u_s, \end{aligned} \quad (1)$$

$$z_s|u_s \sim \text{Exponential Family}, \quad (2)$$

with $z_s|u_s \perp z_t|u_t, \forall s \neq t$ and

$$\mathbf{u} = (u_1, u_2, \dots, u_n)^T \sim N\left(\mathbf{0}, \Sigma = \tau (\mathbf{I} - \rho \mathbf{D})^{-1}\right), \quad (3)$$

where s represents a location identified by the coordinates $(x(s), y(s))$, $\boldsymbol{\beta}$ is a vector of fixed effects, \mathbf{X}_s is the fixed-effect design matrix for location s , u_s being the location specific random effects and \mathbf{Z}_s is a design matrix associated with u_s . While

$$\mathbf{u} = (u_1, u_2, \dots, u_n)^T \sim N\left(\mathbf{0}, \Sigma = \tau (\mathbf{I} - \rho \mathbf{D})^{-1} (\mathbf{I} - \rho \mathbf{D}^T)^{-1}\right) \quad (4)$$

gives a Gaussian SAR random effects structure. The \mathbf{D} matrix in Equations (3) and (4) is some known function of the location coordinates (see, e.g., Clayton and Kaldor, 1987) and ρ is often referred to as the spatial dependence parameter (Hodges, 2013). In application, \mathbf{D} is often a neighborhood matrix whose diagonal elements are all 0, and off-diagonal elements (s, t) are 1 if locations s and t are neighbors.

If $\Sigma = \tau \mathbf{I}$, i.e., there is no spatial correlation, then this model may be estimated by using usual software packages for GLMMs. With $\Sigma = \tau \mathbf{D}$, i.e., defining the spatial correlation directly via the \mathbf{D} matrix, the model can be fitted using earlier versions of the **hglm** package. However, for the general structure in Equations (3) and (4), estimation using explicit maximization of the marginal or profile likelihood involves quite advanced derivations as a consequence of partial differentiation of the likelihood including Σ with ρ and τ as parameters (see, e.g., Lee and Lee, 2012). In this paper, we show that by using eigen decomposition of Σ , we can modify an already existing R package, e.g., **hglm**, with minor programming effort, to fit a HGLM with CAR or SAR random effects.

h-likelihood estimation

In order to explain the specific model fitting algorithm implemented in **hglm**, we present a brief overview of the EQL algorithm for HGLMs (Lee and Nelder, 1996). First, we start with the standard HGLM containing only uncorrelated random effects. Then, we extend the discussion for CAR and SAR random effects. Though it is possible to have more than one random-effect term, in a HGLM, coming from different distributions among the conjugate distributions to GLM families, for presentational simplicity we consider only one random-effect in this section. The (log-)h-likelihood for a HGLM with independent random effects can be presented as

$$h = \sum_i \sum_j \left(\frac{(y_{i,j} \theta_{i,j} - b(\theta_{i,j}))}{\phi} + c(y_{i,j}, \phi) \right) + \sum_i \left(\frac{\psi v_i - b_R(v_i)}{\tau} + c_R(\tau) \right), \quad (5)$$

where, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, k$, θ is the canonical parameter of the mean model, ϕ is the dispersion parameter of the mean model, $b(\cdot)$ a function which satisfies $E(y_{i,j}|u_i) = \mu_{i,j} = b'(\theta_{i,j})$ and $Var(y_{i,j}|u_i) = \phi b''(\theta_{i,j}) = \phi V(\mu_{i,j})$, where $V(\cdot)$ is the GLM variance function. Furthermore, ϕ is the dispersion parameter of the mean model, $\theta_R(u_i) = v_i$ with $\theta_R(\cdot)$ being a so-called weak canonical link for u_i (leading to conjugate HGLMs) or an identity link for Gaussian random effects leading to GLMMs (Lee et al., 2006, pp. 3–4), $\psi = E(u_i)$, τ is the dispersion parameter, and b_R and c_R are some known functions depending on the distribution of u_i . Equation (5) also implies that h can be defined uniquely by using the means and the mean-variance relations of $y_{i,j}|u_i$ and the quasi-response ψ , allowing us representing it as a sum of two extended quasi-likelihoods (double EQL; Lee and Nelder, 2003) as

$$-2D = \sum_i \sum_j \left(\frac{d_{0,i,j}}{\phi} + \log(2\pi\phi V(y_{i,j}|u_i)) \right) + \sum_i \left(\frac{d_{1,i}}{\tau} + \log(2\pi\tau V_1(u_i)) \right), \quad (6)$$

where $d_{0,i,j}$ and $d_{1,i}$ are the deviance components of $y_{i,j}|u_i$ and ψ respectively and $Var(\psi) = \tau V_1(u_i)$. It is worth noting that $-2D$ is an approximation to h if $y|u$ or u (or both) belongs/belong to Binomial, Poisson, double-Poisson (or quasi-Poisson), Beta, or Gamma families. However, no such equivalent relation (even approximately) is known for the quasi-Binomial family.

In order to estimate the model parameters, Lee and Nelder (1996) suggested a two-step procedure in which the first, for fixed dispersion parameters ϕ and τ , h (or equivalently $-2D$) is maximized w.r.t. β and $v = \{v_i\}$. This maximization leads to an iteratively weighted least-square (IWLS) algorithm which solves

$$\mathbf{T}'_a \mathbf{\Delta}^{-1} \mathbf{T}_a \delta = \mathbf{T}'_a \mathbf{\Delta}^{-1} \mathbf{y}_a,$$

where $\mathbf{T}_a = \begin{pmatrix} \mathbf{X} & \mathbf{Z} \\ \mathbf{0} & \mathbf{I}_n \end{pmatrix}$, $\delta = \begin{pmatrix} \beta \\ v \end{pmatrix}$, $\mathbf{y}_a = \begin{pmatrix} \mathbf{y}_{0,a} \\ \mathbf{y}_{1,a} \end{pmatrix}$ with $\mathbf{y}_{0,a} = \left\{ \eta_{i,j} - (y_{i,j} - \mu_{i,j}) \frac{\partial \eta_{i,j}}{\partial \mu_{i,j}} \right\}$ and $\mathbf{y}_{1,a} = \left\{ v_i - (\psi_i - u_i) \frac{\partial v_i}{\partial u_i} \right\}$ are the vectors of GLM working responses for $y_{i,j}|u_i$ and v_i respectively, and $\mathbf{\Delta} = \mathcal{I}\mathbf{W}^{-1}$ with \mathcal{I} being a diagonal matrix whose first n diagonal elements are all equal to ϕ and the remaining k diagonal elements are all equal to τ and $\mathbf{W} = \text{diag}(\mathbf{w}_0, \mathbf{w}_1)$ where $\mathbf{w}_0 = \left\{ \left(\frac{\partial \mu_{i,j}}{\partial \eta_{i,j}} \right)^2 \frac{1}{V(\mu_{i,j})} \right\}$ and $\mathbf{w}_1 = \left\{ \left(\frac{\partial u_i}{\partial v_i} \right)^2 \frac{1}{V_1(u_i)} \right\}$.

In the second step, the following profile likelihood is maximized to estimate the dispersion parameters.

$$p_{\beta,v}(h) = \left(h - \frac{1}{2} \left| \frac{\partial^2 h}{\partial(\beta, v) \partial(\beta, v)^T} \right| \right)_{\beta=\hat{\beta}, v=\hat{v}}, \quad (7)$$

where $\hat{\beta}$ and \hat{v} are obtained from the fist step. One needs to iterate between the two steps until convergence. This procedure is often referred to as "HL(0,1)" (Lee and Lee, 2012) and is available in the R packages **spaMM** and **HGLMMM** (Molas and Lesaffre, 2011). Because there is no unified algorithm to maximize $p_{\beta,v}$ computer packages often carry out the maximization by using general purpose optimization routines, e.g., package **spaMM** uses the `optim` function.

A unified algorithm for estimating the variance components can be derived by using profile likelihood adjustment in Equation (6) instead of (5). This leads to maximizing

$$p_{\beta,v}(Q) = \left(-2D - \frac{1}{2} \left| \frac{\partial^2 (-2D)}{\partial(\beta, v) \partial(\beta, v)^T} \right| \right)_{\beta=\hat{\beta}, v=\hat{v}} \quad (8)$$

for estimating τ and ϕ . The corresponding score equation of the dispersion parameters, after ignoring the fact that $\hat{\beta}$ and \hat{v} are functions of ϕ and τ , can be shown (see, e.g., Lee and Nelder, 2001) to have the form of Gamma family GLMs with $d_{0,i,j}/(1-h_{i,j})$ and $d_{1,i}/(1-h_i)$ as the responses for ϕ and τ respectively, where $h_{i,j}$ and h_i are the hat values corresponding to $y_{i,j}$ and v_i in the first step, and $(1-h_{i,j})/2$ and $(1-h_i)/2$ as the respective prior weights. This procedure is often referred to as EQL (Lee et al., 2006) or DEQL (Lee and Nelder, 2003) and was the only available procedure in **hglm** ($\leq 1.2-8$). An advantage of this algorithm is that fixed effects can also be fitted in the dispersion parameters (Lee et al., 2006) without requiring any major change in the algorithm.

HL(0,1) and EQL were found to be biased, especially for binary responses when the cluster size is small (Lee and Nelder, 2001; Noh and Lee, 2007) and when τ is large in both Binomial and Poisson GLMMs. Several adjustments are suggested and explained in the literature (see, e.g., Lee and Nelder, 2001, 2003; Lee and Lee, 2012) to improve the performance of h-likelihood estimation. Among these alternative suggestions HL(1,1) is the most easily implementable and found to be computationally faster than the other alternatives (Lee and Lee, 2012). The HL(1,1) estimates the fixed effects, β , by maximizing $p_v(h)$ instead of h and can be implemented by adjusting the working response \mathbf{y}_a in the IWLS step, according to Lee and Lee (2012). In **hglm** (≥ 2.0), this correction for the β estimates has been implemented, through the `method = "EQL1"` option, but still $p_{\beta,v}(Q)$ is maximized for the estimation of the dispersion parameters, in order to make use of the unified algorithm for dispersion parameter estimates via Gamma GLMs. Table 1 shows available options for h-likelihood estimation (subject to the corrections mentioned above) in different R packages.

The above EQL method cannot be directly applied to HGLMs with CAR/SAR random effects because the resulting $p_{\beta,v}(Q)$ does not allow us to use the Gamma GLM to estimate the dispersion parameters, τ and ρ . In the following subsection we present a simplification which allows us to use the Gamma GLM for estimating τ and ρ .

	hglm	spaMM	HGLMMM
EQL	method = "EQL"	HLmethod = "EQL-"	-
HL(0,1)	-	HLmethod = "HL(0,1)"	LapFix = FALSE
HL(1,1)	a	HLmethod = "HL(1,1)"	LapFix = TRUE

^a By specifying method = "EQL1" in the **hglm** package, the HL(1,1) correction of the working response is applied in the EQL algorithm.

Table 1: Implemented h-likelihood methods in **hglm** compared to **spaMM** and **HGLMMM** packages.

Simplification of model computation

The main difference between an ordinary GLMM with independent random effects and CAR/SAR random effects models is that the random effects in the later cases are not independent. In the following we show that by using the eigen decomposition we can reformulate a GLMM with CAR or SAR random effects to an equivalent GLMM with independent but heteroscedastic random effects.

Lemma 1 Let $\omega = \{\omega_i\}_{i=1}^n$ be the eigenvalues of \mathbf{D} , and \mathbf{V} is the matrix whose columns are the corresponding orthonormal eigenvectors, then

$$\frac{1}{\tau}(\mathbf{I} - \rho\mathbf{D}) = \mathbf{V}\Lambda\mathbf{V}^T, \quad (9)$$

where Λ is diagonal matrix whose i th diagonal element is given by

$$\lambda_i = \frac{1 - \rho\omega_i}{\tau}. \quad (10)$$

Proof of Lemma 1

$$\begin{aligned} \mathbf{V}\Lambda\mathbf{V}^T &= \mathbf{V}\text{diag}\left\{\frac{1 - \rho\omega_i}{\tau}\right\}\mathbf{V}^T \\ &= \frac{1}{\tau}(\mathbf{V}\mathbf{V}^T - \rho\mathbf{V}\text{diag}\{\omega_i\}\mathbf{V}^T) \\ &= \frac{1}{\tau}(\mathbf{I} - \rho\mathbf{D}) \end{aligned} \quad (11)$$

It is worth noting that the relation between the eigenvalues of \mathbf{D} and Σ was already known as early as in [Ord \(1975\)](#) and was used to simplify the likelihood function of the Gaussian CAR model. Similarly, for simplification of the SAR model, we have

$$\frac{1}{\tau}(\mathbf{I} - \rho\mathbf{D})(\mathbf{I} - \rho\mathbf{D}^T) = \mathbf{V}\text{diag}\left\{\frac{(1 - \rho\omega_i)^2}{\tau}\right\}\mathbf{V}^T. \quad (12)$$

An anonymous referee has pointed out that a somewhat extended version of **Lemma 1**, which deals with simultaneous diagonalization of two positive semi-definite matrices ([Newcomb, 1969](#)), has already been used to simplify the computation of Gaussian response models with intrinsic CAR random effects, especially in a Bayesian context (see, e.g., [He et al., 2007](#), and the related discussions in [Hedges 2013](#), Chap. 5). Therefore, **Lemma 1** is not an original contribution of this paper, however, to the authors' knowledge, no software package has yet utilized this convenient relationship to fit any HGLM by using interconnected GLMs, as discussed below.

Re-arranging Equation (1), we have

$$\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta} + \tilde{\mathbf{Z}}\mathbf{u}^*, \quad (13)$$

where, $\boldsymbol{\eta} = \{\eta_s\}$, $\mathbf{X} = \{\mathbf{X}_s\}$, $\tilde{\mathbf{Z}} = \{\mathbf{Z}_s\}$ \mathbf{V} and $\mathbf{u}^* = \mathbf{V}^T\mathbf{u}$. With the virtue of **Lemma 1** and the properties of the multivariate normal distribution, we see that $\mathbf{u}^* \sim N(\mathbf{0}, \Lambda^{-1})$. This model can now easily be fitted using the **hglm** package in R. Further note that, for the CAR model, from **Lemma 1**, we have

$$\begin{aligned} \lambda_i &= \frac{1 - \rho\omega_i}{\tau} \\ &= \theta_0 + \theta_1\omega_i \end{aligned} \quad (14)$$

Option	Explanation
<code>rand.family = CAR(D = nbr)</code>	The random effect has conditional or simultaneous autoregressive covariance structure. Here <code>nbr</code> is a matrix provided by the user.
<code>rand.family = SAR(D = nbr)</code>	
<code>method = "EQL1"</code>	The first order correction of fixed effects (Lee and Lee, 2012) applied on the EQL estimates.
<code>rand.disp.X = X</code>	Linear predictor for the variance component of a random effect. The matrix <code>X</code> is provided by the user.
<code>rand.family = list(Gamma(),CAR())</code>	This option provides the possibility of having different distributions for the random effects.

Table 2: New options in the `hglm` function.

where $\theta_0 = 1/\tau$ and $\theta_1 = -\rho/\tau$. While for the SAR model,

$$\begin{aligned} \lambda_i &= \frac{(1 - \rho\omega_i)^2}{\tau} \\ \Rightarrow \sqrt{\lambda_i} &= \theta_0 + \theta_1\omega_i, \end{aligned} \quad (15)$$

where $\theta_0 = 1/\sqrt{\tau}$ and $\theta_1 = -\rho/\sqrt{\tau}$. Following Lee et al. (2006), we can use an inverse link for CAR or an inverse square-root link for SAR in a Gamma GLM with response $u_i^{*2}/(1 - h_i)$, where h_i is the corresponding hat value from the mean model (Equation (13)), $(1 - h_i)/2$ as the weight and the linear predictor given by Equations (14) and (15) to obtain an EQL estimate of θ_0 and θ_1 .

Implementation

The spatial models above are implemented in the `hglm` package (≥ 2.0) by defining new families, `CAR()` and `SAR()`, for the random effects. The “CAR” and “SAR” families allow the user to define spatial structures by specifying the `D` matrix. Using Lemma 1, these families are created using `D` with default link function “identity” for the Gaussian random effects `u`, and “inverse” and “inverse.sqrt” for the dispersion models (14) and (15). When the “CAR” or “SAR” family is specified for the random effects, the parameter estimates $\hat{\beta}$ and $\hat{\tau}$ are given in the output in addition to the other summary statistics of the dispersion models. The new input options in `hglm` are described in Table 2.

Throughout the examples below, we use Gaussian CAR random effects to introduce spatial correlation. However, one can also add additional independent non-Gaussian random effects along with the Gaussian CAR random effect. For example, an overdispersed count response can be fitted using a Poisson HGLM with an independent Gamma and Gaussian CAR random effects.

Examples and simulation study

In the `hglm` package vignette, we look at the improvement of EQL1 in comparison to EQL. Here, we focus on the precision of the parameter estimates with spatial HGLMs.

Poisson CAR & SAR model

We study the properties of the estimates produced by `hglm` using a simulation study built around the Scottish Lip Cancer example (see also the examples). We simulate data with the same `X` values, offset and neighborhood matrix as in the Scottish Lip Cancer example data. We use the true values of the parameters, after Lee and Lee (2012), as $(\text{intercept}, \beta_{fpp}, \tau, \rho) = (0.25, 0.35, 1.5, 0.1)$. The parameter estimates for 1000 Monte Carlo iterations are summarized in Table 3.

Both the EQL and the EQL1 correction are slightly biased for τ and ρ (Table 3) though the absolute amount of bias is small and may be negligible in practical applications. The EQL1 correction mainly improves the estimates of the intercept term. There were convergence problems for a small number of replicates, which was not surprising given the small number of observations ($n = 56$) and that the simulated value for the spatial autocorrelation parameter ρ connects the effects in the different Scottish districts rather weakly. Such convergence problems can be addressed by pre-specifying better starting values. For the converged estimates the bias was small and negligible.

Parameter	True value	Bias in the estimation methods			
		CAR		SAR	
		EQL [†]	EQL1 correction [†]	EQL [†]	EQL1 correction [†]
intercept	0.2500	0.0351*	0.0105	0.0541*	0.0307
β_{fpp}	0.3500	-0.0018*	-0.0004	-0.0034*	-0.0022
$1/\tau$	0.6667	0.0660*	0.0658*	n/a	n/a
$-\rho/\tau$	-0.0667	0.0118*	0.0119*	n/a	n/a
$1/\sqrt{\tau}$	0.8165	n/a	n/a	0.0393*	0.0370*
$-\rho/\sqrt{\tau}$	-0.0817	n/a	n/a	0.0037*	0.0049*
τ	1.5000	-0.0770*	-0.0760*	-0.0880*	-0.0860*
ρ	0.1000	-0.0247*	-0.0250*	-0.0097*	-0.0102*

* Significantly different from 0 at the 5% level.

† The estimates are the means from 1000 replicates.

Table 3: Average bias in parameter estimates in the simulation using the Scottish Lip Cancer example.

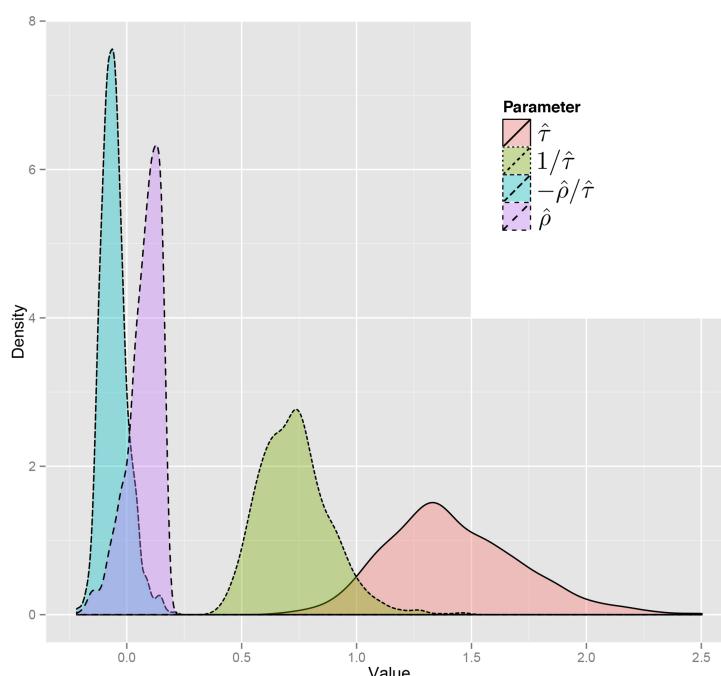


Figure 1: Density plot of the spatial variance-covariance parameter estimates from CAR models via "EQL1" over 1000 Monte Carlo simulations for the Scottish Lip Cancer example.

The simulation results also revealed that the distribution of $\hat{\rho}$ from CAR models is skewed (see Figure 1), which was also pointed out by Lee and Lee (2012). However, the distribution of $-\hat{\rho}/\hat{\tau}$ turned out to be less skewed than the distribution of $\hat{\rho}$. Similar observation was also found for SAR models. This suggests, we might draw any inference on spatial variance-covariance parameters in the transformed scale, θ_0 and θ_1 .

Computational efficiency

Fitting CAR and SAR models for large data sets could be computationally challenging, especially for the spatial variance-covariance parameters. Using our new algorithm in **hglm**, moderately sized problems can be fitted efficiently.

We re-sampled from the ohio data set (for more details on the data set see also the examples) for different number of locations, each with 10 replicates, executed on a single Intel® Xeon® E5520 2.27GHz CPU. In each replicate, the data was fitted using both **hglm** and **spaMM**, for the same model described above. The results regarding average computational time are summarized in Figure 2. **hglm** is clearly more usable for fitting larger sized data sets. In the package vignette, we also show the comparisons of the parameter estimates, where the "EQL" estimates from **hglm** are almost identical

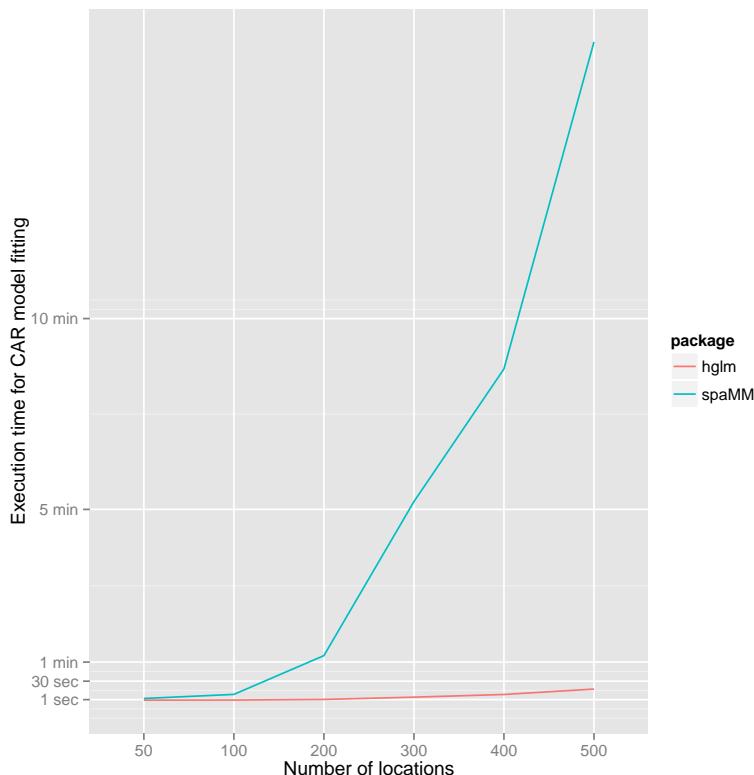


Figure 2: Comparison of the execution time for fitting CAR models using **hglm** and **spaMM**.

to the "EQL—" estimates from **spaMM**, with high correlation coefficients between the two methods: 1.0000, 0.9998, 0.9997 and 0.9999 for the residual variance, ρ , τ and the intercept, respectively.

Examples

Scottish Lip Cancer data set

Here we analyze the cancer data set (source: [Clayton and Kaldor, 1987](#)) from the **hglm** package. Calling `data(cancer)` loads a numeric vector `E` that represents the expected number of skin cancer patients in different districts in Scotland, a numeric vector `O` giving the corresponding observed counts, a numeric vector `Paff` giving proportion of people involved agriculture, farming, and fisheries, and matrix `D` giving the neighborhood structure among Scottish districts. Here we demonstrate how the data is fitted as a CAR model or a SAR model, using the **hglm** package with the EQL method.

```
> library(hglm)
> data(cancer)
> logE <- log(E)
> XX <- model.matrix(~ Paff)
> cancerCAR <- hglm(X = XX, y = O, Z = diag(56),
+                      family = poisson(),
+                      rand.family = CAR(D = nbr),
+                      offset = logE, conv = 1e-8,
+                      maxit = 200, fix.disp = 1)
> summary(cancerCAR)

Call:
hglm.default(X = XX, y = O, Z = diag(56), family = poisson(),
rand.family = CAR(D = nbr), conv = 1e-08, fix.disp = 1, offset = logE)

-----
MEAN MODEL
-----
```

Summary of the fixed effects estimates:

```
Estimate Std. Error t-value Pr(>|t|)
(Intercept) 0.26740   0.20732   1.290  0.20893
Paff        0.03771   0.01215   3.103  0.00471 **  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Note: P-values are based on 25 degrees of freedom
```

Summary of the random effects estimates:

```
Estimate Std. Error
[1,] 0.6407    1.0467
[2,] 0.5533    0.3829
[3,] 0.4124    0.5202  
...
NOTE: to show all the random effects, use print(summary(hglm.object),
      print.ranef = TRUE).
```

```
-----  
DISPERSION MODEL  
-----
```

NOTE: h-likelihood estimates through EQL can be biased.

Dispersion parameter for the mean model:
[1] 1

Model estimates for the dispersion term:

Link = log

Effects:
[1] 1

Dispersion = 1 is used in Gamma model on deviances to calculate the standard error(s).

Dispersion parameter for the random effects:
[1] 656.3

Dispersion model for the random effects:

Link = log

Effects:
. | Random1

	Estimate	Std. Error
1/CAR.tau	6.487	1.727
-CAR.rho/CAR.tau	-1.129	0.303

CAR.tau (estimated spatial variance component): 0.1542
CAR.rho (estimated spatial correlation): 0.174

Dispersion = 1 is used in Gamma model on deviances to calculate the standard error(s).

EQL estimation converged in 10 iterations.

In the above output provided by the `summary()` method, fixed effects estimates are given under MEAN MODEL, and the dispersion parameter estimates are given under DISPERSION MODEL. Here, we have only one random effects term that has a CAR structure, and the corresponding parameter estimates, $\hat{\theta}_0 = 6.487$ and $\hat{\theta}_1 = -1.129$, are given under `. | Random1`. However, these are not the natural parameters of the CAR model (see Section [Simplification of model computation](#)) therefore the estimates of the natural dispersion parameters are given just after them which are, in this case, $\hat{\tau} = 0.15$ and $\hat{\rho} = 0.17$.

Because the \mathbf{D} matrix was a neighborhood matrix (consisting of 0's and 1's), the results imply that the partial correlation of the random effect for any two neighboring districts, given the same for all other districts, is 0.17.

Furthermore, the value 656.3 for the dispersion of the random effects given in the output above is an overall variance of \mathbf{u}^* in Equation (13). This output is usually not of interest to the user and the main results are contained in $\hat{\tau}$ and $\hat{\rho}$.

```
> cancerSAR <- hglm(X = XX, y = 0, Z = diag(56), family = poisson(),
+                      rand.family = SAR(D = nbr), offset = logE,
+                      conv = 1e-08, fix.disp = 1)
> summary(cancerSAR)
```

```
Call:
hglm.default(X = XX, y = 0, Z = diag(56), family = poisson(),
              rand.family = SAR(D = nbr), conv = 1e-08, fix.disp = 1, offset = logE)

-----
```

MEAN MODEL

Summary of the fixed effects estimates:

	Estimate	Std. Error	t-value	Pr(> t)
(Intercept)	0.19579	0.20260	0.966	0.34241
Paff	0.03637	0.01165	3.122	0.00425 **

Signif. codes:	0 ***	0.001 **	0.01 *	0.05 .
	'***'	'**'	'*'	'.'
	0.1	' '	' '	' 1

Note: P-values are based on 27 degrees of freedom

Summary of the random effects estimates:

	Estimate	Std. Error
[1,]	0.7367	1.0469
[2,]	0.6336	0.3930
[3,]	0.4537	0.5784
...		

NOTE: to show all the random effects, use print(summary(hglm.object)),
print.ranef = TRUE).

```
-----
```

DISPERSION MODEL

```
-----
```

NOTE: h-likelihood estimates through EQL can be biased.

Dispersion parameter for the mean model:
[1] 1

Model estimates for the dispersion term:

Link = log

Effects:
[1] 1

Dispersion = 1 is used in Gamma model on deviances to calculate the standard error(s).

Dispersion parameter for the random effects:
[1] 16.3

Dispersion model for the random effects:

Link = log

```

Effects:
. | Random1
          Estimate Std. Error
1/sqrt(SAR.tau)      2.7911   0.4058
-SAR.rho/sqrt(SAR.tau) -0.4397   0.0822
SAR.tau (estimated spatial variance component): 0.1284
SAR.rho (estimated spatial correlation): 0.1575

```

Dispersion = 1 is used in Gamma model on deviances to calculate the standard error(s).

EQL estimation converged in 12 iterations.

For the CAR model, the `hglm` estimates are exactly the same as those labelled as "PQL" estimates in Lee and Lee (2012). To get the EQL1 correction the user has to add the option `method = "EQL1"` and the `hglm` function gives similar results to those reported in Lee and Lee (2012), e.g., their HL(1,1) estimates were $(\text{intercept}, \beta_{fpp}, \tau, \rho) = (0.238, 0.0376, 0.155, 0.174)$ whereas our "EQL1" correction gives $(0.234, 0.0377, 0.156, 0.174)$ (see also Section "Fitting a spatial Markov Random Field model using the CAR family" in the package vignette). A minor difference to the "EQL1" result appears because Lee and Lee (2012) used the HL(1,1) modification to their HL(0,1) whereas we apply such a correction directly to EQL which is slightly different from HL(0,1) (see Table 1).

Ohio elementary school grades data set

We analyze a data set consisting of the student grades of 1,967 Ohio Elementary Schools during the year 2001–2002. The data set is freely available on the internet (URL <http://www.spatial-econometrics.com/>) as a web supplement to LeSage and Pace (2009) but was not analyzed therein. The shape files were downloaded from <http://www.census.gov/cgi-bin/geo/shapefiles2013/main> and the districts of 1,860 schools in these two files could be connected unambiguously. The data set contains information on, for instance, school building ID, Zip code of the location of the school, proportion of passing on five subjects, number of teachers, number of students, etc. We regress the median of 4th grade proficiency scores, \mathbf{y} , on an intercept, based on school districts. The statistical model is given as

$$y_{i,j} = \mu + v_j + \epsilon_{i,j}, \quad (16)$$

where $i = 1, 2, \dots, 1860$ (observations), $j = 1, 2, \dots, 616$ (districts), $\epsilon_{i,j} \sim N(0, \sigma_e^2)$, $\{v_j\} = \mathbf{v} \sim N(\mathbf{0}, \tau (\mathbf{I} - \rho \mathbf{W})^{-1})$ and $\mathbf{W} = \{w_{p,q}\}_{p,q=1}^{616}$ is a spatial weight matrix (i.e., the neighborhood matrix). We construct $w_{p,q} = 1$ if the two districts p and q are adjacent, and $w_{p,q} = 0$ otherwise.

The above choice of constructing the weight matrix is rather simple. Because the aim of this paper is to demonstrate the use of `hglm` for fitting spatial models rather than drawing conclusions from real data analysis, we skip any further discussion on the construction of the weight matrix. Interested, readers are referred to LeSage and Pace (2009) for more discussion on the construction of the spatial weight matrices. With the spatial weight matrix defined as above, we can estimate model (16) by using our `hglm` package in the following way.

```

> ## load the data object 'ohio'
> data(ohio)
>
> ## fit a CAR model for the median scores of the districts
> X <- model.matrix(MedianScore ~ 1, data = ohioMedian)
> Z <- model.matrix(~ 0 + district, data = ohioMedian)
> ohioCAR <- hglm(y = ohioMedian$MedianScore, X = X, Z = Z,
+                     rand.family = CAR(D = ohioDistrictDistMat))
> summary(ohioCAR)

Call:
hglm.default(X = X, y = ohioMedian$MedianScore, Z = Z,
              rand.family = CAR(D = ohioDistrictDistMat))

-----
MEAN MODEL
-----
```

Summary of the fixed effects estimates:

```
Estimate Std. Error t-value Pr(>|t|)
(Intercept) 72.429     0.819   88.44 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Note: P-values are based on 1566 degrees of freedom
```

Summary of the random effects estimates:

```
Estimate Std. Error
[1,] -21.433    11.071
[2,] -17.890    10.511
[3,] -4.537     7.844
...
NOTE: to show all the random effects, use print(summary(hglm.object),
      print.ranef = TRUE).
```

```
-----
DISPERSION MODEL
-----
```

NOTE: h-likelihood estimates through EQL can be biased.

Dispersion parameter for the mean model:
[1] 190.5

Model estimates for the dispersion term:

Link = log

Effects:

	Estimate	Std. Error
	5.2498	0.0357

Dispersion = 1 is used in Gamma model on deviances to calculate the standard error(s).

Dispersion parameter for the random effects:
[1] 1.01

Dispersion model for the random effects:

Link = log

Effects:

```
. | Random1
      Estimate Std. Error
1/CAR.tau       0.0097   8e-04
-CAR.rho/CAR.tau -0.0011   2e-04
CAR.tau (estimated spatial variance component): 103.6
CAR.rho (estimated spatial correlation): 0.1089
```

Dispersion = 1 is used in Gamma model on deviances to calculate the standard error(s).

EQL estimation converged in 5 iterations.

The estimated spatial correlation parameter among school districts is 0.109. We can obtain fitted values from the CAR model and predict the school districts without any observations. The following codes perform such prediction and the results are visualized in Figure 3(B). We remove the estimate of Lake Erie, as estimation for an uninhabited region is meaningless.

```
> ## extract districts from the map data
```

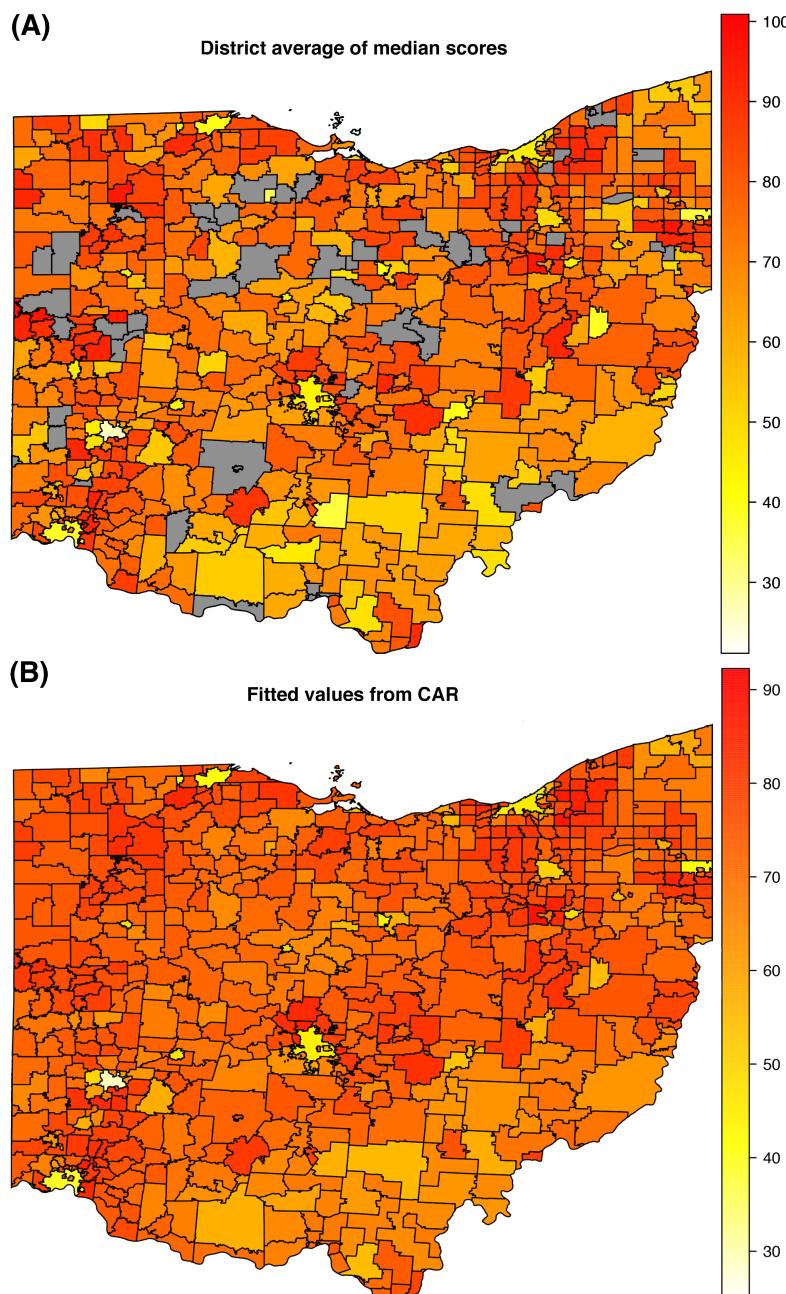


Figure 3: Observed (A) and predicted (B) median 4th grade proficiency scores of the school districts in Ohio. Districts without any observations are displayed in gray.

```
> districtShape <- as.numeric(substr(as.character(ohioShape@data$UNSDIDFP), 3, 7))
>
> ## calculate fitted values from the CAR model
> CARfit <- matrix(ohioCAR$ranef + ohioCAR$fixef,
+                     dimnames = list(rownames(ohioDistrictDistMat), NULL))
> ohioShape@data$CAR <- CARfit[as.character(districtShape),]
> is.na(ohioShape@data$CAR[353]) <- TRUE # remove estimate of Lake Erie
>
> ## visualize the results
> spplot(ohioShape, zcol = "CAR", main = "Fitted values from CAR",
+         col.regions = heat.colors(1000)[1000:1], cuts = 1000)
```

A `predict()` method is not available because predicting spatially correlated random effects for autoregressive models requires re-fitting the whole model. Thus standard kriging cannot be used because the covariance structure changes if the neighborhood matrix is altered, while keeping τ and ρ

unchanged. Instead, the fitted model needs to include the entire neighborhood matrix with districts having missing data as well. Consequently, the incidence matrix Z has more columns than rows. This method of predicting random effects is frequently used in animal breeding applications (Henderson, 1984) but, to our knowledge, has not been applied to spatial autoregressive models previously.

In the example above, `ohioMedian$district` has 616 levels and 54 of the districts have no records (Figure 3(A)). The incidence matrix Z , created using the `model.matrix` function, therefore has 616 columns and 54 of these are columns of zeros. Hence, there are 616 levels in the fitted spatial random effect giving predictions for the districts without records.

Conclusion

The `hglm` package is one of few non-Bayesian packages on CRAN to fit spatial HGLMs, where the fixed and random effects are estimated simultaneously. We have shown how the HGLM framework, allowing linear predictors to model variance components, can be exploited to fit CAR and SAR models. This gives a computationally efficient algorithm for moderately sized problems (number of locations < approx. 5000).

Acknowledgement

X. Shen was supported by a Swedish Research Council grant (No. 2014-371).

Bibliography

- J. Besag. Spatial interaction and the statistical analysis of lattice systems (with discussion). *Journal of the Royal Statistical Society, Series B*, 36:192–236, 1974. [p5]
- D. Clayton and J. Kaldor. Empirical bayes estimation of age-standardized relative risk for use in disease mapping. *Biometrics*, 43:671–681, 1987. [p6, 11]
- N. A. C. Cressie. *Statistics for Spatial Data*. Wiley, New York, revised edition, 1993. [p5, 6]
- Y. He, J. S. Hodges, and B. P. Carlin. Re-considering the variance parameterization in multiple precision models. *Bayesian Analysis*, 2:529–556, 2007. [p8]
- C. R. Henderson. *Applications of Linear Models in Animal Breeding*. University of Guelph, Guelph, Ontario, 1984. [p17]
- J. S. Hodges. *Richly Parametrized Linear Models: Additive Linear and Time Series and Spatial Models Using Random Effects*. Chapman and Hall/CRC, Boca Raton, 2013. [p6, 8]
- W. Lee and Y. Lee. Modifications of REML algorithm for HGLMs. *Statistics and Computing*, 22:959–966, 2012. [p5, 6, 7, 9, 10, 14]
- Y. Lee and J. A. Nelder. Hierarchical generalized linear models (with discussion). *Journal of the Royal Statistical Society, Series B*, 58:619–678, 1996. [p5, 6, 7]
- Y. Lee and J. A. Nelder. Hierarchical generalised linear models: A synthesis of generalised linear models, random effect models and structured dispersions. *Biometrika*, 88:987–1006, 2001. [p7]
- Y. Lee and J. A. Nelder. Extended-REML estimators. *Journal of Applied Statistics*, 30:845–846, 2003. [p6, 7]
- Y. Lee, J. A. Nelder, and Y. Pawitan. *Generalized Linear Models with Random Effects: Unified Analysis via H-Likelihood*. Chapman & Hall/CRC, Boca Raton, 2006. [p5, 6, 7, 9]
- J. LeSage and R. Pace. *Introduction to Spatial Econometrics*. Chapman & Hall/CRC, Boca Raton, 2009. [p14]
- F. Lindgren and H. Rue. Bayesian spatial modelling with R-INLA. *Journal of Statistical Software*, 63(19):1–25, 2015. URL <http://www.jstatsoft.org/v63/i19>. [p5]
- D. J. Lunn, J. A. Thomas, N. Best, and D. Spiegelhalter. WinBUGS – a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing*, 10:325–337, 2000. [p5]

- T. G. Martins, D. Simpson, F. Lindgren, and H. Rue. Bayesian computing with INLA: New features. *Computational Statistics & Data Analysis*, 67:68–83, 2013. [p5]
- M. Molas and E. Lesaffre. Hierarchical generalized linear models: The R package HGLMMM. *Journal of Statistical Software*, 39(13):1–20, 2011. URL <http://www.jstatsoft.org/v39/i13>. [p7]
- R. Newcomb. On the simultaneous diagonalization of two semi-definite matrices. *Quarterly of Applied Mathematics*, 19:144–146, 1969. [p8]
- M. Noh and Y. Lee. REML estimation for binary data in GLMMs. *Journal of Multivariate Analysis*, 98: 896–915, 2007. [p5, 7]
- K. Ord. Estimation methods for models of spatial interaction. *Journal of the American Statistical Association*, 70:120–126, 1975. [p8]
- L. Rönnegård, X. Shen, and M. Alam. hglm: A package for fitting hierarchical generalized linear models. *The R Journal*, 2(2):20–28, 2010. [p5]
- F. Rousset and J.-B. Ferdy. Testing environmental and genetic effects in the presence of spatial autocorrelation. *Ecography*, 37(8):781–790, 2014. [p5]
- M. M. Wall. A close look at the spatial structure implied by the CAR and SAR models. *Journal of Statistical Planning and Inference*, 121:311–324, 2004. [p6]

Moudud Alam

*Statistics, School of Technology and Business Studies
Dalarna University, Sweden
maa@du.se*

Lars Rönnegård

*Statistics, School of Technology and Business Studies
Dalarna University, Sweden
and
Department of Animal Breeding and Genetics
Swedish University of Agricultural Sciences, Sweden
and
Division of Computational Genetics
Department of Clinical Sciences
Swedish University of Agricultural Sciences, Sweden
lrn@du.se*

Xia Shen

*Department of Medical Epidemiology and Biostatistics
Karolinska Institutet, Sweden
and
MRC Human Genetics Unit
MRC Institute of Genetics and Molecular Medicine
University of Edinburgh, United Kingdom
xia.shen@ki.se*

VSURF: An R Package for Variable Selection Using Random Forests

by Robin Genuer, Jean-Michel Poggi and Christine Tuleau-Malot

Abstract This paper describes the R package **VSURF**. Based on random forests, and for both regression and classification problems, it returns two subsets of variables. The first is a subset of important variables including some redundancy which can be relevant for interpretation, and the second one is a smaller subset corresponding to a model trying to avoid redundancy focusing more closely on the prediction objective. The two-stage strategy is based on a preliminary ranking of the explanatory variables using the random forests permutation-based score of importance and proceeds using a stepwise forward strategy for variable introduction. The two proposals can be obtained automatically using data-driven default values, good enough to provide interesting results, but strategy can also be tuned by the user. The algorithm is illustrated on a simulated example and its applications to real datasets are presented.

Introduction

Variable selection is a crucial issue in many applied classification and regression problems (see e.g. [Hastie et al., 2001](#)). It is of interest for statistical analysis as well as for modelisation or prediction purposes to remove irrelevant variables, to select all important ones or to determine a sufficient subset for prediction. These main different objectives from a statistical learning perspective involve variable selection to simplify statistical problems, to help diagnosis and interpretation, and to speed up data processing.

[Genuer et al. \(2010b\)](#) proposed a variable selection method based on random forests ([Breiman, 2001](#)), and the aim of this paper is to describe the associated R package called **VSURF** and to illustrate its use on real datasets. The stable version of the package is available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=VSURF>.¹ In order to make the paper self-contained, the description of the variable selection method is provided. A simulated toy example and a classical real dataset are processed using **VSURF**. In addition, two real examples in a high-dimensional setting, not previously addressed by the authors, are used to illustrate the value of the strategy and the effectiveness of the R package.

Introduced by [Breiman \(2001\)](#), random forests (abbreviated RF in the sequel) are an attractive nonparametric statistical method to deal with these problems, since they require only mild conditions on the model supposed to have generated the observed data. Indeed, since RF are based on decision trees and use aggregation ideas, they allow us to consider in an elegant and versatile framework different models and problems, namely regression, two-class and multiclass classifications.

Considering a learning set $L = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$, supposed to consist of independent observations of the random vector (X, Y) , we distinguish as usual the predictors (or explanatory variables), collected in the vector $X = (X^1, \dots, X^p)$ where $X \in \mathbb{R}^p$, from the explained variable $Y \in \mathcal{Y}$ where Y is either a class label for classification problems or a numerical response for regression ones. Let us recall that a classifier t is a mapping $t : \mathbb{R}^p \rightarrow \mathcal{Y}$ while the regression function naturally corresponds to the function s when we suppose that $Y = s(X) + \varepsilon$ with $E[\varepsilon|X] = 0$. Then random forests provide estimators of either the Bayes classifier, which minimizes the classification error $P(Y \neq t(X))$, or the regression function.

The CART (Classification and Regression Trees) method defined by [Breiman et al. \(1984\)](#) is a well-known way to design optimal single binary decision trees. It proceeds by performing first a growing step and then a pruning one. The principle of random forests is to aggregate many binary decision trees coming from two random perturbation mechanisms: the use of bootstrap samples of L instead of L and the random choice of a subset of explanatory variables at each node instead of all of them. There are two main differences with respect to CART trees: first, in the growing step, at each node, a fixed number of input variables are randomly chosen and the best split is calculated only among them and, second, no pruning step is performed so all the trees of the forest are maximal trees. The RF algorithm is a very popular machine learning algorithm and appears to be powerful in a lot of different applications, see for example [Verikas et al. \(2011\)](#) and [Boulesteix et al. \(2012\)](#) for recent surveys.

Several implementations of these methods are available. Focusing on R packages we must mention **rpart** ([Therneau et al., 2015](#)) for CART, **randomForest** ([Liaw and Wiener, 2002](#)) for RF, **party** ([Hothorn](#)

¹The current development version of the package is also available at <https://github.com/robingenuer/VSURF>.

et al., 2006) for CART and RF (through the function `cforest`) and `ipred` (Peters et al., 2002) for bagging (Breiman, 1996), a closely related method cited here for the sake of completeness. In this paper, we use the `randomForest` procedure, which is based on the initial contribution of Breiman and Cutler (2004). We will concentrate on the prediction performance of RF focusing on the out-of-bag (OOB) error (see Breiman, 2001) and on the quantification of the variable importance (VI in the sequel) which are key ingredients for our variable selection strategy. For a general discussion about variable importance, see Azen and Budescu (2003). In the random forests framework, one of the most widely used scores of importance of a given variable is the increase in mean of the error of a tree (mean square error (MSE) for regression and misclassification rate for classification) in the forest when the observed values of this variable are randomly permuted in the OOB samples (see Archer and Kimes, 2008).

Strobl et al. (2007) showed that VI scores are biased towards correlated variables, and Strobl et al. (2008) proposed an alternative permutation scheme as a solution, which, however, increases the computation cost. This seems to be especially critical for high-dimensional problems with strongly correlated predictors. Nevertheless, our previous experiments (Genuer et al., 2010b) on variable selection and, more recently, the theoretical study of Gregorutti et al. (2013) show that, in some situations, the VI scores are biased towards uncorrelated variables. Additional theoretical results and experiments are needed to more deeply understand these phenomena, but this is out of the scope of the paper.

A lot of variable selection procedures are based on the combination of variable importance for ranking and model estimation to generate, evaluate and compare a family of models, i.e., in particular in the family of “wrapper” methods (Kohavi and John, 1997; Guyon and Elisseeff, 2003) which include the prediction performance in the score calculation, for which a lot of methods can be cited. We choose to highlight one of them which is widely used and close to our procedure. Díaz-Uriarte and Alvarez De Andres (2006) propose a strategy based on recursive elimination of variables. At the beginning, they compute RF variable importance and then, at each step, eliminate iteratively the 20% of the variables having the smallest importance and build a new forest with the remaining variables. The final set of variables is selected by minimizing over the obtained forests, the OOB error rate defined by:

$$err_{OOB} = \begin{cases} \frac{1}{n} \text{Card} \{i \in \{1, \dots, n\} \mid y_i \neq \hat{y}_i\} & \text{in the classification framework} \\ \frac{1}{n} \sum_{i \in \{1, \dots, n\}} (y_i - \hat{y}_i)^2 & \text{in the regression framework} \end{cases}$$

where \hat{y}_i is the aggregation of the predicted values by trees t for which (x_i, y_i) belongs to the associated OOB sample (data not included in the bootstrap sample used to construct t). The proportion of variables to eliminate is an arbitrary parameter of their method and does not depend on the data. Let us remark that we propose an heuristic strategy which does not depend on specific model hypotheses, but which is based on data-driven thresholds to take decisions.

This topic of variable selection still continues to be of interest. Indeed recently Hapfelmeier and Ulm (2012) propose a new variable selection approach using random forests and, more generally, Cadena et al. (2013) describe and compare different approaches in a survey paper.

Some packages are available to cope with variable selection problems. Let us cite, for classification problems the R package `Boruta`, described in Kursa and Rudnicki (2010), which aims at finding all relevant variables using a random forest classification algorithm which iteratively removes the variables using a statistical test. The R package `varSelRF`, described in Díaz-Uriarte (2007), implements the previously described method for selecting very small sets of genes in the context of classification. The R package `ofw` (Lê Cao and Chabrier, 2008), also dedicated to the context of classification, selects relevant variables based on the application of supervised multiclass classifiers such as CART or support vector machines. The R package `spikeSlabGAM` implements Bayesian variable selection via regularized estimation in additive mixed models (Scheipl, 2011). Dedicated to the biomarker identification in the life sciences, the R package `BioMark` implements two meta-statistics for variable selection (Wehrens et al., 2012): the first sets a data-dependent selection threshold for significance, which is useful when two groups are compared, and the second, more general one, uses repeated subsampling and selects the model coefficients remaining consistently important.

The paper is organized as follows. After this introduction, we present the general variable selection strategy. We then describe how to use package `VSURF` on a simple simulated dataset. Finally, we examine three real-life examples to illustrate the method in action in high-dimensional situations as well as in a standard one.

The strategy

Objectives

In [Genuer et al. \(2010b\)](#) we distinguished two variable selection objectives referred to as interpretation and prediction.

Even if this distinction can be a little bit confusing since we use for both objectives the same criterion related to prediction performance, the idea is the following. The first objective, called interpretation, is to find important variables highly related to the response variable, even with some redundancy, possibly high. The second one, namely prediction, is to find a smaller number of variables with very low redundancy and sufficient for a good enough prediction of the response variable. The terminology used here can be misunderstood since usually for interpretation, one usually looks for parsimony but in many situations one may often want to identify all predictor variables associated with the response to interpret correctly the relation. We thank one anonymous reviewer for raising this issue and we detail an example in the next paragraph to clarify the difference.

A typical situation illustrates the distinction between the two kinds of variable selection. Let us consider a high-dimensional ($n \ll p$) classification problem for which the predictor variables are associated to a pixel in an image or a voxel in a 3D-image as in fMRI brain activity classification problems (see e.g. [Genuer et al., 2010a](#)). In such situations, it is supposed that a lot of variables are useless and that there exist unknown groups of highly correlated predictors corresponding to brain regions involved in the response to a given stimulation. The two distinct objectives about variable selection can be of interest. Finding all the important variables highly related to the response variable is useful for interpretation, since it corresponds to the determination of entire regions in the brain or a full parcel in an image. By contrast, finding a small number of variables sufficient for good prediction allows to get the most discriminant variables within the previously highlighted regions. For a more formal approach to this distinction, see also the interesting paper [Nilsson et al. \(2007\)](#).

Principle

The principle of the two-steps algorithm is the following. First, we rank the variables according to a variable importance measure and the unimportant ones are eliminated. Second, we provide two different subsets obtained either by considering a collection of nested RF models and selecting the variables of the most accurate one, or by introducing sequentially the sorted variables.

Since the quantification of the variable importance is crucial for our procedure, let us recall the definition of RF variable importance. For each tree t of the forest, consider the associated OOB_t sample (data not included in the bootstrap sample used to construct t). Denote by $errOOB_t$ the error (MSE for regression and misclassification rate for classification) of a single tree t on this OOB_t sample. Now, randomly permute the values of X^j in OOB_t to get a perturbed sample denoted by \widetilde{OOB}_t^j and compute \widetilde{errOOB}_t^j , the error of predictor t on the perturbed sample. Variable importance of X^j is then equal to:

$$VI(X^j) = \frac{1}{ntree} \sum_t \left(\widetilde{errOOB}_t^j - errOOB_t \right),$$

where the sum is over all trees t of the RF and $ntree$ denotes the number of trees of the RF. Notice that we use this definition of importance and not the normalized one. Indeed, instead of considering (as mentioned in [Breiman and Cutler, 2004](#)) that the raw VI are independent replicates, normalizing them and assuming normality of these scores, we prefer a fully data-driven solution. This is a key point of our strategy: we prefer to estimate directly the variability of importance across repetitions of forests instead of using normality when $ntree$ is sufficiently large, which is only valid under some specific conditions. Those conditions are difficult to check since their validity depends heavily on tuning parameters and problem peculiarities, so data-driven normalization prevents some misspecified asymptotic behavior.

Another useful argument, which provides the rationale for this kind of variable selection procedure based on a classical stepwise method combined with the use of a VI measure, is that a variable not included in the underlying true model has a null “theoretical” importance. In a recent paper [Gregorutti et al. \(2013\)](#) theoretically state that, in the case of additive models, irrelevant variables have null “theoretical” VI. A similar result for the mean decrease impurity index (not used in this paper) has been proven by [Louppe et al. \(2013\)](#).

Detailed strategy

Let us now describe more precisely our two-steps procedure. Note that each RF is typically built using $n_{tree} = 2000$ trees.

- Step 1. Preliminary elimination and ranking:

- Rank the variables by sorting the VI (averaged over typically 50 RF runs) in descending order.
- Eliminate the variables of small importance (let m denote the number of remaining variables).

More precisely, starting from this order, consider the ordered sequence of the corresponding standard deviations (sd) of VI and use it to estimate a threshold value for VI. Since variability of VI is larger for true variables compared to useless ones, the threshold value is given by an estimation of the VI standard deviation of the useless variables. This threshold is set to the minimum prediction value given by a CART model where the Y are the sd of the VI and the X are the ranks.

Then only the variables with an averaged VI exceeding this threshold are retained.

- Step 2. Variable selection:

- For *interpretation*: construct the nested collection of RF models involving the k first variables, for $k = 1$ to m and select the variables involved in the model leading to the smallest OOB error. This leads to consider m' variables.

More precisely, we compute OOB error rates of RF (averaged typically over 25 runs) of the nested models starting from the one with only the most important variable, and ending with the one including all important variables previously kept. Ideally, the variables of the model leading to the smallest OOB error are selected. In fact, in order to deal with instability, we use a classical trick: we select the smallest model with an OOB error less than the minimal OOB error augmented by its standard deviation (based on the same 25 runs).

- For *prediction*: starting with the ordered variables retained for interpretation, construct an ascending sequence of RF models, by invoking and testing the variables in a stepwise way. The variables of the last model are selected.

More precisely, the sequential variable introduction is based on the following test: a variable is added only if the error decrease is larger than a threshold. The idea is that the OOB error decrease must be significantly greater than the average variation obtained by adding noisy variables. The threshold is set to the mean of the absolute values of the first order differentiated OOB errors between the model with m' variables and the one with m variables:

$$\frac{1}{m - m'} \sum_{j=m'}^{m-1} |errOOB(j+1) - errOOB(j),| \quad (1)$$

where $errOOB(j)$ is the OOB error of the RF built using the j most important variables.

Comments

In addition to the detailed strategy, let us give some additional comments. Regarding the first step of our procedure, our previous simulation study in [Genou et al. \(2010b\)](#) shows that both VI level and variability are larger for relevant variables.

We assume that the stabilized ranking performed in the first step of our procedure will not change after the elimination step.

The use of CART to find the threshold is of course not strictly necessary and can be replaced by many other strategies but it is interesting in our case because the idea is typically to find a constant on a large interval. Since CART fits a piece-wise constant function, this desired constant is the threshold for VI and is defined as the minimum prediction value given by a CART model fitting the curve of VI standard deviations. It is obtained automatically by the CART strategy whereas the user needs to select some window parameter if, for example, a simple smoothing method is used.

Note that the threshold value is based on VI standard deviations while the effective thresholding is performed on the VI mean. In general, this rule is conservative and leads to keep more variables than necessary, postponing to the next step a more parsimonious choice.

In addition, we note that several implementation choices have been made: **randomForest** for RF fitting and VI calculation (repeated 50 times to quantify variability) and **rpart** for estimating the VI

variability level associated with irrelevant variables leading to a convenient threshold value. Of course other choices would be possible, for example, using the function `cforest` of the package **party** to implement the same scheme of variable selection.

The next section describes the use of the package **VSURF** together with a complete illustration of the strategy which is not supported by specific model hypotheses but based on data-driven thresholds to take decisions.

But before entering in this description, let us emphasize that the objects of interest here are the subsets of important variables and not precisely the corresponding models. Thus OOB errors from the figures (or from the fitted objects) cannot be used or reported as the prediction error estimation of the final model because this would need a proper cross-validation scheme.

Using the package VSURF

From data to final results

We illustrate the use of the **VSURF** package on a simple simulated moderately high-dimensional dataset ($n = 100 < p = 200$) called `toys` data, introduced by Weston et al. (2003). It comes from an equiprobable two-class problem, $Y \in \{-1, 1\}$, with 6 influential variables and with the others being noise variables. Let us define the simulation model by giving the conditional distribution of the X^i for $Y = y$:

- For the six first variables: with probability 0.7, $X^i \sim \mathcal{N}(y_i, 1)$ for $i = 1, 2, 3$ and $X^i \sim \mathcal{N}(0, 1)$ for $i = 4, 5, 6$; with probability 0.3, $X^i \sim \mathcal{N}(0, 1)$ for $i = 1, 2, 3$ and $X^i \sim \mathcal{N}(y(i-3), 1)$ for $i = 4, 5, 6$.
- The remaining variables are noise: $X^i \sim \mathcal{N}(0, 1)$ for $i = 7, \dots, p$.

The variables obtained in the simulation are standardized before further analysis.

First, we load the **VSURF** package and the `toys` data (and fix the random number generation seed for reproducibility):

```
> library("VSURF")
> data("toys")
> set.seed(3101318)
```

The standard way to use the **VSURF** package is to use the `VSURF` function. This function executes the complete procedure, and is just a wrapper for the three intermediate functions `VSURF_thres`, `VSURF_interp` and `VSURF_pred` which are described in the next section. Typical use of the `VSURF` function is as follows:

```
> toys.vsurf <- VSURF(x = toys$x, y = toys$y, mtry = 100)
```

The only mandatory inputs are `x`, an object containing input variables, and `y`, the output variable.

In this example, we also choose a specific value for `mtry` (default is $p/3$ for classification and regression problems in `VSURF`), which only affects RF runs in the first step of the procedure. In addition, we stress that the default value for `ntree` is 2000 in `VSURF`. Those values were considered as well adapted for VI calculations (see Genuer et al., 2010b, Section 2.2) and these two arguments are passed to the `randomForest` function (we kept the same name for consistency).

The function outputs a list containing all results.

```
> names(toys.vsurf)
[1] "varselect.thres"      "varselect.interp"    "varselect.pred"
[4] "nums.varselect"       "imp.varselect.thres" "min.thres"
[7] "imp.mean.dec"        "imp.mean.dec.ind"   "imp.sd.dec"
[10] "mean.perf"          "pred.pruned.tree"   "err.interp"
[13] "sd.min"              "err.pred"           "mean.jump"
[16] "nmin"                "nsd"                 "nmj"
[19] "overall.time"        "comput.times"      "ncores"
[22] "clusterType"         "call"
```

The most important objects are `varselect.thres`, `varselect.interp` and `varselect.pred`, which contain the set of variables selected after the thresholding, interpretation and prediction step respectively.

The `summary` method gives a short summary of the results: numbers of selected variables and computation times.

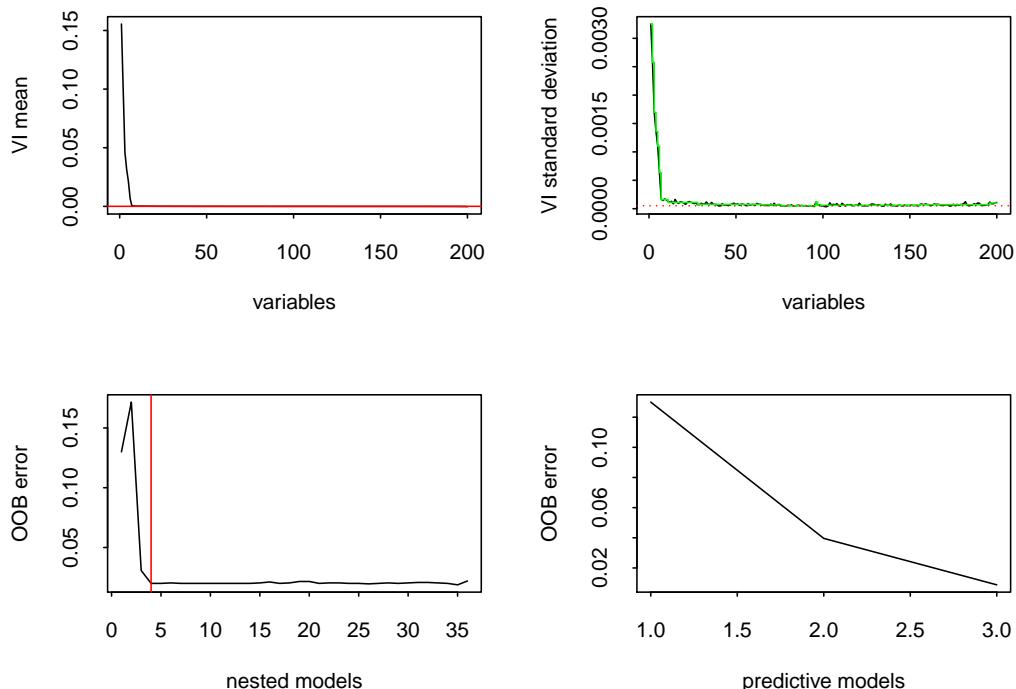


Figure 1: VSURF for the toys data: Top graphs illustrate the thresholding step, bottom left and bottom right graphs are associated with interpretation and prediction steps respectively.

```
> summary(toys.vsurf)

VSURF computation time: 1.6 mins

VSURF selected:
  36 variables at thresholding step (in 57.6 secs)
  4 variables at interpretation step (in 38.4 secs)
  3 variables at prediction step (in 2.2 secs)
```

The plot method gives a plot (see Figure 1) of the results in 4 graphs. To selectively obtain single graphs of Figure 1 one can either suitably specify the arguments of this plot method, or use intermediate plot methods (associated with each procedure step) included in the package.

```
> plot(toys.vsurf)
```

In addition, we include a predict method, which permits to predict the outcomes for new data with RF using only the variables selected by VSURF.

Finally, we point out that all computations of the package can be executed in parallel, whenever it is possible. For example, the following command runs VSURF in parallel on a Linux computing server using 40 cores:

```
> set.seed(2734, kind = "L'Ecuyer-CMRG")
> toys.vsurf.parallel <- VSURF(toys$x, toys$y, mtry = 100, parallel = TRUE,
+                                ncores = 40, clusterType = "FORK")

> summary(toys.vsurf.parallel)

VSURF computation time: 8.3 secs

VSURF selected:
  35 variables at thresholding step (in 3.8 secs)
  4 variables at interpretation step (in 2.3 secs)
  3 variables at prediction step (in 2.2 secs)

VSURF ran in parallel on a FORK cluster and used 40 cores
```

Note that we use the "L'Ecuyer-CMRG" kind in the set.seed function to allow reproducibility (when the call is on a FORK cluster with the same number of cores). Even if one should have a

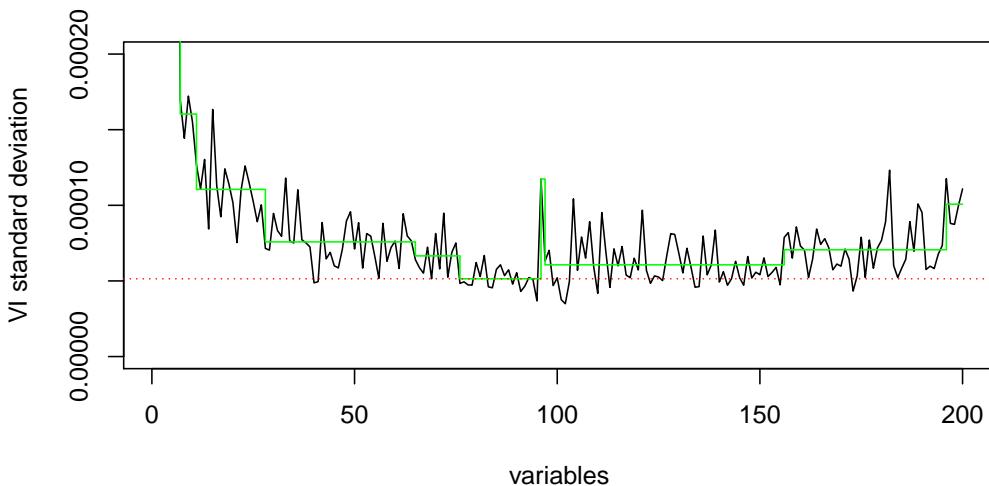


Figure 2: Zoom of the top right graph of Figure 1.

computing server with 40 cores to benefit from this execution time reduction, the resulting ‘VSURF’ object from this call will be the same on a computer with fewer cores available. Parallel calls of VSURF will be used to deal with the two high-dimensional datasets at the end of the paper.

How to get intermediate results

Let us now detail the main stages of the procedure together with the results obtained on the toys data. Note that unless explicitly stated otherwise, all graphs refer to Figure 1.

- Step 1.

- Variable ranking.

The result of variable ranking is drawn on the top left graph. True variables are significantly more important than the noisy ones.

- Variable elimination.

Starting from this order, the plot of the corresponding standard deviations of VI is used to estimate a threshold value for VI. This threshold (figured by the dotted horizontal red line in Figure 2, which is a zoom of the top right graph of Figure 1) is set to the minimum prediction value given by a CART model fitting this curve (see the green piece-wise constant function on the same graph).

Then only the variables with an averaged VI exceeding this level (i.e. above the horizontal red line in the top left graph of Figures 1) are retained.

The computation of the 50 forests, the ranking and elimination steps are obtained with the VSURF_thres function:

```
> set.seed(3101318)
> toys.thres <- VSURF_thres(toys$x, toys$y, mtry = 100)
```

The VSURF_thres function outputs a list containing all results of this step. The main outputs are: viselect.thres which contains the indices of variables selected by this step, imp.mean.dec and imp.sd.dec which hold the VI mean and standard deviation (the order according to decreasing VI mean can be found in imp.mean.dec.ind).

```
> toys.thres$viselect.thres
```

```
[1] 3 2 6 5 1 4 184 37 138 159 81 17 180 131 52 191
[17] 96 192 165 94 198 25 21 109 64 12 29 188 107 157 70 46
[33] 54 143 186 111
```

Finally, Figure 2 can be obtained with the following command:

```
> plot(toys.vsurf, step = "thres", imp.mean = FALSE, ylim = c(0, 2e-4))
```

We can see in the plot of the VI standard deviations (top right graph of Figure 1) that the true variables’ standard deviations are large compared to those of the noisy variables, which are close to zero.

- Step 2.

- Variable selection procedure for interpretation.

We use VSURF_interp for this step. Note that we have to specify the indices of variables selected at the previous step. So we set argument vars to toys.thres\$viselect.thres:

```
> toys.interp <- VSURF_interp(toys$x, toys$y,
+                               vars = toys.thres$viselect.thres)
```

The list resulting from the VSURF_interp function mainly contains viselect.interp: the variables selected by this step, and err.interp: OOB error rates of RF nested models.

```
> toys.interp$viselect.interp
```

```
[1] 3 2 6 5
```

In the bottom left graph, we see that the error decreases quickly. It reaches its (almost) minimum when the first four true variables are included in the model (see the vertical red line) and then it remains nearly constant. The selected model contains variables V3, V2, V6, V5, which are four of the six true variables, while the actual minimum is reached for 35 variables.

Note that, to ensure quality of OOB error estimations (e.g. [Genou et al., 2008](#), Section 2) along embedded RF models, the mtry parameter of the randomForest function is here set to its default value if k (the number of variables involved in the current RF model) is not greater than n , while it is set to $k/3$ otherwise.

- Variable selection procedure for prediction.

We use the VSURF_pred function for this step. We need to specify the error rates and the variables selected in the interpretation step in the err.interp and viselect.interp arguments:

```
> toys.pred <- VSURF_pred(toys$x, toys$y,
+                           err.interp = toys.interp$err.interp,
+                           viselect.interp = toys.interp$viselect.interp)
```

The main outputs of the VSURF_pred function are the variables selected by this final step, viselect.pred, and the OOB error rates of RF models, err.pred.

```
> toys.pred$viselect.pred
```

```
[1] 3 6 5
```

For the toys data, the final model for prediction purpose involves only variables V3, V6, V5 (see the bottom right graph). The threshold is set to the mean of the absolute values of the first order differentiated OOB errors between the model with $m' = 4$ variables and the one with $m = 36$ variables.

Finally, we mention that VSURF_thres and VSURF_interp can be executed in parallel with the same syntax as VSURF (setting parallel = TRUE), while VSURF_pred cannot be parallelized.

Tuning the different steps of the procedure

We provide two additional functions for tuning the thresholding and interpretation steps without having to rerun all computations.

- First, a tune method which, applied to the result of VSURF_thres, can be used to tune the thresholding step. We can use the nmin parameter (which has default value 1) in order to set the threshold to the minimum prediction value given by the CART model times nmin.

```
> toys.thres.tuned <- tune(toys.thres, nmin = 3)
> toys.thres.tuned$viselect.thres
[1] 3 2 6 5 1 4 184 37 138 159 81 17 180 131 52 191
```

We get 16 selected variables instead of 36 previously.

- Secondly, a tune method which, applied to the result of VSURF_interp, is of the same kind and allows to tune the interpretation step. If we now want to be more restrictive in our selection in the interpretation step, we can select the smallest model with an OOB error less than the minimal OOB error augmented by its empirical standard deviation times nsd (with nsd ≥ 1).

```
> toys.interp.tuned <- tune(toys.interp, nsd = 5)
> toys.interp.tuned$viselect.interp
```

```
[1] 3 2 6
```

We get 3 selected variables instead of 4 previously.

We did not write a tuning method for the prediction step because it is a recursive step and needs to recompute the sequence. Hence, to adjust the parameter for this step, we have to rerun the VSURF_pred function with a different value of nmj (which has default value 1). This multiplicative constant allows to modulate the threshold defined in (1).

For example, increasing the value of nmj leads to selection of fewer variables:

```
> toys.pred.tuned <- VSURF_pred(toys$x, toys$y, err.interp = toys.interp$err.interp,
+                                 varselect.interp = toys.interp$varselect.interp,
+                                 nmj = 70)
> toys.pred.tuned$varselect.pred
```

```
[1] 3 6
```

Remark

Thanks to one of the three anonymous reviewers, we would like to give a warning following a remark made by [Svetnik et al. \(2004\)](#). Indeed, this work considers a situation where there is no link between X and Y and for which they use OOB errors in a recursive strategy, different but not too far from our procedure, to select variables. Their results show that this kind of strategy can be seriously biased and can overfit the data. To illustrate explicitly this phenomenon, let us start with the original dataset toys for which our procedure performs quite well (see the paper [Genuer et al., 2010b](#), containing an extensive study of the operating characteristics of the algorithm including no overfitting in presence of many additional dummy variables). Then, modify it by scrambling the Y values thus removing the link between X and Y . Applying VSURF on this modified dataset leads to an OOB error rate, in the interpretation step, starting from 50% (which is correct) and exhibiting a minimum for 10 variables corresponding to 37%. So, in this situation for which there is no optimal solution and the desirable behavior is to find a constant OOB error rate along the sequence, the procedure still provides a solution. So, the conclusion is that even when there is no link between X and Y the procedure can highlight a set of variables.

Of course, using an external 5-fold cross-validation of our entire procedure leads to the correct estimate of 51% error rate for both interpretation and prediction steps and the correct conclusion: there is nothing to find in the data. Alternatively, if we simulate a second sample coming from the same simulation model and use it only to rank the variables, the interpretation step exhibits an OOB error rate curve oscillating around 50%, which leads to the correct conclusion.

Three illustrative examples

In this section we apply the proposed procedure on three real-life examples: two high-dimensional datasets (associated with a regression problem and a classification one respectively) and, before that, a standard one to illustrate the versatility of the procedure.

Let us mention that the VSURF stability is a natural issue to investigate (see e.g. [Meinshausen and Bühlmann, 2010](#)) and is considered in [Genuer et al. \(2010b\)](#) Sections 3 and 4.

Ozone data

The Ozone dataset consists of $n = 366$ observations for 12 independent variables and 1 dependent variable. These variables are numbered as in the R package `mlbench` ([Leisch and Dimitriadou, 2010](#)): 1 – Month, 2 – Day of month, 3 – Day of week, 5 – Pressure height, 6 – Wind speed, 7 – Humidity, 8 – Temperature (Sandburg), 9 – Temperature (El Monte), 10 – Inversion base height, 11 – Pressure gradient, 12 – Inversion base temperature, 13 – Visibility, for independent variables and 4 – Daily maximum one-hour-average ozone, for the dependent variable.

What makes the use of this dataset interesting, is that it has already been extensively studied and that even though it is a real one, it is possible to a priori know which variables are expected to be important. Moreover, this dataset, which is not a high-dimensional one, includes some missing data, allowing us to give an example of how to handle such data using VSURF.

To begin, we load the data:

```
> data("Ozone", package = "mlbench")
> set.seed(221921186)
```

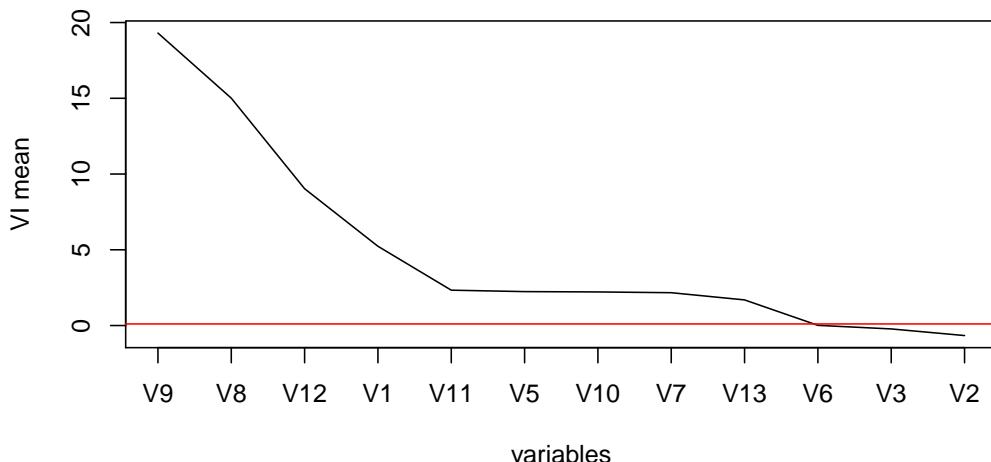


Figure 3: Sorted VI mean associated with the 12 explanatory variables of the Ozone data, with variable names on the x -axis.

Then, we apply the complete procedure via VSURF. Note that the following formula-type call is necessary to handle missing values (as in `randomForest`).

```
> vozone <- VSURF(V4 ~ ., data = Ozone, na.action = na.omit)
> summary(vozone)

VSURF computation time: 1.7 mins

VSURF selected:
  9 variables at thresholding step (in 1.1 mins)
  5 variables at interpretation step (in 26.3 secs)
  5 variables at prediction step (in 12.4 secs)
```

In the first step, we look at the variable importance associated with each of the explanatory variables.

```
> plot(vozone, step = "thres", imp.sd = FALSE, var.names = TRUE)
```

In Figure 3, and as noticed in previous studies, three very sensible groups of variables can be discerned ranging from the most to the least important. The first group contains the two temperatures (8 and 9), the inversion base temperature (12) known to be the best ozone predictors, and the month (1), which is an important predictor since ozone concentration exhibits an heavy seasonal component. The second group of clearly less important meteorological variables consists of: pressure height (5), humidity (7), inversion base height (10), pressure gradient (11) and visibility (13). Finally the last group contains three unimportant variables: day of month (2), day of week (3) of course and more surprisingly wind speed (6). This last fact is classical: wind enters in the model only when ozone pollution arises, otherwise wind and pollution are weakly correlated (see for example Chèze et al., 2003, who highlight this phenomenon using partial estimators).

Let us now examine the results of the selection procedures. To reflect the order used in the definition of the variables, we reorder the output variables of the procedure.

```
> number <- c(1:3, 5:13)
> number[vozone$viselect.thres]

[1] 9 8 12 1 11 5 10 7 13
```

After the first elimination step, the 3 variables of negative importance (variables 6, 3 and 2) are eliminated, as expected.

```
> number[vozone$viselect.interp]

[1] 9 8 12 1 11
```

Then the interpretation procedure leads to select the model with 5 variables, which contains all of the most important variables.

```
> number[vozone$varselect.pred]
[1] 9 8 12 1 11
```

With the default settings, the prediction step does not remove any additional variable.

Remark

Even though the comparison with other variable selection strategies is out of the scope of the paper, one of the three anonymous reviewers has kindly compared the results of the **VSURF** package with the results of the R package **Boruta**, described in [Kursa and Rudnicki \(2010\)](#), on the two datasets toys and Ozone.

Let us recall that this module directly aims at selecting all-relevant features. Hence the comparison with the interpretation set delivered by **VSURF** is of interest. In the toys dataset, the number of truly relevant variables is 6. **VSURF** finds 4 out of 6 variables in the interpretation stage while **Boruta** finds all six truly relevant variables and one more false positive variable. In the case of the Ozone data set **VSURF** finds 5 variables in the interpretation stage, while **Boruta** finds 9. So, this seems to confirm that the heuristic proposed by **VSURF** is prediction oriented (the price to pay is a risk of false negatives) and suggests that the strategy proposed by **Boruta** is more accurate to recover weak redundant correlations between predictors and decision variable (the price to pay seems to be a risk of false positives).

In fact our strategy assumes more or less that there are unnecessary variables in the set of all available variables initially, which is not really the case in the Ozone dataset. However, it is the case in the following two high-dimensional examples.

Toxicity data

This second dataset is also a regression framework, however, unlike before, this case is a high-dimensional problem. The `liver.toxicity` dataset, available in the R package **mixOmics** ([Lê Cao et al., 2015](#)), is a real dataset from a study by [Heinloth et al. \(2004\)](#). In this study, 4 male rats of the inbred strain Fisher 344 were exposed to different doses of acetaminophen (non toxic dose (50 or 150 mg/kg), moderate toxic dose (1500 mg/kg), severe toxic dose (2000 mg/kg)) in a controlled experiment. Necropsies were performed at different hours after exposure (6, 18, 24 and 48 hours) and the mRNA from the liver was extracted. In the original study, 10 clinical chemistry variables containing markers for the liver injury were measured. Those variables are numerical variables since they measure the serum enzymes level. For our analysis, the dataset extracted from this study contains:

- a data frame, called `gene`, with 64 rows representing the subjects and 3116 columns representing explanatory variables which are the gene expression levels after normalization and preprocessing due to [Bushel et al. \(2007\)](#),
- a vector, called `clinic`, with 64 rows and 1 column, one of the 10 clinical variables for the same 64 subjects: more precisely, the variable named `ALB.g.dL.`, which corresponds to the albumin level and which is the one considered in [González et al. \(2012\)](#).

As in previous studies ([Gidskehaug et al., 2007](#); [Lê Cao et al., 2008](#)), our aim is, using **VSURF**, to predict our clinical variable by the genes.

First, we load the data as follows:

```
> data("liver.toxicity", package = "mixOmics")
> clinic <- liver.toxicity$clinic$ALB.g.dL.
> set.seed(7162013, "L'Ecuyer-CMRG")
```

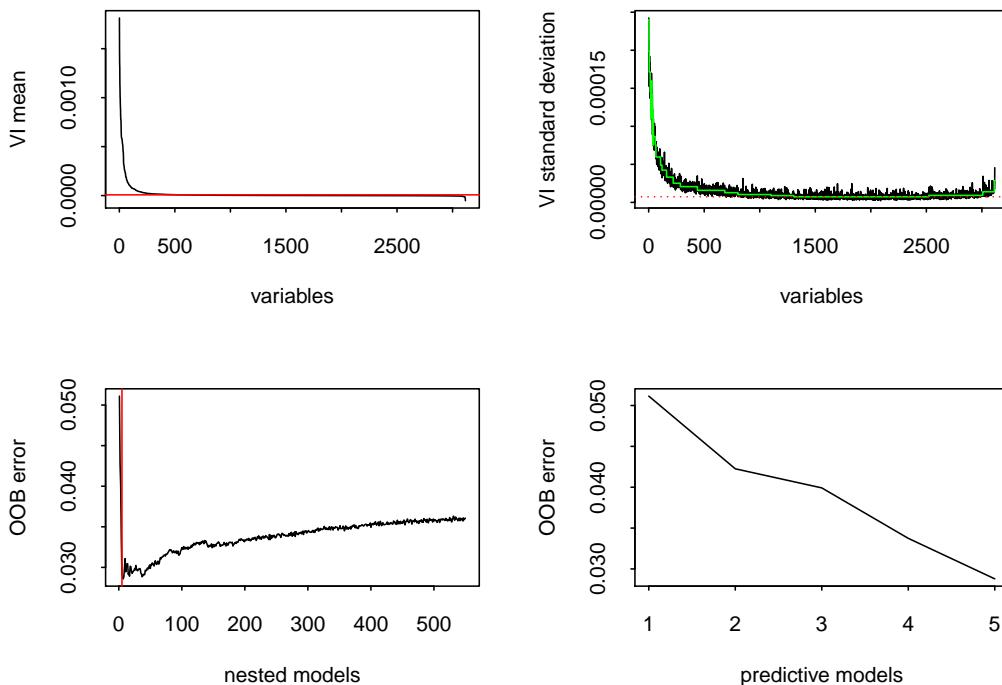
Now we apply our procedure and analyze the results.

```
> vtoxicity <- VSURF(liver.toxicity$gene, clinic, parallel = TRUE, ncores = 40,
+                      clusterType = "FORK")
> summary(vtoxicity)

VSURF computation time: 5.9 mins

VSURF selected:
  550 variables at thresholding step (in 1.1 mins)
  5 variables at interpretation step (in 4.8 mins)
  5 variables at prediction step (in 3.4 secs)

VSURF ran in parallel on a FORK cluster and used 40 cores
```

**Figure 4:** VSURF for the toxicity data.

We notice that after the elimination step, only 550 variables remain, thus the number of variables has been reduced by a factor of six. This ratio is not surprising since we know that there exists extreme redundancy in the gene expression data together with a lot of irrelevant variables.

```
> plot(vtoxicity)
```

By considering the top left graph of Figure 4, it seems quite obvious to keep just a few variables. Indeed, the procedure leads to 5 variables, after both interpretation and prediction steps. Even if the numbers of selected variables are very small, they are not surprisingly low if we refer to the study in [González et al. \(2012\)](#), where 12 variables were selected. It would be noted that, in general, our method failed to deliver all the variables related to the response variable in the case of numerous strongly correlated predictors. In addition, we do not select the same genes but our set of selected variables and the one in [González et al. \(2012\)](#) exhibit strong correlations.

Even if the results are quite similar, an advantage of using VSURF is that this procedure does not involve tuning parameters unlike the procedure developed in [González et al. \(2012\)](#). This difference is the main reason for the gap in computation time: several minutes with 40 cores for VSURF compared to several minutes with 1 core for [González et al. \(2012\)](#).

SRBCT data

The dataset we consider here will allow us to apply our procedure in a classification framework. The real classification dataset is a small version of the small round blue cell tumors of childhood data and contains the expression measure of genes measured on 63 samples. This set is composed of:

- a data frame, called `gene`, of size 63×2308 which contains the 2308 gene expressions;
- a response factor of length 63, called `class`, indicating the class of each sample (4 classes in total).

These data, presented in details in [Khan et al. \(2001\)](#), available in the R package `mixOmics`, have been widely studied but in most cases only 200 genes were considered and data have been transformed to reduce the problem to a regression problem (see e.g. [Lé Cao and Chabrier, 2008](#)). As in [Díaz-Uriarte and Alvarez De Andres \(2006\)](#), we consider the 2308 genes and we deal directly with the classification problem, using VSURF.

```
> data("srbc", package = "mixOmics")
> set.seed(10131419, "L'Ecuyer-CMRG")
```

```

> vSRBCT <- VSURF(srbct$gene, srbct$class, parallel = TRUE, ncores = 40,
+                      clusterType = "FORK")

> summary(vSRBCT)

VSURF computation time: 3.6 mins

VSURF selected:
  676 variables at thresholding step (in 22.6 secs)
  25 variables at interpretation step (in 3 mins)
  13 variables at prediction step (in 13.6 secs)

VSURF ran in parallel on a FORK cluster and used 40 cores

```

On this dataset, the procedure leads to 25 and 13 selected variables after the interpretation and prediction step respectively, and the selected variable sets are stable.

We can compare these results with those obtained in Díaz-Uriarte and Alvarez De Andres (2006) where the authors select 22 genes on the original dataset and their number of selected variables is quite stable.

To get an idea of the performance of our procedure on the dataset, we perform an error rate estimation using an external 5-fold cross-validation scheme (meaning that we apply VSURF on each fold of the cross-validation). We obtain the following error rates² for interpretation and prediction sets respectively:

interp	pred
0.01587302	0.07936508

The comparison with error rates evaluated using 200 bootstrap samples in Díaz-Uriarte and Alvarez De Andres (2006) suggests that our selections are reasonable.

Acknowledgements

We thank the editor and the three anonymous referees for their thorough comments and suggestions which really helped to improve the clarity of the paper.

Bibliography

- K. J. Archer and R. V. Kimes. Empirical characterization of random forest variable importance measures. *Computational Statistics & Data Analysis*, 52(4):2249–2260, 2008. [p20]
- R. Azen and D. V. Budescu. The dominance analysis approach for comparing predictors in multiple regression. *Psychological Methods*, 8(2):129–148, 2003. [p20]
- A.-L. Boulesteix, S. Janitza, J. Kruppa, and I. R. König. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):493–507, 2012. [p19]
- L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. [p20]
- L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. [p19, 20]
- L. Breiman and A. Cutler. Random forest manual. 2004. URL http://www.stat.berkeley.edu/~breiman/RandomForests/cc_manual.htm. [p20, 21]
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984. [p19]
- P. Bushel, R. Wolfinger, and G. Gibson. Simultaneous clustering of gene expression data with clinical chemistry and pathological evaluations reveals phenotypic prototypes. *BMC Systems Biology*, 1(1):15, 2007. [p29]
- J. M. Cadenas, M. Carmen Garrido, and R. Martínez. Feature subset selection filter-wrapper based on low quality data. *Expert Systems with Applications*, 40(16):6241–6252, 2013. [p20]

²The script is available at https://github.com/robingenuer/VSURF/blob/master/Example/srbct_cv.R

- N. Chèze, J.-M. Poggi, and B. Portier. Partial and recombined estimators for nonlinear additive models. *Statistical Inference for Stochastic Processes*, 6(2):155–197, 2003. [p28]
- R. Díaz-Uriarte. GeneSelRF and varSelRF: A web-based tool and R package for gene selection and classification using random forest. *BMC Bioinformatics*, 8(1):328, 2007. [p20]
- R. Díaz-Uriarte and S. Alvarez De Andres. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1):3, 2006. [p20, 30, 31]
- R. Genuer, J.-M. Poggi, and C. Tuleau. Random forests: Some methodological insights. 2008. arXiv:0811.3619. [p26]
- R. Genuer, V. Michel, E. Eger, and B. Thirion. Random forests based feature selection for decoding fMRI data. In *Proceedings of COMPSTAT'2010 – 19th International Conference on Computational Statistics*, pages 1071–1078, 2010a. [p21]
- R. Genuer, J.-M. Poggi, and C. Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225–2236, 2010b. [p19, 20, 21, 22, 23, 27]
- L. Gidskehaug, E. Anderssen, A. Flatberg, and B. K. Alsberg. A framework for significance analysis of gene expression data using dimension reduction methods. *BMC Bioinformatics*, 8(1):346, 2007. [p29]
- I. González, K.-A. Lê Cao, M. J. Davis, and S. Déjean. Visualising associations between paired ‘omics’ data sets. *BioData Mining*, 5(1):1–23, 2012. [p29, 30]
- B. Gregorutti, B. Michel, and P. Saint-Pierre. Correlation and variable importance in random forests. 2013. arXiv:1310.5726. [p20, 21]
- I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. [p20]
- A. Hapfelmeier and K. Ulm. A new variable selection approach using random forests. *Computational Statistics & Data Analysis*, 60:50–69, 2012. [p20]
- T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, New York, 2001. [p19]
- A. N. Heinloth, R. D. Irwin, G. A. Boorman, P. Nettesheim, R. D. Fannin, S. O. Sieber, M. L. Snell, C. J. Tucker, L. Li, G. S. Travlos, G. Vasant, P. E. Blackshear, R. W. Tennant, M. L. Cunningham, and R. S. Paules. Gene expression profiling of rat livers reveals indicators of potential adverse effects. *Toxicological Sciences*, 80(1):193–202, 2004. [p29]
- T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006. [p19]
- J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson, and P. S. Meltzer. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine*, 7(6):673–679, 2001. [p30]
- R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997. [p20]
- M. B. Kursa and W. R. Rudnicki. Feature selection with the Boruta package. *Journal of Statistical Software*, 36(11):1–13, 2010. [p20, 29]
- K.-A. Lê Cao and P. Chabrier. ofw: An R package to select continuous variables for multiclass classification with a stochastic wrapper method. *Journal of Statistical Software*, 28(9):1–16, 2008. [p20, 30]
- K.-A. Lê Cao, D. Rossouw, C. Robert-Granié, and P. Besse. A sparse PLS for variable selection when integrating omics data. *Statistical Applications in Genetics and Molecular Biology*, 7(1), 2008. [p29]
- K.-A. Lê Cao, I. Gonzalez, and S. Dejean. *mixOmics: Omics Data Integration Project*, 2015. URL <https://CRAN.R-project.org/package=mixOmics>. R package version 5.0-4, with key contributions from F. Rohart and B. Gautier and contributions from P. Monget, J. Coquery, F. Yao and B. Liquet. [p29]
- F. Leisch and E. Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2010. URL <https://CRAN.R-project.org/package=mlbench>. R package version 2.1-1. [p27]

- A. Liaw and M. Wiener. Classification and regression by randomForest. *R News*, 2(3):18–22, 2002. [p19]
- G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts. Understanding variable importances in forests of randomized trees. In *Advances in Neural Information Processing Systems*, pages 431–439, 2013. [p21]
- N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Statistical Society B*, 72(4):417–473, 2010. [p27]
- R. Nilsson, J. M. Peña, J. Björkegren, and J. Tegnér. Consistent feature selection for pattern recognition in polynomial time. *Journal of Machine Learning Research*, 8:589–612, 2007. [p21]
- A. Peters, T. Hothorn, and B. Lausen. ipred: Improved predictors. *R News*, 2(2):33–36, 2002. [p20]
- F. Scheipl. spikeSlabGAM: Bayesian variable selection, model choice and regularization for generalized additive mixed models in R. *Journal of Statistical Software*, 43(14):1–24, 2011. [p20]
- C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1):25, 2007. [p20]
- C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(1):307, 2008. [p20]
- V. Svetnik, A. Liaw, C. Tong, and T. Wang. Application of Breiman’s random forest to modeling structure-activity relationships of pharmaceutical molecules. In *Multiple Classifier Systems*, pages 334–343. Springer, 2004. [p27]
- T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2015. URL <https://CRAN.R-project.org/package=rpart>. R package version 4.1-9. [p19]
- A. Verikas, A. Gelzinis, and M. Bacauskiene. Mining data with random forests: A survey and results of new tests. *Pattern Recognition*, 44(2):330–349, 2011. [p19]
- H. R. Wehrens, M. Johan, and P. Franceschi. Meta-statistics for variable selection: The R package BioMark. *Journal of Statistical Software*, 51(10):1–18, 2012. [p20]
- J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping. Use of the zero norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–1461, 2003. [p23]

Robin Genuer
University of Bordeaux
ISPED, Centre INSERM U-897-Epidemiologie-Biostatistique
33000 Bordeaux, France
and
INRIA Bordeaux Sud Ouest, SISTM team
33400 Talence, France
Robin.Genuer@isped.u-bordeaux2.fr

Jean-Michel Poggi
University of Orsay
Lab. Mathematics
bat 425
91405 Orsay, France
Jean-Michel.Poggi@math.u-psud.fr

Christine Tuleau-Malot
University of Nice Sophia Antipolis
CNRS, LJAD, UMR 7351
06100 Nice, France
Malot@unice.fr

zoib: An R package for Bayesian Inference for Beta Regression and Zero/One Inflated Beta Regression

by Fang Liu and Yunchuan Kong

Abstract The beta distribution is a versatile function that accommodates a broad range of probability distribution shapes. Beta regression based on the beta distribution can be used to model a response variable y that takes values in open unit interval $(0, 1)$. Zero/one inflated beta (ZOIB) regression models can be applied when y takes values from closed unit interval $[0, 1]$. The ZOIB model is based a piecewise distribution that accounts for the probability mass at 0 and 1, in addition to the probability density within $(0, 1)$. This paper introduces an R package – **zoib** that provides Bayesian inferences for a class of ZOIB models. The statistical methodology underlying the **zoib** package is discussed, the functions covered by the package are outlined, and the usage of the package is illustrated with three examples of different data and model types. The package is comprehensive and versatile in that it can model data with or without inflation at 0 or 1, accommodate clustered and correlated data via latent variables, perform penalized regression as needed, and allow for model comparison via the computation of the DIC criterion.

Introduction

The beta distribution has two shape parameters α_1 and α_2 : Beta(α_1, α_2). The mean and variance of a variable y that follows the beta distribution are $E(y) = \mu = \alpha_1(\alpha_1 + \alpha_2)^{-1}$ and $V(y) = \mu(1 - \mu)(\alpha_1 + \alpha_2 + 1)^{-1}$, respectively. A broad spectrum of distribution shapes can be generated by varying the two shapes values of α_1 and α_2 , as demonstrated in Figure 1. The beta regression has become more popular in recent years in modeling data bounded within open interval $(0, 1)$ such as rates and proportions, and more generally, data bounded within (a, b) as long as a and b are fixed and known and it is sensible to transform the raw data onto the scale of $(0, 1)$ by shifting and scaling, that is, $y' = (y - a)(b - a)^{-1}$.

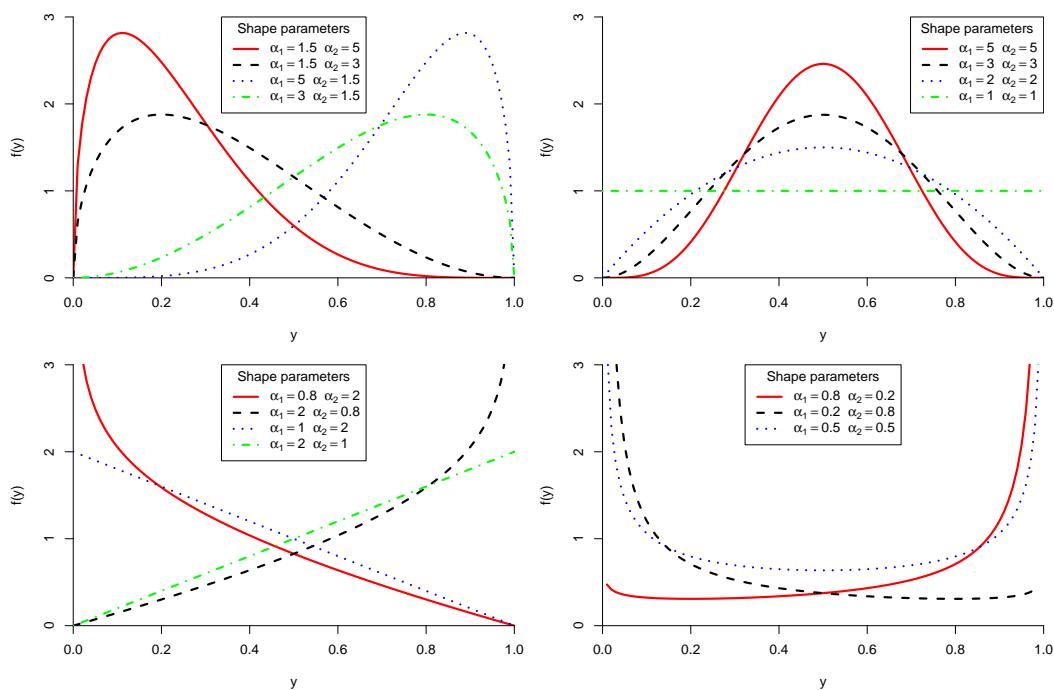


Figure 1: Beta distribution with various values of the two shape parameters.

Given the flexibility and increasing popularity of the beta regression, significant development has been made in the theory, methodology, and practical applications of the beta regression (Cepeda-Cuervo, 2001; Paolino, 2001; Williams, 1982; Prentice, 1986; Ferrari and Cribari-Neto, 2004; Smithson and Verkuilen, 2006; Simas et al., 2010; Smithson and Verkuilen, 2006; Hatfield et al., 2012;

Ospina and Ferrari, 2012; Cepeda-Cuervo, 2015). Mostly recently, Grün et al. (2012) apply the techniques of the model-based recursive partitioning (Zeileis et al., 2008) and the finite mixture model (Dalrymplea et al., 2003) in the framework of beta regression to account for heterogeneity between groups/clusters of observations. They also propose bias-corrected or bias-reduced estimation in the beta regression by applying the unifying iteration technique (Kosmidis and Firth, 2010).

In many cases of real life data, exact 0's and 1's occur in addition to y values between 0 and 1, producing zero-inflated, one-inflated, or zero/one-inflated outcomes. Though the beta distribution covers a variety of the distribution shape, it does not accommodate excessive values at 0 and 1. Smithson and Verkuilen (2006) propose transformation $n^{-1}(y(n-1) + 0.5)$, where n is the sample size, so all data points after transformation are bounded within 0 and 1 and the regular beta regression can be applied. This approach, while offering a simple way to circumvent the complexity from modeling the boundary values, only shifts the excessiveness in point mass from one location to another. Hatfield et al. (2012) model the zero/one inflated VAS responses by relocating all 1 to 0.9995 and keep 0 as is, and apply the zero-inflated beta (ZIB) regression. The approach of only shifting 1 but not 0 when there is inflation at both is ad-hoc especially if there is no justification for treating 0 differently from 1. From a practical perspective, the observed 0's and 1's might carry practical meanings that would be otherwise lost if being replacing with other values, regardless how close the raw and substitutes values are. Ospina and Ferrari (2012) propose the zero-or-one inflated beta regression model (inflation at either 0 or 1, but not both) and obtain inferences via the maximum likelihood estimation (MLE). When there is inflation at both 0 and 1, it is sensible to model the excessiveness explicitly with the zero/one inflated beta (ZOIB) regression, especially when population 0's and 1's are real. For example, if the response variable is the death proportion of mice on different doses of a chemical entity; the death rate caused by administration of the chemical entity theoretically can be 0 when its dosage is 0, and 1 when the dosage increases to a 100% lethal level. The ZOIB regression technique has been previously discussed in the literature (Swearingen et al., 2012). Most beta regression and zoib models focus on fixed effects models only, and thus cannot handle clustered or repeated measurements. Liu and Li (2014) apply a joint model with latent variables to model the dependency structure among multiple [0, 1]-bounded responses with repeated measures in the Bayesian framework.

From a software perspective, beta regression can be implemented in a software suite or package that accommodate nonlinear regression models, such as SPSS (NLR and CNLR) and SAS (PROC NLIN, PROC NLMIXED). There are also contributed packages or macros devoted specifically to beta regression, such as the SAS macro developed by Swearingen et al. (2011), which implements the beta regression directly and provides residuals plots for model fit diagnostics. In R, there are a couple of packages targeted specifically at beta regression. **betareg** (Zeileis et al., 2014) models a single response variable bounded within (0, 1), with fixed-effects linear predictors in the link functions for the mean and precision parameter of the beta distribution (Cribari-Neto and Zeileis, 2010). The package is later updated by Grün et al. (2012) to perform bias correction/reduction, model-based recursive partitioning, and finite mixture models with added functions betatree() and betamix() in package **betareg**. In **betareg**, the coefficients of the regression are estimated by the MLE and inferences are based on large sample assumptions. **Bayesianbetareg** (Marin et al., 2014) allows the joint modelling of mean and precision of a single response in the Bayesian framework, as is proposed in Cepeda-Cuervo (2001), with logit link for the mean and logarithmic for the precision. Neither **betareg** nor **Bayesianbetareg** accommodate inflation at 0 or 1 (**betareg** transforms y with inflation at 0 and 1 using $(y(n-1) + 0.5)n^{-1}$; neither can model multiple response variables, repeated measures, or clustered/correlated response variables. In other words, the linear predictors in the link functions of the mean and precision parameters of the beta distribution in both **betareg** and **Bayesianbetareg** contain fixed effects only.

In this discussion, we introduce a new R package **zoib** (Liu and Kong, 2014) that models responses bounded within [0, 1] – without inflation at 0 nor 1, with inflation at 0 only, at 1 only, or at both 0 and 1. The package can model a single response with or without repeated measures, or multiple or clustered [0, 1]-bounded response variables, taking into account the dependency among them. Compared to the existing packages on beta regression in R, **zoib** is more comprehensive and flexible from the modeling perspective and can accommodate more data types. The inferences of the mdoel parameters in package **zoib** are obtained in the Bayesian framework via the Markov Chain Monte Carlo (MCMC) approach as implemented in JAGS (Plummer, 2014a).

The rest of the paper is organized as follows. Section **Zero/one inflated beta regression** describes the methodology underlying the ZOIB regression. Section **Implementation in R** introduces the package **zoib**, including its functionality and outputs. Section **Examples** illustrates the usage of the package with 3 real-life data sets and 1 simulated data of different types. The paper ends in Section **Summary** with summaries and discussions.

Zero/one inflated beta regression

The ZOIB model

Suppose y_j is the j^{th} variable out of a total p response variables measured on n independent units, that is, $\mathbf{y}_j = (y_{1j}, \dots, y_{nj})^t$. The zoib model assumes y_{ij} follows a piecewise distribution when y_{ij} has inflation at both 0 and 1.

$$f(y_{ij}|\eta_{ij}) = \begin{cases} p_{ij} & \text{if } y_{ij} = 0 \\ (1 - p_{ij})q_{ij} & \text{if } y_{ij} = 1 \\ (1 - p_{ij})(1 - q_{ij})\text{Beta}(\alpha_{ij1}, \alpha_{ij2}) & \text{if } y_{ij} \in (0, 1). \end{cases} \quad (1)$$

p_{ij} is the probability of $y_{ij} = 0$, and q_{ij} is the conditional probability $\Pr(y_{ij} = 1|y_{ij} \neq 0)$, and α_{ij1} and α_{ij2} are the shape parameters of the beta distribution when $y_{ij} \in (0, 1)$. The probability parameters from the binomial distributions and the two shape parameters from the beta distributions are linked to observed explanatory variables \mathbf{x}_{ij} or unobserved latent variable \mathbf{z}_{ij} via link functions. Some natural choices for the link functions for p_{ij} , q_{ij} , and the mean of the beta distribution $\mu_{ij}^{(0,1)} = E(y_{ij}|y_{ij} \in (0, 1)) = \alpha_{ij1}(\alpha_{ij1} + \alpha_{ij2})^{-1}$, which are all parameters within $(0, 1)$, include the logit function, the probit function, or the complementary log-log (cloglog) function. While the binomial distribution is described by a single probability parameter, the beta distribution is characterized by two parameters. The variance of the beta distribution is not only a function of its mean but also the sum of two shape parameters $v_{ij} = \alpha_{ij1} + \alpha_{ij2}$; that is, $V(y_{ij}|y_{ij} \in (0, 1)) = \mu_{ij}^{(0,1)}(1 - \mu_{ij}^{(0,1)})(\alpha_{ij1} + \alpha_{ij2} + 1)^{-1} = \mu_{ij}^{(0,1)}(1 - \mu_{ij}^{(0,1)})(v_{ij} + 1)^{-1}$. v_{ij} is often referred to as the precision (dispersion) parameter and can also be affected by external explanatory variables or latent variables (Simas et al., 2010; Cribari-Neto and Zeileis, 2010). An example of the formulation of the zoib model, if the logit function is applied to p_{ij} , q_{ij} , and $\mu_{ij}^{(0,1)}$, and the log link function is applied to v_{ij} , is

$$\text{logit}(\mu_{ij}^{(0,1)}) = \mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j} + I_1(\mathbf{z}_{1,ij}\gamma_{1,i}) \quad (2)$$

$$\log(v_{ij}) = \mathbf{x}_{2,ij}\boldsymbol{\beta}_{2j} + I_2(\mathbf{z}_{2,ij}\gamma_{2,i}) \quad (3)$$

$$\text{logit}(p_{ij}) = \mathbf{x}_{3,ij}\boldsymbol{\beta}_{3j} + I_3(\mathbf{z}_{3,ij}\gamma_{3,i}) \quad (4)$$

$$\text{logit}(q_{ij}) = \mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j} + I_4(\mathbf{z}_{4,ij}\gamma_{4,i}), \quad (5)$$

where $\boldsymbol{\beta}_{m,j}$ represents the linear fixed effects in link function m ($m = 1, 2, 3, 4$) for response j ($j = 1, \dots, p$); $\mathbf{x}_{m,ij}$ is the design matrix for the fixed effects; $I_m(\mathbf{z}_{m,ij}\gamma_{m,i})$ is an indicator function on whether link function m has a random component or not, that is, $I_m(\mathbf{z}_{m,ij}\gamma_{m,i}) = \mathbf{z}_{m,ij}\gamma_{m,i}$ if link function has a random component, $I_m(\mathbf{z}_{m,ij}\gamma_{m,i}) = 0$ otherwise. $\mathbf{z}_{m,i}$ represents the design matrix associated with the random components; Unless stated otherwise, we assume $\gamma_{m,i} \stackrel{\text{ind}}{\sim} N(\mathbf{0}, \Sigma_m)$ for $i = 1, \dots, n$ in link function m throughout this discussion. When $p \geq 2$, dependency among the p response variables are modeled through their sharing of $\gamma_{m,i}$ for each m .

When there are no random/latent components in the linear predictors, the mean of the beta distribution for $y_{ij} \in (0, 1)$ is given by $\exp(\mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j})(1 + \exp(\mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j}))^{-1}$ while $\exp(\mathbf{x}_{2,ij}\boldsymbol{\beta}_{2j})$ is the sum of the two shape parameters. $\exp(\mathbf{x}_{3,ij}\boldsymbol{\beta}_{3j})(1 + \exp(\mathbf{x}_{3,ij}\boldsymbol{\beta}_{3j}))^{-1}$ is $\Pr(y_{ij} = 0)$, and $\exp(\mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j})(1 + \exp(\mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j}))^{-1}$ is $\Pr(y_{ij} = 1|y_{ij} > 0)$. The overall mean of y_{ij} is thus given by

$$\begin{aligned} E(y_{ij}) &= (1 - p_{ij}) \left(q_{ij} + (1 - q_{ij})\mu_{ij}^{(0,1)} \right) \\ &= \frac{\exp(\mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j})(1 + \exp(\mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j}))^{-1} + \exp(\mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j})}{(1 + \exp(\mathbf{x}_{3,ij}\boldsymbol{\beta}_{3j}))(1 + \exp(\mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j}))} \end{aligned}$$

When there are random/latent components, then the conditional mean of y_{ij} given $\mathbf{z}_{m,i}$ is

$$\begin{aligned} E(y_{ij}|\gamma_{1,i}, \gamma_{2,i}, \gamma_{3,i}, \gamma_{4,i}) &= (1 - p_{ij}) \left(q_{ij} + (1 - q_{ij})\mu_{ij}^{(0,1)} \right) \\ &= \frac{\exp\{\mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j} + I_1(\mathbf{z}_{1,ij}\gamma_{1,i})\}(1 + \exp\{\mathbf{x}_{1,ij}\boldsymbol{\beta}_{1j} + I_1(\mathbf{z}_{1,ij}\gamma_{1,i})\})^{-1} + \exp\{\mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j}I_4(\mathbf{z}_{4,ij}\gamma_{4,i})\}}{(1 + \exp\{\mathbf{x}_{3,ij}\boldsymbol{\beta}_{3j} + I_1(\mathbf{z}_{3,ij}\gamma_{3,i})\})(1 + \exp\{\mathbf{x}_{4,ij}\boldsymbol{\beta}_{4j} + I_2(\mathbf{z}_{4,ij}\gamma_{4,i})\})} \end{aligned} \quad (6)$$

Calculation of the marginal mean of y_{ij} involves integrating out $\gamma_{m,i}$ over its distribution; that is, $E(y_{ij}) = \int E(y_{ij}|\gamma_{1,i}, \gamma_{2,i}, \gamma_{3,i}, \gamma_{4,i})f(\gamma_{1,i}|\Sigma_1)f(\gamma_{2,i}|\Sigma_2)f(\gamma_{3,i}|\Sigma_3)f(\gamma_{4,i}|\Sigma_4)d\gamma_{i,1}d\gamma_{2,i}d\gamma_{3,i}d\gamma_{4,i}$, which can become computationally and analytically tractable if the MLE approach is taken. In contrast, $E(y_{ij})$

is easier to obtain by the Monte Carlo approach in the Bayesian computational framework.

Equations (2) to (5) give a full parameterization of the ZOIB model. Various reduced forms of the fully parameterized model as given in equations (2) to (5) are available. For example, if a constant dispersion parameter is assumed, then equation (3) can be simplified $\log(v_{ij}) = c_j$ that differs only by response variable. In practice, it might also be reasonable to assume $\mathbf{z}_{m,ij}\gamma_{m,i}$ is the same across all links functions, that is, $\Sigma_m = \Sigma$, since information to distinguish among Σ_m 's is unlikely available in many real life applications.

Bayesian inference

Though the inferences of the parameters in the proposed ZOIB model can be obtained via the MLE approach, the task can be analytically and computationally challenging, considering the nonlinear nature of the model and existence of possible random effects. We adopt the Bayesian inferential approach in package **zoib**. Let $\Theta = \{\beta_1, \beta_2, \beta_3, \beta_4, \Sigma\}$ denote the set of the parameters from the ZOIB model (**zoib** sets $\gamma_{m,i} = \gamma_i$ and $\Sigma_m = \Sigma \forall m$). The joint posterior distribution of Θ and the random effects γ given data \mathbf{y} is $p(\Theta, \gamma|\mathbf{y}) \propto p(\mathbf{y}|\Theta, \gamma)p(\gamma|\Theta)p(\Theta)$. The likelihood $p(\mathbf{y}|\Theta, \gamma)$ is constructed from the ZOIB model in equation (1)

$$p(\mathbf{y}|\Theta, \gamma) \propto \prod_i \prod_j \left\{ p_{ij}^{I(y_{ij}=0)} (1-p_{ij})^{I(y_{ij}>0)} q_{ij}^{I(y_{ij}=1)} \right\} \times \\ \left\{ (1-q_{ij}) \frac{\Gamma(v_{ij})}{\Gamma(v_{ij}\mu_{ij}^{(0,1)})\Gamma(v_{ij}(1-\mu_{ij}^{(0,1)}))} (y_{ij})^{v_{ij}\mu_{ij}^{(0,1)}-1} (1-y_{ij})^{v_{ij}(1-\mu_{ij}^{(0,1)})-1} \right\}^{I(y_{ij} \in (0,1))},$$

noting p_{ij}, q_{ij}, v_{ij} and $\mu_{ij}^{(0,1)}$ are functions of Θ , and $p(\gamma|\Theta) \sim N(0, \Sigma)$. **zoib** assumes all the parameters in Θ are *a priori* independent, thus $f(\Theta) = f(\Sigma) \prod_{j=1}^p \prod_{m=1}^4 f(\beta_{mj})$. **zoib** offers the following prior choices on $\beta_{m,j}$:

- Diffuse normal (DN) on all intercept terms $\beta_{m,j0} \sim N(0, C)$, where C is the precision of the normal distribution that can be specified by users. The smaller C is, the more “diffuse” the normal distribution is (the less *a priori* information there is about $\beta_{m,j0}$). The default $C = 10^{-3}$.
- For the rest of elements in $\beta_{m,j}$ (minus the intercept term), there are 4 options:
 - diffuse normal (DN, default): $\beta_{m,jk} \stackrel{\text{ind}}{\sim} N(0, C)$ across $k = 1, \dots, p_m$ for a given j ($j = 1, \dots, q$) and m ($m = 1, \dots, 4$). C is the precision of the normal distribution that can be specified by users; the default $C = 10^{-3}$.
 - L2-prior (L2): The L2 prior shrinks the regression coefficients in the same link function m on the same variable y_j in a L_2 manner as in ridge regression (Lindley and Smith, 1972). The L2 prior helps when there is non-orthogonality among the covariates. $\beta_{m,jk} | \lambda_{m,jk} \stackrel{\text{ind}}{\sim} N(0, \lambda_{m,j})$ for $k = 1, \dots, p_m$ and the precision parameter $\lambda_{m,j} \stackrel{\text{ind}}{\sim} \text{gamma}(\alpha, \beta)$ given j and m . α and β , the shape and scale parameters of the gamma distribution, are small constants that can be specified by the user. The default is $\alpha = \beta = 10^{-3}$ for all m and j .
 - L1-prior (L1): The L1 prior shrinks the regression coefficients in the same link function m on the same variable y_j in a L_1 manner (Lindley and Smith, 1972) as in Lasso regression (Park and Casella, 2008). As such, the L1-prior helps there is a large of covariates and sparsity in the regression coefficients is desirable. $\beta_{m,jk} | \lambda_{m,jk} \stackrel{\text{ind}}{\sim} N(0, \lambda_{m,jk})$ and $\lambda_{m,jk} \stackrel{\text{ind}}{\sim} \exp(\epsilon_{m,j})$ for $k = 1, \dots, p_m$ given j and m . $\epsilon_{m,j}$ is a small constant that can be specified by users. The default $\epsilon_{m,j} = 10^{-3}$ for all m and j .
 - automatic relevance determination (ARD): ARD, as the L2 and L1 priors, regularizes the regression coefficients toward sparsity. Different from the L2 prior, where every coefficient has the same precision parameter $\lambda_{m,j}$, the precision is coefficient-specific in the ARD prior (MacKay, 1996; Neal, 1994): $\beta_{m,jk} | \lambda_{m,jk} \stackrel{\text{ind}}{\sim} N(0, \lambda_{m,jk})$ and $\lambda_{m,jk} \stackrel{\text{ind}}{\sim} \text{gamma}(\alpha_{m,j}, \beta_{m,j})$ for $k = 1, \dots, p_m$ given j and m . $\alpha_{m,j}, \beta_{m,j}$ are small constants that can be specified by users. The default $\alpha_{m,j} = \beta_{m,j} = 10^{-3}$ for all m and j .

Regarding the random effects specification in **zoib**, it is assumed $z \sim N(0, \sigma^2)$ in the case of a single random variable z ; when there are multiple random variables z_1, \dots, z_L , it is assumed $\mathbf{z} \sim N(0, \Sigma)$. **zoib** offers two structures on Σ : variance components (VC) and unstructured (UN).

- In the VC case, Σ is diagonal, indicating all the random variables are independent. **zoib** offers two priors on σ_l , the standard deviation of z_l ($l = 1, \dots, L$): 1) $\sigma_l \sim \text{unif}(0, C)$, where C is a large constant that can be specified by users (default $C = 20$); 2) $\sigma_l \sim \text{half-Cauchy}(C)$, the half- t distribution with degree freedom equal to 1. Symbolically, $f(\sigma_j) \propto (1 + \sigma_j^2 C^{-2})^{-1}$, where C is the scale parameter (Gelman, 2006) (default $C = 20$). The half-Cauchy distribution is the default in **zoib**.
- When Σ is fully parameterized with $L(L + 1)/2$ parameters (the UN structure), we write $\Sigma = \text{Diag}(\sigma_l) \cdot \mathbf{R} \cdot \text{Diag}(\sigma_j)$, where \mathbf{R} is the correlation matrix. The priors for σ_l for $l = 1, \dots, L$ are the same as in the VC case. **zoib** supports L up to 3 in the UN structure. When $L = 2$, there is a single correlation parameter and a uniform prior is imposed $\rho \sim \text{unif}(0, 1)$. When $L = 3$, the uniform prior is imposed on two out of three correlation coefficients, say $\rho_{12} \sim \text{unif}(0, 1)$ and $\rho_{13} \sim \text{unif}(0, 1)$. In order to ensure positive definitiveness of \mathbf{R} , ρ_{23} has to be bounded within (L, U) , where $L = \rho_{12}\rho_{13} - \sqrt{(1 - \rho_{12}^2)(1 - \rho_{13}^2)}$ and $U = \rho_{12}\rho_{13} + \sqrt{(1 - \rho_{12}^2)(1 - \rho_{13}^2)}$. The prior on ρ_{13} is thus specified as $\text{unif}(L, U)$.

All taken together, **zoib** offers 4 options on the prior for the covariance matrix Σ in the case of more than one random variables: VC.unif, VC.halft, UN.unif, and UN.halft.

Implementation in R

The joint distribution $f(\Theta, \gamma | \mathbf{y})$ in the zoib model is not available in closed form. We apply slice sampling(Neal, 2003), a Markov chain Monte Carlo (MCMC) method, to draw posterior samples on the parameters leveraging on the available software JAGS (Plummer, 2014a). Before using **zoib**, users need to download JAGS and the R package **rjags** that offers a connection between R and JAGS. The main function in **zoib** generates a JAGS model object, and the posterior samples on the model parameters, the observed y and their posterior predictive values, and the design matrices $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and \mathbf{x}_4 , as applicable. Convergence diagnostics, mixing of the MCMC chains, summary of the posterior draws, and the deviance information criterion (DIC) (Spiegelhalter et al., 2002) of the model can be calculated using the functions already available in packages **coda** (Plummer et al., 2006) and **rjags** (Plummer, 2014b). Trace plots and auto-correlation plots can be generated, the Gelman-Rubin's potential scale reduction factor (psrf) (Gelman and Rubin, 1992) and multivariate psrf (Brooks and Gelman, 1998) can be computed. To check on the mixing and convergence of the Markov chains, multiple independent Markov chains should be run. More details on the output and functions of **zoib** are provided in Section [Implementation in R](#) below.

The package **zoib** contains 23 functions (Table 1). Users can call the main function `zoib()`, which produces a MCMC (JAGS) model object and posterior samples of model parameters as an MCMC object, among others. Convergence of the MCMC chains can be checked using the `traceplot(MCMC.object)`, `autocorr.plot(MCMC.object)` and `gelman.diag(MCMC.object)` functions provided by package **coda**. Posterior summary of the parameters can be obtained by function `summary()` if the posterior draws are in a format of a `mcmc.list`. The DIC of the proposed model can be calculated using function `dic.samples(JAGS.object)` available in **rjags** for model comparison purposes. Besides these existing functions, **zoib** provides an additional function `check.psrf()` that checks whether multivariate psrf value can be calculated for multi-dimensional model parameters, provides box plots and summary statistics on multiple univariate psrf values, and the `paraplot()` function which provides a visual display on the posterior inferences on the model parameters. The remaining 20 functions are called internally by function `zoib()`. The main function `zoib()` is used as follows:

```
zoib(model, data, zero.inflation = TRUE, one.inflation = TRUE, joint = TRUE,
      random = 0, EUID, link.mu = "logit", link.x0 = "logit", link.x1 = "logit",
      prior.beta = rep("DN", 4), prec.int = 0.001, prec.DN = 0.001,
      lambda.L2 = 0.001, lambda.L1 = 0.001, lambda.ARD = 0.001,
      prior.Sigma = "VC.halft", scale.unif = 20, scale.halft = 20,
      n.chain = 2, n.iter = 5000, n.burn=200 , n.thin = 2)
```

`data` represents the data set to be modeled. `model` presents a symbolic description of the zoib model in the format of formula responses $y \sim \text{covariates } x$. `zero.inflation` and `one.inflation` contain the information on whether the data has inflation at zero or one. `joint` specifies whether to model multiple response variables jointly or separately. `random = 0` indicates the ZOIB model has no random effects; `random = m` (for $m = 1, 2, 3, 4$) instructs `zoib` which linear predictor(s) out of the four (as given in equations (2) to (5)) have a random component. For example, if `random = 13`, then the linear predictors associated with the link function of the mean of the beta regression (1) and the probability of zero inflation (3) have a random component, while the linear predictors associated with the link function of the precision parameters of (2) and the probability one inflation (4) do not have a random

Function	Description
Functions called by users	
<code>zoib()</code>	main function; produces a MCMC (JAGS) model object and posterior samples of model parameters
<code>check.psrf()</code>	checks whether the multivariate psrf value can be calculated for multi-dimensional parameters; provides a box plot and summary statistics for multiple univariate psrf values
<code>paraplot()</code>	plots the posterior mode, mean, or median with Bayesian credible intervals for the parameters from a zoib model.
Internal functions called by <code>zoib()</code>	
<i>fixed-effect model</i>	
<code>fixed()</code>	without y inflation at 0 or 1
<code>fixed0()</code>	with y inflation at 0 only
<code>fixed1()</code>	with y inflation at 1 only
<code>fixed01()</code>	with y inflation at both 0 and 1
<i>joint modeling of ≥ 2 response variables when there is a single random variable</i>	
<code>join.1z()</code>	without y inflation at 0 or 1
<code>join.1z0()</code>	with y inflation at 0 only
<code>join.1z1()</code>	with y inflation at 1 only
<code>join.1z01()</code>	at both 0 and 1
<i>joint modeling of ≥ 2 response variables when there are ≥ 2 random variables</i>	
<code>join.2z()</code>	without y inflation at 0 or 1
<code>join.2z0()</code>	with y inflation at 0 only
<code>join.2z1()</code>	with y inflation at 1 only
<code>join.2z01()</code>	with y inflation at both 0 and 1
<i>separate modeling of ≥ 2 response variables when there is a single random variable</i>	
<code>sep.1z()</code>	without y inflation at 0 or 1
<code>sep.1z0()</code>	with y inflation at 0 only
<code>sep.1z1()</code>	with y inflation at 1 only
<code>sep.1z01()</code>	at both 0 and 1. called by function <code>zoib()</code> .
<i>separate modeling of ≥ 2 response variables when there are ≥ 2 random variables</i>	
<code>sep.2z()</code>	without y inflation at 0 or 1
<code>sep.2z0()</code>	with y inflation at 0
<code>sep.2z1()</code>	with y inflation at 1 only
<code>sep.2z01()</code>	with y inflation at both 0 and 1

Table 1: Functions developed in package **zoib**.

component. Similarly, if `random = 124`, then the linear predictors associated with the mean (1) and precision parameters (2) of the beta distribution, and the probability of one inflation (4) have a random component, but the link function associated with the probability of zero inflation (4) does not. `random = 1234` would suggest all 4 linear predictors have random components. There are total $2^4 - 1 = 15$ possibilities to specify the random components and **zoib** supports all 15 possibilities.

The remaining arguments in function `zoib()` are necessary for Bayesian model formulation and computation, including the hyper-parameter specification in the prior distributions of the parameters in the ZOIB model (`prior.beta`, `prec.int`, `prec.DN`, `lambda.L2`, `lambda.L1`, `lamdda.ARD`, `scale.unif`, `scale.halft`, `prior.Sigma`), the number of Markov chains to run (`n.chain`), the number of MCMC iterations per chain (`n.iter`), and the burn-in period (`n.burn`) and thinning period (`n.thin`). In addition, the link functions that relate linear predictors to $\Pr(y = 0)$, $\Pr(y = 1)$, and $\mu^{(0,1)}$ can be chosen from `logit` (the default), `probit`, and `cloglog`. The link function that links a linear predictor to the sum of the two shape parameters of the beta distribution is the `log` function.

Table 2 lists the functions offered by packages **coda** and **rjags** that can be used to check the convergence of the MCMC chains from the ZOIB models, to compute the posterior summaries of the model parameters, and to calculate the penalized deviance of the converged models.

Examples

We apply the package **zoib** to three examples. In the first example **zoib** is applied to analyze the `GasolineYield` data available in R package **betareg**, to provide a comparison between the results obtained from the two packages. There is no inflation in either 0 or 1 in data `GasolineYield`. In

Function	Description
traceplot()	plots number of iterations vs. drawn values for each parameter in per Markov chain (from package coda)
autocorr.plot()	plots the autocorrelation for each parameter in each Markov chain (from package coda)
gelman.diag()	calculates the potential scale reduction factor (psrf) value for each variable drawn from at least two Markov chains, together with the upper and lower 95% confidence limits. When there are multiple variables, a multivariate psrf value is calculated (from package coda)
dic.samples()	extracts random samples of the penalized deviance from a jags model (from package rjags)
summary()	calculates posterior mean, standard deviation, 50%, 2.5% and 97.5% for each parameters using the posterior draws from Markov chains

Table 2: Existing functions for checking the convergence and mixing of the Markov chain of the ZOIB model and summarizing the posterior samples.

Example 2, **zoib** is applied to a simulated data with two correlated beta variables, where joint modeling of the variable is used with a single random variable. In Example 3, **zoib** is applied to a real life data on alcohol use in California teenagers. Example 3 is used to demonstrate how to model clustered beta variables via **zoib**. The data set in Example 3 can be downloaded from website <http://www.kidsdata.org>. In all three example, the ZOIB model is specified using the generic function **formula** in R. When there are multiple response variables, each variable should be separated by |, such as $y_1|y_2|y_3$ on the left hand side of the formula. On the right side of the formula, it can take up to 5 parts in the following order:

1. fixed-effect variables x_1 in the link function of the mean of the beta distribution;
2. fixed-effect variables x_2 in the link function of the precision parameter of the beta distribution;
3. fixed-effect variables x_3 in the link function of $\Pr(y = 0)$;
4. fixed-effect variables x_4 in the link function of $\Pr(y = 1)$; and
5. random-effects variables z .

x_1 and x_2 should always be specified, even if x_2 contains only an intercept (represented by 1). If there is no zero inflation in any of the y 's, then the x_3 part can be omitted, similarly with x_2 and the random component z . For example, if there are 3 response variables y_1, y_2, y_3 and 2 independent variables ($xx1, xx2$), and none of the y 's has zero inflation, then model $y_1 | y_2 | y_3 \sim xx1 + xx2 | 1 | xx1 | xx2$ implies $x_1 = (1, xx1, xx2)$, $x_2 = 1$ (intercept), $x_3 = \text{NULL}$, $x_4 = (1, xx1)$, $z = (1, xx2)$. If y_1 has inflation at zero, y_3 has inflation at one, and there is no random effect, model $y_1 | y_2 | y_3 \sim xx1 + xx2 | xx1 | xx1$ implies $x_1 = (1, xx1, xx2)$, $x_2 = (1, xx1)$, $x_3 = c(1, xx1)$ for y_1 , $x_4 = (1, xx1)$ for y_3 . The details on how to specify the model using **formula** can be found in the user manual of package **zoib**.

Example 1: univariate fixed-effect beta regression

According to the description in **betareg**, the **GasolineYield** data was collected by [Prater \(1956\)](#) and analyzed by [Atkinson \(1985\)](#). The data set contains 32 observations and 6 variables. The dependent variable is the proportion of crude oil after distillation and fractionation. There is no 0 or 1 inflation in y . **betareg** fits a beta regression model with all 32 observations and 2 independent variables: batch ID (1, ..., 10) corresponding to 10 different crudes that were subjected to experimentally controlled distillation conditions, and temp (quantitative, Fahrenheit temperature at which all gasoline has vaporized). Both batch and temp are treated as fixed effects.

$$\text{logit} \left(\frac{\alpha_1}{\alpha_1 + \alpha_2} \right) = \beta_0 + \beta_1 \cdot \text{temp} + \beta_2 \cdot \text{batch1} + \dots + \beta_{10} \cdot \text{batch9}, \\ \log(\alpha_1 + \alpha_2) = \eta.$$

The R command for fitting the model using **betareg** is

```
library(zoib)
library(betareg)
data("GasolineYield", package = "zoib")
GasolineYield$batch <- as.factor(GasolineYield$batch)
```

```
#### betareg
gy <- betareg(yield ~ temp + batch, data = GasolineYield)
summary(gy)$coeff
```

Fitting the same model in **zoib**, we have

```
#### zoib: fixed effect on batch.
d <- GasolineYield
eg1.fixed <-
  zoib(yield ~ temp + as.factor(batch)| 1, data = GasolineYield, joint = FALSE,
        random = 0, EUID = 1:nrow(d), zero.inflation = FALSE,
        one.inflation = FALSE, n.iter = 1050, n.thin = 5, n.burn = 50)
sample1 <- eg.fixed$coeff
# check convergence of the MCMC chains
traceplot(sample1); autocorr.plot(sample2); gelman.diag(sample1)
```

The procedure took about 25 seconds on a PC with Intel Core-i5 2520M CPU 2.5GHz (2 chains, 1050 iterations, burin-in periods = 5, and thinning period = 5 per chain). The trace plots, the autocorrelation plots, and the values of the potential scale reduction factors (psrf) (Gelman and Rubin, 1992; Brooks and Gelman, 1998) suggest the Markov chains mixed well and reached satisfactory convergence (Appendix Figure A1 and Table A1).

If the 10 batches constitute a random sample of many possible batches and the mean of each batch is of little interest, we can treat batch as a random variable rather than dummying code batch. The following model is a mixed-effects model with a random component in the link function of the mean of the beta distribution.

```
#### zoib: random effect on batch
eg1.random <- zoib(yield ~ temp | 1 | 1, data = GasolineYield, joint = FALSE,
                     random = 1, EUID = GasolineYield$batch, zero.inflation = FALSE,
                     one.inflation = FALSE, n.iter = 10200, n.thin = 50, n.burn = 200)
sample2 <- eg1.random$oripara
summary(sample2); traceplot(sample2); autocorr.plot(sample2); gelman.diag(sample2)
```

The above procedure took about 42 seconds on a PC with Intel Core-i5 2520M CPU 2.5GHz (2 chains, 10200 iterations, burin-in periods = 50, and thinning period = 50 per chain). The trace plots, the auto-correlation plots, and the psrf values suggest that the Markov chains mixed well and converged (Appendix Figure A2 and Table A1).

The inferential results on the parameters from all three analyses (betareg, zoib-fixed, zoib-random) are depicted in Figure 2, which is generated using the **paraplot** function.

```
## posterior inferences from zoib: fixed
summ1 <- summary(sample1); summ1 <- cbind(summ1$stat[, 1], summ1$quant[, c(1, 5)])
## posterior inferences from zoib: random
summ2 <- summary(sample2); summ2 <- cbind(summ2$stat[, 1], summ2$quant[, c(1, 5)])
summ2<- summ2[-4, ]
### inferences from betareg
summ3 <- cbind(c(summ3$mean[, 1], summ3$precision[, 1]), confint(gy))
summ3[12, ] <- log(summ3[12, ]) # log-precision
### plot
names <- rownames(summ3); names[1] <- "intercept"; names[12] <- "log(precision)"
rownames(summ1) <- names; rownames(summ3) <- names
rownames(summ2) <- names[c(1, 2, 12)]
paraplot(summ1, summ2, summ3, legpos = c(2, 10), annotate = TRUE,
         legtext = c("zoib: fixed", "zoib: random", "betareg"))
```

Figure 2 suggests minimal difference between the Bayesian and the frequentist approaches in the inferences of the intercept and regression coefficients in the fixed-effects model. The posterior mean of η (log sum of the two shape parameters in the beta distribution) is numerically smaller from the Bayesian approach compared to the MLE of the frequentist approach. The mixed-effects approach yields similar estimates on η and the regression coefficients as in the fixed effect approaches, but the point estimate on the intercept is smaller. Appendix Figure A3 also presents the posterior mean of y plotted against the observed y for the two zoib models, and suggests both zoib models provide satisfactory goodness-of-fit.

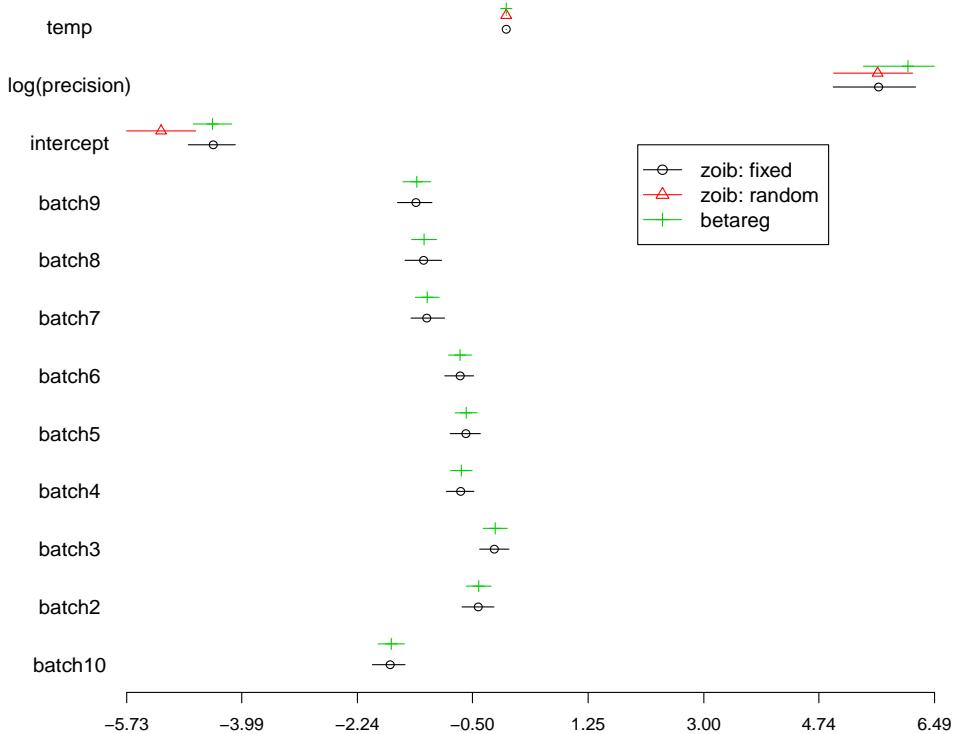


Figure 2: Inferences of model parameters in Example 1 (posterior mean and 95% posterior interval from **zoib**; MLE and 95% CI from **betareg**).

Example 2: bivariate repeated measures

Example 2 demonstrates how to jointly model multiple $[0, 1]$ -bounded response variables using **zoib**. The data set contains two beta variables ($\mathbf{y}_{i1}, \mathbf{y}_{i2}$) from 200 independent cases ($i = 1, \dots, 200$). Both \mathbf{y}_{i1} and \mathbf{y}_{i2} are repeatedly measured at a set of covariate values $\mathbf{x} = (0.1, 0.2, 0.3, 0.4, 0.5, 0.6)$. That is, $\mathbf{y}_{i1} = (y_{i11}, y_{i12}, \dots, y_{i16})$, and $\mathbf{y}_{i2} = (y_{i21}, y_{i22}, \dots, y_{i26})$. BiRepeated is a simulated data set from the following model,

$$\text{logit} \left(\frac{\alpha_{ijk,1}}{\alpha_{ijk,1} + \alpha_{ijk,2}} \right) = (\beta_{0j} + u_{i1}) + (\beta_{1j} + u_{i2})x_{ijk} \\ \log(\alpha_{ijk,1} + \alpha_{ijk,2}) = \eta_j, \text{ and} \\ \mathbf{u}_i = (u_{i1}, u_{i2}) \sim N_{(2)}(0, \Sigma), \text{ where } \Sigma = \text{Diag}(\sigma) \mathbf{R} \text{ Diag}(\sigma) \quad (7)$$

where $i = 1, \dots, 200$ and $k = 1, \dots, 6$, $\beta_{01} = -1, \beta_{11} = 1, \beta_{02} = -2, \beta_{12} = 2, \rho = 0$, and $\sigma^2 = (\sigma_1^2, \sigma_2^2) = (0.2, 0.2)$. σ_1 and σ_2 are the marginal standard deviation of the two random variables u_{i1} and u_{i2} , and $\mathbf{R} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$ is the correlation matrix. The data is available in R by name BiRepeated in package **zoib**. The joint model as given in equation (7) is applied to the data. The priors for the model parameters are $\beta_{0j} \sim N(0, 10^{-3}), \beta_{1j} \sim N(0, 10^{-3}), \eta_j \sim N(0, 10^{-3}), \sigma_j \sim \text{unif}(0, 20)$, and $\rho \sim \text{unif}(-1, 1)$. The R codes for realizing the above model are

```
library(zoib)
data("BiRepeated", package = "zoib")
eg2 <- zoib(y1|y2 ~ x|1|x, data = BiRepeated, random = 1, EUID = BiRepeated$id,
             zero.inflation = FALSE, one.inflation = FALSE, prior.Sigma = "UN.unif",
             n.iter = 7000, n.thin = 25, n.burn = 2000)
post.sample <- eg2$oripara; summary(post.sample)
```

The above procedure took about 3 hours 55 minutes on a Linux server with 2.4 GHz AMD Opteron processors (2 chains, 7000 iterations, burin-in periods is 2000, and thinning period is 25 per chain). The trace plots, the auto-correlation plots, and the psrf values suggest that the Markov chains mixed well

and converged (Appendix Figures A4 and Table A2).

The Bayesian inferences on β_{0j} , β_{1j} and η_j for $j = 1, 2$ are given in Table 3. Note the purpose of Example 2 is to demonstrate how **zoib** can model the correlated data; so only one simulated data set is used. The posterior means of β and η_j are nevertheless close to the true parameter values used to simulate the data, even with the finite not-so-large sample size ($n = 200$).

Parameter	posterior mean	posterior median	2.5% quantile	97.5% quantile
β_{01}	-0.957	-0.958	-1.059	-0.866
β_{11}	0.906	0.903	0.720	1.092
β_{02}	-2.022	-2.020	-2.119	-1.935
β_{12}	2.040	2.040	1.865	2.234
η_1	2.505	2.506	2.433	2.582
η_2	3.014	3.017	2.926	3.091
σ_1^2	0.180	0.179	0.135	0.235
σ_2^2	0.334	0.331	0.152	0.530
ρ	-0.70	-0.70	-0.84	-0.62

Table 3: Bayesian inferences of the joint ZOIB model parameters in Example 2.

Example 3: clustered zero-inflated beta regression

In this example, **zoib** is applied to the county-level monthly alcohol use data collected from students in California from year 2008 to 2010. The data is available in **zoib** by name `AlcoholUse`. The data can be downloaded at <http://www.kidsdata.org>. `AlcoholUse` contains the percentage of public school students in grades 7, 9, and 11 reporting the number of days in which they drank alcohol in the past 30 days by gender (students at the "Non-Traditional" grade level refer to those enrolled in Community Day Schools or Continuation Education and are not included in this analysis). The following model is fitted to the data

$$\begin{aligned} \text{logit}\left(\frac{\alpha_{ij,1}}{\alpha_{ij,1} + \alpha_{ij,2}}\right) &= (\beta_{1,0} + u_i) + \beta_1 \mathbf{x}_{ij} \\ \log(\alpha_{ij,1} + \alpha_{ij,2}) &= \eta \\ \text{logit}(p_{ij}) &= \beta_{2,0} + \beta_2 \mathbf{x}_{ij} \\ u_i &\sim N(0, \sigma^{-2}) \end{aligned}$$

where u_i is the cluster-level (county-level) random variable ($i = 1, \dots, 56$) and j indexes the j^{th} case in cluster i . $\beta_{1,1}$ contains the regression coefficients associated with the main effects associated with gender, grade, and the mid-point of each days bucket on which teenagers drank alcohol, and the interaction between gender and grade; so does $\beta_{2,1}$. σ^{-2} is the precision of the distribution of random effect u_i . The prior specification of the model parameters are: $\beta_{1,0} \sim N(0, 10^{-3})$, $\beta_{2,0} \sim N(0, 10^{-3})$, $\beta_{1,k} \stackrel{\text{ind}}{\sim} N(0, 10^{-3})$ and $\beta_{2,k} \stackrel{\text{ind}}{\sim} N(0, 10^{-3})$ for $k = 1, \dots, 6$, $\eta \sim N(0, 10^{-3})$, and $\sigma \sim \text{unif}(0, 20)$. The R codes for realizing the model in **zoib** are

```
data("AlcoholUse", package = "zoib")
AlcoholUse$Grade <- as.factor(AlcoholUse$Grade)
eg3 <- zoib(Percentage ~ Grade * Gender + MedDays|1|Grade * Gender + MedDays|1,
            data = AlcoholUse, random = 1, EUID = AlcoholUse$County,
            zero.inflation = TRUE, one.inflation = FALSE, joint = FALSE,
            n.iter = 5000, n.thin = 20, n.burn = 1000)
sample1 <- eg3$coeff
summary(sample1)
```

The above procedure took about 10 hours 56 minutes on a Linux server with 2.4 GHz AMD Opteron processors (2 chains, 5000 iterations, burin-in periods is 1000, and thinning period is 20 per chain). The trace plots, the auto-correlation plots, and the psrf values suggest that the Markov chains mixed well and reached satisfactory convergence (Appendix Figures A5 and Table A4).

The results from Example 3 are presented in Table 4. The posterior mean difference in the logit(mean) of the beta distribution between a 9-th grader and a 7-th grader is 0.702 ($\beta_{1,1}$), assuming they are of the same gender, and fall in the same Days Bucket. Similarly, the posterior mean difference in logit(mean) of the beta distribution between a male and a female students is -0.0503 ($\beta_{1,3}$), assuming

they are equal with regard to other covariates. The posterior mean difference logit($\text{Pr}(y = 0)$) between a 9-th grader and a 7-th grader is $\beta_{2,1} = -0.563$; in other words, the ratio in the odds of not drinking alcohol between the a 7-th grader and a 9-th grader is $\exp(0.563) = 1.75$. The other parameters in β_1 and β_2 can be interpreted in a similar manner. The posterior mean of the log(sum of the two shape parameters) η in the beta distribution is 4.389, and the posterior mean of the variance of the random effect u_i is 0.021.

Effect	Parameter	mean	median	2.5% quantile	97.5% quantile
Intercept	$\beta_{1,0}$	-2.392	-2.392	-2.484	-2.299
Grade 9	$\beta_{1,1}$	0.702	0.702	0.609	0.791
Grade 11	$\beta_{1,2}$	0.955	0.956	0.869	1.036
Gender M	$\beta_{1,3}$	-0.053	-0.052	-0.156	0.054
MedDays	$\beta_{1,4}$	-0.092	-0.092	-0.096	-0.088
Grade 9*Gender M	$\beta_{1,5}$	-0.123	-0.118	-0.255	0.003
Grade 11*Gender M	$\beta_{1,6}$	0.053	0.055	-0.087	0.193
intercept	$\beta_{2,0}$	-3.365	-3.332	-4.158	-2.635
Grade 9	$\beta_{2,1}$	-0.563	-0.572	-1.648	0.427
Grade 11	$\beta_{2,2}$	-0.874	-0.884	-2.027	0.181
Gender M	$\beta_{2,3}$	0.465	0.469	-0.382	1.329
MedDays	$\beta_{2,4}$	0.028	0.028	-0.003	0.062
Grade 9*Gender M	$\beta_{2,5}$	-0.246	-0.213	-1.628	0.999
Grade 11*Gender M	$\beta_{2,6}$	-0.664	-0.695	-2.117	1.015
η		4.384	4.385	4.302	4.463
σ^2		0.021	0.020	0.011	0.034

Table 4: Posterior inferences (Example 3).

Discussion

We have introduced an R package for obtaining the Bayesian inferences from the beta regression and zero/one inflated beta regression. We have provided the methodological background behind the package and demonstrated how to apply the package using both real-life and simulated data. **zoib** is more versatile and comprehensive from a modeling perspective compared to other R packages **betareg** and **Bayesainbetareg** on beta regression. First, **zoib** accommodates boundary inflation at 0 or 1. Second, it models clustered and correlated beta variables by introducing random components into the linear predictors of the link functions, and users can specify which linear predictors have a random component. Last but not least, the Bayesian inferential approach provides a convenient way for obtaining inferences for parameters that can be computationally expensive in the frequentist approach, such as the marginal means of response variables when there are random effects. For the regression coefficients in a linear predictor, 4 different priors are offered with options for penalized regression if needed. DIC criteria can be calculated using existing function from package **rjags** for model comparison purposes. Future updates to the **zoib** package include the development of more efficient computational algorithms to shorten the computational time in running the MCMC chains, especially when a zoib model contains a relatively large number of parameters.

Acknowledgements

The authors would like to thank two anonymous reviewers for their valuable comments and suggestions that have greatly improve the quality of the manuscript.

Bibliography

- A. Atkinson. *Plots, Transformations and Regression: An Introduction to Graphical Methods of Diagnostic Regression Analysis*. New York: Oxford University Press, 1985. [p40]
- S. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7:434–455, 1998. [p38, 41]
- E. Cepeda-Cuervo. *Modelagem da variabilidade em modelos lineares generalizados*. PhD thesis, Instituto de Matemáticas, Universidade Federal de Rio de Janeiro, 2001. [p34, 35]

- E. Cepeda-Cuervo. Beta regression models: Joint mean and variance modeling. *Journal of Statistical Theory and Practice*, 9(1):134–145, 2015. [p35]
- F. Cribari-Neto and A. Zeileis. Beta regression in R. *Journal of Statistical Software*, 34(2):1–24, 2010. [p35, 36]
- M. Dalrymplea, I. Hudsona, and R. Ford. Finite mixture, zero-inflated poisson and hurdle models with application to sids. *Computational Statistics & Data Analysis*, 41:491–504, 2003. [p35]
- S. Ferrari and F. Cribari-Neto. Beta regression for modelling rates and proportions. *Journal of the Royal Statistical Society, Series B*, 31:799–815, 2004. [p34]
- A. Gelman. Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis*, 1: 515–533, 2006. [p38]
- A. Gelman and D. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–511, 1992. [p38, 41]
- B. Grün, I. Kosmidis, and A. Zeileis. Extended beta regression in R: Shaken, stirred, mixed, and partitioned. *Journal of Statistical Software*, 48(11):1–25, 2012. [p35]
- L. Hatfield, M. Boye, M. Hackshaw, and B. Carlin. Models for survival times and longitudinal patient-reported outcomes with many zeros. *Journal of the American Statistical Association*, 107:875–885, 2012. [p34, 35]
- I. Kosmidis and D. Firth. A generic algorithm for reducing bias in parametric estimation. *Electronic Journal of Statistics*, 4:1097–1112, 2010. [p35]
- D. V. Lindley and A. F. M. Smith. Bayes estimates for the linear model. *Journal of the Royal Statistical Society: Series B*, 34(1):1–41, 1972. [p37]
- F. Liu and Y. Kong. *zoib: Bayesian Inference for Beta Regression and Zero/One Inflated Beta Regression*, 2014. URL <https://CRAN.R-project.org/package=zoib>. R package version 1.0. [p35]
- F. Liu and Q. Li. A Bayesian model for joint analysis of multivariate repeated measures and time to event data in crossover trials. *Statistics Methods in Medical Research*, 2014. doi: 10.1177/0962280213519594. [p35]
- D. MacKay. Bayesian methods for back-propagation networks. In E. Domany, J. van Hemmen, and K. Schulten, editors, *Models of Neural Networks III: Association, Generalization, and Representation*, pages 211–254. Springer-Verlag, New York, 1996. [p37]
- M. Marin, J. Rojas, D. Jaimes, H. A. G. Rojas, and E. Cepeda-Cuervo. *Bayesianbetareg: Bayesian Beta Regression: Joint Mean and Precision Modeling*, 2014. URL <https://CRAN.R-project.org/package=Bayesianbetareg>. R package version 1.2. [p35]
- R. Neal. *Bayesian Learnings for Neural Networks*. University of Toronto, Canada, 1994. [p37]
- R. M. Neal. Slice sampling. *Annals of Statistics*, 31(3):705–767, 2003. [p38]
- R. Ospina and S. L. Ferrari. A general class of zero-or-one inflated beta regression models. *Computational Statistics & Data Analysis*, 56(6):1609–1623, 2012. [p35]
- P. Paolino. Maximum likelihood estimation of models with beta-distributed dependent variables. *Political Analysis*, 9(4):325–346, 2001. [p34]
- T. Park and G. Casella. Bayesian lasso. *Journal of the American Statistical Association*, 103(482):681–686, 2008. [p37]
- M. Plummer. *JAGS*. <http://mcmc-jags.sourceforge.net/>, 2014a. [p35, 38]
- M. Plummer. *rjags: Bayesian Graphical Models using MCMC*, 2014b. URL <https://CRAN.R-project.org/package=rjags>. R package version 3-14. [p38]
- M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, 2006. URL <https://CRAN.R-project.org/doc/Rnews/>. [p38]
- N. Prater. Estimate gasoline yields from crudes. *Petroleum Refiner*, 35:236–238, 1956. [p40]
- R. Prentice. Binary regression using an extended beta-binomial distribution, with discussion of correlation induced by covariate measurement. *Journal of the American Statistical Association*, 81: 321–327, 1986. [p34]

- A. Simas, W. Barreto-Souza, and A. Rocha. Improved estimators for a general class of beta regression models. *Computational Statistics & Data Analysis*, 54:348–366, 2010. [p34, 36]
- M. Smithson and J. Verkuilen. A better lemon squeezer? Maximum-likelihood regression with beta-distributed dependent variables. *Psychological Methods*, 11:54–71, 2006. [p34, 35]
- D. J. Spiegelhalter, N. G. Best, B. P. Carlin, and A. van der Linde. Bayesian measures of model complexity and fit (with discussion). *Journal of the Royal Statistical Society, Series B*, 64(4):583–639, 2002. [p38]
- C. Swearingen, M. Melguizo Castro, and Z. Bursac. Modeling percentage outcomes: The %beta_regression macro. *SAS Global Forum 2011: Statistics and Data Analysis*, paper 335, 2011. [p35]
- C. Swearingen, M. Melguizo Castro, and Z. Bursac. Inflated beta regression: Zero, one, and everything in between. *SAS Global Forum 2012: Statistics and Data Analysis*, paper 325, 2012. [p35]
- D. Williams. Extra binomial variation in logistic linear models. *Applied Statistics*, 31:144–148, 1982. [p34]
- A. Zeileis, T. Hothorn, and K. Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statisticss*, 17:492–514, 2008. [p35]
- A. Zeileis, F. Cribari-Neto, B. Grün, I. Kosmidis, A. B. Simas, and A. V. Rocha. *Beta Regression for Rates and Proportions*, 2014. URL <https://CRAN.R-project.org/package=betareg>. R package version 3.0-5. [p35]

Fang Liu
University of Notre Dame
Notre Dame, IN 46530
USA
fang.liu.131@nd.edu

Yunchuan Kong
Chinese University of Hong Kong
Shatin, N.T.
Hong Kong
ykong@cuhk.edu.hk

Appendices

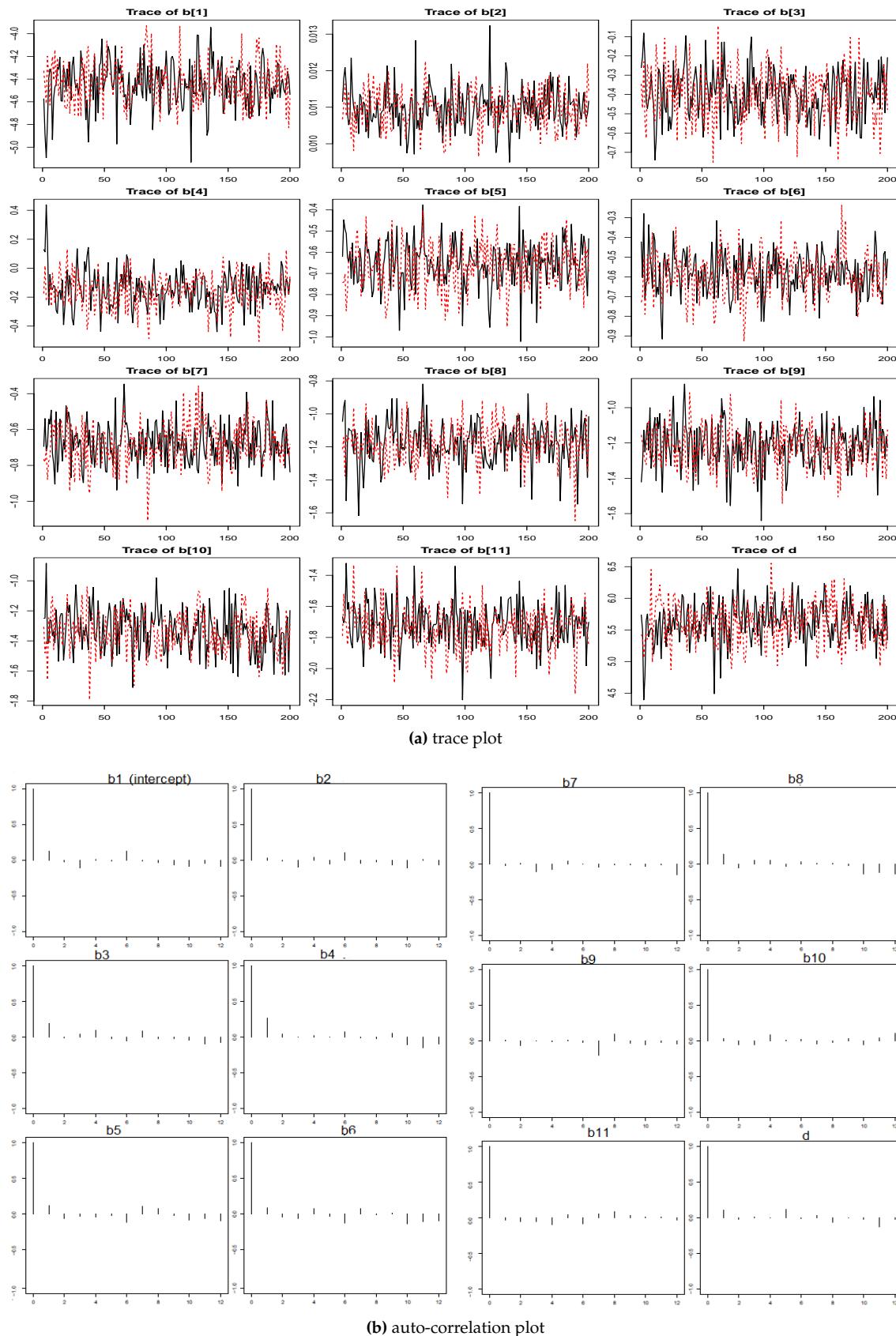


Figure A1: Trace and auto-correlation plots in the zoib-fixed model in Example 1.

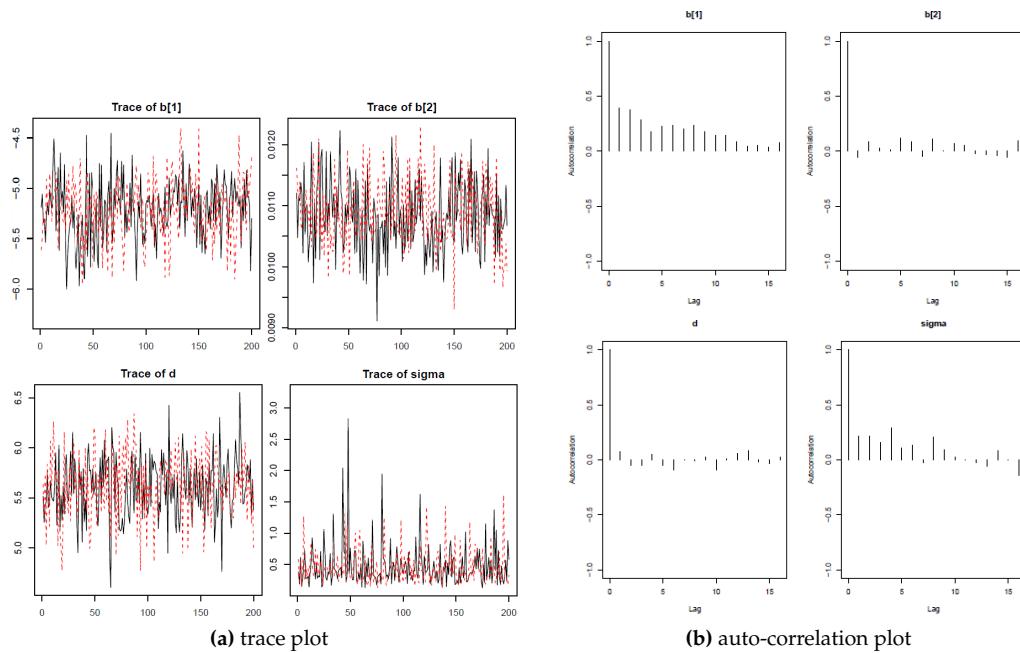


Figure A2: Trace and auto-correlation plots in the zoib-random model in Example 1.

Parameter	zoib-fixed		zoib-random	
	point	upper bound (95%)	point	upper bound (95%)
$\beta_{1,0}$ (intercept)	0.997	0.998	1.036	1.037
$\beta_{1,1}$	1.001	1.002		
$\beta_{1,2}$	1.008	1.008		
$\beta_{1,3}$	1.013	1.040		
$\beta_{1,4}$	0.998	1.006		
$\beta_{1,5}$	1.005	1.007		
$\beta_{1,6}$	1.000	1.014		
$\beta_{1,7}$	0.999	1.004		
$\beta_{1,8}$	1.003	1.033		
$\beta_{1,9}$	1.001	1.015		
$\beta_{1,10}$	1.007	1.048	1.003	1.017
β_{20} (temperature)	1.018	1.032	1.002	1.006
σ^2			0.997	0.997

Table A1: Potential scale reduction factors of the zoib model parameters in Example 1.

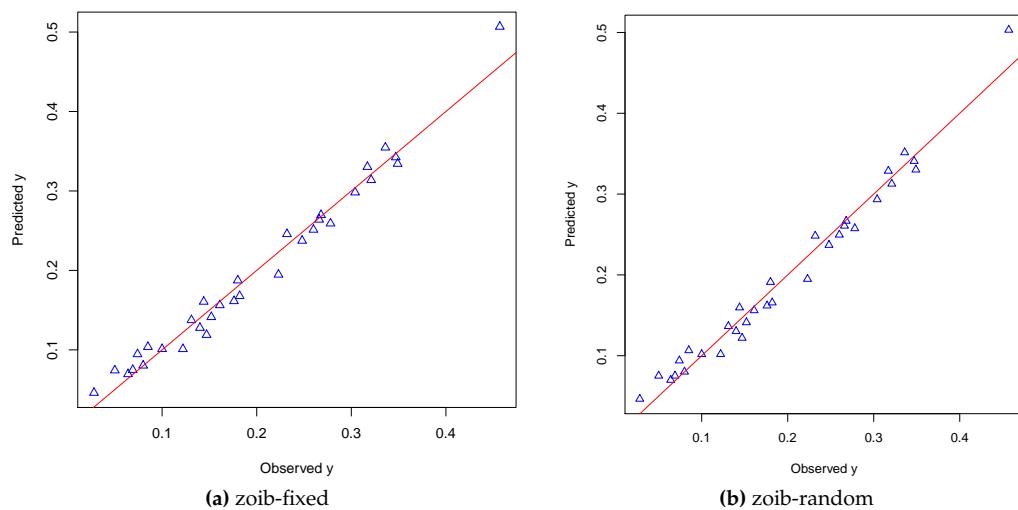


Figure A3: Posterior mean of Y vs. observed Y in Example 1.

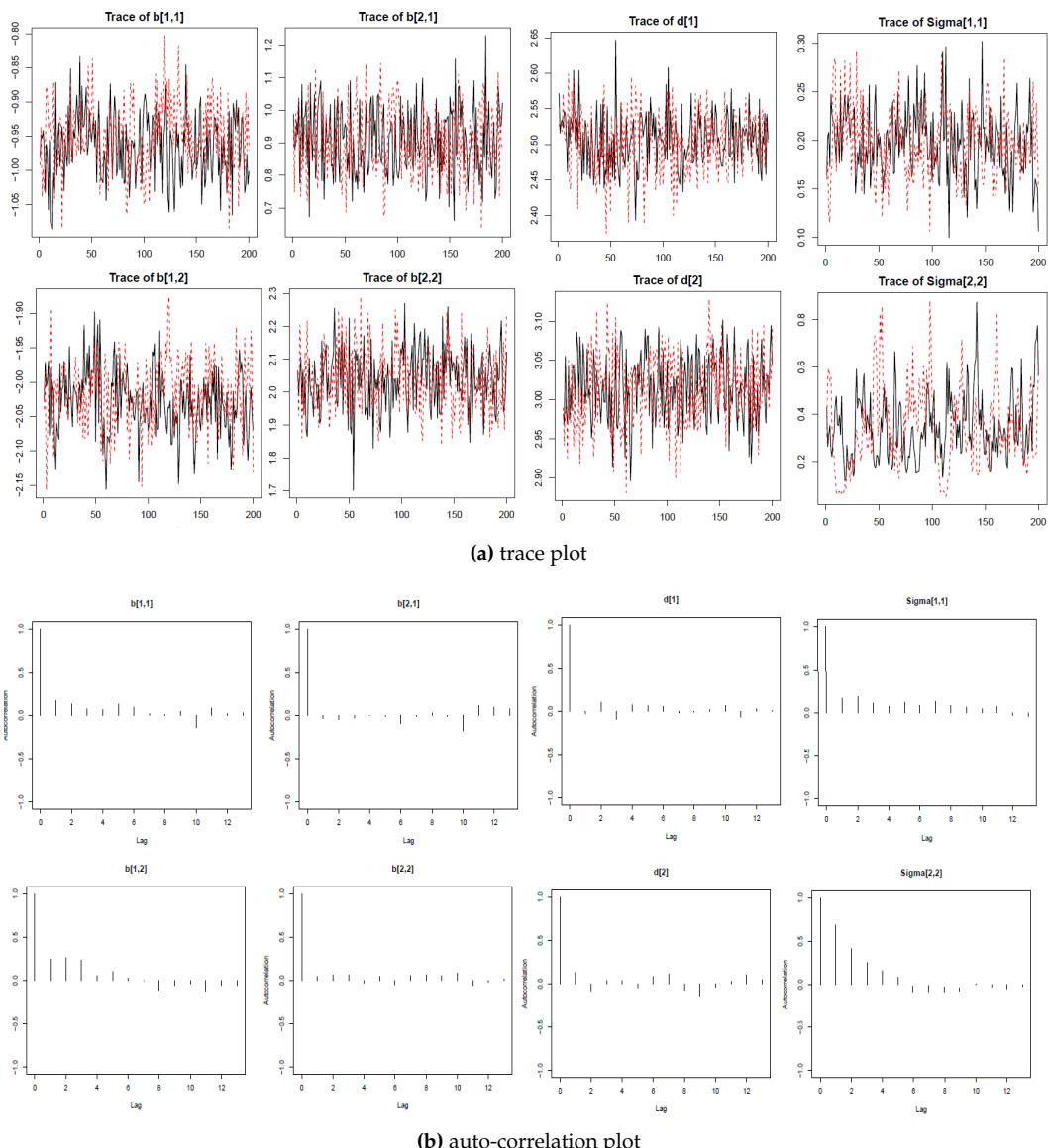


Figure A4: Example 1, zoib-fixed.

Parameter	point	upper bound (95%)
$\beta_{0,1}$	1.083	1.332
$\beta_{1,1}$	1.004	1.035
$\beta_{0,2}$	1.163	1.582
$\beta_{1,2}$	1.008	1.100
σ_1^2	0.997	0.997
σ_2^2	1.036	1.165

Table A2: Potential scale reduction factors in Example 2.

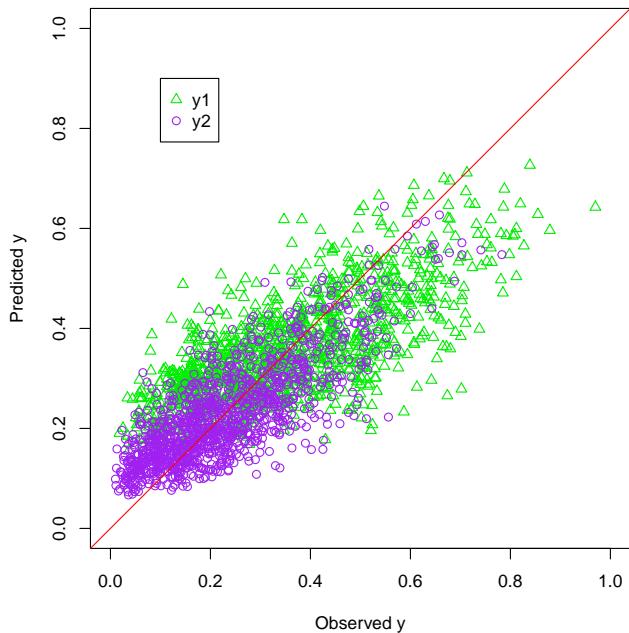


Table A3: Posterior mean of Y vs. observed Y (Example 2).

Parameter	$\beta_{1,0}$	$\beta_{1,1}$	$\beta_{1,2}$	$\beta_{1,3}$
point	1.018	1.005	1.000	0.997
upper limit	1.092	1.042	1.015	0.999
(95% CI)				
parameter	$\beta_{1,4}$	$\beta_{1,5}$	$\beta_{1,6}$	η
point	0.997	1.006	0.997	1.068
upper limit	1.000	1.047	0.998	1.279
(95% CI)				
parameter	$\beta_{2,0}$	$\beta_{2,1}$	$\beta_{2,2}$	$\beta_{2,3}$
point	0.998	1.001	0.998	0.998
upper limit	1.000	1.001	0.999	1.004
(95% CI)				
parameter	$\beta_{2,4}$	$\beta_{2,5}$	$\beta_{2,6}$	σ^2
point	0.996	1.012	0.996	1.001
upper limit	0.997	1.012	0.999	1.020
(95% CI)				

Table A4: Potential scale reduction factors in Example 3.

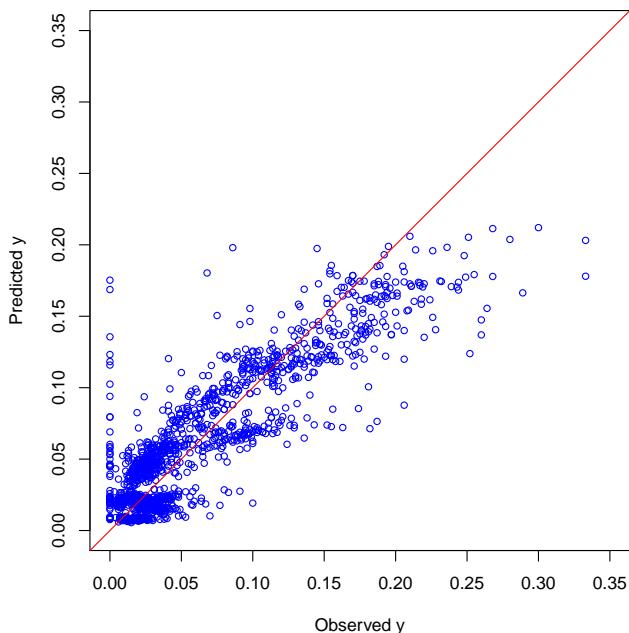
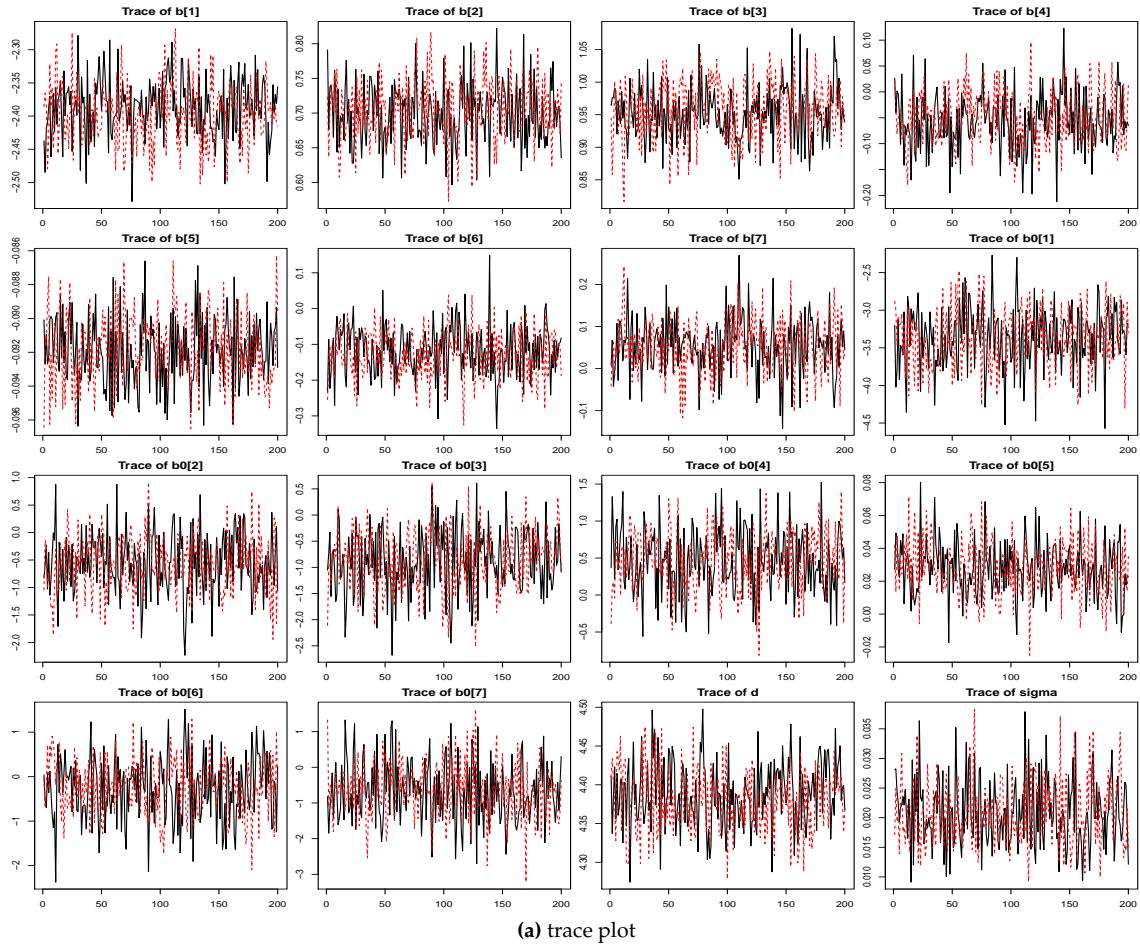
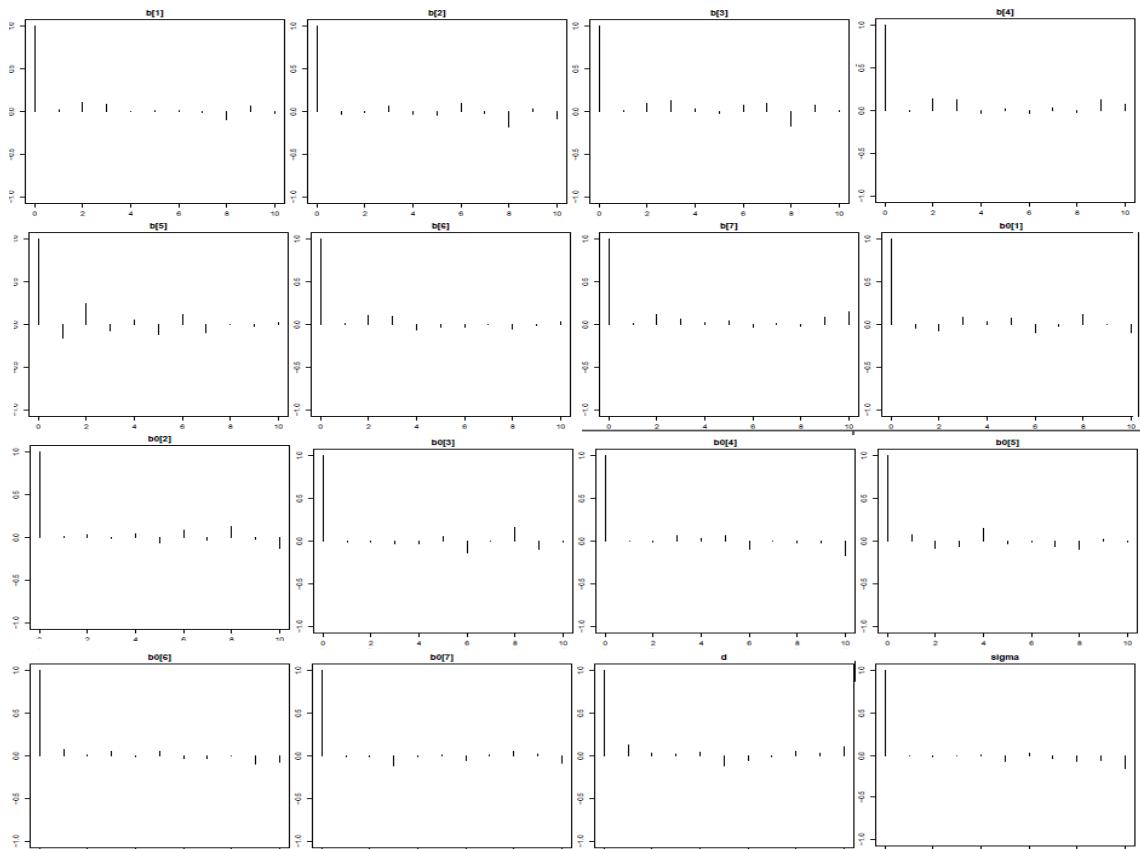


Table A5: Posterior mean of Y vs. observed Y (Example 2).



(a) trace plot



(b) auto-correlation plot

Figure A5: Trace and auto-correlation plots in the zoib-fixed model in Example 1.

apc: An R Package for Age-Period-Cohort Analysis

by Bent Nielsen

Abstract The **apc** package includes functions for age-period-cohort analysis based on the canonical parametrisation of Kuang et al. (2008a). The package includes functions for organizing the data, descriptive plots, a deviance table, estimation of (sub-models of) the age-period-cohort model, a plot for specification testing, plots of estimated parameters, and sub-sample analysis.

Introduction

Age-period-cohort models are extensively used in actuarial sciences, demography, epidemiology and social sciences. They have an identification problem in that the predictor is defined from time effects for age, period and cohort, but these time effects cannot be fully recovered from the predictor. The **apc** package, see Nielsen (2015a), implements the solution proposed by Kuang et al. (2008a) and Nielsen (2014), which is to abandon the time effects and reparametrise the predictor in terms of freely varying parameters. The vector of freely varying parameters is of a lower dimension than the vector of the original time effects. These freely varying parameters describe the variation of the likelihood function fully. The intention with the package is to focus on the aspects of the time effect that are identified by the likelihood.

The age-period-cohort model has three time scales: *age*, *period* and *cohort*. These are linked through the identity $age + cohort = period$. The package is concerned with the situation where two of the time scales are measured in discrete and equidistant time. The third time scale can then be computed through $age + cohort = period$. The choice of these two indices vary from application to application. For instance, the example in this paper is a an age-period array of annual counts of mesothelioma deaths by age group. The interface of the package is constructed in such a way that the user does not need to keep track of the coordinate system. Internally, the package uses an age-cohort coordinate system to exploit that period is a symmetric function of age and cohort.

The statistical model is a generalized linear model with a predictor of the form

$$\mu_{age,cohort} = \alpha_{age} + \beta_{period} + \gamma_{cohort} + \delta. \quad (1)$$

The likelihood is a function of the predictor $\mu_{age,cohort}$. In turn, the predictor is constructed from time effects for age, α_{age} , period, β_{period} , and cohort, γ_{cohort} . If only we could estimate the time effects, we could learn about the predictor through manipulations of the time effects. This would be done by treating these as time series: plot them, fit time series models to them, perhaps forecast future values, and finally combine them to get the predictor. However, the time effects are not fully identifiable from the predictor so this approach has to be pursued with some care.

The identification problem is that linear trends can be moved between the time effects without changing the predictor. Indeed, the predictor in (1) satisfies

$$\begin{aligned} \mu_{age,cohort} &= (\alpha_{age} + a + d \times age) + (\beta_{period} + b - d \times period) \\ &\quad + (\gamma_{cohort} + c + d \times cohort) + (\delta - a - b - c), \end{aligned} \quad (2)$$

for any choice of a , b , c and d . In other words, knowledge of the predictor from the likelihood is not enough to pin down the time effects. The problem is discussed, for instance, by Carstensen (2007), Clayton and Schifflers (1987a,b), Holford (1985), Kuang et al. (2008a), Luo (2013), Nielsen and Nielsen (2014), O'Brien (2011), and Yang and Land (2013). There appears to be two types of solutions to the problem: either to introduce four constraints to the time effects or to abandon the time effects and seek a parsimonious and freely varying parametrisation of the predictor. The **apc** package follows the latter approach. With the former approach, the constraints must come from some external argument as the likelihood carries no information in this respect. An example is the ‘intrinsic estimator’, which is based on a particular choice of a generalized matrix inverse, see Yang and Land (2013). The constraints need to be tracked carefully through the analysis to clarify which inferences are driven by data and which inferences are driven by the constraints; see Nielsen and Nielsen (2014) for an algebraic analysis both for frequentist and Bayesian settings.

The **apc** package addresses the identification through the parsimonious parametrization of the predictor suggested by Kuang et al. (2008a), see also Nielsen (2014). This exploits the fact that the second differences of the time effects are identified and that the predictor itself is also identifiable.

As an example, the age second difference is $\Delta^2\alpha_{age} = \Delta\alpha_{age} - \Delta\alpha_{age-1}$, where the first differences are $\Delta\alpha_{age} = \alpha_{age} - \alpha_{age-1}$. The second differences are identifiable from the predictor through

$$\Delta^2\alpha_{age} = \mu_{age,cohort} - \mu_{age-1,cohort+1} - \mu_{age-1,cohort} + \mu_{age-2,cohort+1}. \quad (3)$$

The double differences are well-known to be identifiable in the age-period-cohort model (1), see for instance [Clayton and Schifflers \(1987b\)](#). [Martínez Miranda et al. \(2015\)](#) give a log-odds interpretation of the double differences. If the double differences are all zero, or equivalently, all time effects are linear then the model reduces to a linear plane. That linear plane is parametrised by the any three elements of the predictor, $\mu_{age,cohort}, \mu_{age^+,cohort^+}, \mu_{age^+,cohort^{\ddagger}}$ say, for which the coordinates form a triangle rather than a line. Accordingly, [Kuang et al. \(2008a\)](#) suggest to use

$$\xi = (\mu_{age,cohort}, \mu_{age^+,cohort^+}, \mu_{age^+,cohort^{\ddagger}}, \dots, \Delta^2\alpha_{age}, \dots, \Delta^2\beta_{period}, \dots, \Delta^2\gamma_{cohort}, \dots)' \quad (4)$$

A dimension reduction of 4 is achieved, since double differencing reduces each set of time effects by two elements. The parameter is invariant to the identification problem (2) due to (3). To be of any use it has to be shown how the predictor can be formed from the parsimonious parameter. Double summation of double differences of the time effects results in the original time effects up to a linear trend. Thus, for the user of the package it is sufficient to know that predictor can be found from the parsimonious parameter through a formula of the form

$$\mu_{age,cohort} = a \text{ linear plane} + \sum_{age} \sum \Delta^2\alpha_s + \sum_{period} \sum \Delta^2\beta_s + \sum_{cohort} \sum \Delta^2\gamma_s. \quad (5)$$

The formula that is actually used internally in the package is shown in (11). In that parametrisation the linear plane is a function exclusively of $\mu_{age,cohort}, \mu_{age^+,cohort^+}, \mu_{age^+,cohort^{\ddagger}}$ so that linear plane parameters and double differences are separated. The formula that is used in default plots uses detrended versions of the double sums of double differences, see (15). This allows the user to focus on deviations from linearity. The parsimonious predictor is identified since it can be shown that different values $\xi^+ \neq \xi^{\ddagger}$ imply different predictors $\mu^+ \neq \mu^{\ddagger}$, see [Kuang et al. \(2008a\)](#). In the context of an exponential family ξ is therefore the canonical parameter and the family is regular.

An existing package, **Epi**, for age-period-cohort analysis is created by [Carstensen et al. \(2014\)](#). It is based on [Carstensen \(2007\)](#). It has a series of functions for demographic and epidemiological analysis as well as some functions for age-period-cohort analysis. There are several differences between the packages **apc** and **Epi**. First, **apc** uses the canonical parametrization of [Kuang et al. \(2008a\)](#), whereas **Epi** does not. Second, **apc**, at present, is concerned with age-period-cohort data in various matrix formats. These have to be vectorized before fitting the generalized linear model, but this is done internally, so that the user only has to consider the original matrix format, while **Epi** takes data in vectorized form and uses the data frame format. Third, at present, **apc** cannot handle the problem of over-lapping cohorts: the people of age 25 in April 2015 will have been born either in 1989 or in 1990. Conversely those born in 1990 will either be 24 or 25 in April 2015. When data on all three time scales are available, cells can be sub-divided into two Lexis triangles with non-overlapping cohorts. **Epi** has functions for exploiting such information.

The main contributions of the **apc** are therefore

1. to consider data in matrix format indexed in a number of different ways;
2. to provide specification graphics illustrating the quality of the fit;
3. to estimate the model parametrised in terms of the canonical parameter ξ in (4);
4. to visualize the components of the representation of the predictor μ in (5) as time series;
5. and to do this from a range of sub-models where some of the components of ξ or, correspondingly, of the time effects, are set to zero.

The remainder of the paper will illustrate this. It is envisaged to extend the package with further time series tools in the future. For reference, a theory of forecasting in the age-period-cohort model is given in [Kuang et al. \(2008b\)](#) and used in for instance [Martínez Miranda et al. \(2015\)](#).

The **apc** package

The **apc** package includes functions for organizing the data, descriptive plots, a deviance table, estimation of (sub-models of) the age-period-cohort model, a plot for specification testing, plots of estimated parameters, and sub-sample analysis. These are described in turn.

The example for this analysis is a data set for annual mesothelioma deaths in the UK taken from [Martínez Miranda et al. \(2015\)](#). It is thought that most mesothelioma deaths are caused by exposure to

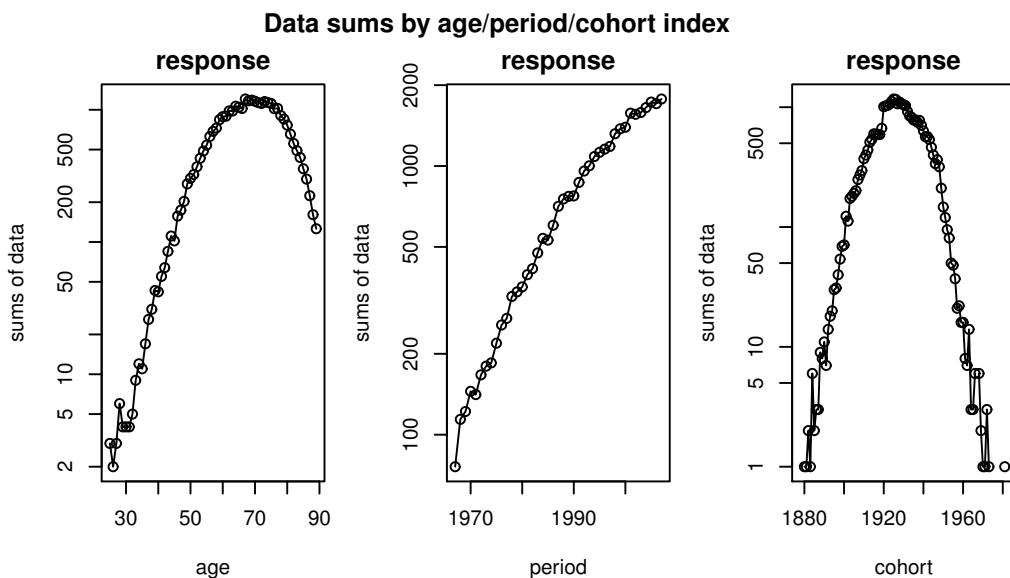


Figure 1: Data sums by age, by period and by cohort.

asbestos. The data set has counts of male deaths by age 25–89 and by 1967–2007. There is no direct measure for the exposure to asbestos.

Organizing the data

Age-period-cohort data may include doses and responses or just responses. They come in different types of data arrays. **apc** allows eight matrix formats arising from the choice of two indices from the age, period, and cohort time scales, a triangular format for chain-ladder analysis, as well as a generalized trapezoid format encompassing the other options, see (8). A special data format **apc.data.list** is used to keep track of the data format and the time scales. An artificial response-only data set organized in age-period format can be coded as follows

```
> library(apc)
> m.data <- matrix(data = seq(12), nrow = 3, ncol = 4)
> data.artificial <- apc.data.list(m.data, "AP", age1 = 25, per1 = 1990, unit = 5)
> data.artificial$response
[1,] [2,] [3,] [4,]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

The value returned to the variable **data.artificial** from **apc.data.list** is a list with ten elements. The list includes the response, **m.data**; a dose which is set to **NULL** in this example; the data format "AP"; and information about the real time scales. This is all based on the arguments of the function **apc.data.list**. The first argument defines the response data, while the second argument signifies that the response matrix is rectangular with coordinates in age-period format. The remaining arguments are optional. In this case information about the times scales have been given. This shows that the real time scales are 25, 30, 35 for age and 1990, 1995, 2000, 2005 for period, which in turn implies that the cohorts are 1955, 1960, ..., 1980. At this point **data.artificial\$response** simply stores the input matrix. We can think of it as varying in a simple age-period coordinate system. From a practical viewpoint this is not particularly helpful. Therefore **apc** will exploit the optional information on the real time scales when reporting estimators in the subsequent analysis.

A variety of data from the literature are pre-coded including the asbestos data from [Martínez Miranda et al. \(2015\)](#). The available information for that data set is exactly as in the previous example: a data matrix for responses in period-age format, though much larger, along with information about the time scales. It can be called through

```
> data.asbestos <- data.asbestos()
```

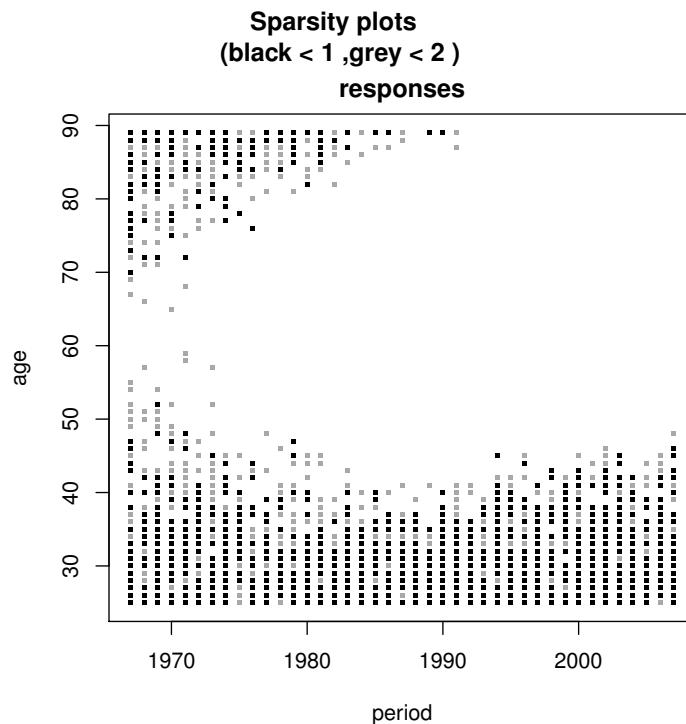


Figure 2: Sparsity plot.

Descriptive plots

The **apc** package has a variety of plots for descriptive analysis. These include plots of sums of the data by age, period, or cohort to get an idea of the aggregate development; plots of the data matrix against two of the three time indices to spot patterns in the data; and sparsity plots indicating if some entries in the data matrix are very small. For instance, there are very few mesothelioma deaths for young people. These plots can be called and manipulated individually or they can be called with a single command, for example:

```
> apc.plot.data.all(data.asbestos)
```

Figure 1 shows the plots of data sums. The responses are seen to be sparse for young people and for old and recent cohorts. The sparsity plot in Figure 2 illustrates this in more detail. It shows with black and grey entries in the data matrix with zero or one. The data are very sparse for young age groups and for old cohorts.

Deviance analysis

At this point the distribution is chosen. Currently four distributions are implemented: A Poisson response model, a Poisson dose-response model, a logistic dose-response model, and a Gaussian model giving least squares regression. The sampling theory for the two Poisson models is described in [Martínez Miranda et al. \(2015\)](#) and [Nielsen \(2014\)](#), respectively.

The age-period-cohort model has a variety of interesting sub-models. These arise by setting some of the coordinates of the canonical parameter ξ to zero. [Nielsen \(2014\)](#) gives a detailed discussion of the interpretation of the sub-models. An age-cohort model "AC" arises by setting the period double-differences to zero, so $\Delta^2\beta_j = 0$ for $j = 1, \dots, J$. The drift models of [Clayton and Schifflers \(1987a,b\)](#) arise by setting two sets of double-differences to zero. An age-drift model "Ad" arises by setting the double differences $\Delta^2\beta_j$ and $\Delta^2\gamma_k$ to zero. Thus, it is a sub-model of "AC". An age model code "A" arises by setting $\Delta^2\beta_j$, $\Delta^2\gamma_k$, and the cohort slope to zero. Thus, it is a sub-model of "Ad". A trend model "t" is the linear plane where all double differences $\Delta^2\alpha_i$, $\Delta^2\beta_j$ and $\Delta^2\gamma_k$ are set to zero. Thus, it is a sub-model of "Ad", but not nested in "A" as it has a cohort slope. An age trend model arises by setting all double differences $\Delta^2\alpha_i$, $\Delta^2\beta_j$ and $\Delta^2\gamma_k$ as well as the cohort slope to zero. Thus it is a sub-model of both "t" and "Ad". Finally, an intercept model is denoted "1". A deviance table gives an overview of the relative performance of the different models. For the mesothelioma data we get the

following output.

```
> apc.fit.table(data.asbestos, "poisson.response")
   -2logL df.residual prob(>chi_sq) LR.vs.APC df.vs.APC prob(>chi_sq)      aic
APC  2384.923     2457     0.848     NA     NA     NA 10805.81
AP    5336.034     2560     0.000 2951.111     103     0.000 13550.92
AC   2441.728     2496     0.778    56.805     39     0.033 10784.61
PC   8265.746     2520     0.000 5880.823     63     0.000 16560.63
Ad   5912.422     2599     0.000 3527.499     142     0.000 14049.31
Pd   23461.384     2623     0.000 21076.461     166     0.000 31550.27
Cd   8494.658     2559     0.000 6109.735     102     0.000 16711.54
A    21948.036     2600     0.000 19563.113     143     0.000 30082.92
P    34391.044     2624     0.000 32006.121     167     0.000 42477.93
C    28415.983     2560     0.000 26031.060     103     0.000 36630.87
t    24037.772     2662     0.000 21652.849     205     0.000 32048.66
tA   40073.386     2663     0.000 37688.463     206     0.000 48082.27
tP   34967.432     2663     0.000 32582.509     206     0.000 42976.32
tC   50558.531     2663     0.000 48173.607     206     0.000 58567.42
I    51003.046     2664     0.000 48618.123     207     0.000 59009.93
```

The first column in the table has the heading `-2logL`, noting that the deviance for Poisson and logistic models can be interpreted as minus twice the log likelihood for the model normalized to be zero in the saturated model. The deviance table indicates that the reduction worth considering is an age-cohort model, which is denoted "AC". Moreover, the likelihood value and *p*-value for the "APC" model indicate that the quality of the unrestricted model is quite good, with a deviance smaller than the degrees of freedom.

Estimation of a particular model

We can look a bit closer at a particular sub-model. For instance, in the case of the asbestos data the unrestricted age-period-cohort model is estimated as follows. The estimation in `apc` is based on the representation (11). Along with the estimates we get standard errors, which are discussed below. The canonical parameter has 208 parameters, so only the first 8 estimates are reported here.

```
> fit.apc<- apc.fit.model(data.asbestos, "poisson.response", "APC")
> fit.apc$coefficients.canonical[1:8, ]
   Estimate Std. Error      z value Pr(>|z|)
level     1.041126756     NA     NA     NA
age slope  0.379386996  0.1115535 3.400941274 0.0006715425
cohort slope 0.358297074  0.1125026 3.184789061 0.0014485956
DD_age_27  1.029446394  1.6467618 0.625133761 0.5318832712
DD_age_28  0.065309039  1.4311381 0.045634337 0.9636017004
DD_age_29  -1.097279478 1.1180554 -0.981417831 0.3263867366
DD_age_30  0.414467808  1.1902557 0.348217448 0.7276768856
DD_age_31  0.003217972  1.2247555 0.002627441 0.9979036081
```

Note that the names for the parameters utilize the information about the real time scales coded through `apc.data.list()`.

For this data set exposure or dose is not available. We therefore apply the multinomial sampling scheme used in Martinez Miranda et al. (2015). With this approach we condition on the overall level of the data. The asymptotic distribution approximations will therefore be good in a situation where the dimension of the data is fixed and the total number of responses is large. Thus, in this response model we do not get standard errors for the level.

The level and the age and period slope define the linear plane that would arise if all double differences were set to zero. The interpretation derives from the general representation (11). The level is the estimate of the predictor $\mu_{57,1967,1910}$, which is the predictor of the middle age group for the lowest period. The slopes have more interesting interpretations. The age (cohort) slope shows how much the predictor changes when increasing age (cohort) by one, while keeping cohort (age) fixed. Thus, the age and cohort slopes estimate

$$\mu_{58,1968,1910} - \mu_{57,1967,1910}, \quad \mu_{57,1968,1911} - \mu_{57,1967,1910}, \quad (6)$$

where any cell could be taken as a reference point. While the level and slopes have an explicit interpretation it is perhaps easier to interpret them in terms of a plot. Plots of the estimates are discussed below. While the package parametrises the linear plane in terms of the age and cohort slopes

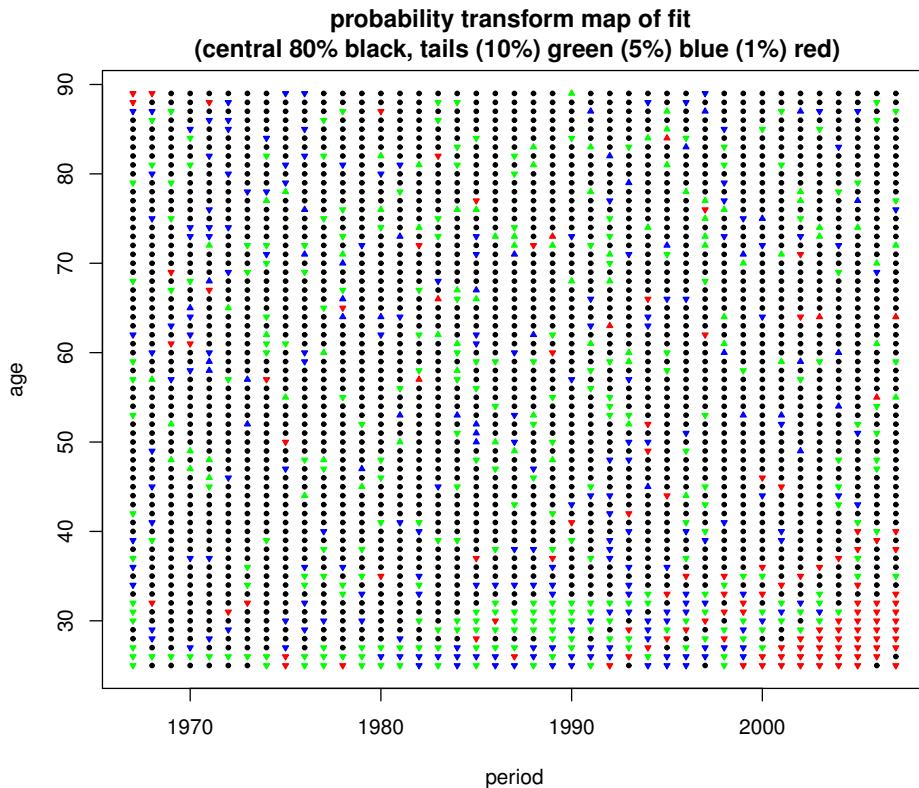


Figure 3: Probability transform plot of age-period-cohort fit to asbestos data.

other choices could be made, such as age and period slopes. The age and cohort slopes are chosen due to the age and cohort symmetry of the model.

A probability transform plot for the fit

The quality of the fit can be illustrated using a probability transform plot. Using the estimates it plots probability transforms of responses given the fitted value. In other words: are the actual observations probable given the estimated model? The plot is given in the original coordinate system. Colours and symbols are used to indicate whether responses are central to the fitted distribution or in the tails of the fitted distribution. The intention of the plot is to reveal if there are particularly many extreme observations given the fit and if they form a particular pattern.

For the asbestos data the probability transform plot is coded as:

```
> apc.plot.fit.pt(fit.apc)
```

Figure 3 shows the result. For instance, all red point triangles indicates observations in the extreme 1 % of the distribution. Those pointing down indicate the lower end of the distribution. The number of red triangles is not particularly large given the number of observations, $n = 2665$, but, they form a pattern among the most recent cohorts. Therefore, a sub-sample analysis is performed below.

Plots of the estimates

The estimates can be plotted using a single command. This command will automatically pick up information about which sub-model and adjust accordingly, based on the analysis in Nielsen (2014). There are two types of plots, which are illustrated using a sequence of three plots. Details follow.

1. Figure 4. Plot of type "sum.sum". This is illustrates the canonical parameter and the representation (11), but it is possibly the less useful choice in practical work.
2. Figure 5. Plot of type "detrend". This is illustrates the representation (15). It is the default choice.
3. Figure 6. Plot of type "detrend" for a sub-sample. The above plots appear very messy, in part because of the sparsity. This evidence leads to a sub-sample, for which the estimates look much cleaner.

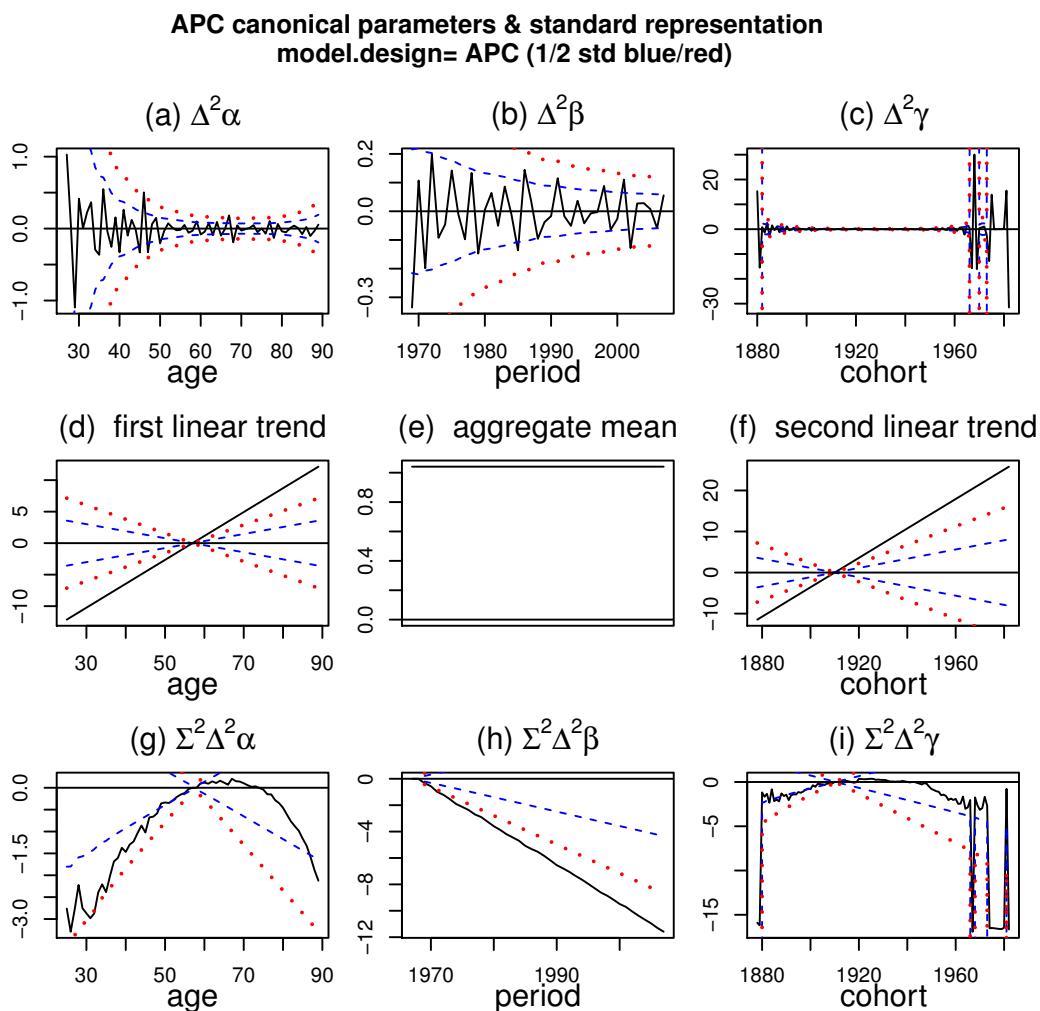


Figure 4: Plots of the fitted canonical parameters illustrating representation (11).

Plot of type "sum.sum". Figure 4 is generated by:

```
> apc.plot.fit(fit.apc, type = "sum.sum")
```

It shows the canonical parameter estimates and illustrates the representation (11).

Figure 4 (a)–(c) shows the estimated second difference parameters $\Delta^2\alpha_i$, $\Delta^2\beta_j$, $\Delta^2\gamma_k$. The estimates are plotted with pointwise confidence bands centered around zero. The age double differences are noisy for young ages while the cohort double differences are noisy for young and old cohorts. This is due to the sparsity of observations for those age and cohort groups as shown in Figure 2. This calls for a sub-sample analysis, which is described below.

The next row of panels in Figure 4 illustrates the estimated level and the slopes (6). Panel (e) shows the estimated level of 1.04. No confidence bands are shown due to the multinomial sampling scheme. Panels (d), (f) show the age and cohort slopes anchored at age 57 and cohort 1910 as discussed above.

Figure 4(g)–(i) shows double sums of double difference based on the representation (11). In each plot two values of the double sums are set to zero. In other words, the degrees of freedom, that is the number of non-zero values, in these plots are exactly the same as for the double differences. For exact values of the double sums see the last section of the paper.

The sum of the information in Figure 4(d)–(i) gives the linear predictor of the model. That is for someone born in 1920 and dying at age 70 in 1990, the predictor is the sum of the linear age trend in (d) evaluated at 70, the level in (e), the linear cohort trend in (f) evaluated at 1920, the age effect in (g) evaluated at 70, the period effect in (h) evaluated at 1990, and the cohort effect in (i) evaluated at 1920.

The plots have a messy appearance. There are several reasons. First, double sums of double differences are only identified up to arbitrary linear trends. It can be difficult to abstract from that arbitrary linear trend with these plots. In particular the period effect in (h) has a strong linear trend. It is hard to discern what the variation around that linear trend could be. We address this in the

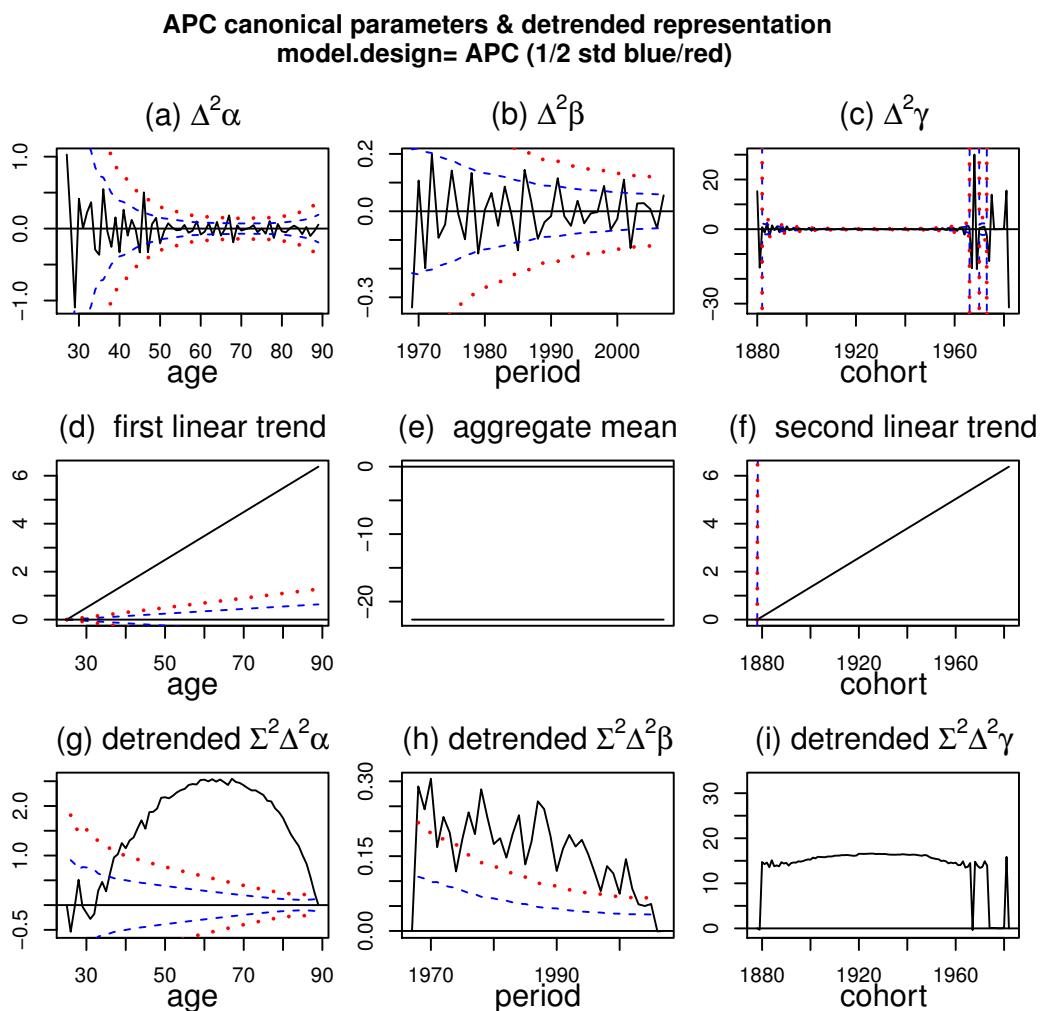


Figure 5: Plots of the fitted values using the detrended representation (15).

sub-sample analysis in connection with Figure 5. Second, the data are sparse for young age groups and for young and old cohorts. This shows up in panels (a),(c),(g),(i). We address this in the sub-sample analysis in connection with Figure 6.

Plot of type "detrend". Figure 5 is generated by:

```
> apc.plot.fit(fit.apc)
```

This plot illustrates the detrended representation (15). Figure 5(a)–(c) show exactly the same double differences as before.

The level, slopes and double sums in Figure 5(d)–(i) are now changed. The idea is to give a good visual impression of variation over and above a linear trend while preserving the degrees of freedom in panels (a)–(c). In this way Figure 5(g)–(i) show double sums of double differences detrended so as to start and end in zero. The level and slopes in Figure 5(d)–(f) are then changed according to representation (15). The interpretation is as before: The linear predictor for someone born in 1920 and dying at age 70 in 1990 is the sum of the linear age trend in (d) evaluated at 70, the level in (e), the linear cohort trend in (f) evaluated at 1920, the detrended age effect in (g) evaluated at 70, the detrended period effect in (h) evaluated at 1990, and the detrended cohort effect in (i) evaluated at 1920.

There are several noteworthy features of the detrended plots. The detrended double sums in Figure 5(g)–(i) fill the plot area better than those in Figure 4(g)–(i). Visually, it is easier to abstract from the arbitrary linear trend and focus on deviation from the linear trend. A possible drawback of the detrended plot is that age-period-cohort models can have perfect fit in some corners of the data array. For an age-period array, the very first and last cohort double differences will therefore be based on one data entry each. When looking at the detrended double sums those double differences are, however,

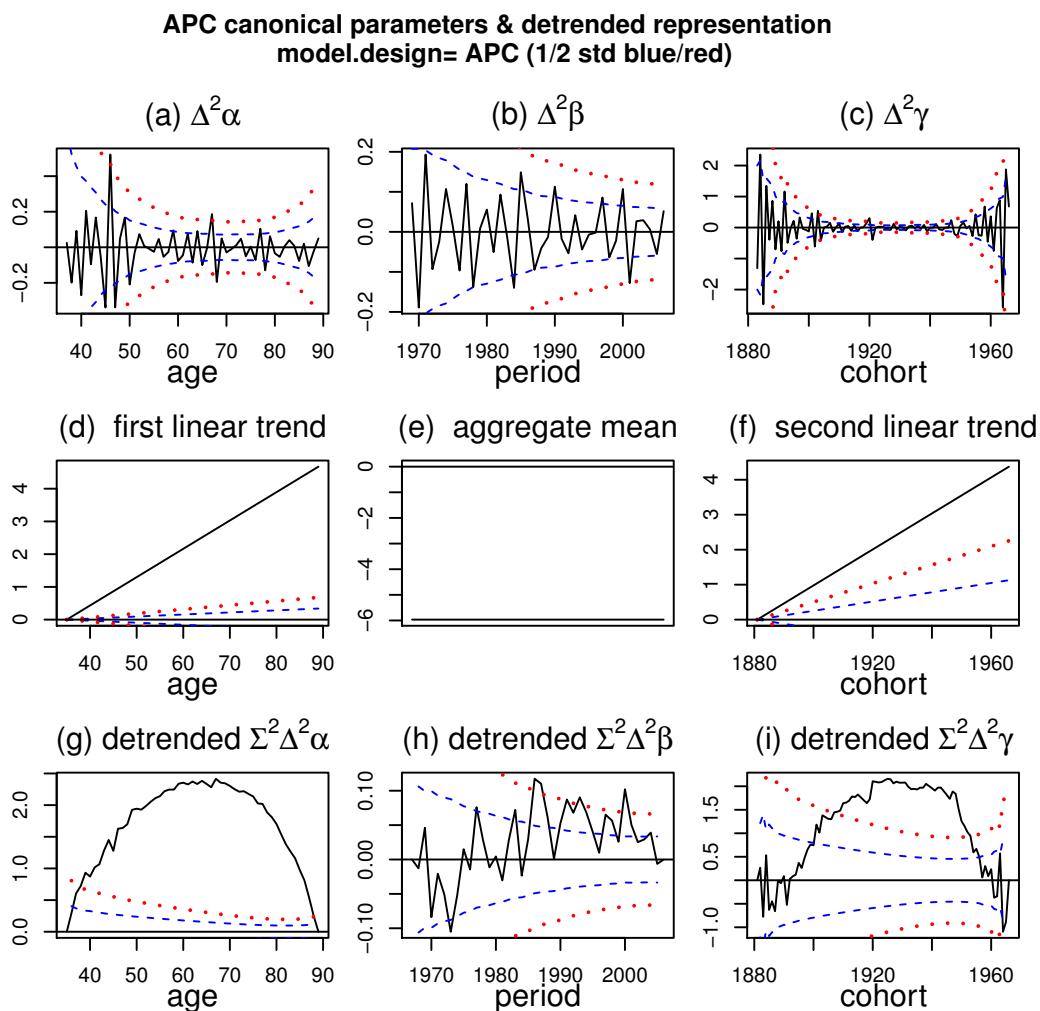


Figure 6: Plots of the fitted values for sub-sample.

combined with double sums of all the other double differences which are better determined.

The detrended age double sums in Figure 5(g) are broadly similar to those in Figure 4(g) apart from a lift in the scale and a slight change of slope. The plot indicates a near concave deviation from linearity after age 35. The development over the range 25-35 could be driven by the sparsity of the data in that region. We return to this point in the sub-sample analysis.

The detrended period double sums in Figure 5(h) have a very different appearance from those in Figure 4(h). The appearance is now seen to be a ragged concave shape. The first period stands out. Abstracting from that, the plot looks very linear. We return to this point in the sub-sample analysis.

The detrended cohort double sums in Figure 5(h) are just as messed up in appearance as those in Figure 4(h). The confidence bands have dropped off the plot and a warning is given. Again, the sub-sample analysis will address this point.

Plot of type "detrend" for a sub-sample. Figure 6 shows the result of a sub-sample analysis.

The asbestos data is sparse for low ages and for old and young cohorts. A recursive analysis can be used to check how sensitive the above analysis is in this respect. The idea is to cut parts of observations away and redo the analysis. This can be done through the command:

```
> data.asbestos.subset <- apc.data.list.subset(data.asbestos, 10, 0, 0, 0, 3, 16)
```

which cuts the lower 10 age groups, the lower 3 cohort groups and the upper 16 groups. The subset of the data is no longer a rectangle in the period-age coordinate system, but rather a rectangle with some corners cut off. This is a generalized trapezoid, see (8) for details. The above analysis can now be redone. The deviance table, which is not reported, gives approximately the same information as before, with only weak support for the "AC" sub-model ($p = 4\%$).

Prompted by the jump in the period effect in Figure 5(h) we can go one step further and drop the

data for the first period, 1967. It is possible that the data collection scheme was slightly different the first year. This is also consistent with [Tan et al. \(2010\)](#). We achieve this with only a small modification of the previous code

```
> data.asbestos.subset <- apc.data.list.subset(data.asbestos, 10, 0, 1, 0, 3, 16)
> fit.apc.subset <- apc.fit.model(data.asbestos, "poisson.response", "APC")
> apc.plot.fit(fit.apc.subset)
```

The deviance table, which is not reported, now gives stronger support for the "AC" sub-model ($p = 12\%$).

Figure 6 shows plots of the estimates. The difference relative to Figure 5 is that the noise from the youngest age groups, the first period group and the youngest and oldest cohorts group has been eliminated. Remarkably, the estimates for the remaining age, period and cohort groups are very similar. This is most clear in the plot of $\Delta^2\beta$ in panels (b) of Figure 5,6, which are nearly identical. The plots of $\Delta^2\alpha$ and $\Delta^2\gamma$ in panels (a), (c) are also very similar, although this is masked by the difference in scales. For a good empirical model the predictors for the sub-sample should be the same in the full sample and in the sub-sample. Since the double differences are identifiable from the predictors, the same should apply to them.

The double sums of double differences in panels (g)–(i) and the consequent level and linear slopes in (d)–(f) are changed. They depend on the normalisation, which depends on the choice of sample. For the sub-sample, we now see a concave shape in the double sums for age and cohort. Since the sums are pinned down to be zero at both ends this is very visible. This is quite common in cancer studies. [Nielsen \(2014\)](#) argues that this is consistent with double differences that increase from a negative value and sub log linear age effects.

It is worth noting that the sub-sample analysis is used in two ways here. First, it is used to trim off noise from sparse parts of the data set. Secondly, it is used to show that estimates do not depend very much on the choice of data array. [Martínez Miranda et al. \(2015\)](#) are concerned with forecasting future mortality and use the sub-sample analysis to show that forecasts are robust to the choice of data array. However, at present forecast methods are not implemented in **apc**. For an identification theory of forecasting see [Kuang et al. \(2008b\)](#). Recursive sub-sample graphs are very common in time series econometrics, see [Hendry and Nielsen \(2007, §13.4\)](#) and could, with advantage, be developed further.

Some details on the representation

Internally **apc** uses a representation developed in [Nielsen \(2014\)](#) that generalises the representation (5) from [Kuang et al. \(2008a\)](#). An overview of the representation is given along with some notes on the level and slope estimates in the mesothelioma example as well as on the ad hoc identification of double sums of double differences and the application to the mesothelioma data.

The representation

The package can handle data arrays that are generalized trapezoids. To illustrate this, while keeping a notation that is consistent with [Nielsen \(2014\)](#) the age-period-cohort model (1) is now written as

$$\mu_{ik} = \alpha_i + \beta_j + \gamma_k + \delta, \quad (7)$$

where i is age, j is period and k is cohort, so that $i + k = j + 1$. The generalized trapezoids are arrays of the form

$$\mathcal{I} = \{i, k : 1 \leq i \leq I, 1 \leq k \leq K, L+1 \leq j \leq L+J\}, \quad (8)$$

where I, J and K are the numbers of age, period and cohort indices, while $L+1$ is the lower period index. An age-cohort rectangular array arises when $L=0$ and $J=I+K-1$. A reserving triangle is a triangular age-cohort array where $I=J=K=L=0$. A period-age rectangular array is an age-cohort trapezoid where $L=I-1$ and $K=I+J-1$. The above sub-sample analysis is based on a rectangular age-cohort array with two corners chopped off.

It is convenient to choose a representation of the model that is symmetric in age and cohort. [Nielsen \(2014\)](#) derives such a representation. The level is anchored in the middle of the first diagonal of odd length. Thus, define $U = \text{integer}\{(L+3)/2\}$. For a period-age array where $L=I-1$ this reduces to $U = \text{integer}(I+2)/2$. If I is odd the anchoring point will be the middle age group $(I+1)/2$ for the first period. If I is even the anchoring point will be the age group $I/2+1$ for the second period. The age and cohort slopes are then defined as the one-step slopes in age and cohort directions from that point. The canonical parameter is then chosen as

$$\xi = (\nu_0, \nu_a, \nu_c, \Delta^2\alpha_3, \dots, \Delta^2\alpha_I, \Delta^2\beta_{L+2}, \dots, \Delta^2\beta_{L+J}, \Delta^2\gamma_3, \dots, \Delta^2\gamma_K)', \quad (9)$$

where

$$\nu_0 = \mu_{UU}, \quad \nu_a = (i - U)(\mu_{U+1,U} - \mu_{UU}), \quad \nu_c = (k - U)(\mu_{U,U+1} - \mu_{UU}). \quad (10)$$

It can then be shown that the predictor has the representation

$$\mu_{ik} = \nu_0 + \nu_a + \nu_c + A_i + B_j + C_k, \quad (11)$$

where

$$A_i = 1_{(i < U)} \sum_{t=i+2}^{U+1} \sum_{s=t}^{U+1} \Delta^2 \alpha_s + 1_{(i > U+1)} \sum_{t=U+2}^i \sum_{s=U+2}^t \Delta^2 \alpha_s \quad (12)$$

$$B_j = 1_{(L \text{ odd } \& j=2U-2)} \Delta^2 \beta_{2U} + 1_{(j > 2U)} \sum_{t=2U+1}^j \sum_{s=2U+1}^t \Delta^2 \beta_s \quad (13)$$

$$C_k = 1_{(k < U)} \sum_{t=k+2}^{U+1} \sum_{s=t}^{U+1} \Delta^2 \gamma_s + 1_{(k > U+1)} \sum_{t=U+2}^k \sum_{s=U+2}^t \Delta^2 \gamma_s \quad (14)$$

Estimates for the time effects A_i, B_j, C_k can be found by the code

```
> id.apc <- apc.identify(fit.apc)
> id.apc$coefficients.sssd
```

The canonical parameter (9) and the predictor (11) can be visualized through the command

```
> apc.plot.fit(fit.apc, "sum.sum")
```

The interpretation is similar to that given in the discussion of Figure 5. The package includes a vignette, [Nielsen \(2015b\)](#), showing how the parameters A_i, B_j, C_k are computed from the canonical parameter.

The representation (11) has the advantage that it is symmetric in age and cohort, reducing to that of [Kuang et al. \(2008a\)](#) for age-cohort data arrays. There is some separation between the linear plane parameters and the non-linear parameters. Indeed, the transformation from (9) to (11) does not mix the two. The choice of parametrisation is primarily for internal uses and will usually not be of importance to the user. It should be noted that any bijective transformation of ξ could be used as the parsimonious parameter. If the transformation is linear, the transformed parameter will also be the canonical parameter in an exponential family context. Some times non-linear transformations of the parameter are preferred. An example is the chain ladder model, which is an age-cohort model for an age-cohort triangle. This is often parametrised in terms of the development factors, see [Kuang et al. \(2009\)](#) for a discussion.

The detrended representation

The default plot of the parameters uses a detrended version of the parameters A_i, B_j, C_k . To be specific, it uses the representation

$$\mu_{ik} = \nu_0^{detrend} + (i - 1)\nu_a^{detrend} + (k - 1)\nu_c^{detrend} + A_i^{detrend} + B_j^{detrend} + C_k^{detrend}, \quad (15)$$

where

$$A_i^{detrend} = A_i - A_1 - \frac{i-1}{I-1}(A_I - A_1) \quad (16)$$

$$B_j^{detrend} = B_j - B_1 - \frac{j-L-1}{J-1}(B_{L+J} - B_{L+1}) \quad (17)$$

$$C_k^{detrend} = C_k - C_1 - \frac{k-1}{K-1}(C_K - C_1), \quad (18)$$

which all start and end in zero. Consequently, it must hold that

$$\nu_0^{detrend} = \nu_0 - (U-1)(\nu_a + \nu_c) - A_1 - B_{J+1} - C_1 - \frac{L}{J-1}(B_{L+J} - B_{L+1}), \quad (19)$$

$$\nu_a^{detrend} = \nu_a + \frac{1}{I-1}(A_I - A_1) + \frac{1}{J-1}(B_{J+L} - B_{J+1}), \quad (20)$$

$$\nu_c^{detrend} = \nu_c + \frac{1}{K-1}(C_K - C_1) + \frac{1}{J-1}(B_{J+L} - B_{J+1}). \quad (21)$$

The parameters $\nu_0^{detrend}, \nu_a^{detrend}, \nu_c^{detrend}, A_i^{detrend}, B_j^{detrend}, C_k^{detrend}$ can be found by the code

```
> fit.apc$coefficients.detrend
```

They are visualized in Figures 5. In particular, the plotted level and slopes are those derived in (19)–(21). The vignette [Nielsen \(2015b\)](#) shows how to check the transformation from the representation (11) to (15).

The level and slope estimates for the mesothelioma data

Recall that the canonical parameter estimates for the mesothelioma data are available through:

```
> fit.apc$coefficients.canonical[1:5, ]
      Estimate Std. Error     z value   Pr(>|z|)
level      1.041126756    NA      NA      NA
age slope   0.379386996  0.1115535  3.400941274 0.0006715425
cohort slope 0.358297074  0.1125026  3.184789061 0.0014485956
DD_age_27   1.029446394  1.6467618  0.625133761 0.5318832712
DD_age_28   0.065309039  1.4311381  0.045634337 0.9636017004
```

The level estimate for the mesothelioma arises as follows. The data is organised in an period-age array with $I = 65$ age groups and $J = 41$ cohort groups. Thus, $L = I - 1 = 64$ and $U = \text{integer}\{(L + 3)/2\} = \text{integer}(67/2) = 33$. The anchoring point for the level is therefore in the age-cohort coordinate system $\mu_{33,33}$, or, in period-age coordinates $\mu_{1,33}$. The corresponding, real time period-age coordinates are $\mu_{1967,57}$. To check this predictor estimates the level, run the following code, which organises the linear predictor for the vectorized data as a matrix in the original format.

```
> # create matrix of same dimension as response matrix
> m.linear.predictor <- data.asbestos$response
> m.linear.predictor[fit.apc$index.data] <- fit.apc$linear.predictor
> m.linear.predictor[1,33]
[1] 1.041127
```

For comparison, there are 5 observed deaths of age 57 in 1967, which is not far from $\exp(\hat{\mu}_{1967,57}) = 2.8$.

The slope estimates arise as follows. The age and cohort slopes are now, in age-cohort coordinates, $\mu_{34,33} - \mu_{33,33}$ and $\mu_{33,34} - \mu_{33,33}$, or, in period-age coordinates, $\mu_{2,34} - \mu_{1,33}$ and $\mu_{2,33} - \mu_{1,33}$. The estimates are

```
> m.linear.predictor[2,34]-m.linear.predictor[1,33]
[1] 0.379387
> m.linear.predictor[2,33]-m.linear.predictor[1,33]
[1] 0.3582971
```

The estimates of the double sums appearing in (11) can be computed by as follows.

```
> id.apc$coefficients.ssdd[c(35:38, 69:71, 141:144), ]
      Estimate Std. Error     z value   Pr(>|z|)
SS_DD_age_56   -0.02995345 0.09120714 -0.3284113 0.7426007
SS_DD_age_57    0.00000000    NA      NA      NA
SS_DD_age_58    0.00000000    NA      NA      NA
SS_DD_age_59    0.09970809  0.08965228  1.1121646 0.2660674
SS_DD_period_1967 0.00000000    NA      NA      NA
SS_DD_period_1968 0.00000000    NA      NA      NA
SS_DD_period_1969 -0.33556503  0.21566380 -1.5559636 0.1197167
SS_DD_cohort_1909 -0.15147783  0.12248429 -1.2367124 0.2161939
SS_DD_cohort_1910  0.00000000    NA      NA      NA
SS_DD_cohort_1911  0.00000000    NA      NA      NA
SS_DD_cohort_1912  0.02337873  0.12074650  0.1936183 0.8464748
```

Summary

This article describes the **apc** package for age-period-cohort modelling. It implements the canonical parametrisation of [Kuang et al. \(2008a\)](#). The package includes functions for organizing the data, a descriptive plot, a deviance table, estimation of sub-models of the age-period-cohort model, a plot for specification testing, plots of estimated parameters, and sub-sample analysis.

Acknowledgements

Comments from an anonymous referee and from B. Carstensen are gratefully acknowledged.

Bibliography

- B. Carstensen. Age-period-cohort models for the Lexis diagram. *Statistics in Medicine*, 26(15):3018–3045, 2007. [p52, 53]
- B. Carstensen, M. Plummer, E. Laara, and M. Hills. *Epi: A Package for Statistical Analysis in Epidemiology*, 2014. URL <https://CRAN.R-project.org/package=Epi>. R package version 1.1.67. [p53]
- D. Clayton and E. Schifflers. Models for temporal variation in cancer rates. I: Age-period and age-cohort models. *Statistics in Medicine*, 6(4):449–467, 1987a. [p52, 55]
- D. Clayton and E. Schifflers. Models for temporal variation in cancer rates. II: Age-period-cohort models. *Statistics in Medicine*, 6(4):469–481, 1987b. [p52, 53, 55]
- D. F. Hendry and B. Nielsen. *Econometric Modeling*. Princeton University Press, 2007. [p61]
- T. R. Holford. An alternative approach to statistical age-period-cohort analysis. *Journal of Chronic Diseases*, 38(10):831–836, 1985. [p52]
- D. Kuang, B. Nielsen, and J. P. Nielsen. Identification of the age-period-cohort model and the extended chain ladder model. *Biometrika*, 95(4):979–986, 2008a. [p52, 53, 61, 62, 63]
- D. Kuang, B. Nielsen, and J. P. Nielsen. Forecasting with the age-period-cohort model and the extended chain-ladder model. *Biometrika*, 95(4):987–991, 2008b. [p53, 61]
- D. Kuang, B. Nielsen, and J. P. Nielsen. Chain-ladder as maximum likelihood revisited. *Annals of Actuarial Science*, 4(1):105–121, 2009. [p62]
- L. Luo. Assessing validity and application scope of the intrinsic estimator approach of the age-period-cohort problem. *Demography*, 50(6):1945–1967, 2013. [p52]
- M. D. Martínez Miranda, B. Nielsen, and J. P. Nielsen. Inference and forecasting in the age-period-cohort model with unknown exposure with an application to mesothelioma mortality. *Journal of the Royal Statistical Society A*, 178(1):29–55, 2015. [p53, 54, 55, 56, 61]
- B. Nielsen. Deviance analysis of age-period-cohort models. Nuffield College Discussion Paper, 2014. [p52, 55, 57, 61]
- B. Nielsen. *apc: A Package for Age-Period-Cohort Analysis*, 2015a. URL <https://CRAN.R-project.org/package=apc>. R package version 1.1. [p52]
- B. Nielsen. *Checking the identification of the apc package*, 2015b. URL <https://CRAN.R-project.org/package=apc>. Vignette for the apc package. [p62, 63]
- B. Nielsen and J. P. Nielsen. Identification and forecasting in mortality models. *The Scientific World Journal*, pages 1–24, 2014. Article ID 347073. [p52]
- R. M. O'Brien. Constrained estimators and age-period-cohort models. *Sociological Methods & Research*, 40(3):419–452, 2011. [p52]
- E. Tan, N. Warran, A. J. Darnton, and J. T. Hodgson. Projection of mesothelioma mortality in Britain using Bayesian methods. *British Journal of Cancer*, 103(3):430–436, 2010. [p61]
- Y. Yang and K. C. Land. *Age-Period-Cohort Analysis: New Models, Methods, and Empirical Applications*. Chapman & Hall/CRC, Boca Raton, FL, 2013. [p52]

Bent Nielsen

Nuffield College, Oxford OX1 1NF, UK

and

Department of Economics, University of Oxford

and

Programme for Economic Modelling, INET, Oxford

bent.nielsen@nuffield.ox.ac.uk

QuantifQuantile: An R Package for Performing Quantile Regression Through Optimal Quantization

by Isabelle Charlier, Davy Paindaveine and Jérôme Saracco

Abstract In quantile regression, various quantiles of a response variable Y are modelled as functions of covariates (rather than its mean). An important application is the construction of reference curves/surfaces and conditional prediction intervals for Y . Recently, a nonparametric quantile regression method based on the concept of optimal quantization was proposed. This method competes very well with k -nearest neighbor, kernel, and spline methods. In this paper, we describe an R package, called **QuantifQuantile**, that allows to perform quantization-based quantile regression. We describe the various functions of the package and provide examples.

Introduction

In numerous applications, quantile regression is used to evaluate the impact of a d -dimensional covariate X on a (scalar) response variable Y . Quantile regression is an interesting alternative to standard regression whenever the conditional mean does not provide a satisfactory picture of the conditional distribution. Denoting by $F(\cdot|x)$ the conditional distribution of Y given $X = x$, the conditional quantile functions

$$x \mapsto q_\alpha(x) = \inf \{y \in \mathbb{R} : F(y|x) \geq \alpha\}, \quad \alpha \in (0,1), \quad (1)$$

indeed always yield a complete description of the conditional distribution. For our purposes, it is useful to recall that the conditional quantiles in (1) can be equivalently defined as

$$q_\alpha(x) = \arg \min_{a \in \mathbb{R}} E[\rho_\alpha(Y - a)|X = x], \quad (2)$$

where $\rho_\alpha(z) = \alpha z \mathbb{I}_{[z \geq 0]} - (1 - \alpha)z \mathbb{I}_{[z < 0]}$ is the so-called *check function*.

For fixed α , the quantile functions $x \mapsto q_\alpha(x)$ provide reference curves (when $d = 1$), one for each value of α . For fixed x , they provide conditional prediction intervals of the form $I_\alpha = [q_\alpha(x), q_{1-\alpha}(x)]$ ($\alpha < 1/2$). Such reference curves and prediction intervals are widely used, e.g. in economics, ecology, or lifetime analysis. In medicine, they are used to provide reference growth curves for children's height and weight given their age.

Many approaches have been developed to estimate conditional quantiles. After the seminal paper of Koenker and Bassett (1978) that introduced linear quantile regression, much effort has been made to consider nonparametric quantile regression. The most classical procedures in this vein are the nearest neighbor estimators (Bhattacharya and Gangopadhyay, 1990), the (kernel) local linear estimators (Yu and Jones, 1998) or the spline-based estimators (Koenker et al., 1994; Koenker and Mizera, 2004). For related work, we also refer to, e.g. Fan et al. (1994), Gannoun et al. (2002), Muggeo et al. (2013) and Yu et al. (2003). There also exists a wide variety of R functions/packages dedicated to the estimation of conditional quantiles. Among them, let us cite the functions `rqss` (only for $d \leq 2$) and `gcrq` (only for $d = 1$) from the packages `quantreg` (Koenker, 2015) and `quantregGrowth` (Muggeo, 2015), respectively.

Recently, Charlier et al. (2015a) proposed a nonparametric quantile regression method based on the concept of *optimal quantization*. Optimal quantization replaces the (typically continuous) covariate X with a discretized version \tilde{X}^N obtained by projecting X on a collection of N points (these N points, that form the *quantization grid*, are chosen to minimize the L_p -norm of $X - \tilde{X}^N$; see below). As shown in Charlier et al. (2015b), the resulting conditional quantile estimators compete very well with their classical competitors.

The goal of this paper is to describe an R package, called **QuantifQuantile** (Charlier et al., 2015c), that allows to perform quantization-based quantile regression. This includes the data-driven selection of the grid size N (that plays the role of a tuning parameter), the construction of the corresponding quantization grid, the computation of the resulting sample conditional quantiles, as well as (for $d \leq 2$) their graphical representation.

The paper is organized as follows. The first section briefly recalls the construction of quantization-based quantile regression introduced in Charlier et al. (2015a,b) and explains the various steps needed to obtain the resulting estimators. The second section lists the main functions of **QuantifQuantile** and

describes their inputs and outputs. Finally, the third section provides several examples that illustrate the use of the various functions. We conclude the paper by comparing our method with R alternatives on a real data set. An illustration of the function computing optimal quantization grids is given in the Appendix, which can be of independent interest in numerical probability, finance or numerical integration where quantization is extensively used (Pagès, 1998; Pagès et al., 2004).

Quantile regression through quantization

As mentioned above, the R package we describe in this paper implements the Charlier et al. (2015a,b) quantization-based methodology to perform nonparametric quantile regression. This section describes this methodology.

Approximating population conditional quantiles through quantization

Let $\gamma^N \in (\mathbb{R}^d)^N$ be a grid of size N , that is, a collection of N points in \mathbb{R}^d . For any $x \in \mathbb{R}^d$, we will denote by $\tilde{x}^{\gamma^N} = \text{Proj}_{\gamma^N}(x)$ the projection of x onto this grid, that is, the point of γ^N that is closest to x (absolute continuity assumption makes ties unimportant in the sequel). This allows to approximate the d -dimensional covariate X by its quantized version \tilde{X}^{γ^N} .

Obviously, the choice of the grid has a significant impact on the quality of this approximation. Under the assumption that $\|X\|_p := E[|X|^p]^{1/p} < \infty$ (throughout, $|\cdot|$ denotes the Euclidean norm), optimal quantization selects the grid γ^N that minimizes the L_p -quantization error $\|X - \tilde{X}^{\gamma^N}\|_p$. Such an optimal grid exists under the assumption that the distribution P_X of X does not charge any hyperplane, i.e. under the assumption that $P_X[H] = 0$ for any hyperplane H (Pagès, 1998). In practice, an optimal grid is constructed using a *stochastic gradient algorithm* (see the following section). For more details on quantization, the reader may refer to Pagès (1998) and Graf and Luschgy (2000).

Based on optimal quantization of X , we can approximate the conditional quantile $q_\alpha(x)$ in (2) by

$$\tilde{q}_\alpha^N(x) := \arg \min_{a \in \mathbb{R}} E[\rho_\alpha(Y - a) | \tilde{X}^N = \tilde{x}^N], \quad (3)$$

where \tilde{X}^N (resp., \tilde{x}^N) denotes the projection of X (resp., x) onto an optimal grid. It is shown in Charlier et al. (2015a) that under mild assumptions,¹ $\tilde{q}_\alpha^N(x)$ converges to $q_\alpha(x)$ as $N \rightarrow \infty$, uniformly in x .

Obtaining an optimal N -grid

As we will see below, whenever independent copies $(X'_1, Y_1)', \dots, (X'_n, Y_n)'$ of $(X', Y)'$ are available, the first step to obtain a sample version of (3) is to compute an optimal N -grid (we assume here that N is fixed). As already mentioned, this can be done through a stochastic gradient algorithm. This algorithm, called *Competitive Learning Vector Quantization (CLVQ)* when $p = 2$, is an iterative procedure that operates as follows :

- First, an initial grid – $\hat{\gamma}^{N,0}$, say – is chosen by sampling randomly without replacement among the X_i 's.
- Second, n iterations are performed (one for each observation available). The grid $\hat{\gamma}^{N,t} = (\hat{\gamma}_1^{N,t}, \dots, \hat{\gamma}_N^{N,t})$ at step t is obtained through

$$\hat{\gamma}_i^{N,t} = \begin{cases} \hat{\gamma}_i^{N,t-1} - \delta_t |\hat{\gamma}_i^{N,t-1} - X_t|^{p-1} \frac{\hat{\gamma}_i^{N,t-1} - X_t}{|\hat{\gamma}_i^{N,t-1} - X_t|} & \text{if } \text{Proj}_{\hat{\gamma}^{N,t-1}}(X_t) = \hat{\gamma}_i^{N,t-1}, \\ \hat{\gamma}_i^{N,t-1} & \text{otherwise,} \end{cases}$$

where $(\delta_t), t \in \mathbb{N}_0$, is a deterministic sequence in $(0, 1)$ such that $\sum_t \delta_t = \infty$ and $\sum_t \delta_t^2 < \infty$. At the t^{th} iteration, only one point of the grid moves, namely the one that is closest to X_t .

The optimal grid provided by this algorithm is then $\hat{\gamma}^{N,n}$.

Estimating conditional quantiles

Assume now that a sample $(X'_1, Y_1)', \dots, (X'_n, Y_n)'$ as above is indeed available. The sample analog of (3) is then defined as follows :

¹Our method actually requires assumptions on the link function between Y and X and on the error term of the model. These assumptions in particular guarantee that, for any x , the conditional distribution of Y given $X = x$ is absolutely continuous; we refer to Charlier et al. (2015a) for details.

- (S1) First, we compute the optimal grid $\hat{\gamma}^{N,n}$ through the stochastic gradient algorithm just described, and we write $\hat{X}_i^N = \text{Proj}_{\hat{\gamma}^{N,n}}(X_i)$, $i = 1, \dots, n$.
 (S2) Then, the conditional quantiles are estimated by

$$\hat{q}_\alpha^{N,n}(x) = \arg \min_{a \in \mathbb{R}} \sum_{i=1}^n \rho_\alpha(Y_i - a) \mathbb{I}_{[\hat{X}_i^N = \hat{x}^N]}, \quad (4)$$

where $\hat{x}^N = \text{Proj}_{\hat{\gamma}^{N,n}}(x)$. In practice, $\hat{q}_\alpha^{N,n}(x)$ is simply evaluated as the sample α -quantile of the Y_i 's for which $\hat{X}_i^N = \hat{x}^N$.

It is shown in [Charlier et al. \(2015a\)](#) that under mild assumptions, $\hat{q}_\alpha^{N,n}(x)$, for any fixed N and x , converges in probability to $\tilde{q}_\alpha^n(x)$ as $n \rightarrow \infty$, provided that quantization is based on $p = 2$.

When the sample size n is small to moderate ($n \leq 300$, say), the estimated reference curves $x \mapsto \hat{q}_\alpha^{N,n}(x)$ typically are not smooth. To improve on this, [Charlier et al. \(2015a,b\)](#) introduced the following bootstrapped version of the estimator in (4). For some positive integer B , generate B samples of size n from the original sample $\{(X'_i, Y_i)'\}_{i=1,\dots,n}$ with replacement. From each of these bootstrap samples, the stochastic gradient algorithm provides an “optimal” grid, using these bootstrapped samples to perform the iterations. The bootstrapped estimator of conditional quantile is then

$$\bar{q}_{\alpha,B}^{N,n}(x) = \frac{1}{B} \sum_{b=1}^B \hat{q}_\alpha^{(b)}(x), \quad (5)$$

where $\hat{q}_\alpha^{(b)}(x) = \hat{q}_\alpha^{(b),N,n}(x)$ denotes the estimator in (4) computed on the basis of the b^{th} optimal grid. We stress that, when computing $\hat{q}_\alpha^{(b)}(x)$, the original sample is used in (S2); the bootstrapped samples are only used to provide the B different grids. As shown in [Charlier et al. \(2015a,b\)](#), the bootstrapped reference curves are much smoother than the original ones. Of course, B should be chosen large enough to make the bootstrap useful, but also small enough to keep the computational burden under control. For $d = 1$, we usually choose $B = 50$.

Selecting the grid size N

Both for the original estimators $\hat{q}_\alpha^{N,n}(x)$ and for their bootstrapped versions $\bar{q}_{\alpha,B}^{N,n}(x)$, an appropriate value of N should be identified. If N is too small, then reference curves will have a large bias, while if N is too large, then they will show much variability. This leads to the usual bias/variance trade-off that is to be achieved when selecting the value of a smoothing parameter in nonparametric statistics.

[Charlier et al. \(2015b\)](#) proposed the following data-driven method to choose N . Let $\{x_1, \dots, x_J\}$ be a set of x -values at which we want to estimate $q_\alpha(x)$ (the x_j 's are for instance chosen equispaced on the support of X) and let \mathcal{N} be a finite collection of N -values (this represents the values of N one allows for and should typically be chosen according to the sample size n). Ideally, we would like to select the optimal value of N as

$$\bar{N}_{\alpha;\text{opt}} = \arg \min_{N \in \mathcal{N}} \text{ISE}_\alpha(N), \quad \text{with } \text{ISE}_\alpha(N) = \frac{1}{J} \sum_{j=1}^J (\bar{q}_{\alpha,B}^{N,n}(x_j) - q_\alpha(x_j))^2. \quad (6)$$

This, however, is infeasible, since the population quantiles $q_\alpha(x_j)$ are unknown. This is why we draw \tilde{B} extra bootstrap samples (still of size n) from the original sample and consider

$$\hat{N}_{\alpha;\text{opt}} = \arg \min_{N \in \mathcal{N}} \widehat{\text{ISE}}_\alpha(N), \quad \text{with } \widehat{\text{ISE}}_\alpha(N) = \frac{1}{J} \sum_{j=1}^J \left(\frac{1}{\tilde{B}} \sum_{\tilde{b}=1}^{\tilde{B}} (\hat{q}_{\alpha,B}^{N,n}(x_j) - \hat{q}_\alpha^{(\tilde{b})}(x_j))^2 \right), \quad (7)$$

where $\hat{q}_\alpha^{(\tilde{b})}(x_j)$, for $\tilde{b} = 1, \dots, \tilde{B}$, makes use of this \tilde{b}^{th} new bootstrap sample; more precisely, the bootstrap sample is still only used to perform the iterations of the algorithm, whereas the original sample is used in both the initial grid and in (S2).

As shown in [Charlier et al. \(2015b\)](#) through simulations, both $N \mapsto \text{ISE}_\alpha(N)$ and $N \mapsto \widehat{\text{ISE}}_\alpha(N)$ are essentially convex in N and lead to roughly the same minimizers. This therefore provides a feasible way to select a reasonable value of N for the estimator $\bar{q}_{\alpha,B}^{N,n}(x)$ in (5). Note that this also applies to $\hat{q}_\alpha^{N,n}(x)$ by simply taking $B = 1$ in the procedure above.

If quantiles are to be estimated at various orders α , (7) will provide an optimal N -value for each α . It may then happen, in principle, that the resulting reference curves cross, which is of course undesirable.

One way to guarantee that no such crossings occur is to identify a common N -value for the various α 's. In such a case, N will be chosen as

$$\hat{N}_{\text{opt}} = \arg \min_{N \in \mathcal{N}} \widehat{\text{ISE}}(N), \quad \text{with } \widehat{\text{ISE}}(N) = \sum_{\alpha} \widehat{\text{ISE}}_{\alpha}(N), \quad (8)$$

where the sum is computed over the various α -values considered.

[Charlier et al. \(2015b\)](#) performed extensive comparisons between the quantization-based estimators in (5) – based on the efficient data-driven selection method for N just described – and some of their main competitors, namely estimators obtained from spline, k -nearest neighbor, and kernel methods. This revealed that the quantization-based estimators compete well in all cases, and often dominates their competitors (in terms of integrated square errors), particularly so for complex link functions; see [Charlier et al. \(2015b\)](#) for details.

Unlike the local linear and local constant estimators from [Yu and Jones \(1998\)](#), that are usually based on a global-in- x bandwidth, our quantization-based estimators are locally adaptive in the sense that, when estimating $q_{\alpha}(x)$, the “working bandwidth” – that is, the size of the quantization cell containing x – depends on x . The k -nearest neighbor (k NN) estimator is closer in spirit to quantization-based estimators but always selects k neighbors, irrespective of the x -value considered, whereas the number of X_i 's in the quantization cell of x may depend on x . This subtle local-in- x behavior may explain the good empirical performances of quantization-based estimators over kernel and nearest-neighbor competitors. Finally, spline methods (implemented in R with the rqss and qss functions) tend to perform poorly for complex link functions, since they always provide piecewise linear reference curves ([Koenker et al., 1994](#)). Moreover, the current implementation of the rqss function only supports dimensions 1 and 2, whereas our package allows to compute quantization-based estimators in any dimension d .

The QuantifQuantile package

This section provides a description of the various functions offered in the R package **QuantifQuantile**. We first detail the three functions that allow to estimate conditional quantiles through quantization. Then we describe a function computing optimal quantization grids.

Conditional quantile estimation

QuantifQuantile is composed of three main functions that each provide estimated conditional quantiles in (4)-(5). These functions work in a similar way but address different values of d (recall that d is the dimension of the covariate vector X) :

- The function `QuantifQuantile` is suitable for $d = 1$.
- The function `QuantifQuantile.d2` addresses the case $d = 2$.
- Finally, `QuantifQuantile.d` can deal with an arbitrary value of d .

Combined with the `plot` function, the first two functions provide reference curves and reference surfaces, respectively. No graphical outputs can be obtained from the third function if $d > 2$.

The three functions share the same arguments, but not necessarily the same default values. For each function, using `args()` displays the various arguments and corresponding default values :

```
QuantifQuantile(X, Y, alpha = c(0.05, 0.25, 0.5, 0.75, 0.95), x = seq(min(X), max(X),
length = 100), testN = c(35, 40, 45, 50, 55), p = 2, B = 50, tildeB = 20,
same_N = TRUE, ncores = 1)
```

```
QuantifQuantile.d2(X, Y, alpha = c(0.05, 0.25, 0.5, 0.75, 0.95),
x = matrix(c(rep(seq(min(X[1, ]), max(X[1, ]), length = 20), 20),
sort(rep(seq(min(X[2, ]), max(X[2, ]), length = 20), 20))), nrow = 2, byrow = TRUE),
testN = c(110, 120, 130, 140, 150), p = 2, B = 50, tildeB = 20, same_N = TRUE,
ncores = 1)
```

```
QuantifQuantile.d(X, Y, x, alpha = c(0.05, 0.25, 0.5, 0.75, 0.95),
testN = c(35, 40, 45, 50, 55), p = 2, B = 50, tildeB = 20, same_N = TRUE, ncores = 1)
```

We now give more details on these arguments.

- X : a $d \times n$ real array (required by all three functions, a vector of length n for `QuantifQuantile`). The columns of this matrix are the X_i 's, $i = 1, \dots, n$.

- Y : an $n \times 1$ real array (required by all three functions). This vector collects the Y_i 's, $i = 1, \dots, n$.
- alpha : an $r \times 1$ array with components in $(0, 1)$ (optional for all three functions). This vector contains the orders for which $q_\alpha(x)$ should be estimated.
- x : a $d \times J$ real array (optional for QuantifQuantile and QuantifQuantile.d2, required by QuantifQuantile.d). The columns of this matrix are the x_j 's at which the quantiles $q_\alpha(x_j)$ are to be estimated. If x is not provided when calling QuantifQuantile, then it is set to a vector of $J = 100$ equispaced values between the minimum and the maximum of the X_i 's. If this argument is not provided when calling QuantifQuantile.d2, then the default for x is a matrix whose $J = 20^2 = 400$ column vectors are obtained as follows: 20 equispaced values are considered between the minimum and maximum values of the $(X_i)_1$'s and similarly for the second component. The 400 column vectors of the default x are obtained by considering all combinations of those 20 values for the first component with the 20 values for the second one².
- testN : an $m \times 1$ vector of pairwise distinct positive integers (optional for all three functions). The entries of this vector are the elements of the set \mathcal{N} in (7)-(8), hence are the N -values for which the $\widehat{\text{ISE}}_\alpha$ quantity considered will be evaluated. The default is $(35, 40, \dots, 55)$ but it is strongly recommended to adapt it according to the sample size n at hand.
- p : a real number larger than or equal to one (optional for all three functions). This is the parameter p to be used when performing optimal quantization in L_p -norm.
- B : a positive integer (optional for all three functions). This is the number of bootstrap replications B to be used in (5).
- \tilde{B} : a positive integer (optional for all three functions). This is the number of bootstrap replications \tilde{B} to be used when determining $\hat{N}_{\alpha,\text{opt}}$ or \hat{N}_{opt} .
- same_N : a boolean variable (optional for all three functions). If $\text{same_N}=\text{TRUE}$, then a common value of N (that is, \hat{N}_{opt} in (8)) will be selected for all α 's. If $\text{same_N}=\text{FALSE}$, then optimal values of N will be chosen independently for the various of α (which will provide several $\hat{N}_{\alpha,\text{opt}}$, as in (7)).
- ncores : number of cores to use. These functions can use parallel computation to save time by increasing this parameter. Parallel computation relies on `mclapply` from **parallel** package, hence is not available on Windows.

All three functions return the following list of objects, which is of class 'QuantifQuantile':

- hatq_opt : an $r \times J$ real array (where r is the number of α -values considered). If $\text{same_N}=\text{TRUE}$, then the entry (i, j) of this matrix is $\bar{q}_{\alpha_i, B}^{N_{\text{opt}}, n}(x_j)$. If $\text{same_N}=\text{FALSE}$, then it is rather $\bar{q}_{\alpha_i, B}^{N_{\alpha_i, \text{opt}}, n}(x_j)$. This object can also be returned using the usual `fitted.values` function.
- N_{opt} : a positive integer (if $\text{same_N}=\text{TRUE}$) or an $r \times 1$ array of positive integers (if $\text{same_N}=\text{FALSE}$). Depending on same_N , this provides the value of \hat{N}_{opt} or the vector $(\hat{N}_{\alpha_1, \text{opt}}, \dots, \hat{N}_{\alpha_r, \text{opt}})$.
- hatISE_N : an $r \times m$ real array. The entry (i, j) of this matrix is $\widehat{\text{ISE}}_{\alpha_i}(N_j)$. Plotting this for fixed α or plotting its average over the various α , in both cases over testN , allows to assess the global convexity of these ISEs. Hence, it can be used to indicate whether or not the choice of testN was adequate. This will be illustrated in the examples below.
- hatq_N : an $r \times J \times m$ real array. The entry (i, j, ℓ) of this matrix is $\bar{q}_{\alpha_i, B}^{N_\ell, n}(x_j)$, where N_ℓ is the ℓ^{th} entry of the argument testN . From this output, it is easy by fixing the third entry to get the matrix of the $\bar{q}_{\alpha_i, B}^{N, n}(x_j)$ values for any N in testN .
- The arguments X , Y , x , alpha , and testN are also reported in this response list.

Moreover, when the optimal value N_{opt} selected is on the boundary of testN , which means that testN most likely was not well chosen, a warning message is printed.

The 'QuantifQuantile' class response can be used as argument of the functions `plot` (only for $d \leq 2$), `summary` and `print`. The `plot` function draws the observations and plots the estimated conditional quantile curves ($d = 1$) or surfaces ($d = 2$) – for $d = 2$, the `rgl` package is used (Adler et al., 2015), which allows to change the perspective in a dynamic way. In order to illustrate the selection of N , the function `plot` also has an optional argument `ise`. Setting this argument to `TRUE` (the default is `FALSE`), this function, that can be used for any dimension d , provides the plot (against N) of the $\widehat{\text{ISE}}_\alpha$ and $\widehat{\text{ISE}}$ quantities in (7) or in (8), depending on the choice $\text{same_N}=\text{FALSE}$ or $\text{same_N}=\text{TRUE}$, respectively; see the examples below for details. If $d \leq 2$, it also returns the fitted quantile curves or surfaces.

²Since the number J of points in a default value of x obtained in this fashion would increase exponentially with the dimension d , we did not adopt the same approach for $d \geq 3$.

Computing optimal grids

Besides the functions that allow to estimate conditional quantiles and to plot the corresponding reference curves/surfaces, **QuantifQuantile** provides a function that computes optimal quantization grids. This function, called `choice.grid`, admits the following arguments :

- `X`: a $d \times n$ real array (required). The columns of this matrix are the X_i 's, $i = 1, \dots, n$, for which the optimal quantization grid should be determined. Each point of `X` is used as a stimulus in the stochastic gradient algorithm to get an optimal grid.
- `N`: a positive integer (required). The size of the desired quantization grid.
- `ng`: a positive integer (optional). The number of desired quantization grids. The default is 1.
- `p`: a real number larger than or equal to one (optional). This is the parameter p used in the quantization error. The default is 2.

In some cases, it may be necessary to have several quantization grids, such as in (5), where `B + tildeB` grids are needed. The three functions computing quantization-based conditional quantiles then call the function `choice.grid` with `ng > 1`. In such case, the various grids are obtained using as stimuli a resampling version of `X` (the X_t 's in the previous section).

The output is a list containing the following elements :

- `init_grid`: a $d \times N \times ng$ real array. The entry (i, j, ℓ) of this matrix is the i^{th} component of the j^{th} point of the ℓ^{th} initial grid.
- `opti_grid`: a $d \times N \times ng$ real array. The entry (i, j, ℓ) of this matrix is the i^{th} component of the j^{th} point of the ℓ^{th} optimal grid provided by the algorithm.

Illustrations

In this section, we illustrate on several examples the use of the functions described above. Examples 1–3 restrict to `QuantifQuantile`/`QuantifQuantile.d2` and provide graphical representations in each case. Example 4 deals with a three-dimensional covariate, without graphical representation. An illustration of the function `choice.grid` is given in the Appendix.

Example 1: Simulated data with one-dimensional covariate

We generate a random sample $(X_i, Y_i)', i = 1, \dots, n = 300$, where the X_i 's are uniformly distributed over the interval $(-2, 2)$ and where the Y_i 's are obtained by adding to X_i^2 a standard normal error term that is independent of X_i :

```
set.seed(258164)
n <- 300
X <- runif(n, -2, 2)
Y <- X^2 + rnorm(n)
```

We test the number N of quantizers between 10 and 30 by steps of 5 and we do not change the default values of the other arguments. We then run the function `QuantifQuantile` with these arguments and stock the response in `res`.

```
testN <- seq(10, 30, by = 5)
res <- QuantifQuantile(X, Y, testN = testN)
```

No warning message is printed, which means that this choice of `testN` was adequate. To assess this in a graphical way, we use the function `plot` with `ise` argument set to TRUE that plots `hatISEmean_N` (obtained by taking the mean of `hatISE_N` over `alpha`) against the various N -values in `testN`.

```
plot(res, ise = TRUE)
```

Figure 1a provides the resulting graph, which confirms that `testN` was well chosen since `hatISEmean_N` is larger for smaller and larger values of N than `N_opt`. We then plot the corresponding estimated conditional quantiles curves in Figure 1b. The default colors of the points and of the curves are changed by using the `col.plot` argument. This argument is a vector of size `1+length(alpha)`, whose first component fixes the color of the data points and whose remaining components determine the colors of the various reference curves.

```
col.plot <- c("grey", "red", "orange", "green", "orange", "red")
plot(res, col.plot = col.plot, xlab = "X", ylab = "Y")
```

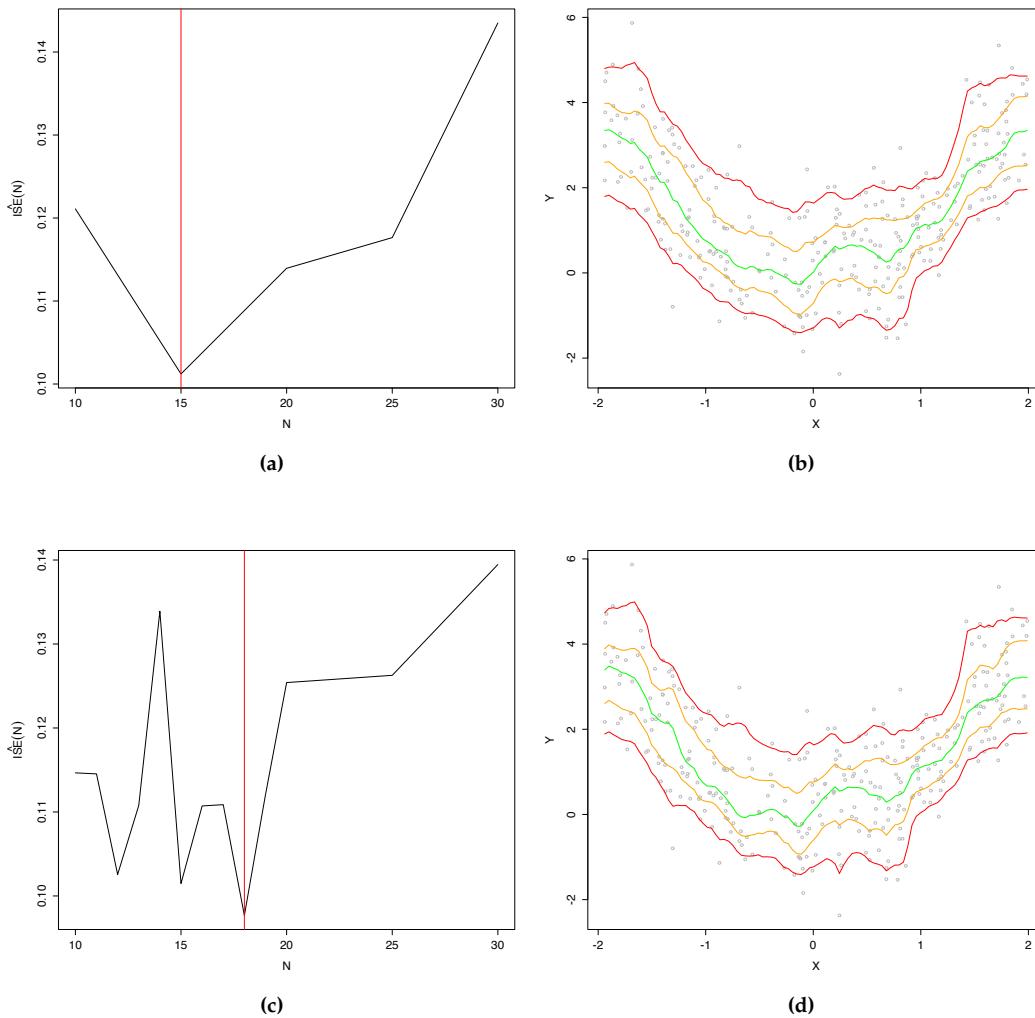


Figure 1: For the sample considered in Example 1, this figure provides (a) the plot of $N \mapsto \widehat{ISE}(N)$ with $N \in \{10, 15, 20, 25, 30\}$, and (b) the resulting reference curves. The panels (c)–(d) provide the corresponding plots when taking $N \in \{10, 11, 12, \dots, 19, 20, 25, 30\}$.

It is natural to make the grid `testN` finer. Of course, the more N -values we test, the longer it takes. This is why we adopted this two-stage approach, where the goal of the first stage was to get a rough approximation of the optimal N -value. In the second stage, we can then refine the grid only in the vicinity of the value N_{opt} obtained in the first stage.

```
testN <- c(seq(10, 20, by = 1), seq(25, 30, by = 5))
res_step1 <- QuantifQuantile(X, Y, testN = testN)
plot(res_step1, ise = TRUE, col.plot = col.plot, xlab = "X", ylab = "Y")
```

The resulting graphs are provided in Figures 1c–1d, respectively. We observe that the value of N_{opt} is made more precise, since we now get $N_{opt}=18$ instead of 15. The resulting estimated conditional quantiles curves in Figure 1b are very similar to the ones in Figure 1d.

So far, we used the default value `same_N=TRUE`, which leads to selecting an N -value that is common to all α 's. For the sake of comparison, we now explore the results for `same_N=FALSE`.

```
testN <- c(seq(10, 30, by = 5))
res2 <- QuantifQuantile(X, Y, testN = testN, same_N = FALSE)
plot(res2, ise = TRUE, col.plot = col.plot, xlab = "X", ylab = "Y")
testN <- c(seq(10, 20, by = 1), seq(25, 30, by = 5))
res2_step1 <- QuantifQuantile(X, Y, testN = testN, same_N = FALSE)
plot(res2_step1, ise = TRUE, col.plot = col.plot, xlab = "X", ylab = "Y")
```

The results are provided in Figure 2. Comparing the left panels of Figures 1 and 2, we see that

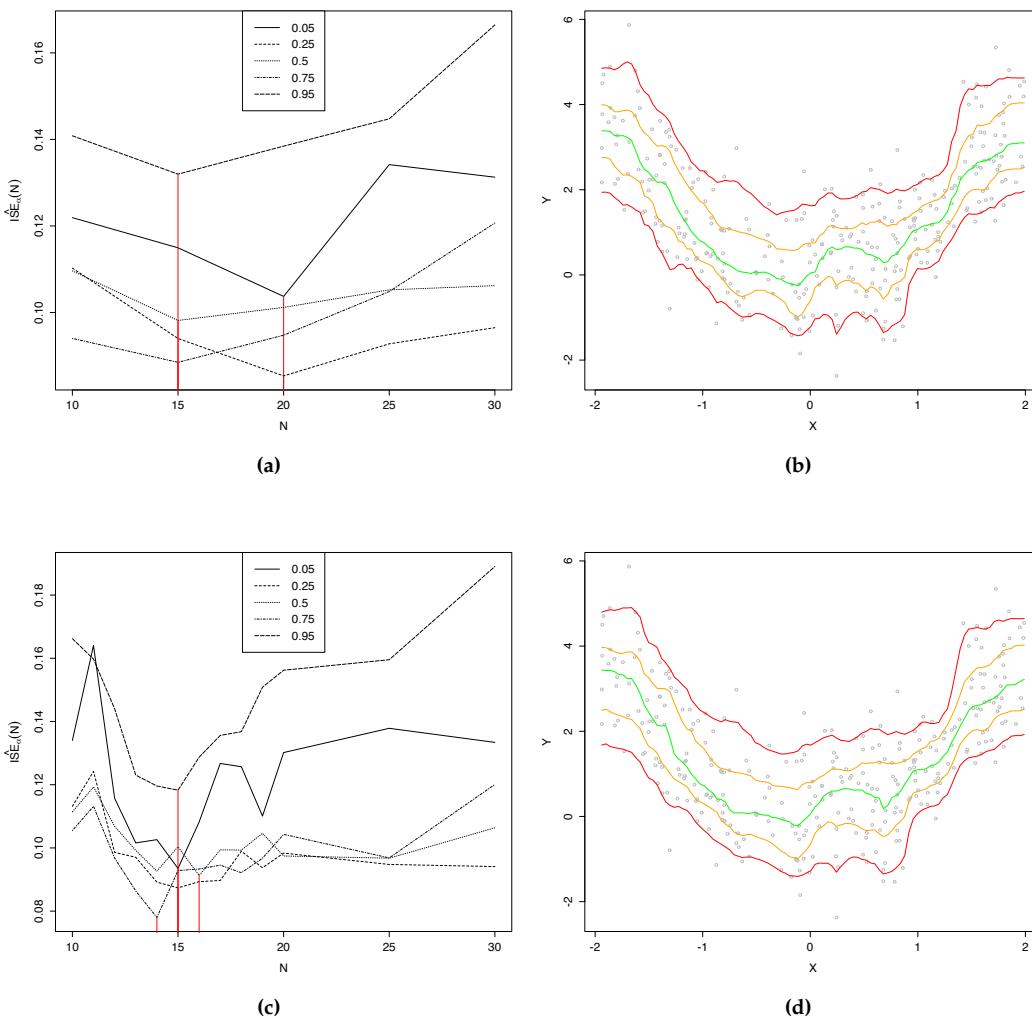


Figure 2: The same results as in Figure 1, but when selecting optimal values of N separately for each α .

when choosing N by steps of five, we find $N_{\text{opt}} = 15$ with $\text{same_N} = \text{TRUE}$ and $N_{\text{opt}} = 15$ or 20 (depending on α) for $\text{same_N} = \text{FALSE}$. When we refine the grid testN , we find analogously $N_{\text{opt}} = 18$ for $\text{same_N} = \text{TRUE}$ and $N_{\text{opt}} = 14, 15$, or 16 for $\text{same_N} = \text{FALSE}$. In the present setup, thus, both methods provide relatively close optimal N -values, which explains why the corresponding estimated reference curves are so similar (see the right panels of Figures 1 and 2). Therefore, the grid of N -values tested in Figure 1, that may seem too coarse at first sight, actually provides fitted curves that are as satisfactory as those associated with the finer grid in Figure 2.

Example 2: Simulated data with two-dimensional covariate

The sample considered here is made of $n = 1,000$ independent realizations of a random vector $(X', Y)'$, where $X = (X_1, X_2)'$ is uniformly distributed on the square $(-2, 2)^2$ and where Y is obtained by adding to $X_1^2 + X_2^2$ an independent standard normal error term.

```
set.seed(642516)
n <- 1000
X <- matrix(runif(n*2, -2, 2), ncol = n)
Y <- apply(X^2, 2, sum) + rnorm(n)
```

We test N between 40 and 90 by steps of 10. We change the values of B and \tilde{B} to reduce the computational burden, which is heavier for $d = 2$ than for $d = 1$. We keep the default values of all other arguments when running the function `QuantifQuantile.d2`. Here, a warning message is printed informing us that `testN` was not well-chosen. We confirm it with the function `plot` with `ise` argument set to `TRUE`.

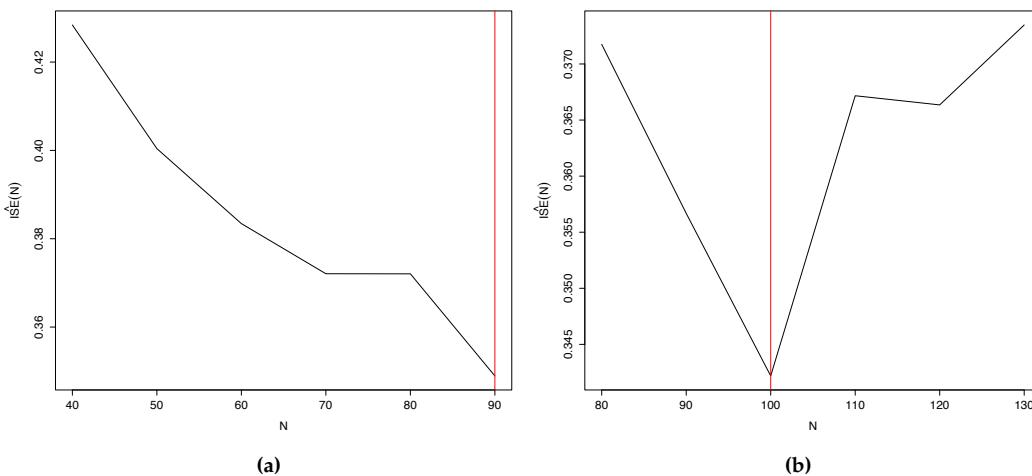


Figure 3: For the sample considered in Example 2, this figure plots $N \mapsto \widehat{\text{ISE}}(N)$ (a) for $N \in \{40, 50, 60, 70, 80\}$ and (b) for $N \in \{80, 90, \dots, 120, 130\}$.

```
testN <- seq(40, 90, by = 10)
B <- 20
tildeB <- 15
res <- QuantifQuantile.d2(X, Y, testN = testN, B = B, tildeB = tildeB)
plot(res, ise = TRUE)
```

Figure 3a provides the resulting graph. The parameter `testN` was not well chosen since `hatISEmean_N` becomes smaller and smaller as `N_opt` increases. We then adapt the choice of `testN` accordingly and rerun the procedure, which identifies an optimal N -value equal to 100; see Figure 3b.

```
testN <- seq(80, 130, by = 10)
res <- QuantifQuantile.d2(X, Y, testN = testN, B = B, tildeB = tildeB)
plot(res, ise = TRUE)
```

We then plot the corresponding estimated conditional quantile surfaces in Figure 4.

```
col.plot <- c("black", "red", "orange", "green", "orange", "red")
plot(res, col.plot = col.plot, xlab = "X_1", ylab = "X_2", zlab = "Y")
```

Example 3: Real data study and comparison with some competitors

This example aims at illustrating the proposed estimated reference curves on a real data set and at comparing them with some competitors. In this example, the `ncores` parameter of `QuantifQuantile` function was set to the number of cores detected by R minus 1. The data used here, that are included in the `QuantifQuantile` package, involves several variables related to employment, housing and environment associated with $n = 542$ towns/villages in Gironde, France. For the present illustration, we restrict to the regressions R_1 and R_2 involving $(X, Y) = (\text{percentage of owners living in their primary residence, percentage of buildings area})$ and $(X, Y) = (\text{percentage of middle-range employees, population density})$, respectively. In both cases, $n = 542$ observations are available and we are interested in the estimation of reference curves for $\alpha = 0.05, 0.25, 0.50, 0.75$ and 0.95 . For both R_1 and R_2 , we tested the number N of quantizers to be used between 5 and 15 by step of 1, using the methodology described in Example 1.

```
set.seed(644925)
data(gironde)
X <- gironde[[2]]$owners
Y <- gironde[[4]]$building
testN <- seq(5, 15, by = 1)
res <- QuantifQuantile(X, Y, testN = testN, same_N = F, ncores = detectCores() - 1)
col.plot <- c("grey", "red", "orange", "green", "orange", "red")
plot(res, col.plot = col.plot, xlab = "X", ylab = "Y")
```

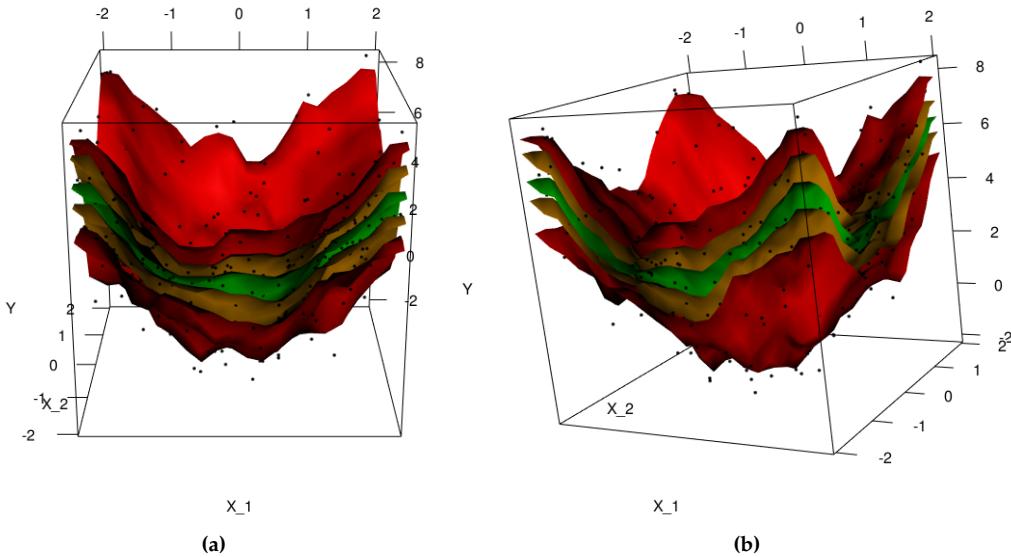


Figure 4: For the sample considered in Example 2, this figure plots (with two different views) the estimated conditional quantile surfaces obtained with the `plot` function for $\alpha = 0.05, 0.25, 0.50, 0.75$ and 0.95 .

The same exercise is repeated with $(X, Y) = (\text{percentage of middle-range employees}, \text{population density})$. For each α -value considered, we obtained $\hat{N}_{\alpha;\text{opt}} = 13$ for R_1 and $\hat{N}_{\alpha;\text{opt}} = 7$ for R_2 . The corresponding quantization-based reference curves are plotted in Figures 5a and 5c, respectively. For the sake of comparison, spline-based curves are provided in Figures 5b and 5d. These were obtained from the function `rqss` in the package `quantreg`. Since the parameter λ involved, that governs the trade-off between fidelity and smoothness, is not automatically selected by `rqss`, we selected it through AIC (via the `AIC` function), separately for each order α .

```
rank <- rank(X, ties.method = "random")
X[rank] <- X
Y[rank] <- Y
alpha <- c(0.05, 0.25, 0.5, 0.75, 0.95)
x <- seq(min(X), max(X), length = 100)
n <- length(X)
lambda <- array(0, dim = c(length(alpha), 1))
interval = c(0.2, 10)
for(i in 1:length(alpha)){
  AIC_crit <- function(lambda){
    AIC(rqss(Y ~ qss(X, lambda = lambda), tau = alpha[i]))[1]
  }
  select_lambda <- optimize(AIC_crit, interval = interval)
  lambda[i] <- select_lambda$min
}
hatq <- array(0, dim = c(length(x), length(alpha)))
fitted_matrix <- array(0, dim = c(n,length(alpha)))
for(l in 1:length(alpha)){
  res_rqss <- rqss(Y ~ qss(X, lambda = lambda[l]), tau = alpha[l])
  fitted_matrix[,l] <- fitted(res_rqss)
}
plot(X, Y, col = col.plot[1], cex = 0.7);
for(i in 1:length(alpha)){
  lines(fitted_matrix[, i] ~ X, type = "l", col = col.plot[i+1])
}
```

The same exercise is repeated for R_2 , but with λ tested between 0.5 and 15. Since they are piecewise linear, the resulting spline-based reference curves are less smooth than the one based on quantization. Arguably, the latter better adapt to the samples even though they are sometimes quite wiggly.

Of course, the computational burden is also an important issue. Therefore, Table 1 gathers, for each method and each regression problem, the average and standard deviation of the computing times

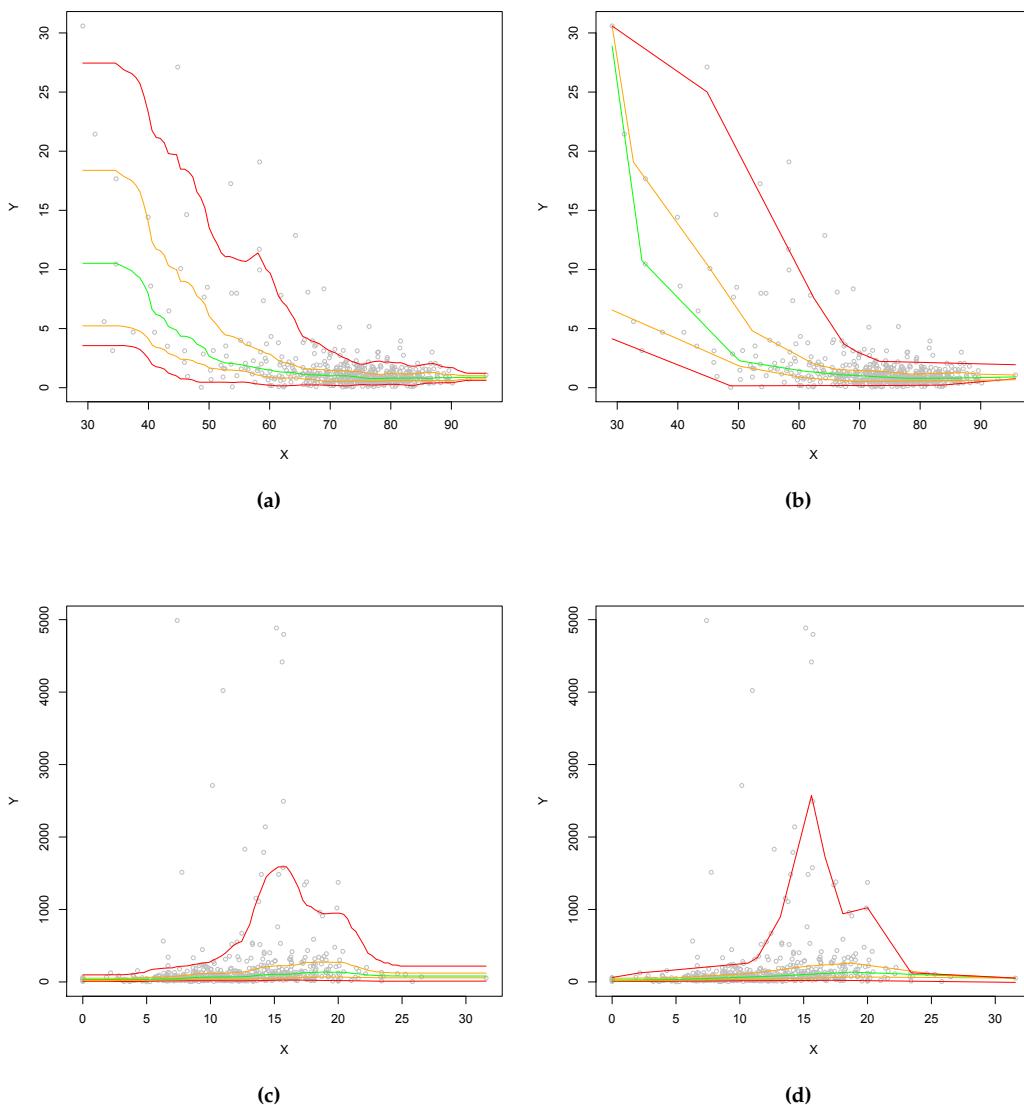


Figure 5: Estimated conditional quantile curves obtained with QuantifQuantile (left) and rqss (right), for regression R_1 (top) and regression R_2 (bottom). In each case, the quantile orders considered are $\alpha = 0.05, 0.25, 0.50, 0.75$ and 0.95 .

	QuantifQuantile	rqss
R_1	2.83 (0.117)	4.39 (0.119)
R_2	2.47 (0.085)	4.08 (0.115)

Table 1: Averages of the computing times (in seconds) to obtain 50 times the conditional quantile curves in Example 3 for QuantifQuantile and rqss, respectively; standard deviations are reported in parentheses.

in a collection of 50 runs (these 50 runs were considered to make results more reliable). In each case, our method is faster than its spline-based competitor.

Example 4: Real data with three-dimensional covariate

To treat an example with $d > 2$, we reconsider the data set in Example 3, this time with the response $Y =$ population density and the three covariates $X_1 =$ percentage of farmers, $X_2 =$ percentage of unemployed workers, and $X_3 =$ percentage of managers. In this setup, no graphical output is

available. We therefore restrict to a finite collection of x -values where conditional quantiles are to be estimated. Denoting by M_j and \bar{X}_j , $j = 1, 2, 3$, the maximal value and the average of X_{ij} , $i = 1, \dots, n = 542$, respectively, we consider the following eight values of x :

$$x_1 = \begin{pmatrix} \bar{X}_1 \\ \bar{X}_2 \\ \bar{X}_3 \end{pmatrix}, x_2 = \begin{pmatrix} \frac{1}{2}(\bar{X}_1 + M_1) \\ \bar{X}_2 \\ \bar{X}_3 \end{pmatrix}, x_3 = \begin{pmatrix} \bar{X}_1 \\ \frac{1}{2}(\bar{X}_2 + M_2) \\ \bar{X}_3 \end{pmatrix}, x_4 = \begin{pmatrix} \bar{X}_1 \\ \bar{X}_2 \\ \frac{1}{2}(\bar{X}_3 + M_3) \end{pmatrix},$$

$$x_5 = \begin{pmatrix} \frac{1}{2}(\bar{X}_1 + M_1) \\ \frac{1}{2}(\bar{X}_2 + M_2) \\ \bar{X}_3 \end{pmatrix}, x_6 = \begin{pmatrix} \frac{1}{2}(\bar{X}_1 + M_1) \\ \bar{X}_2 \\ \frac{1}{2}(\bar{X}_3 + M_3) \end{pmatrix}, x_7 = \begin{pmatrix} \bar{X}_1 \\ \frac{1}{2}(\bar{X}_2 + M_2) \\ \frac{1}{2}(\bar{X}_3 + M_3) \end{pmatrix}, x_8 = \begin{pmatrix} \frac{1}{2}(\bar{X}_1 + M_1) \\ \frac{1}{2}(\bar{X}_2 + M_2) \\ \frac{1}{2}(\bar{X}_3 + M_3) \end{pmatrix}.$$

The function `QuantifQuantile.d` is then evaluated for the response and covariates indicated above, and with the arguments $\text{alpha} = (0.25, 0.5, 0.75)'$, $\text{testN} = (5, 6, 7, 8, 9, 10)'$, x being the 3×8 matrix whose columns are the vectors x_1, x_2, \dots, x_8 just defined and ncores being the number of cores detected by R minus 1.

```
data(gironde)
set.seed(729848)
X1 <- gironde[[1]]$farmers
X2 <- gironde[[1]]$unemployed
X3 <- gironde[[1]]$managers
Y <- gironde[[2]]$density
X <- matrix(c(X1, X2, X3), nr = 3, byrow = TRUE)
n <- length(X)/3
d <- 3
alpha <- c(0.25, 0.5, 0.75)
x1 <- round(c(mean(X1), mean(X2), mean(X3)))
x2 <- round(c((mean(X1) + max(X1))/2, mean(X2), mean(X3)))
x3 <- round(c(mean(X1), (mean(X2) + max(X2))/2, mean(X3)))
x4 <- round(c(mean(X1), mean(X2), (mean(X3) + max(X3))/2))
x5 <- round(c((mean(X1) + max(X1))/2, (mean(X2) + max(X2))/2, mean(X3)))
x6 <- round(c((mean(X1) + max(X1))/2, mean(X2), (mean(X3) + max(X3))/2))
x7 <- round(c(mean(X1), (mean(X2) + max(X2))/2, (mean(X3) + max(X3))/2))
x8 <- round(c((mean(X1) + max(X1))/2, (mean(X2) + max(X2))/2, (mean(X3) + max(X3))/2))
x <- matrix(c(x1, x2, x3, x4, x5, x6, x7, x8), nr = d)
res <- QuantifQuantile.d(X, Y, x, alpha = alpha, testN = seq(5, 10, by = 1),
  same_N = F, ncores = detectCores() - 1)
round(fitted.values(res), 2)
```

This provided $\hat{N}_{\alpha,\text{opt}} = 8, 7$ and 7 , for $\alpha = 0.25, 0.50$ and 0.75 , respectively. The total computation time is 6.86 seconds. The `fitted.values` function then allowed to return the following matrix `hatq_opt` of estimated conditional quantiles :

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
[1,]	44.30	22.59	39.50	71.59	25.05	22.40	76.37	24.19
[2,]	80.07	32.31	81.24	161.85	35.01	31.92	145.29	38.18
[3,]	139.16	46.50	223.13	344.92	53.73	47.01	402.98	73.19

This collection of (estimated) conditional quartiles allows to appreciate the impact of a marginal perturbation of the covariates on Y 's conditional median (location) or interquartile range (scale). For instance, the results suggest that Y 's conditional median decreases with X_1 , is stable with X_2 , and increases with X_3 , whereas its conditional interquartile range decreases with X_1 but increases much with X_2 and with X_3 . The eight x -values considered further allow to look at the joint impact of two or three covariates on Y 's conditional location and scale. Of course, other shifts in the covariates (and other orders α) should further be considered to fully appreciate the dependence of Y on X .

Conclusion

In this paper, we described the package `QuantifQuantile` that allows to implement the quantization-based quantile regression method introduced in [Charlier et al. \(2015a,b\)](#). The package is simple to use, as the function `QuantifQuantile` and its multivariate versions essentially only require providing the covariate and response as arguments. Since the choice of the tuning parameter N is crucial, a warning

message is printed if it is not well-chosen and the function plot can also be used as guide to change adequately the value of the parameter testN in the various functions. Moreover, a graphical illustration is directly provided by the same function plot when the dimension of the covariate is smaller than or equal to 2. Finally, this package also contains a function that provides optimal quantization grids, which might be useful in other contexts, too.

Finally, we stress that quantization-based estimators, like most nonparametric smoothing procedures, are likely to perform poorly in high-dimensional situations due to the curse of dimensionality. For large d , it is therefore unclear how to assess whether a given covariate has a significant impact on the response variable. For small d , however, it is always possible, in the absence of a formal testing procedure, to resort to visual inspection. In the simplest case of a single covariate ($d = 1$), this would lead to looking whether or not fitted curves approximately are horizontal lines. This can be extended to the case $d = 2$.

Acknowledgments

The authors are grateful to the Editor and two anonymous referees for their careful reading and insightful comments that led to a significant improvement of the original manuscript. The first author's research is supported by a Bourse FRIA of the Fonds National de la Recherche Scientifique, Communauté française de Belgique. The second author's research is supported by an A.R.C. contract from the Communauté Française de Belgique and by the IAP research network grant P7/06 of the Belgian government (Belgian Science Policy).

Appendix

Illustration of choice.grid

We here put to work the function choice.grid in the univariate and bivariate cases. This function provides the “optimal” grid generated by the stochastic gradient algorithm described earlier. As above mentioned, quantization was extensively used in many other fields as numerical integration, cluster analysis, numerical probability or finance (Pagès, 1998; Pagès et al., 2004). Therefore, this function can be of interest outside the regression setup considered here.

We start with the univariate case and generate a random sample of size $n = 500$ from the uniform distribution over $(-2, 2)$. With $N = 15$ and $ng = 1$, this function provides a single initial grid (obtained by sampling without replacement among the uniform sample) and the corresponding optimal grid returned by the algorithm. Figure 6 represents the observations (in grey), the initial grid (in red), and the optimal grid (in green). The same exercise is repeated with sample size $n = 5,000$, and the results are also given in Figure 6.

```
set.seed(643625)
n <- 500
X <- runif(n, -2, 2)
N <- 15
ng <- 1
res <- choice.grid(X, N, ng)
# Plots of the initial and optimal grids
plot(X, rep(1, n), col = "grey", cex = 0.5, ylim = c(-0.1, 1.1), yaxt = "n",
     ylab = "")
points(res$init_grid, rep(0.5, N), col = "red", pch = 16, cex = 1.2)
points(res$opti_grid, rep(0, N), col = "forestgreen", pch = 16, cex = 1.2)
```

Since the parent distribution is uniform over $(-2, 2)$, the population optimal grid is the equispaced grid on that interval (Pagès, 1998). For both sample sizes considered, the optimal grid provided by the choice.grid function is much closer to the population optimal grid than the initial one. Recalling that the stochastic gradient algorithm in choice.grid performs as many iterations as observations in the original sample, it is not surprising that the optimal grid associated with the sample of size 5,000 better approximates the population optimal grid than the optimal grid associated with the sample of size 500.

Finally, we turn to the bivariate case and generate two random samples of size $n = 2,000$ and size $n = 20,000$ from the uniform distribution over the square $(-2, 2)^2$. The function choice.grid was applied to these samples with $N = 30$ and $ng = 1$. The resulting couple of initial and optimal grids are plotted in Figure 7. As in the univariate case, we observe an improvement when going from the initial grids to the corresponding optimal grids provided by the function choice.grid (here as well, the

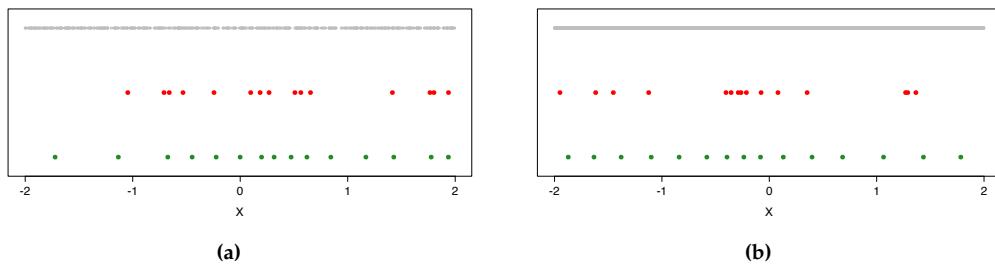


Figure 6: For $n = 500$ (left) and $n = 5,000$ (right), a random sample of size n from the uniform distribution over $(-2, 2)$ (in grey), an initial grid of size 15 obtained by sampling without replacement among these n observations (in red), and the “optimal” grid returned by choice.grid (in green).

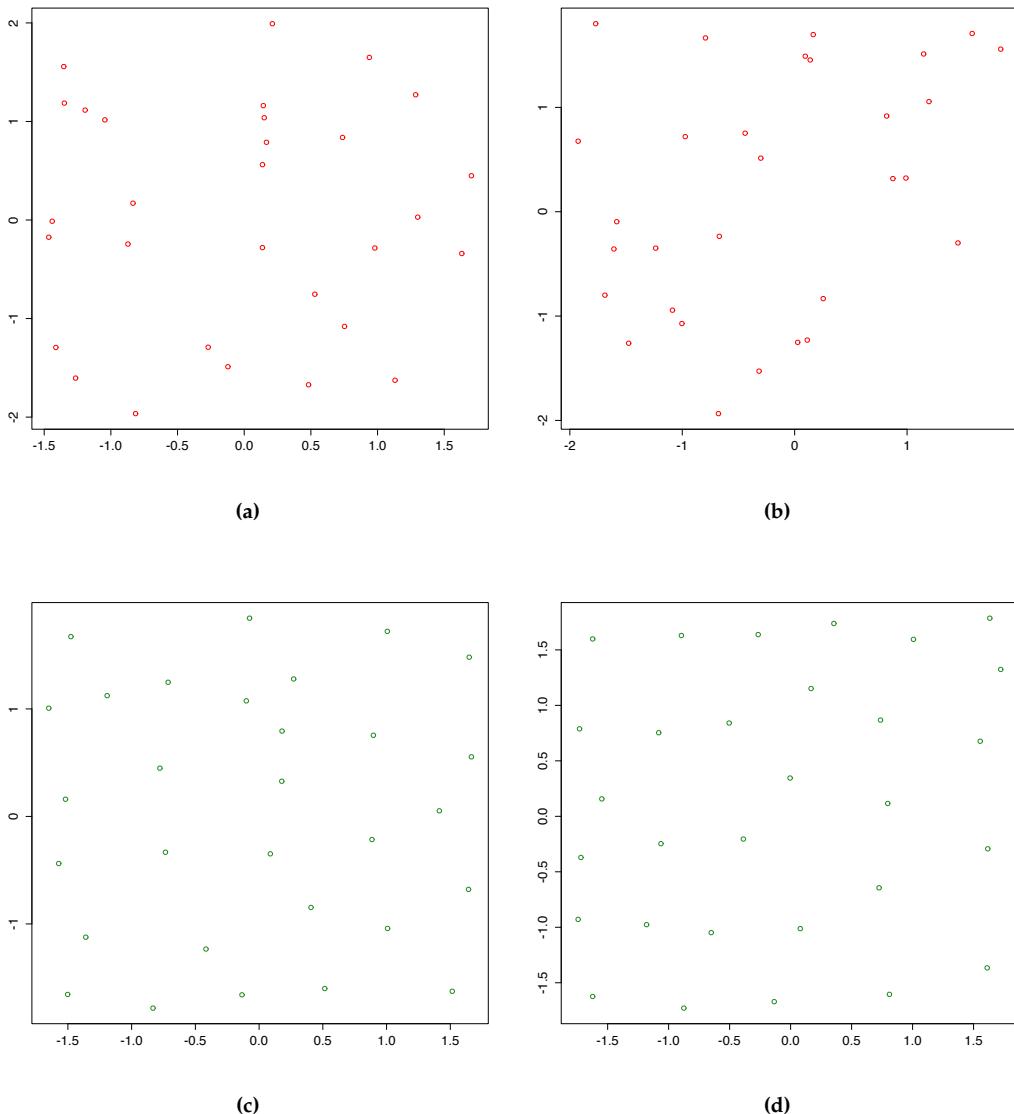


Figure 7: For $n = 2,000$ (left) and $n = 20,000$ (right), an initial grid of size 15 obtained by sampling without replacement among a random sample of size n from the uniform distribution over $(-2, 2)$ (top), and the corresponding optimal grid returned by choice.grid (bottom).

population optimal grid should be uniformly spread over the support of the underlying distribution). Also, it is still the case that the resulting optimal grid is better when based on a larger sample size n .

```

set.seed(345689)
n <- 2000
X <- matrix(runif(n*2, -2, 2), nc = n)
N <- 30
ng <- 1
res <- choice.grid(X, N, ng)
col <- c("red", "forestgreen")
plot(res$init_grid[1,,1], res$init_grid[2,,1], col = col[1], xlab = "", ylab = "")
plot(res$opti_grid[1,,1], res$opti_grid[2,,1], col = col[2], xlab = "", ylab = "")

```

Bibliography

- D. Adler, D. Murdoch, and others. *rgl: 3D Visualization Device System (OpenGL)*, 2015. URL <https://CRAN.R-project.org/package=rgl>. R package version 0.95.1367. [p69]
- P. K. Bhattacharya and A. K. Gangopadhyay. Kernel and nearest-neighbor estimation of a conditional quantile. *The Annals of Statistics*, 18(3):1400–1415, 1990. [p65]
- I. Charlier, D. Paindaveine, and J. Saracco. Conditional quantile estimation through optimal quantization. *Journal Statistical Planning and Inference*, 156:14–30, 2015a. [p65, 66, 67, 76]
- I. Charlier, D. Paindaveine, and J. Saracco. Conditional quantile estimation based on optimal quantization: from theory to practice. *Computational Statistics & Data Analysis*, 91:20–39, 2015b. [p65, 66, 67, 68, 76]
- I. Charlier, D. Paindaveine, and J. Saracco. *QuantifQuantile: Estimation of Conditional Quantiles using Optimal Quantization*, 2015c. URL <https://CRAN.R-project.org/package=QuantifQuantile>. R package version 2.0. [p65]
- J. Fan, T.-C. Hu, and Y. Truong. Robust nonparametric function estimation. *Scandinavian Journal of Statistics*, 21(4):433–446, 1994. [p65]
- A. Gannoun, S. Girard, C. Guinot, and J. Saracco. Reference curves based on non-parametric quantile regression. *Statistics in Medicine*, 21(4):3119–3135, 2002. [p65]
- S. Graf and H. Luschgy. *Foundations of Quantization for Probability Distributions*, volume 1730 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2000. [p66]
- R. Koenker. *quantreg: Quantile Regression*, 2015. URL <https://CRAN.R-project.org/package=quantreg>. R package version 5.19. [p65]
- R. Koenker and G. Bassett, Jr. Regression quantiles. *Econometrica*, 46(1):33–50, 1978. [p65]
- R. Koenker and I. Mizera. Penalized triograms: Total variation regularization for bivariate smoothing. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 66(1):145–163, 2004. [p65]
- R. Koenker, P. Ng, and S. Portnoy. Quantile smoothing splines. *Biometrika*, 81(4):673–680, 1994. [p65, 68]
- V. M. R. Muggeo. *quantregGrowth: Growth Charts via Regression Quantiles*, 2015. URL <https://CRAN.R-project.org/package=quantregGrowth>. R package version 0.3-1. [p65]
- V. M. R. Muggeo, M. Sciandra, A. Tomasello, and S. Calvo. Estimating growth charts via nonparametric quantile regression: A practical framework with application in ecology. *Environmental and Ecological Statistics*, 20(4):519–531, 2013. [p65]
- G. Pagès. A space quantization method for numerical integration. *Journal of Computational and Applied Mathematics*, 89(1):1–38, 1998. [p66, 77]
- G. Pagès, H. Pham, and J. Printems. Optimal quantization methods and applications to numerical problems in finance. In *Handbook of Computational and Numerical Methods in Finance*, pages 253–297. Birkhäuser Boston, Boston, MA, 2004. [p66, 77]
- K. Yu and M. C. Jones. Local linear quantile regression. *Journal of the American Statistical Association*, 93(441):228–237, 1998. [p65, 68]
- K. Yu, Z. Lu, and J. Stander. Quantile regression: Applications and current research areas. *Journal of the Royal Statistical Society, Series D (The Statistician)*, 52(3):331–350, 2003. [p65]

Isabelle Charlier
Université Libre de Bruxelles
Département de Mathématique and ECARES
Campus Plaine, Boulevard du Triomphe, CP210
1050 Brussels
Belgium
and
Université de Bordeaux
IMB, UMR 5251
33400 Talence
France
and
Inria Bordeaux Sud-Ouest
200 Avenue Vieille Tour
33400 Talence
France
ischarli@ulb.ac.be

Davy Paindaveine
Université Libre de Bruxelles
ECARES and Département de Mathématique
Avenue F.D. Roosevelt, 50, ECARES - CP114/04
1050 Brussels
Belgium
dpaindav@ulb.ac.be

Jérôme Saracco
Université de Bordeaux
IMB, UMR 5251
33400 Talence
France
and
Inria Bordeaux Sud-Ouest
200 Avenue Vieille Tour
33400 Talence
France
Jerome.Saracco@math.u-bordeaux1.fr

Numerical Evaluation of the Gauss Hypergeometric Function with the `hypergeo` Package

by Robin K. S. Hankin

Abstract This paper introduces the `hypergeo` package of R routines for numerical calculation of hypergeometric functions. The package is focussed on efficient and accurate evaluation of the Gauss hypergeometric function over the whole of the complex plane within the constraints of fixed-precision arithmetic. The hypergeometric series is convergent only within the unit circle, so analytic continuation must be used to define the function outside the unit circle. This short document outlines the numerical and conceptual methods used in the package; and justifies the package philosophy, which is to maintain transparent and verifiable links between the software and Abramowitz and Stegun (1965). Most of the package functionality is accessed via the single function `hypergeo()`, which dispatches to one of several methods depending on the value of its arguments. The package is demonstrated in the context of game theory.

Introduction

The geometric series $\sum_{k=0}^{\infty} t_k$ with $t_k = z^k$ may be characterized by its first term and the constant ratio of successive terms $t_{k+1}/t_k = z$, giving the familiar identity $\sum_{k=0}^{\infty} z^k = (1-z)^{-1}$. Observe that while the series has unit radius of convergence, the right hand side is defined over the whole complex plane except for $z = 1$ where it has a pole. Series of this type may be generalized to a hypergeometric series in which the ratio of successive terms is a rational function of k :

$$\frac{t_{k+1}}{t_k} = \frac{P(k)}{Q(k)}$$

where $P(k)$ and $Q(k)$ are polynomials. If both numerator and denominator have been completely factored we would write

$$\frac{t_{k+1}}{t_k} = \frac{(k+a_1)(k+a_2) \cdots (k+a_p)}{(k+b_1)(k+b_2) \cdots (k+b_q)(k+1)} z$$

where z is the ratio of the leading terms of $P(k)$ and $Q(k)$ (the final term in the denominator is due to historical reasons), and if we require $t_0 = 1$ then we write

$$\sum_{k=0}^{\infty} t_k z^k = {}_p F_q \left[\begin{matrix} a_1, a_2, \dots, a_p \\ b_1, b_2, \dots, b_q \end{matrix}; z \right] \quad (1)$$

where it is understood that $q \geq p - 1$. The series representation, namely

$$1 + \frac{\prod_{i=1}^p a_i}{\prod_{i=1}^q b_i} z + \frac{\prod_{i=1}^p a_i (a_i + 1)}{\prod_{i=1}^q b_i (b_i + 1) 2!} z^2 + \cdots + \frac{\prod_{i=1}^p a_i (a_i + 1) \cdots (a_i + k)}{\prod_{i=1}^q b_i (b_i + 1) \cdots (b_i + k) k!} z^k + \cdots \quad (2)$$

is implemented in the package as `genhypergeo_series()` and operates by repeatedly incrementing the upper and lower index vectors (a_1, \dots, a_p) and (b_1, \dots, b_q) , and taking an appropriate running product. Terms are calculated and summed successively until a new term does not change the sum.

In most cases of practical interest one finds that $p = 2, q = 1$ suffices (Seaborn, 1991). Writing a, b, c for the two upper and one lower argument respectively, the resulting function ${}_2 F_1(a, b; c; z)$ is known as the hypergeometric function, or Gauss's hypergeometric function. Many functions of elementary analysis are of this form; examples would include logarithmic and trigonometric functions, Bessel functions, etc. For example, ${}_2 F_1\left(\frac{1}{2}, 1; \frac{3}{2}; -z^2\right) = z^{-1} \arctan z$.

Michel and Stoitsov (2008) state that physical applications are “plethora”; examples would include atomic collisions (Alder et al., 1956), cosmology (de la Cruz-Dombriz and Dobado, 2006), and analysis of Feynman diagrams (Davydychev and Kalmykov, 2004). In addition, naturally-occurring combinatorial series frequently have a sum expressible in terms of hypergeometric functions (Petkovsek et al., 1997). One meets higher-order hypergeometric functions occasionally; the hypergeometric distribution, for example, has a cumulative distribution function involving the ${}_3 F_2$ generalized hypergeometric

function. An example from the author's work in the field of game theory is given below.

Numerical implementations

There are two other numerical implementations for the hypergeometric function for R: the `gsl` package (Hankin, 2006b), a wrapper for the Gnu Scientific Library, although this does not cover complex values (Galassi et al., 2013); and the `appell` package (Bove et al., 2013) which implements the Gauss hypergeometric function as `hyp2f1()`.

Outside the R world, there are several proprietary implementations but the evaluation methodology is not available for inspection. Open-source implementations include that of Sage (Stein et al., 2015) and Maxima (2014). The `hypergeo` package is offered as an R-centric suite of functionality with an emphasis on multiple evaluation methodologies, and transparent coding with nomenclature and structure following that of Abramowitz and Stegun (1965). An example is given below in which the positions of the cut lines may be modified.

Equivalent forms

The hypergeometric function's series representation, namely

$$_2F_1(a, b; c; z) = \sum_{k=0}^{\infty} \frac{(a)_k (b)_k}{(c)_k k!} z^k, \quad (a)_k = \Gamma(a+k)/\Gamma(a) \quad (15.1.1)$$

has unit radius of convergence by the ratio test [NB: equations with three-part numbers, as 15.1.1 above, are named for their reference in Abramowitz and Stegun (1965)]. However, the integral form

$$_2F_1(a, b; c; z) = \frac{\Gamma(c)}{\Gamma(b)\Gamma(c-b)} \int_{t=0}^1 t^{b-1} (1-t)^{c-b-1} (1-tz)^{-a} dt, \quad (15.3.1)$$

due to Gauss, furnishes analytic continuation; it is usual to follow Riemann and define a cut along the positive real axis from 1 to ∞ and specify continuity from below (but see below). This is implemented as `f15.3.1()` in the package and exhibits surprisingly accurate evaluation.

Gauss also provided a continued fraction form for the hypergeometric function (implemented as `hypergeo_continfrac()` in the package) which has superior convergence rates for parts of the complex plane at the expense of more complicated convergence properties (Cuyt et al., 2008).

The `hypergeo` package

The `hypergeo` package provides some functionality for the hypergeometric function. the emphasis is on fast vectorized R-centric code, complex z and moderate real values for the auxiliary parameters a, b, c . Extension to complex auxiliary parameters might be possible but Michel and Stoitsov (2008) caution that this is not straightforward. The package is released under GPL-2.

The majority of the package functionality is accessed via the `hypergeo()` function whose behaviour is discussed below.

Observing the slow convergence of the series representation 15.1.1, the complex behaviour of the continued fraction representation, and the heavy computational expense of the integral representation 15.3.1, it is clear that non-trivial numerical techniques are required for a production package.

The package implements a generalization of the method of Forrey (1997) to the complex case. It utilizes the observation that the ratio of successive terms approaches z , and thus the strategy adopted is to seek a transformation which reduces the modulus of z to a minimum. Abramowitz and Stegun give the following transformations:

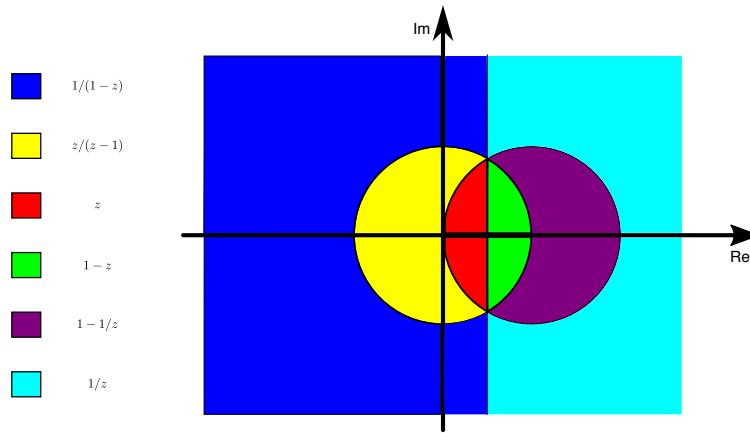


Figure 1: View of the complex plane showing which of equations 15.3.4 to 15.3.9 transforms to the value of smallest modulus. The yellow, green, and red region is the unit circle.

$$_2F_1(a, b; c; z) = (1-z)^{-a} {}_2F_1\left(a, c-b; c; \frac{z}{z-1}\right) \quad (15.3.4)$$

$$= (1-z)^{-b} {}_2F_1\left(a, c-a; c; \frac{z}{z-1}\right) \quad (15.3.5)$$

$$= \frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)} {}_2F_1(a, b; a+b-c+1; 1-z) + (1-z)^{c-a-b} \frac{\Gamma(c)\Gamma(a+b-c)}{\Gamma(a)\Gamma(b)} {}_2F_1(c-a, c-b; c-a-b+1; 1-z) \quad (15.3.6)$$

$$= \frac{\Gamma(c)\Gamma(b-a)}{\Gamma(b)\Gamma(c-a)} (-z)^{-a} {}_2F_1\left(a, 1-c+a; 1-b+a; \frac{1}{z}\right) + \frac{\Gamma(c)\Gamma(a-b)}{\Gamma(a)\Gamma(c-b)} (-z)^{-b} {}_2F_1\left(b, 1-c+b; 1-a+b; \frac{1}{z}\right) \quad (15.3.7)$$

$$= (1-z)^{-a} \frac{\Gamma(c)\Gamma(b-a)}{\Gamma(b)\Gamma(c-a)} {}_2F_1\left(a, c-b; a-b+1; \frac{1}{1-z}\right) + (1-z)^{-b} \frac{\Gamma(c)\Gamma(a-b)}{\Gamma(a)\Gamma(c-b)} {}_2F_1\left(b, c-a; b-a+1; \frac{1}{1-z}\right) \quad (15.3.8)$$

$$= \frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)} z^{-a} {}_2F_1\left(a, a-c+1; a+b-c+1; 1-\frac{1}{z}\right) + \frac{\Gamma(c)\Gamma(a+b-c)}{\Gamma(a)\Gamma(b)} (1-z)^{c-a-b} z^{a-c} {}_2F_1\left(c-a, 1-a; c-a-b+1; 1-\frac{1}{z}\right). \quad (15.3.9)$$

The primary argument in equations 15.3.4–15.3.9 is a member of the set

$$M = \left\{ z, \frac{z}{z-1}, 1-z, \frac{1}{z}, \frac{1}{1-z}, 1-\frac{1}{z} \right\};$$

and, observing that M is closed under functional composition, we may apply each of the transformations to the primary argument z and choose the one of smallest absolute value to evaluate using `genhypergeo_series()`; see Figure 1 for a diagram showing which parts of the complex plane use which transformation.

Given the appropriate transformation, the right hand side is evaluated using direct summation. If $|z| < 1$, the series is convergent by the ratio test, but may require a large number of terms to achieve acceptable numerical precision. Summation is dispatched to `genhypergeo_series()` which evaluates the generalized hypergeometric function, Equation 1; the R implementation uses multiplication by repeatedly incremented upper and lower indices a_i, b_i .

Thus for example if $(1-z)^{-1}$ is small in absolute value we would use function `f15.3.8()`:

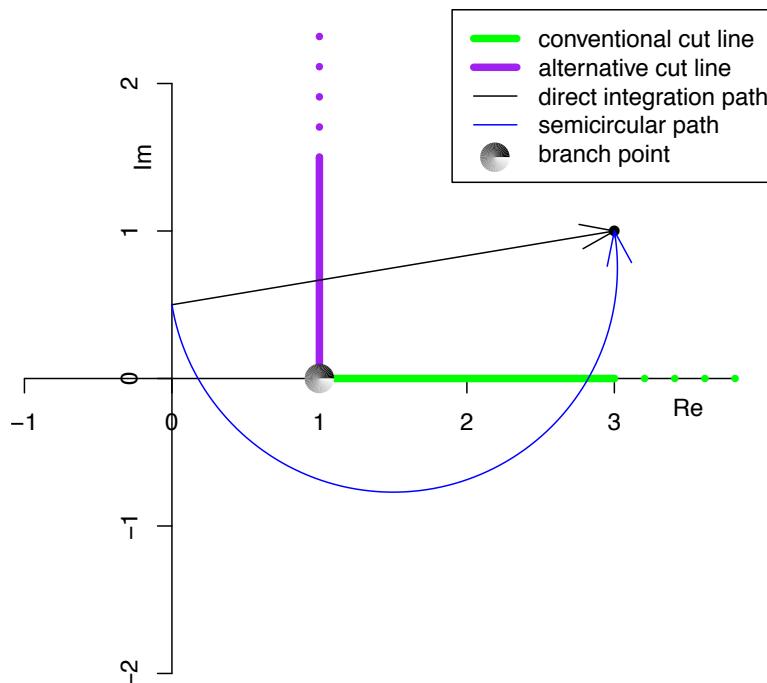


Figure 2: Different integration paths for evaluating ${}_2F_1(3+i)$ from a start point of $i/2$. The straight line path avoids the conventional cut line (green), unlike the semicircular path, which would be consistent with the alternative cut line (purple). The values at $z = 3 + i$ differ because of the residue at $z = 1$.

```
> require("hypergeo")
> f15.3.8

function(A, B, C, z, tol = 0, maxiter = 2000) {
  jj <- i15.3.8(A, B, C)
  jj[1] * (1-z)^(-A) * genhypergeo(U = c(A, C-B), L = A-B+1, z = 1/(1-z), tol = tol,
    maxiter = maxiter) + jj[2] * (1-z)^(-B) * genhypergeo(U = c(B, C-A), L = B-A+1,
    z = 1/(1-z), tol = tol, maxiter = maxiter)
}
```

(slightly edited in the interests of visual clarity). This is a typical internal function of the package and like all similar functions is named for its equation number in Abramowitz and Stegun (1965). Note the helper function `i15.3.9()`, which calculates the Gamma coefficients of the two hypergeometric terms in the identity. This structure allows transparent checking of the code.

Cut lines

The hypergeometric differential equation

$$z(1-z)F''(z) + [c - (a+b+1)z]F'(z) - abF(z) = 0, \quad (15.5.1)$$

together with a known value of $F(z)$ and $F'(z)$ may be used to define ${}_2F_1(z)$. Because $z = 1$ and $z = \infty$ are in general branch points, requiring $F(\cdot)$ to be single valued necessitates a cut line that connects these two points. It is usual to specify a a cut line following the real axis from 1 to ∞ ; but sometimes this is inconvenient. Figure 2 shows an example of different integration paths being used to relocate the cut line.

The package includes functionality for solving equation 15.5.1 using `ode()` from the `deSolve` package (Soetaert et al., 2010):

```
> f15.5.1(
+   A = 1.1, B = 2.2, C = 3.5, z = 3+1i, startz = 0.5i,
+   u = function(u) straight(u, 0.5i, 3+1i),
+   udash = function(u) straighthdash(u, 0.5i, 3+1i))

[1] -0.5354302+0.7081344i
```

```

> f15.5.1(
+   A = 1.1, B = 2.2, C = 3.5, z = 3+1i, startz = 0.5i,
+   u = function(u) semicircle(u, 0.5i, 3+1i, FALSE),
+   udash = function(u) semidash(u, 0.5i, 3+1i, FALSE))

[1] -1.395698-0.043599i

> hypergeo(1.1, 2.2, 3.5, 3+1i)

[1] -0.5354302+0.7081338i

```

See how the different integration paths give different results; the straight path value matches that of `hypergeo()`. The package also provides `hypergeo_press()`, which is somewhat more user-friendly but less flexible, and uses the method recommended by Press et al. (1992).

Special cases

The series methods detailed above are not applicable for all values of the parameters a, b, c . If, for example, $c = a + b \pm m$, $m \in \mathbb{N}$ (a not uncommon case), then equation 15.3.6 is not useful because each term has a pole; and it is numerically difficult to approach the limit. In this case the package dispatches to `hypergeo_cover1()` which uses 15.3.4 through 15.3.9 but with 15.3.6 replaced with suitable limiting forms such as

$${}_2F_1(a, b; a+b; z) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(n!)^2} [2\psi(n+1) - \psi(a+n) - \psi(b+n) - \log(1-z)] (1-z)^n, \\ \pi < |\arg(1-z)| < \pi, |1-z| < 1 \quad (15.3.10)$$

This equation is comparable to 15.3.6 in terms of computational complexity but requires evaluation of the digamma function ψ . Equation 15.3.10 is evaluated in the package using an algorithm similar to that for `genhypergeo_series()` but includes a runtime option which specifies whether to evaluate $\psi(\cdot)$ *ab initio* each time it is needed, or to use the recurrence relation $\psi(z+1) = \psi(z) + 1/z$ at each iteration after the first. These two options appear to be comparable in terms of both numerical accuracy and speed of execution, but further work would be needed to specify which is preferable in this context.

A similar methodology is used for the case $b = a \pm m$, $m = 0, 1, 2, \dots$ in which case the package dispatches to `hypergeo_cover2()`.

However, the case $c - a = 0, 1, 2, \dots$ is not covered by Abramowitz and Stegun (1965) and the package dispatches to `hypergeo_cover3()` which uses formulae taken from the Wolfram functions site (Wolfram, 2014). For example `w07.23.06.0026.01()` gives a straightforwardly implementable numerical expression for ${}_2F_1$ as a sum of two *finite* series and a generalized hypergeometric function ${}_3F_2$ with primary argument z^{-1} .

In all these cases, the limiting behaviour is problematic. For example, consider a case where $|1-z| \ll 1$ and $a + b - c$ is close to, but not exactly equal to, zero. Then equation 15.3.10 is not applicable. The analytic value of the hypergeometric function in these circumstances is typically of moderate modulus, but both terms of equation 15.3.6 have large modulus and the numerics are susceptible to cancellation errors. However, in practice this issue seems to be rare as it arises only in contrived situations where one is deliberately testing the system. If a user really was interested in exploring this part of parameter space to high numerical precision then the package provides alternative methodologies such as the integral form `f15.3.1()` or the continued fraction form `genhypergeo_confrac()`.

Critical points

All the above methods fail when $z = \frac{1}{2} \pm \frac{i\sqrt{3}}{2}$, because none of the transformations 15.3.6–15.3.9 change the modulus of z from 1. The function is convergent at these points but numerical evaluation is difficult. This issue does not arise in the real case considered by Forrey (1997).

These points were considered by Buhring (1987) who presented a computational method for these values; however, his method is not suitable for finite-precision arithmetic (a brief discussion is presented at `?buhring`) and the package employs either `hypergeo_gosper()` which uses iterative scheme due to Gosper (Johansson et al., 2013), or the residue theorem if z is close to either of these points.

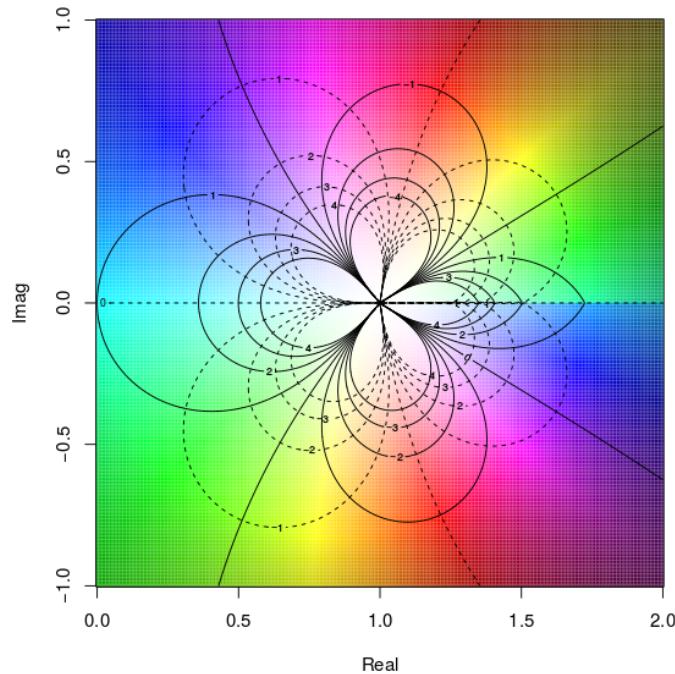


Figure 3: View of the function ${}_2F_1\left(2, \frac{1}{2}; \frac{2}{3}; z\right)$ evaluated over a part of the complex plane using the **hypergeo** package. Function visualization following Thaller (1998) and the **elliptic** package (Hankin, 2006a); hue corresponds to argument and saturation to modulus. Solid contour lines correspond to real function values and dotted to imaginary function values. Note the cut line along the real axis starting at $(1, 0)$, made visible by an abrupt change in hue.

Package testing suite

The package comes with an extensive test suite in the tests/ directory. The tests fall into two main categories, firstly comparison with either Maple or Mathematica output following Becken and Schmelcher (2000); and secondly, verification of identities which appear in Abramowitz and Stegun (1965) as elementary special cases. Consider, for example,

$${}_2F_1\left(a, 1-a; \frac{3}{2}; \sin^2(z)\right) = \frac{\sin[(2a-1)z]}{(2a-1)\sin z} \quad (15.1.15)$$

The left and right hand sides are given by eqn15.1.15a() and eqn15.1.15b() respectively which agree to numerical precision in the test suite; but care must be taken with regard to the placing of branch cuts. Further validation is provided by checking against known analytical results. For example, it is known that

$${}_2F_1\left(2, b; \frac{5-b}{2}; -\frac{1}{2}\right) = 1 - \frac{b}{3} \quad (3)$$

so, for example,

```
> hypergeo(2, 1, 2, -1/2)
[1] 0.66666666666667+0i
```

The package in use

The **hypergeo** package offers direct numerical functionality to the R user on the command line. The package is designed for use with R and Figure 3 shows the package being used to visualize ${}_2F_1\left(2, \frac{1}{2}; \frac{2}{3}; z\right)$ over a region of the complex plane.

A second example is given from the author's current work in game theory. Consider a game in which a player is given n counters each of which she must allocate into one of two boxes, A or B .

At times $t = 1, 2, 3 \dots$ a box is identified at random and, if it is not empty, a counter removed from it; box A is chosen with probability p and box B with probability $1 - p$. The object of the game is to remove all counters as quickly as possible. If the player places a counters in box A and b in B , then the probability mass function (PMF) of removing the final counter at time $t = a + b + r$ is

$$p^a(1-p)^b \left[\binom{a+b+r-1}{a-1, b+r} (1-p)^r + \binom{a+b+r-1}{a+r, b-1} p^r \right], \quad r = 0, 1, 2, \dots \quad (4)$$

The two terms correspond to the final counter being removed from box A or B respectively. The PMF for r has expectation

$$\begin{aligned} p^a(1-p)^b & \left[p \binom{a+b}{a+1, b-1} {}_2F_1(a+b+1, 2; a+2; p) + \right. \\ & \left. (1-p) \binom{a+b}{a-1, b+1} {}_2F_1(a+b+1, 2; b+2; 1-p) \right] \end{aligned} \quad (5)$$

with R idiom:

```
> expected <- function(a, b, p) {
+   Re(
+     choose(a+b, b) * p^a * (1-p)^b *
+     (p * b/(1+a) * hypergeo(a+b+1, 2, a+2, p) +
+      (1-p) * a/(1+b) * hypergeo(a+b+1, 2, b+2, 1-p)))
+ }
```

Thus if $p = 0.8$ and given $n = 10$ counters we might wonder whether it is preferable to allocate them $(8, 2)$ or $(9, 1)$:

```
> c(expected(8, 2, 0.8), expected(9, 1, 0.8))
[1] 3.019899 1.921089
```

showing that the latter allocation is preferable in expectation.

Conclusions

Evaluation of the hypergeometric function is hard, as evidenced by the extensive literature concerning its numerical evaluation (Becken and Schmelcher, 2000; Michel and Stoitsov, 2008; Forrey, 1997; Buhring, 1987). The **hypergeo** package is presented as a modular, R-centric implementation with multiple evaluation methodologies, providing reasonably accurate results over the complex plane and covering moderate real values of the auxiliary parameters a, b, c .

Bibliography

- M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, New York, 1965. [p81, 82, 84, 85, 86]
- K. Alder, A. S. Bohr, T. Huus, B. Mottelson, and A. Winther. Study of nuclear structure by electromagnetic excitation with accelerated ions. *Reviews of Modern Physics*, 28:432, 1956. [p81]
- W. Becken and P. Schmelcher. The analytic continuation of the Gaussian hypergeometric function ${}_2F_1(a, b; c; z)$ for arbitrary parameters. *Journal of Computational and Applied Mathematics*, 126:449–478, 2000. [p86, 87]
- D. S. Bove et al. *appell: Compute Appell's F1 Hypergeometric Function*, 2013. URL <https://CRAN.R-project.org/package=appell>. R package version 0.0-4. [p82]
- W. Buhring. An analytic continuation of the hypergeometric series. *SIAM Journal on Mathematical Analysis*, 18(3):884–889, 1987. [p85, 87]
- A. Cuyt et al. *Handbook of Continued Fractions for Special Functions*. Springer-Verlag, 2008. [p82]
- A. I. Davydychev and M. Y. Kalmykov. Massive Feynman diagrams and inverse binomial sums. *Nuclear Physics B*, 699:3–64, 2004. [p81]

- Á. de la Cruz-Dombriz and A. Dobado. $f(r)$ gravity without a cosmological constant. *Physical Review D*, 74(8–15), Oct. 2006. [p81]
- R. C. Forrey. Computing the hypergeometric function. *Journal of Computational Physics*, 137:79–100, 1997. [p82, 85, 87]
- M. Galassi et al. *GNU Scientific Library*, 2013. URL <http://www.gnu.org/software/gsl/>. Reference Manual edition 1.16. [p82]
- R. K. S. Hankin. Introducing elliptic, an R package for elliptic and modular functions. *Journal of Statistical Software*, 15, Feb. 2006a. [p86]
- R. K. S. Hankin. Special functions in R: Introducing the gsl package. *R News*, 6, Oct. 2006b. [p82]
- F. Johansson et al. *mpmath: A Python Library for Arbitrary-Precision Floating-Point Arithmetic (Version 0.18)*, Oct. 2013. <http://code.google.com/p/mpmath/>. [p85]
- Maxima. *Maxima, a Computer Algebra System. Version 5.34.1*. <http://maxima.sourceforge.net/>, 2014. URL <http://maxima.sourceforge.net/>. [p82]
- N. Michel and M. V. Stoitsov. Fast computation of the Gauss hypergeometric function with all its parameters complex with application to the Pöschl-Teller-Ginocchio potential wave functions. *Computer Physics Communications*, 178:535–551, 2008. URL <http://arxiv.org/abs/0708.0116v1>. [p81, 82, 87]
- M. Petkovsek, H. S. Wilf, and D. Zeilberger. *A = B*. A. K. Peters, Ltd., 1997. [p81]
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in C: The art of scientific computing, 1992. [p85]
- J. B. Seaborn. *Hypergeometric Functions and Their Applications*. Springer-Verlag, 1991. [p81]
- K. Soetaert, T. Petzoldt, and R. W. Setzer. Solving differential equations in R: Package desolve. *Journal of Statistical Software*, 33(9):1–25, 2010. URL <http://www.jstatsoft.org/v33/i09>. [p84]
- W. Stein et al. *Sage Mathematics Software (Version 5.8)*. The Sage Development Team, 2015. URL <http://www.sagemath.org>. [p82]
- B. Thaller. Visualization of complex functions. *The Mathematica Journal*, 7(2):163–180, 1998. [p86]
- S. Wolfram. The hypergeometric function, 2014. File Hypergeometric2F1.pdf, downloaded from <http://functions.wolfram.com/HypergeometricFunctions/>. [p85]

Robin K. S. Hankin
Auckland University of Technology
New Zealand
hankin.robin@gmail.com

SRCS: Statistical Ranking Color Scheme for Visualizing Parameterized Multiple Pairwise Comparisons with R

by Pablo J. Villacorta and José A. Sáez

Abstract The problem of comparing a new solution method against existing ones to find statistically significant differences arises very often in sciences and engineering. When the problem instance being solved is defined by several parameters, assessing a number of methods with respect to many problem configurations simultaneously becomes a hard task. Some visualization technique is required for presenting a large number of statistical significance results in an easily interpretable way. Here we review an existing color-based approach called Statistical Ranking Color Scheme (SRCS) for displaying the results of multiple pairwise statistical comparisons between several methods assessed separately on a number of problem configurations. We introduce an R package implementing SRCS, which performs all the pairwise statistical tests from user data and generates customizable plots. We demonstrate its applicability on two examples from the areas of dynamic optimization and machine learning, in which several algorithms are compared on many problem instances, each defined by a combination of parameters.

Introduction

When carrying out research in statistics, operational research and computer science, the problem of comparing a novel algorithm against other state-of-the-art techniques arises very often. The same idea can be applied to many other fields of science when introducing a new method for solving a well-known task, with the purpose of demonstrating the superiority of the proposed approach by numerically comparing the results with those obtained by already existing methods.

For some time now, it is widely accepted that statistical tests are required to compare several techniques that solve one given task (Demšar, 2006; García et al., 2010). This is motivated by the fact – also shown by Eugster et al. (2014) – that the performance of a technique for solving a task (for example, supervised classification) heavily depends on the characteristics of the concrete task instance (in this case, the data to which a classifier is to be fitted) and thus the experiments should randomize over a large number of datasets. Even with the same dataset, the results may vary when considering different subsets of training/test data (the former are used for fitting the model, and the latter for evaluating the model once it has been learned and does not change any more). The same applies to other very common machine learning tasks such as regression (Graczyk et al., 2010), approximate optimization using metaheuristics (García et al., 2009), and computational intelligence in general (Derrac et al., 2011). It should be noted that metaheuristics employed in optimization are by themselves randomized algorithms. Therefore, multiple runs of the same algorithm on the same optimization problem are required to assess an algorithm, as well as testing the performance over several different functions; we will further elaborate on this later. In order to analyze the results of these randomized trials, statistical tests are applied to draw a conclusion about the superiority of one method over the rest. A vast amount of literature exists dealing with this specific problem, see Coffin and Saltzman (2000); Shilane et al. (2008); García et al. (2010) and references therein, just to cite a few.

If one aims to visualize the results of statistical pairwise comparisons, the volume of data to display grows a lot if we take into account many problem configurations at the same time. The use of tables is very common as they summarize a lot of data in a compact way but they become hard to interpret when the results they contain are grouped in more than two parameters. It is usually very difficult to draw conclusions from big result tables, and for that reason, authors have developed data visualization techniques more sophisticated than boxplots or line charts, such as the figures presented in Demšar (2006) to distinguish between statistically different and indistinguishable algorithms, and other approaches explained in Bartz-Beielstein et al. (2010). A tool for the same purpose that is worth mentioning is the Model Viewer feature of the SPSS software (IBM Corp., 2012). When applied to hypothesis testing, it displays the multiple pairwise comparisons output as a complete graph where nodes represent the groups being compared, and arcs between them are colored differently according to the p -value of the corresponding comparison (in orange when the p -value is below a fixed significance threshold, and in black otherwise). Two remarkable tools are available for the R language. The `paircompviz` package (Burda, 2014), closely related to ours, makes use of Hasse diagrams with p -values in the arcs to represent the outcome of statistical tests. However, it does not use colors and it is not well suited for representing a large number of comparisons at once (as happens when we deal

with many different problem configurations) since the resulting Hasse diagram would be too complex. The `factorplot` package recently published in this journal (Armstrong, 2013) focuses on hypothesis testing concerning the coefficients of generalized linear models or coefficients in multinomial logistic regression models, representing the results of the comparisons in grayscale grid plots. Our approach is more general and is oriented mainly to simulation-based studies.

Approximate optimization and machine learning constitute two areas of knowledge in which the problem of representing statistical results under several factors arises naturally. In both cases, we often want to compare the algorithm performance separately on different problem setups to highlight the conditions under which certain algorithms may work specially well. Existing studies in the field of dynamic optimization employ up to 40 numeric tables or graphs in a paper to summarize their results, due to the number of different experimental settings tested and the large amount of parameters involved in each problem configuration. Obviously, interpreting such a huge amount of numeric results becomes unfeasible. Moreover, none of the aforementioned visualization approaches deals well with multiple factor problems.

In order to solve this problem, a novel color-based technique for multiple pairwise statistical comparisons under several factors, called Statistical Ranking Color Scheme (SRCS), was introduced in del Amo and Pelta (2013) for comparing the performance of several dynamic optimization algorithms under a number of different problem configurations (del Amo et al., 2012). The method relies on a wise use of color scales that simplifies the identification of overall trends along many different problem settings simultaneously, thus enabling better understanding and interpretation of the results, and providing an overview of the circumstances under which each algorithm outperforms (or is outperformed by) the rest. However, no software package was available so far to automatically generate this kind of graphs at once from a dataset that collects the numerical results. The code published in del Amo and Pelta (2013) only calculates the ranking obtained by several algorithms on a fixed problem configuration, but does not plot the results nor allows for an automatic computation over a whole set of different problem configurations in order to obtain the images shown in del Amo et al. (2012).

Our aim here is to present an easy-to-use R package called `SRCS` (Villacorta, 2015) for creating fully customizable plots from a results file in experiments involving several factors, so that the user can configure how the plots should be arranged in the figure and has control over all graphical details of it, such as colors, fonts, titles, etc. Furthermore, we demonstrate the applicability of our package in two different contexts. The first is the comparison of algorithms to solve dynamic optimization problems (DOPs), which is the setting for which SRCS was originally conceived. The second is a novel application to machine learning tasks, where SRCS is used to compare the performance of several supervised classification algorithms over synthetic datasets created based on several parameters. Examples of these are noisy and/or imbalanced data for which parameters like the severity and type of noise, or the imbalance ratio are considered when generating the dataset from an originally clear one.

The remainder of this contribution is structured as follows. After the introduction the foundation of the SRCS technique and how multiple statistical significance results are displayed in color plots is reviewed. The next section presents an R package implementing SRCS, with a detailed description of the most important functions, their common uses and how they should be called. Then we explain two case studies where `SRCS` has been applied to visualize the statistical results of comparing a number of algorithms for two very different tasks, namely dynamic optimization and supervised classification when the data from which the classifier is learned contain noise or are imbalanced. Finally, the last section is devoted to conclusions and further work.

Statistical ranking color scheme

In this section we briefly review the foundations of SRCS (del Amo and Pelta, 2013). SRCS was developed for analyzing the relative performance of algorithms on a problem, rather than the absolute one. In other words, the outcome is a rank for each algorithm that depends on how many algorithms are better, equal or worse than the algorithm being ranked, where the decision on each pairwise comparison is given by a non-parametric statistical test over two sets of samples corresponding to multiple runs of each algorithm in exactly the same conditions. No distinction is made concerning the magnitude of the advantage or disadvantage in the performance comparison: SRCS is interested only in whether one algorithm is statistically better or worse than another, but not in how much.

The rank assigned to an algorithm A_i on a problem configuration c (determined by at most 3 parameters) is the sum of the scores obtained by the algorithm when comparing its performance $\{\text{perf}_k\}_i^c$, $k = 1, \dots, K$ against the rest of the algorithms (all pairwise comparisons) over the same problem configuration c . The performance is given by a sample composed by K repeated observations obtained after K independent runs of A_i over the same problem configuration. It is assumed that

either the nature of A_i is itself randomized and gives a different output in each run, as happens with stochastic optimization algorithms, or the input data used by A_i are a random sample and thus differ for each run, as happens for instance when using cross-validation (CV) for assessing a classification algorithm with a given dataset. In the m -fold CV method (typically $m = 5$ or $m = 10$), $m - 1$ folds are used for building a model and the remaining fold is used for evaluating it and collecting the performance measure (accuracy or any other). This is repeated until every fold has been used exactly once as the test fold, hence collecting m different performance values. If the complete m -fold CV process is repeated r times, each time taking a different m -fold partition of the whole dataset, we obtain $K = m \cdot r$ independent measurements of the classifier's performance.

Ranks are calculated as follows. For each $j \neq i$, if the sample $\{\text{perf}_k\}_i^c$ is statistically better (in the sense of the performance measure we are using) than $\{\text{perf}_k\}_j^c$, then A_i adds 1 point to its rank, and A_j subtracts 1 point; if the opposite occurs, A_i subtracts 1 point and A_j adds 1 point. Otherwise, both algorithms are statistically equivalent so none of them modifies its rank. The initial rank of every algorithm is 0. With this approach, when comparing N algorithms, the maximum rank attainable by an algorithm is $N - 1$, which means it outperforms the rest, and the minimum is $-(N - 1)$, meaning it is outperformed by the rest.

The statistical test applied in pairwise comparisons could be customized by the user. In our implementation, we abide by the original proposal of [del Amo and Pelta \(2013\)](#) and use the pairwise Wilcoxon rank sum test with Holm's correction for multiple comparisons. Whether the test should be paired or not depends on the concrete problem we are facing, and can be set by the user. When assessing optimization algorithms, for instance, the test will most likely be non-paired since usually there is no relation between, say, the first execution of A_i and the first execution of A_j on the same problem configuration. In machine learning, the test should most likely be paired because all algorithms should be evaluated exactly with the same folds, hence the performance of the first execution of A_i is paired with the first execution of A_j because both were done with the same training and test subsets.

The strength of SRCS lies in its capability of arranging in a single plot the ranks obtained by many algorithms when tested separately over a lot of different problem configurations. Therefore, one can quickly visualize which configurations are the most favorable to each algorithm. This is done by using a grid of heatmaps. A heatmap represents three variables, namely the rank using a color scheme, and two variables in the X and Y axis of the heatmap, which we call the inner X and Y variables. At the same time, the whole heatmap is associated with one level of the other two variables, called the outer X and Y variables.

Figure 1 shows a toy example¹ of ranking calculation and depiction of a simulated problem involving four algorithms that constitute the four levels of the outer Y variable. The problem involves three more variables, namely the outer X variable (from which only the level *outX1* is displayed), the inner Y variable with four possible levels, and the inner X variable with four possible levels as well. In Figure 1c the arrangement within the global plot is displayed for a concrete problem configuration that is allocated in the top left-most corner (as *inner X variable* = 1, *inner Y variable* = 4) of the left-most column of heatmaps (since *outer X variable* = *outX1*). The number of levels of all variables does not have to be the same as in this particular case.

An R package implementing SRCS

The aim of the **SRCS** package is to offer a set of functions to obtain figures similar to the one above in a straightforward manner and, at the same time, provide R users with full customization capabilities over graphical aspects like font size, color, axes aspect and so on. This has been accomplished by accepting tagged lists that are passed almost unchanged to some built-in graphical functions of the base package **graphics** on which our code relies. This package is very flexible and can be easily adapted so that the final plot has exactly the desired appearance. Package **grid** was also considered initially, but the adaptation would require more coding since the default aspect (more elegant) is slightly more complicated to fit our exact needs.

The general workflow can be summarized as:

1. Use function **SRCSranks** on the data being analyzed in order to compute the rank for each combination of factors according to the performance exhibited by that combination, following the rules explained in the preceding section.
2. Use function **plot** on the object returned by **SRCSranks**, indicating where each factor should be placed in the resulting plot, in order to obtain a color plot depicting the ranks calculated previously for all the factor combinations.

¹Refer to Section [Case studies](#) for real examples.

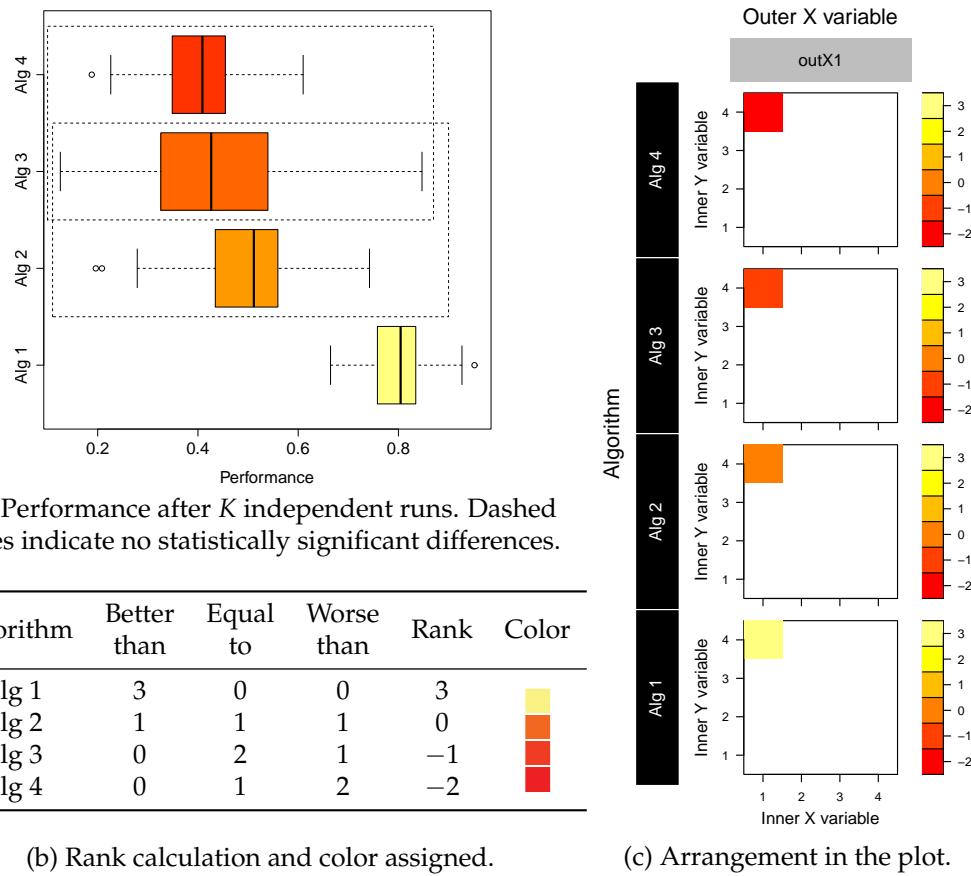


Figure 1: Rank calculation of the relative performance of four algorithms in a problem configuration defined by $\text{Inner X variable} = 1$, $\text{Inner Y variable} = 4$, $\text{Outer X variable} = \text{outX1}$.

3. (If needed) Use function `SRCScomparison` on the object returned by `SRCSranks`, specifying a concrete factor combination as well, to obtain a qualitative table describing the relative performance (measured from a statistical point of view) of every pair of levels of the target variable on the factor combination indicated. Each cell of the table contains a sign "=", ">" or "<" comparing the level on that row with the level on that column, where "=" stands for "no statistically significant differences found".
4. (If needed) Use function `animatedplot` on the object returned by `SRCSranks`, provided that the user data had more than one performance column, to visualize a video in which each video frame displays the ranks plot obtained by one performance column.
5. (If needed) Use function `singleplot` on the object returned by `SRCSranks`, specifying a factor combination that leaves two factors free, to visualize the ranks of one square of the full grid.

Functions `SRCSranks` and `SRCScomparison`

Our package exports five functions. Note that most of the arguments have default values to allow for a straightforward use if no customization is needed. The one that should be called first, prior to the plotting functions, is the following:

```
SRCSranks(data, params, target, performance, pairing.col = NULL,
          test = c("wilcoxon", "t", "tukeyHSD", "custom"), fun = NULL,
          correction = p.adjust.methods, alpha = 0.05, maximize = TRUE, ncores = 1,
          paired = FALSE)
```

We review the meaning of the arguments below. For further details please refer to the corresponding help pages.

- `data` is a data frame containing the repeated performance measures together with their problem configuration (Table 1).
- `params` is a vector of strings with the names of the columns that define a problem configuration (here: `c("A", "B", "C")`).

A	B	C	Target	Performance	Fold
a_1	b_1	c_1	Alg1	72.45	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
a_1	b_1	c_1	Alg1	72.36	K
a_1	b_1	c_1	Alg2	70.12	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
a_1	b_1	c_1	Alg2	69.89	K
a_1	b_1	c_1	Alg3	85.40	1
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
a_1	b_1	c_1	Alg3	85.21	K
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

Table 1: A subset of the input data in a problem with a 3-level target variable, three problem-defining parameters A, B, and C, with K observations of the performance per problem configuration, and pairing between the samples. Showing only a fixed problem configuration defined by $A = a_1$, $B = b_1$, $C = c_1$. In this case, the column called *Fold* acts as the pairing column as it links the performance values within a given problem configuration for the paired statistical tests.

- target is the name of the target column whose levels are compared within each problem configuration (here, "Target").
- performance is the name of the column containing one or more observations of the response (or performance) variable associated to a problem configuration and a target level. It can be a string or a vector of strings, in which case the ranking process will be done independently for each of the performance columns indicated in the vector. This feature is used for composing videos showing the evolution of the performance at several time instants.
- pairing.col is the name of the column that indicates which performance values (corresponding to the same parameter configuration but different levels of the target variable) are linked with respect to the statistical tests. This value only makes sense if we set paired = TRUE; otherwise, it will be ignored.
- test is the statistical test (defaults to Wilcoxon) to be used for the pairwise comparisons (paired indicates whether a paired version of the test will be used or not). "custom" means a custom test will be applied, implemented by the function passed in the fun argument (which otherwise will be ignored).
- fun is a function implementing a custom statistical test for two samples that should return a tagged list with a p.values field, as occurs with pairwise.t.test and paired.wilcox.test, containing a matrix of p-values whose rows and columns have proper names.
- correction is the p-value adjustment method for multiple pairwise comparisons (defaults to Holm's procedure). It must be one of those natively implemented by R (ignored when test = "tukeyHSD").
- alpha is the significance threshold for the statistical test.
- maximize indicates whether the larger the performance, the better (default) or vice versa.
- ncores is the number of physical cores to be used in the computations. Parallelization is achieved through the function parLapply of the parallel package.
- paired indicates whether the multiple pairwise comparison tests should be paired or not (defaults to FALSE). When set to TRUE, the repeated performance observations are taken to be linked according to the values of the pairing.col column. For a given combination of params, the multiple observations associated to distinct levels of the target variable but sharing the same value of pairing.col are linked, as shown in column Fold of Table 1. Hence, all the pairwise comparisons between any two levels of the target variable will be paired.

The above function receives a data frame, chunks it according to all possible combinations of the values of params, and compares the levels of the target variable within each group by applying a statistical test to each binary comparison with the selected p-value adjustment method. When running in parallel, each processor manages a subset of all the chunks generated, where a chunk is composed

of all the rows corresponding to a problem configuration. Therefore the input data are distributed among the processors by subsets of consecutive rows.

The output of the function is an object belonging to the S3 class ‘SRCS’ and extending class ‘`data.frame`’, which is actually a data frame containing all the `params` and `target` columns, a new rank column, two more columns with the average and the standard deviation of the performance for each problem combination, and additional columns summarizing the p -values of pairwise comparisons. In case more than one performance column was passed to the function, the output data frame will not contain the average, standard deviation and p -values columns, but just one rank column for each of the performance columns of the input data. The resulting object has been given an S3 class name ‘SRCS’ so that function `plot` can be applied on it after properly implementing a specific S3 method described below.

Function `SRCScomparison` receives the ‘SRCS’ object calculated by `SRCSranks` together with a problem configuration, and summarizes the p -values of the multiple pairwise comparisons. All the data are already present in the data frame returned by `SRCSranks` but not in an easily interpretable disposition. Therefore this function basically collects the p -values and prints them on screen in a nice way, either as a p -value table or showing only the qualitative result of every statistical comparison, i.e., $>$, $=$, $<$ for a fixed α , without presenting the actual p -values. The function only works if the previous call to `SRCSranks` was done with only one performance column, because otherwise no p -values or average performances are calculated in the output data frame. The signature is the following:

```
SRCScomparison(rankdata, target, alpha = 0.05, pvalues = FALSE, ...)
```

where `rankdata` is the data frame produced by `SRCSranks`, `target` is the name of the target column in `rankdata`, `alpha` is the significance threshold, `pvalues` indicates whether p -values or qualitative results of the comparisons should be printed, and `...` is a succession of named arguments corresponding to columns of `rankdata` and their values to fully determine a problem configuration. These named arguments are used for subsetting `rankdata`. The number of rows of this subset should be equal to the number of levels of the target variable; otherwise an error is thrown.

The S3 plot method for ‘SRCS’ objects

The data frame produced by `SRCSranks` is usually passed on to the next function, which is the S3 plot method for ‘SRCS’ objects and constitutes the main component of the package:

```
plot(x, yOuter, xOuter, yInner, xInner, zInner = "rank",
      out.Y.par = list(), out.X.par = list(),
      inner.X.par = list(), inner.Y.par = list(),
      colorbar.par = list(), color.function = heat.colors, heatmaps.per.row = NULL,
      heatmaps.titles = NULL, annotation.lab = NULL, show.colorbar = TRUE,
      heat.cell.par = list(), heat.axes.par = list(), colorbar.cell.par = list(),
      colorbar.axes.par = list(), annotation.text.par = list())
```

Below we provide a brief description of all the parameters. For further details please refer to the package help pages. Notice only the first five arguments are mandatory.

- `x` is an ‘SRCS’ object usually generated by a call to `SRCSranks` but can also be directly composed by the user. This way, the user can create his own ranks and use the **SRCS** package only to plot them, as long as they are properly arranged in a data frame with class ‘SRCS’ as those generated by `SRCSranks`.
- `yOuter`, `xOuter`, `yInner`, `xInner`, `zInner` are the names of the columns that will be plotted in each of the dimensions of the plot; see Figure 1c, where the *Algorithm* plays the role of the outer Y variable. The `zInner` variable corresponds to the rank column, which is plotted using colors in the heatmaps: the higher the value of `zInner`, the better, and hence, the lighter the color assigned to it. The location of the levels both in the outer and inner variables depends on the factor levels for these variables when transforming them to factors, a conversion that takes place inside the function.
- `out.Y.par`, `out.X.par`, `inner.X.par`, `inner.Y.par` are tagged lists to customize how variable labels and level labels are displayed. Some options include hiding a label, setting the character size, color, location, orientation, whether it should be placed inside a rectangle or not, border and background color of such a rectangle, and other parameters that will be passed directly to the `text` function in the **graphics** package. Arguments `heat.cell.par`, `heat.axes.par`, `colorbar.cell.par`, `colorbar.axes.par` and `annotation.text.par` play a similar role.
- `color.function` is a function returning a vector of hexadecimal color codes of length (`maxrank` – `minrank` + 1) which will be used for displaying the heatmaps. Can be either a custom

function or one of the existing palettes such as `heat.colors`, `terrain.colors`, etc. The function will be called with one argument, namely the number of colors to be returned, (`maxrank - minrank + 1`).

- `heatmaps.per.row` is an integer indicating whether all the levels of the outer X variable are shown horizontally, or broken in two or more sub-rows.
- `show.colorbar` is a Boolean that displays or hides a colorbar used as the legend of the figure.
- `heatmaps.titles` is a vector of strings with the titles of every individual heatmap, if needed.
- `annotation.lab` is an annotation string that will be displayed on the top left corner of the plot. This is useful for labeling individual plots when composing videos.

The function relies on the `layout` function of the `graphics` package to automatically compose a suitable layout, taking into account the number of levels of each variable and the user's choices.

Functions `animatedplot` and `singleplot`

Function `animatedplot` enables composing videos from sequences of plots like Figure 3. This enables the user to visualize time as a new dimension by plotting statistical pairwise comparison results at different time moments. This can be useful, for instance, when comparing convergence speed between many algorithms about which the best solution so far has been annotated at different moments of the optimization process. The function relies on R's built-in capability to automatically compose counter-based filenames when successively generating plots to image files, and then calls ImageMagick (Still, 2005), a widely used open-source software for Windows and Linux, to join them together into a video file. A number of image formats can be used for the images generated prior to composing the video. Note that those files are not automatically deleted; the user will have to do it by himself. It is necessary that the user has previously installed ImageMagick.

The function signature is the following:

```
animatedplot(x, filename, path.to.converter,
             yOuter, xOuter, yInner, xInner, zInner,
             width = 800, height = 800, res = 100, pointsize = 16,
             delay = 30, type = c("png", "jpeg", "bmp", "tiff"), quality = 75,
             compression = c("none", "rle", "lzw", "jpeg", "zip"),
             annotations = NULL, ...)
```

In this case, `zInner` should be a vector with the names of the columns in `x` containing the performance measures to be plotted successively. The video will have as many frames as elements there are in `zInner`. The argument `path.to.converter` is a string with the full path of the converter program that comes with ImageMagick, e.g., "C:/Program Files/ImageMagick-<version>/convert.exe". The rest of the arguments allow setting the name of the output video file (including the file format) and configure the size, resolution, delay between the frames (in 1/100th of a second), percentage of quality and type of compression. The function also gives the possibility to set an independent annotation in the upper-left corner of each frame by passing a vector of strings, where each element is the annotation of the corresponding frame of the sequence. The `...` argument accepts any subset of the optional arguments to be passed to the S3 plot method for 'SRCS' objects that plots every frame.

Function `singleplot` creates a single heatmap focused on the problem configuration defined by the user. It has the following signature:

```
singleplot(x, yInner, xInner, zInner = "rank", color.function = heat.colors,
           labels.par = list(), colorbar.par = list(), heat.axes.par = list(),
           colorbar.axes.par = list(), haxis = TRUE, vaxis = TRUE, title = "",
           show.colorbar = TRUE, ...)
```

The parameters are similar to those already described. The `...` argument in this case stands for a succession of named arguments (as many as necessary) that will be used to subset the data argument. From that subset, the values of the `zInner` column will be depicted in a single heatmap, in the locations indicated by `yInner` and `xInner` columns. If any pair of values of columns (`xInner, yInner`) is found more than once after subsetting, an error is thrown.

Case studies

In this section, two examples representative of those typically faced by potential users will be presented, with the purpose of illustrating the package capabilities and ease of use. None of them is aimed at finding the best algorithm for the posed problems, but at showing the applicability of the package

in different areas of knowledge when analyzing experimental results. Therefore, the details of the experimental framework in each example (such as the tuning of the parameters, the concrete algorithms and datasets tested and so on) are not relevant for the aforementioned purpose².

The first example analyses the results of dynamic optimization algorithms while the second deals with typical machine learning problems where several classification algorithms are compared under different settings or problem configurations. Note that the package is oriented at the analysis of experimental results, which do not necessarily come from R code or even from a computer program. In our case, the techniques assessed in the first example have been implemented in Java and are not available in R.

Application to dynamic optimization problems

DOPs ([Branke, 2001](#)) are a variant of classical optimization problems in which the function being optimized has time-dependent properties, i.e., changes along the time during the execution of the optimization algorithm itself. The changes may affect the fitness function, the constraints, the number of variables of the function or their domain, etc. DOPs have attracted increasing attention due to their closeness to many real-world changing problems, as explained in the aforementioned work.

Many algorithms have been proposed to solve DOPs as explained in [Cruz et al. \(2011\)](#), most of them based on Evolutionary Algorithms and other population-based metaheuristics. Here we will reproduce one of the plots published in [del Amo et al. \(2012\)](#) representing a broad DOP algorithm comparison, including the R code necessary to obtain them in a straightforward way. The numerical results represented in the plots have been included as a data frame object called MPB in the **SRCS** package. Details on the algorithms compared can be found in the aforementioned work. Below we briefly comment on the meaning of the parameters involved in a problem configuration, the performance measure collected in the file and the fitness function we are optimizing.

In a DOP, the fitness function changes along the time. Several aspects modulate how this happens, such as the time passed between two consecutive changes, or the severity of the change (how different the function is with respect to the previous version). None of these parameters is known in advance by any algorithm. The third parameter known to affect the performance is the dimension of the function, which is user-configurable but remains invariant during the execution.

The fitness function employed, known as the Moving Peaks Benchmark (MPB, [Branke 1999](#); see Figure 2a), was specifically designed as a DOP benchmark. The problem consists in maximizing a continuous n -dimensional function that results from the superposition of m peaks, each one characterized by its own height ($h^j \in \mathbb{R}$), width ($w^j \in \mathbb{R}$) and location of its centre ($p^j \in \mathbb{R}^n$):

$$\text{MPB}(\mathbf{x}) = \max_j \left\{ h^j - w^j \sqrt{\sum_{i=1}^n (x_i - p_i^j)^2} \right\}, \quad j = 1, \dots, m. \quad (1)$$

The global optimum is the centre of the peak with the highest parameter h^j . To make this function dynamic, the parameters of the peaks are initialized to some prefixed values, but then change every ω function evaluations according to certain laws (refer to [Branke 1999; del Amo et al. 2012](#) for details). The values of the parameters used in the experiments are summarized in Figure 2b. The first three rows can vary to define every single problem configuration, while the rest are fixed for all problem configurations.

A lot of different performance measures have been proposed for DOPs as mentioned in [Cruz et al. \(2011\)](#). Here we employ the most widely accepted one, namely the *offline error* ([del Amo et al., 2012](#)):

$$e_{off} = \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{1}{N_e(i)} \sum_{j=1}^{N_e(i)} (f_i^* - f_{ij}), \quad (2)$$

where N_c is the total number of changes in the environment during the execution, $N_e(i)$ is the total number of evaluations allowed in the i -th change, f_i^* is the optimum value of the i -th change, and f_{ij} is the best value found by the algorithm since the beginning of the i -th change up to the j -th evaluation. It is defined this way to favor those algorithms which converge to good solutions very quickly after each change. Furthermore, since changes take place at a fixed rate in our experiments ($N_e(i_1) = N_e(i_2) = \dots = N_e$), the formula simplifies to

$$e_{off} = \frac{1}{N_c N_e} \sum_{i=1}^{N_c} \sum_{j=1}^{N_e} (f_i^* - f_{ij}). \quad (3)$$

²This section has been expanded in the package vignette with a third case study.

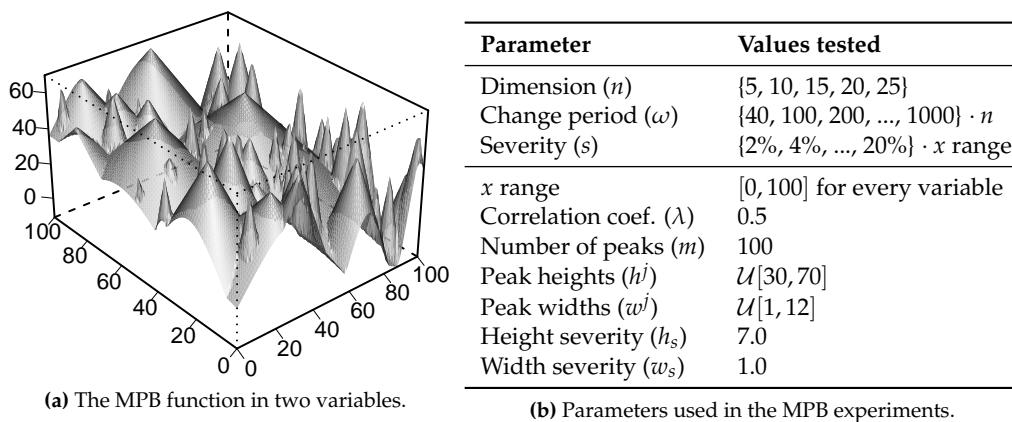


Figure 2: Experimental setup in the MPB.

As this is a maximization problem, $f_i^* - f_{ij}$ remains always positive. For each problem configuration {change period, severity, dimension}, every algorithm is run $K = 50$ independent times, thus collecting 50 offline error measurements which will be used to perform the pairwise statistical tests. In each run, the fitness function changes $N_c = 100$ times along the execution, at regular time intervals.

The R code used to plot the results is the following:

```
> library(SRCS)
> str(MPB)
'data.frame': 220000 obs. of 5 variables:
 $ Algorithm: Factor w/ 8 levels "reactive-cs",...: 7 7 7 7 7 7 7 7 7 7 ...
 $ Dim      : int 5 5 5 5 5 5 5 5 5 ...
 $ CF       : int 40 40 40 40 40 40 40 40 40 ...
 $ Severity : int 2 2 2 2 2 2 2 2 2 ...
 $ OffError : num 21.8 19.5 16.7 18.3 22.6 ...

> head(MPB)
  Algorithm Dim CF Severity OffError
1 agents    5 40          2 21.81232
2 agents    5 40          2 19.53094
3 agents    5 40          2 16.73922
4 agents    5 40          2 18.32204
5 agents    5 40          2 22.61913
6 agents    5 40          2 19.17223
```

The above output is the first part of the 50 performance observations of algorithm *agents* in the problem configuration defined by $Dim = 5$, $CF = 40$, $Severity = 2$. Note that the tests should be non-paired as there is no relation between the runs of the algorithms.

```
> ranks <- SRCSranks(MPB, params = c("Dim", "CF", "Severity"),
+   target = "Algorithm", performance = "OffError", maximize = FALSE, ncores = 2)
> head(ranks);
  Algorithm Dim CF Severity rank  mean   sd agents.pval independent-cs.pval
1      agents    5 40          2    0 18.16 2.698        NA 8.993e-12
2 independent-cs 5 40          2   -5 22.86 1.850 8.993e-12        NA
3           mqso 5 40          2    3 16.86 1.824 1.346e-01 1.324e-15
4      mqso-both 5 40          2    4 16.58 2.094 2.646e-02 5.349e-15
5     mqso-change 5 40          2    4 16.58 2.094 2.646e-02 5.349e-15
6     mqso-rand 5 40          2    4 16.58 2.094 2.646e-02 5.349e-15
  reactive-cs.pval mqso.pval mqso-both.pval mqso-change.pval mqso-rand.pval
1 8.993e-12 1.346e-01 2.646e-02 2.646e-02 2.646e-02
2 1.000e+00 1.324e-15 5.349e-15 5.349e-15 5.349e-15
3 1.324e-15        NA 1.000e+00 1.000e+00 1.000e+00
4 5.349e-15 1.000e+00          NA 1.000e+00 1.000e+00
5 5.349e-15 1.000e+00 1.000e+00          NA 1.000e+00
6 5.349e-15 1.000e+00 1.000e+00 1.000e+00          NA
  soriga.pval
1 2.812e-12
```

```

2  1.466e-01
3  4.419e-15
4  5.349e-15
5  5.349e-15
6  5.349e-15

```

Note that the K rows per problem configuration present in the input data have now collapsed into one row per problem configuration, containing the average performance of the K observations, their standard deviation, the rank achieved by that configuration, and the corrected p -values of the multiple pairwise statistical tests against the other levels of the target variable in the same problem configuration. We can now plot these results with the following code:

```

> plot(ranks, yOuter = "Algorithm", xOuter = "Dim", yInner = "CF", xInner = "Severity",
+ ## all the remaining arguments are optional, for customizing the appearance
+   inner.Y.par = list(levels.at = c("40", "200", "400", "600", "800", "1000"),
+     lab = "Change\n period", levels.loc = "left"),
+   out.Y.par = list(levels.lab.textpar = list(cex = 1, col = "white"),
+     levels.bg = "black", levels.border = "white"),
+   out.X.par = list/lab = "Dimension", levels.bg = "gray"),
+   colorbar.par = list(levels.at = c("-7", "0", "7")),
+   colorbar.axes.par = list(cex.axis = 0.8),
+   show.colorbar = TRUE
+ )

```

The results are depicted in Figure 3, which should be interpreted as follows: for a given value of *Dimension*, one should look at the whole column of heatmaps vertically to know how the algorithms behave for that dimension. The arrangement of the cells within the heatmaps is analogous to Figure 1c. From the figure, we can see that, for instance, *soriga* only behaves well (although it is not the best one) when the change period is short, and this is enhanced when increasing the dimensionality of the problem. This amounts to say that *soriga* is specially good at detecting a change and recovering from it by quickly discovering promising regions after the change, although it is not so good at exploiting these regions (it is beaten by other algorithms when the change period gets larger). On the other hand, *agents* also improves its behaviour when the dimensionality grows above 15 (otherwise, *mqso-rand* dominates the rest when considering a 5- or 10-variable fitness function), but also when severity increases, becoming the best algorithm in those cases (right part of the heatmaps).

We could ask for a single heatmap as well, defined by some values of the outer Y and X variables, for instance *Algorithm* = *soriga* and *Dimension* = 25, using the following call:

```

> singleplot(x = ranks, zInner = "rank", yInner = "CF",
+ xInner = "Severity", colorbar.par = list(levels.at = c("-7", "0", "7")),
+ labels.par = list(ylab = "Change period"), Algorithm = "soriga", Dim = "25")

```

The output is shown in Figure 4. To obtain a qualitative performance comparison for a given problem configuration, for instance when *Change period* = 40, *Dimension* = 25, *Severity* = 20, we can use the following call:

```
SRCScomparison(ranks, "Algorithm", CF = 40, Dim = 25, Severity = 20, pvalues = FALSE)
```

which will produce the following matrix object as a result:

	agents	indep-cs	mqso	mqso-both	mqso-change	mqso-rand	reactive-cs	soriga
agents	NA	"<"	"<"	"<"	"<"	"<"	"<"	"<"
indep-cs	">"	NA	"<"	"<"	"<"	"<"	"="	">"
mqso	">"	">"	NA	">"	"="	">"	">"	">"
mqso-both	">"	">"	"<"	NA	"<"	"="	">"	">"
mqso-change	">"	">"	"="	">"	NA	">"	">"	">"
mqso-rand	">"	">"	"<"	"="	"<"	NA	">"	">"
reactive-cs	">"	"="	"<"	"<"	"<"	"<"	NA	">"
soriga	">"	"<"	"<"	"<"	"<"	"<"	"<"	NA

Composing a video file to visualize convergence In the vignette associated to this package it is shown how to create a video to visualize an extra temporal component in the results. The data were collected from the aforementioned executions of dynamic optimization algorithms over the MPB problem, annotating the offline error at the instant before every change. Both the data, the R script and the resulting video can be downloaded from the first author's personal home page³, which is also given as URL in the package description. Please refer to the vignette for further details.

³<http://decsai.ugr.es/~pjvi/r-packages.html>

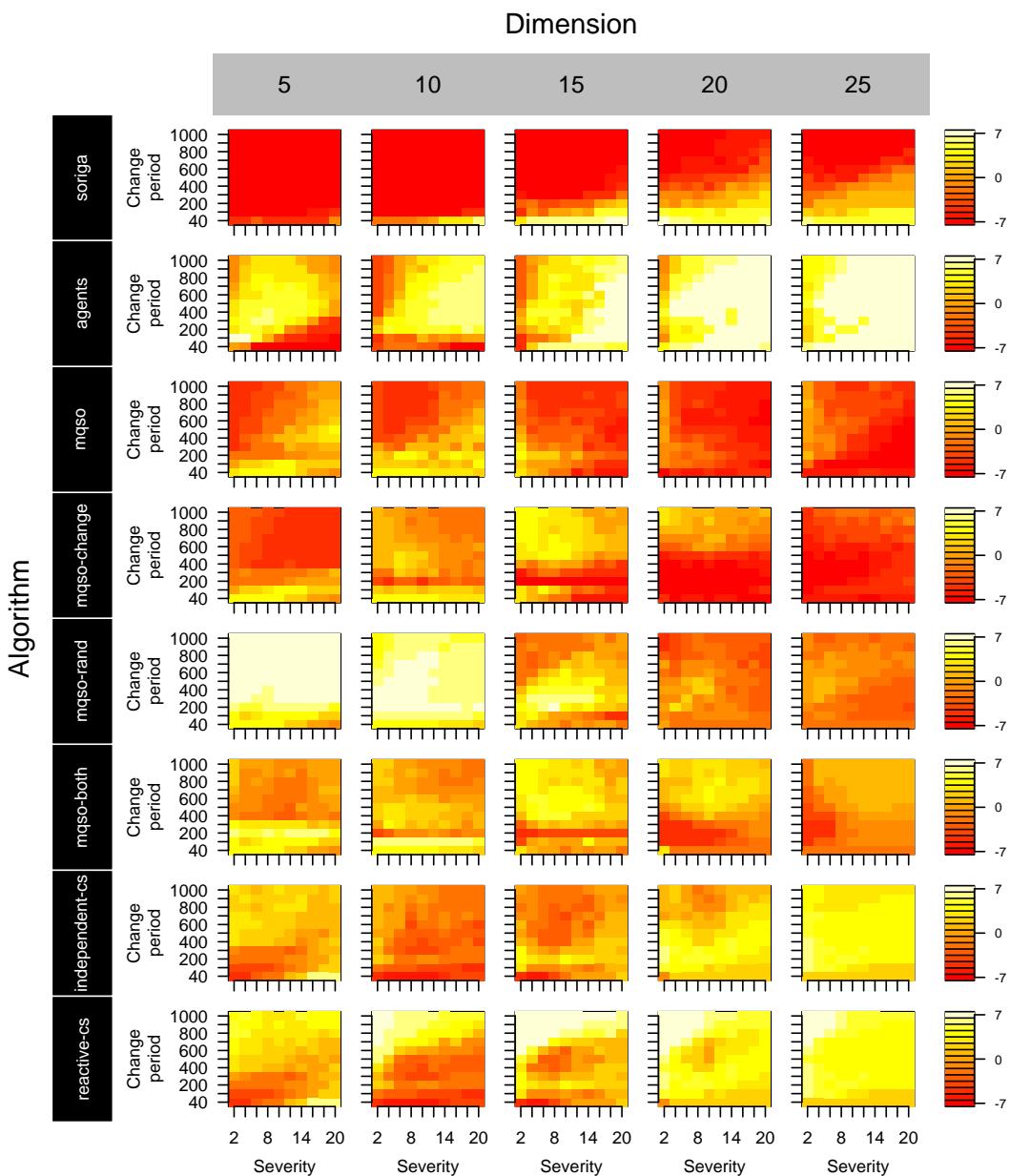


Figure 3: Results of several dynamic optimization algorithms on the MPB. This plot mirrors Figure 5 of [del Amo et al. \(2012\)](#).

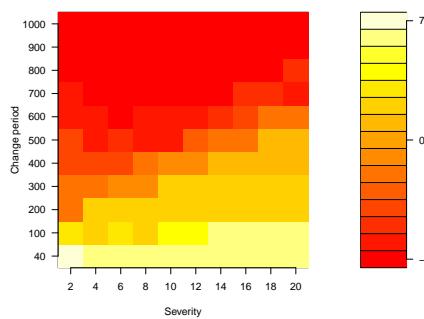


Figure 4: A single heatmap generated by `singleplot` for problem configurations where *Algorithm* = *soriga*, *Dimension* = 25.

Dataset	#EX	#AT	#CL	Dataset	#EX	#AT	#CL
automobile	159	25 (15/10)	6	glass	214	9 (9/0)	7
balance	625	4 (4/0)	3	ionosphere	351	33 (33/0)	2
cleveland	297	13 (13/0)	5	pima	768	8 (8/0)	2
ecoli	336	7 (7/0)	8	vehicle	846	18 (18/0)	4

Table 2: Description of the classification datasets.

Application to machine learning: Noisy datasets

The second case study is a machine learning experiment involving six supervised classification algorithms tested over a number of noisy datasets with several types of noise and noise severities. The aim is to assess how noise affects each of the classifiers and whether the behaviour changes with those parameters. The algorithms tested were the following: (a) the *k*-Nearest-Neighbours classifier with three different values of *k* (*k* = 1, *k* = 3 and *k* = 5), (b) *Repeated Incremental Pruning to Produce Error Reduction* (RIPPER), (c) a *Support Vector Machine* (SVM), and (d) the C4.5 tree-based rule induction classifier. The reader may refer to [Cohen \(1995\)](#); [Bishop \(2006\)](#) for a review of all these algorithms.

R implementations have been used in all cases. We coded the *k*-NN to incorporate the HVDM distance ([Wilson and Martinez, 1997](#)) for heterogeneous (continuous and nominal) attributes. The SVM was taken from the `e1071` package, version 1.6-4 ([Meyer et al., 2014](#)), which contains a wrapper for the libsvm C++ library ([Chang and Lin, 2001](#)). Algorithms C4.5 and RIPPER were taken from [RWeka](#), version 0.4-24 ([Hornik et al., 2009](#)), which offers an R interface to the Weka framework ([Witten and Frank, 2005](#)). The datasets employed in the experiment (Table 2) have been taken from the UCI repository ([Lichman, 2013](#)), and are among the most widely used in machine learning studies. For each dataset, the number of classes (#CL), the number of examples (#EX) and the number of attributes (#AT), along with the number of numeric and nominal attributes are presented.

In the literature, two types of noise can be distinguished in a dataset ([Zhu and Wu, 2004](#)): (i) *class noise* (examples labeled with a class distinct from the true one) and *attribute noise* (that usually refers to erroneous attribute values). The amount and type of noise present in real-world datasets are usually unknown. In order to control the amount of noise in the datasets and check how it affects the classifiers, noise is introduced into each dataset in a controlled manner. Four different noise schemes have been used in order to introduce a noise level $x\%$ into each dataset ([Zhu and Wu, 2004](#)):

1. Introduction of class noise.

- *Random class noise* (CLA_RAND). $x\%$ of the examples are randomly selected and turned corrupt. The class labels of these examples are randomly replaced by another one from the M classes.
- *Pairwise class noise* (CLA_PAIR). Let X be the majority class and Y the second majority class. An example with the label X has a probability of $x/100$ of being incorrectly labeled as Y .

2. Introduction of attribute noise.

- *Random attribute noise* (ATT_RAND). $x\%$ of the values of each attribute in the dataset are randomly selected and turned corrupt. To corrupt each attribute AT_i , $x\%$ of the examples in the dataset are chosen, and their AT_i value is replaced by a random value from the

domain \mathbb{D}_i of the attribute AT_i . An uniform distribution is used for both numerical and nominal attributes.

- *Gaussian attribute noise* (ATT_GAUS). This scheme is similar to the uniform attribute noise, but in this case, the AT_i values are corrupted, adding a random value to them following a Gaussian distribution of $mean = 0$ and $standard deviation = (max - min)/5$, being max and min the limits of the attribute domain (\mathbb{D}_i). Nominal attributes are treated as in the case of the uniform attribute noise.

The four noise schemes have been considered independently and for each type of noise, the noise levels ranging from $x = 0\%$ (base datasets) to $x = 50\%$, by increments of 5%, have been studied. The accuracy estimation of the classifiers in a dataset is obtained by means of a stratified 5-fold cross-validation, which is the standard in the field. For obtaining multiple observations, the cross-validation procedure was repeated five times, thus obtaining $K = 25$ performance (accuracy rate) values for each algorithm in each problem configuration, defined by {dataset, noise type, noise severity}. These values will later be used in pairwise statistical comparisons. For a given problem configuration, exactly the same partitions of a dataset were used with all the algorithms, and for that reason, the observations are paired (recall Table 1).

Performing pairwise comparisons separating the results by dataset can be particularly useful in certain machine learning works which include a very small number of datasets. In those studies, the conventional approach consisting in summarizing the performance of an algorithm over a dataset with a single value and applying post-hoc pairwise comparisons between the algorithms with these summaries does not work, because each of the samples being compared has too few elements (due to the reduced number of datasets) to apply a statistical test. In such cases, the SRCS approach would be more suitable and would yield a reliable comparison for each dataset separately.

The R script which runs the algorithms over the datasets mentioned and generates the results to be analyzed can be downloaded from the first author's home page mentioned before, together with the datasets. The performance results obtained are already included in the package, to save time. When **SRCS** is loaded, a data frame object called **ML1** containing the results of this experiment is created:

```
> str(ML1)

'data.frame': 52800 obs. of 6 variables:
 $ Algorithm : Factor w/ 6 levels "1-NN","3-NN",...: 1 1 1 1 1 1 1 1 1 ...
 $ Dataset   : Factor w/ 8 levels "automobile","balance",...: 1 1 1 1 1 1 1 1 ...
 $ Noise type: Factor w/ 4 levels "ATT_GAUS","ATT_RAND",...: 1 1 1 1 1 1 1 1 ...
 $ Noise ratio: num  0 0 0 0 0 0 0 0 0 ...
 $ Fold      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Performance: num  77.4 54.5 86.7 81.2 84.8 ...
```

```
> head(ML1)

  Algorithm Dataset Noise type Noise ratio Fold Performance
1     1-NN automobile    ATT_GAUS       0     1    77.41935
2     1-NN automobile    ATT_GAUS       0     2    54.54545
3     1-NN automobile    ATT_GAUS       0     3    86.66667
4     1-NN automobile    ATT_GAUS       0     4    81.25000
5     1-NN automobile    ATT_GAUS       0     5    84.84848
6     1-NN automobile    ATT_GAUS       0     6    84.37500
```

The R code to compute and plot the ranks with **SRCS** is the following.

```
> ranks <- SRCSranks(ML1, params = c("Dataset", "Noise type", "Noise ratio"),
+   target = "Algorithm", performance = "Performance", pairing.col = "Fold",
+   maximize = TRUE, ncores = 1, paired = TRUE)
> plot(ranks, yOuter = "Dataset", xOuter = "Algorithm", yInner =
+   "Noise type", xInner = "Noise ratio", zInner = "rank", out.X.par =
+   list(levels.lab.textpar = list(col = "white"), levels.bg = "black",
+   levels.border = "white"), out.Y.par = list(levels.bg = "gray"),
+   colorbar.axes.par = list(cex.axis = 0.8), show.colorbar = TRUE)
```

The results are summarized in Figure 5. This figure shows that higher values of k in the k -NN classifier make the model perform better than lower values of k (with the exception of the automobile dataset, where the opposite happens). Thus, 5-NN generally is better than 3-NN, and 3-NN is better than 1-NN for the different datasets considered. This fact is in accordance with the work of Kononenko and Kukar (2007) that claimed that the value of k in k -NN determines a higher or lower sensitivity to

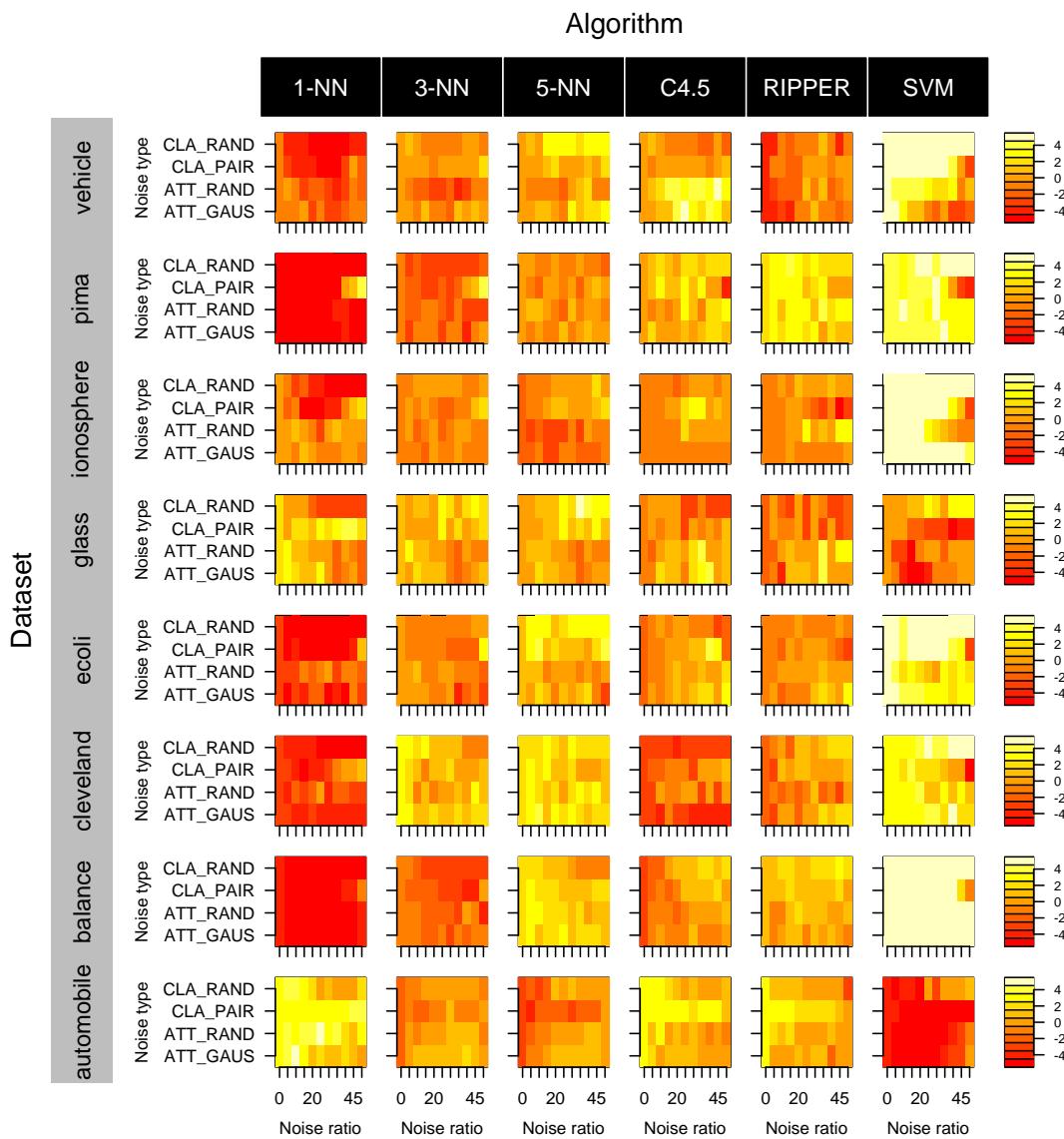


Figure 5: Results of six supervised classification algorithms on eight noisy datasets.

noise. SVM presents variable results, depending on the dataset analyzed. For some of them, such as automobile or glass, the results are predominantly in red colours. Other datasets, such as vehicle or cleveland, show that SVM can work relatively well when the noise level is low, but its performance is deteriorated when the noise level increases. These facts agree with the results of the literature that state that SVM is usually noise-sensitive, particularly with high noise levels (Nettleton et al., 2010). However, for other datasets considered, such as balance, SVM obtains good results. Finally, one must note that both C4.5 and RIPPER, which are considered robust to noise (Zhu and Wu, 2004), obtain intermediate results in the eight datasets considered.

Conclusions and further work

In this paper we have introduced an R package called **SRCS**, aimed at testing and plotting the results of multiple pairwise statistical comparisons in different configurations of a problem, defined by several parameters. The package implements a previously published visualization technique to summarize the output of many comparisons at the same time by using a careful spatial arrangement to display the result for each problem configuration defined by a parameter combination. As we have explained, our code gives the user full control over all the graphical options so as to fully customize the plot. Furthermore, we have taken this approach a step further by considering the time as another parameter. This turns static images into videos to take into account this new dimension, but allows constructing convergence plots for all problem configurations simultaneously. It should be noticed that, while videos have been conceived to represent convergence, they can also be used with another variable in

any setting in which it makes sense to watch the evolution of statistical results.

We have successfully applied our package to two very different problems, namely dynamic optimization problems and machine learning problems. The latter represents a novel use of SRCS that has proven very helpful for comparing classification algorithms under different circumstances of noise type, noise levels, imbalance ratios and shape of the data. The SRCS approach enables visualizing the results of a number of algorithms at a glance, which in turns leads to an easier interpretation and may also reveal trends relating different problem configurations that otherwise would be harder to uncover, such as the configurations where each algorithm (or family of algorithms) performs best.

An interesting improvement would consist in adding interactivity to the plots. The user could manually re-arrange the plots or add/remove problem parameters and/or target levels, and visually check whether such modifications cause a strong change in the results or not as the plot would be automatically updated.

Acknowledgments

This work is supported by projects TIN2011-27696-C02-01 from the Spanish Ministry of Science and Innovation, P11-TIC-8001 from the Andalusian Government, and FEDER funds. P. J. Villacorta acknowledges support from an FPU scholarship from the Spanish Ministry of Education, and J. A. Sáez, from EC under FP7, Coordination and Support Action, Grant Agreement Number 316097, ENGINE European Research Centre of Network Intelligence for Innovation Enhancement (<http://engine.pwr.wroc.pl/>). We are thankful to Dr. Antonio D. Masegosa from the University of Deusto, Spain, for suggesting the use of video sequences for visualizing the convergence of dynamic optimization algorithms, which has proven very successful for this purpose.

Bibliography

- D. A. Armstrong. *factorplot: Improving presentation of simple contrasts in generalized linear models*. *The R Journal*, 5(2):4–15, 2013. [p90]
- T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer-Verlag, Berlin Heidelberg, 2010. [p89]
- C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. [p100]
- J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1875–1882, 1999. [p96]
- J. Branke. *Evolutionary Optimization in Dynamic Environments*. Genetic Algorithms and Evolutionary Computation (Book 3). Kluwer Academic Publishers, MA, USA, 2001. [p96]
- M. Burda. *paircompviz: An R Package for Visualization of Multiple Pairwise Comparison Test Results*, 2014. URL <https://www.bioconductor.org/packages/release/bioc/html/paircompviz.html>. R package version 1.0.0. [p89]
- C.-C. Chang and C.-J. Lin. *LIBSVM: A Library for Support Vector Machines*, 2001. URL <http://www.csie.ntu.edu.tw/~cjlin/libsvm>. [p100]
- M. Coffin and M. J. Saltzman. Statistical analysis of computational tests of algorithms and heuristics. *INFORMS Journal on Computing*, 12(1):24–44, 2000. [p89]
- W. W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann Publishers, 1995. [p100]
- C. Cruz, J. R. González, and D. A. Pelta. Optimization in dynamic environments: A survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448, 2011. [p96]
- I. G. del Amo and D. A. Pelta. SRCS: A technique for comparing multiple algorithms under several factors in dynamic optimization problems. In E. Alba, A. Nakib, and P. Siarry, editors, *Metaheuristics for Dynamic Optimization*, volume 433 of *Studies in Computational Intelligence*, pages 61–77. Springer-Verlag, Berlin Heidelberg, 2013. [p90, 91]
- I. G. del Amo, D. A. Pelta, J. R. González, and A. D. Masegosa. An algorithm comparison for dynamic optimization problems. *Applied Soft Computing*, 12(10):3176–3192, 2012. [p90, 96, 99]

- J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. [p⁸⁹]
- J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011. [p⁸⁹]
- M. J. Eugster, F. Leisch, and C. Strobl. (Psycho-)analysis of benchmark experiments: A formal framework for investigating the relationship between data sets and learning algorithms. *Computational Statistics & Data Analysis*, 71:986–1000, 2014. [p⁸⁹]
- S. García, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009. [p⁸⁹]
- S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010. [p⁸⁹]
- M. Graczyk, T. Lasota, Z. Telec, and B. Trawiński. Nonparametric statistical analysis of machine learning algorithms for regression problems. In R. Setchi, I. Jordanov, R. Howlett, and L. Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, volume 6276 of *Lecture Notes in Computer Science*, pages 111–120. Springer-Verlag, Berlin Heidelberg, 2010. [p⁸⁹]
- K. Hornik, C. Buchta, and A. Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24:225–232, 2009. [p¹⁰⁰]
- IBM Corp. *IBM SPSS Statistics for Windows, Version 21*. Armonk, NY, 2012. URL <http://www-01.ibm.com/software/analytics/spss/products/statistics/>. [p⁸⁹]
- I. Kononenko and M. Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, 2007. [p¹⁰¹]
- M. Lichman. UCI Machine Learning Repository, 2013. URL <http://archive.ics.uci.edu/ml>. [p¹⁰⁰]
- D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics (e1071)*, TU Wien, 2014. URL <https://CRAN.R-project.org/package=e1071>. R package version 1.6-4. [p¹⁰⁰]
- D. Nettleton, A. Orriols-Puig, and A. Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33:275–306, 2010. [p¹⁰²]
- D. Shilane, J. Martikainen, S. Dudoit, and S. J. Ovaska. A general framework for statistical performance comparison of evolutionary computation algorithms. *Information Sciences*, 178(14):2870–2879, 2008. [p⁸⁹]
- M. Still. *The Definitive Guide to ImageMagick*. Apress, NY, 2005. [p⁹⁵]
- P. J. Villacorta. *SRCS: Statistical Ranking Color Scheme for Multiple Pairwise Comparisons*, 2015. URL <https://CRAN.R-project.org/package=SRCS>. R package version 1.1. [p⁹⁰]
- D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997. [p¹⁰⁰]
- I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005. [p¹⁰⁰]
- X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22: 177–210, 2004. [p^{100, 102}]

Pablo J. Villacorta
Department of Computer Science and Artificial Intelligence,
University of Granada,
ETSIIT, C/Periodista Daniel Saucedo Aranda s/n, 18071 Granada, Spain
pjvi@decsai.ugr.es

José A. Sáez
ENGINE Centre,
Wrocław University of Technology,
Wybrzeże Wyspińskiego 27, 50-370 Wrocław, Poland
jose.saezmunoz@pwr.edu.pl

An R Package for the Panel Approach Method for Program Evaluation: pampe

by Ainhoa Vega-Bayo

Abstract The **pampe** package for R implements the panel data approach method for program evaluation designed to estimate the causal effects of political interventions or treatments. This procedure exploits the dependence among cross-sectional units to construct a counterfactual of the treated unit(s), and it is an appropriate method for research events that occur at an aggregate level like countries or regions and that affect only one or a small number of units. The implementation of the **pampe** package is illustrated using data from Hong Kong and 24 other units, by examining the economic impact of the political and economic integration of Hong Kong with mainland China in 1997 and 2004 respectively.

An introduction to the panel data approach and program evaluation methods

Program evaluation methodologies have long been used by social scientists to measure the effect of different economic or political interventions (treatments). The problem is, of course, that you cannot observe the outcome both under the intervention and in the absence of the intervention simultaneously, hence the need for program evaluation methods. Traditionally, comparative case studies have been the preferred method by researchers in order to compare units affected by a treatment or event (dubbed the treatment group) to one or more units not affected by this intervention (the control group). The idea is to use the outcome of the control group to obtain an approximation of what would have been the outcome of the treated group had it not been treated. In more recent years, synthetic control methods ([Abadie and Gardeazabal, 2003](#); [Abadie et al., 2010](#)) have addressed these issues by introducing a data-driven procedure for selecting the control group. However, the synthetic control methods are not without shortcomings: since the synthetic control is calculated as a convex combination of the units in the donor pool, and thus it does not allow for extrapolation, it might be that a suitable synthetic control for our treated unit does not exist. Furthermore, the synthetic control is designed to be used with explanatory variables or covariates that help explain the variance in the outcome variable. For the cases when the researcher finds that extrapolation is needed to obtain a suitable comparison for the treated unit, or when the covariates available do not properly explain the outcome on which the effect of the treatment is intended to be measured, he or she might prefer to use the panel data approach for program evaluation by [Hsiao et al. \(2012\)](#). The panel data approach for constructing the counterfactual of the unit subjected to the intervention is to use other units that are not subject to the treatment to predict what would have happened to the treated unit had it not been subject to the policy intervention. The basic idea behind this approach is to rely on the correlations among cross-sectional units. They attribute the cross-sectional dependence to the presence of common factors that drive all the relevant cross-sectional units.

As such, the aim of this article is to present the package **pampe** that implements the panel data approach for program evaluation procedures in R, which is available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=pampe>. The main function in the package is `pampe()`, which computes the counterfactual for the treated unit using the modeling strategy outlined by [Hsiao et al. \(2012\)](#). The function includes an option to obtain placebo tests. There is an additional function `robustness()`, which conducts a leave-one-out robustness on the results. The data example is also from [Hsiao et al. \(2012\)](#), which introduced the panel data approach methodology to study the effect of the political and economic integration of Hong Kong with mainland China using other countries geographically and economically close to Hong Kong as possible controls.

The article is organized as follows. The following section is a brief overview of the panel data approach as defined by [Hsiao et al. \(2012\)](#). The main section of the paper, titled [Implementing pampe in R](#), demonstrates the implementation of this method and the use of the **pampe** package with an example, including how to perform inference and robustness checks.

The panel data approach method for program evaluation

The panel data approach for program evaluation exploits the dependence among cross-sectional units to construct a counterfactual of the treated unit(s), to estimate how the affected unit would have developed in the absence of an intervention. The estimated effect of the policy intervention is therefore simply the difference between the actual observed outcome of the treated unit and this estimated

counterfactual. Hsiao et al. (2012) provide a thorough description of the methodology. Here the focus is on how this method is implemented in the **pampe** package and thus only a brief overview of the procedure is provided.

Let us consider $J + 1$ units over $t = 1, \dots, T, \dots, T'$ periods. Without loss of generality, only the first unit is affected uninterruptedly by an intervention in period T during periods $T, T + 1, \dots, T'$, after an initial pre-intervention period $1, \dots, T - 1$. The left over J units are the controls that form the so-called “donor pool”, and they are not affected by the intervention. Let Y_{jt} denote the outcome variable – the variable for which the intervention effect is being measured – of unit j at period t . Y_{jt}^1 and Y_{jt}^0 denote the outcome of unit j at time t under treatment and in the absence of treatment respectively. We usually do not simultaneously observe both Y_{jt}^1 and Y_{jt}^0 , but instead we observe Y_{jt} , which can be written as

$$Y_{jt} = d_{jt} Y_{jt}^1 + (1 - d_{jt}) Y_{jt}^0;$$

where d_{jt} is a dummy variable that takes value 1 if unit j is under treatment at time t , and value 0 otherwise (Rubin, 1974). In this case and without loss of generality, only the first unit is under intervention, so we have that

$$d_{jt} = \begin{cases} 1 & \text{if } j = 1 \text{ and } t \geq T, \\ 0 & \text{otherwise.} \end{cases}$$

The treatment or intervention effect for the treated unit can therefore be expressed as

$$\alpha_{1t} = Y_{1t}^1 - Y_{1t}^0.$$

Of course, we do not observe Y_{1t}^0 for $t \geq T$. Thus, the goal of the panel data approach is to obtain an estimate for the effect of the intervention, $\hat{\alpha}_{1t}$, during the post-treatment period T, \dots, T' by attempting to replicate the economy of the treated unit in the pre-intervention period $1, \dots, T - 1$; that is, by obtaining an estimate of the outcome variable under no treatment Y_{1t}^0 . It is assumed that there is no treatment interference between units, i.e., the outcome of the untreated units is not affected by the treatment of the treated unit.

The panel data approach developed by Hsiao et al. (2012) attempts to predict Y_{1t}^0 for $t \geq T$ and therefore to estimate the treatment effect α_{1t} by exploiting the dependency among cross-sectional units in the donor pool and the treated unit, using the following modeling strategy: use R^2 (or likelihood values) in order to select the best OLS estimator for Y_{1t}^0 using j out of the J units in the donor pool, denoted by $M(j)^*$ for $j = 1, \dots, J$; then choose $M(m)^*$ from $M(1)^*, \dots, M(J)^*$ in terms of a model selection criterion, like AICc, AIC or BIC.¹

This strategy is founded on the following underlying model. Hsiao et al. (2012) assume that Y_{it}^0 is generated by a dynamic factor model of the form:

$$Y_{jt}^0 = \gamma_j + \mathbf{f}_t \mathbf{b}_j + \varepsilon_{it}, \quad (1)$$

where γ_j denotes an individual-specific effect, \mathbf{f}_t is a $(1 \times K)$ vector that denotes time varying unobserved common factors, \mathbf{b}_j denotes a $(K \times 1)$ vector of constants that can vary across units, K is the number of common factors, and ε_{it} is the time varying idiosyncratic component of individual j .

Y_{1t}^0 could be predicted using the underlying model Hsiao et al. (2012) specify and the assumptions they delineate. Instead, they suggest a more practical approach, i.e., using the remaining non-intervened units in the donor pool $Y_{-1t} = (Y_{2t}, \dots, Y_{Jt})$ to predict Y_{1t}^0

$$Y_{1t}^0 = \alpha + \beta Y_{-1t} + \varepsilon_{1t}. \quad (2)$$

Note that the panel data approach calculates OLS models of up to $J + 1$ parameters; so that if the length of the pre-treatment period $t = 1, 2, \dots, T' - 1$ is not of a much higher order than that, the regressions $M(J - 1)^*, M(J)^*$ cannot be calculated because there are not enough degrees of freedom. To avoid this problem, we propose the following slight modification to the previously outlined modeling strategy: use R^2 in order to select the best OLS estimator for Y_{1t}^0 using j out of the J units in the donor pool, denoted by $M(j)^*$ for $j = 1, \dots, T_0 - 4$; then choose $M(m)^*$ from $M(1)^*, \dots, M(T_0 - 4)^*$ in terms of a model selection criterion (in our case AICc). Note that the key difference is that while we allowed models up to $M(J)^*$, this is now modified to allow models up to $M(T_0 - 4)^*$, with $T_0 - 4 < J$.²

To implement the method, the **pampe** package relies on the use of the `regsubsets()` function from the **leaps** package by Lumley (2014). The main user-available function of the **pampe** package, also called `pampe()`, calculates all OLS models for Y_{1t}^0 as dependent variable and using j out of the J units

¹Hsiao et al. (2012) conduct the analysis using both AIC and AICc criteria; in the implementation of the method in the **pampe** package both criteria plus BIC are included.

² $T_0 - 4$ is to allow for at least three degrees of freedom.

in the donor pool as explanatory variables, denoted by $M(j)^*$ for $j = 1, \dots, J$ or up to order $J' < J$ if specified by the user, which would override the default outlined above; then the best one is kept in terms of a model selection criterion (AIC , $AICc$, or BIC) also specified by the user.

In order to perform inference on the results obtained, the package implements the so-called placebo studies procedure outlined in [Abadie and Gardeazabal \(2003\)](#); [Abadie et al. \(2010\)](#) and [Abadie et al. \(2015\)](#).³ The basic idea behind the placebo studies is to iterate the application of the panel data approach by reassigning the treatment to other non-treated units, i.e., to the controls in the donor pool; or by reassigning the treatment to other pre-intervention periods, when the treatment had yet to occur. The set of placebo effects can therefore be compared to the effect that was estimated for the “real” time and unit, in order to evaluate whether the effect estimated by the panel data approach when and where the treatment actually occurred is large relative to the placebo effects.

Implementing pampe in R

This section expands on the implementation of the method itself as well as the placebo studies and how they can be interpreted by the user by means of two examples: the political and economic integration of Hong Kong with mainland China in 1997 and 2004, plus the reassignment of the treatment to other units in the control group and different pre-treatment dates. [Hsiao et al. \(2012\)](#) use a combination of other countries to construct a counterfactual for Hong Kong that resembled the economy prior to the political and economic integration. The growth dataset, obtained from the supplemental materials of [Hsiao et al. \(2012\)](#) contains information on the quarterly real GDP growth rate of 24 countries (the donor pool) and Hong Kong from 1993 Q1 to 2008 Q1, computed as the change with respect to the same quarter in the previous year.

```
> library("pampe")
> data("growth")
```

The data is organized in standard cross-sectional data format, with the variables (the quarterly real GDP growth rate of the countries in the donor pool act as explanatory variables) extending across the columns and the quarters (time-periods) extending across rows. It is important for the user to have his or her data in this standard format to correctly apply the methodology. Naming the rows and especially the columns is also strongly recommended though not required.

If the user does not have the data in standard wide format, the pampe package also includes an optional pampeData function that prepares the data according to the required format. It also helps reshape the data in case the user has it in long format. This function should be run prior to the pampe function. For example, if we had a dataset in long format such as the Produc dataset from the plm package:

```
> library("plm")
> data(Produc, package = "plm")
> long.data <- plm.data(Produc)
> head(long.data)

  state year    pcap    hwy   water   util     pc    gsp    emp unemp
1 ALABAMA 1970 15032.67 7325.80 1655.68 6051.20 35793.80 28418 1010.5   4.7
2 ALABAMA 1971 15501.94 7525.94 1721.02 6254.98 37299.91 29375 1021.9   5.2
3 ALABAMA 1972 15972.41 7765.42 1764.75 6442.23 38670.30 31303 1072.3   4.7
4 ALABAMA 1973 16406.26 7907.66 1742.41 6756.19 40084.01 33430 1135.5   3.9
5 ALABAMA 1974 16762.67 8025.52 1734.85 7002.29 42057.31 33749 1169.8   5.5
6 ALABAMA 1975 17316.26 8158.23 1752.27 7405.76 43971.71 33604 1155.4   7.7
```

Note that “year” is the name of the time index and “state” is the id index, and that there is data on eight variables. If we want to keep data on the variable “pcap” and transform that into a wide format, in which “year” spans across rows and “state” across columns, we can use the following call to pampeData for that.

```
> wide.data <- pampeData(long.data, start = 1970, frequency = 1,
+                         timevar = "year", idvar = "state", "yvar" = "pcap")
> wide.data[1:5, 1:5]
```

³The user will notice that these are not the same inference tests as the ones proposed by [Hsiao et al. \(2012\)](#). The problem with those tests is that they cannot be carried out in a systematic way and therefore, they cannot be built into the package. However, the user can still choose to use the pampe() function without placebo tests and carry out the inference in such a way if they wish to do so, using the acf() and arima() functions from the base package stats.

time.pretr	The pre-treatment periods, up to the introduction of the treatment. For example, if you have 10 pre-treatment periods and the treatment was introduced in the 11th period, this argument could be 1:10.
time.tr	The treatment period plus the periods for which you want to check the effect. For example, if the treatment was introduced in the 11th period and you want results for 9 more periods, it would be 11:20.
treated	The treated unit, i.e., the unit that receives the intervention. It can be a name or the index of the column if the columns in the data matrix are named (which is recommended).
data	The name of the data matrix to be used, e.g., growth.

Table 1: Necessary arguments for the pampe function.

	ALABAMA	ARIZONA	ARKANSAS	CALIFORNIA	COLORADO
1970	15032.67	10148.42	7613.26	128545.4	11402.52
1971	15501.94	10560.54	7982.03	132263.3	11682.06
1972	15972.41	10977.53	8309.01	134451.5	12010.91
1973	16406.26	11598.26	8399.59	135988.4	12473.28
1974	16762.67	12129.06	8512.05	136827.3	12964.14

Of course, the data above is for the "pcap" variable.

Having introduced the optional data preparation, let us now continue with the main function and example of this paper, the growth dataset and the pampe function. Observe how the wide.data above is in an equivalent format as the growth data below after having applied the pampeData function.

```
> growth[1:10, 1:5]
```

	HongKong	Australia	Austria	Canada	Denmark
1993Q1	0.062	0.040489125	-0.013083510	0.01006395	-0.012291821
1993Q2	0.059	0.037856919	-0.007580798	0.02126387	-0.003092842
1993Q3	0.058	0.022509481	0.000542671	0.01891943	-0.007764421
1993Q4	0.062	0.028746550	0.001180751	0.02531683	-0.004048589
1994Q1	0.079	0.033990391	0.025510849	0.04356715	0.031094401
1994Q2	0.068	0.037919372	0.019941313	0.05022538	0.064280003
1994Q3	0.046	0.052289413	0.017087875	0.06512183	0.045955455
1994Q4	0.052	0.031070896	0.023035197	0.06733068	0.055166411
1995Q1	0.037	0.008696091	0.025292696	0.05092120	0.048057177
1995Q2	0.029	0.006773674	0.021849955	0.03152506	0.011953605

In this example, the treated unit – Hong Kong – is in the first column, while the 24 non-treated units are in columns 2 to 25; and the time-periods (quarters) are in rows. Note how both the rows and the columns are named for ease of use and interpretation.

Using the function pampe()

Once the data is in the correct format, it is just a matter of applying the pampe() command to the dataset. Note that it requires a balanced dataset, i.e., no missing values are allowed.⁴ As the bare minimum, the command requires the arguments specified in Table 1.

No additional arguments are necessary, though one may choose to pass other arguments as well. Setting the controls argument is especially recommended, otherwise the default is to use all the remaining (non-treated) columns in the dataset as controls. For example, let us run the pampe() functions using only the bare-bones arguments for the economic integration of Hong Kong as carried out by Hsiao et al. (2012). We first set the pre-treatment and treatment periods; the economic integration of Hong Kong happened in 2004Q1. The pre-treatment period therefore ranges from 1993Q1 to 2003Q4, and the treatment and post-treatment goes from 2004Q1 to 2008Q1. It is useful to define the periods

⁴This might change in future versions.

objects before calling the function so that you can use them later when processing the results, although inputting them directly into the function call is of course an option.

```
> time.pretr <- c("1993Q1", "2003Q4")
> time.tr <- c("2004Q1", "2008Q1")
> ## Or if you know the row indexes use those directly, e.g.
> time.tr <- 45:61
> ## The treated unit
> treated <- "HongKong"
> ## Call the function with the bare minimum arguments specified
> econ.integ <- pampe(time.pretr = time.pretr, time.tr = time.tr,
+                      treated = treated, data = growth)
```

Notice that the defaults, which are used in this case, are to use all the controls and the AICc criterion. You can print out a summary of the optimal model:

```
> summary(econ.integ$model)

Call:
lm(formula = fmla, data = data[time.pretr, ])

Residuals:
    Min      1Q  Median      3Q     Max 
-0.016264 -0.008368 -0.001369  0.008332  0.031529 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.001940   0.003703  -0.524  0.60339    
Austria     -1.011560   0.168235  -6.013 6.03e-07 ***  
Italy        -0.317654   0.159060  -1.997  0.05321 .    
Korea        0.344735   0.046899   7.351 9.72e-09 ***  
Mexico       0.312858   0.051008   6.134 4.14e-07 ***  
Norway       0.322183   0.053776   5.991 6.45e-07 ***  
Singapore    0.184509   0.054569   3.381  0.00172 **  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0117 on 37 degrees of freedom
Multiple R-squared:  0.931, Adjusted R-squared:  0.9198 
F-statistic: 83.16 on 6 and 37 DF,  p-value: < 2.2e-16
```

This replicates the results obtained by [Hsiao et al. \(2012\)](#) for the economic integration of Hong Kong using all 24 units as the possible controls, and the AICc selection criterion. That is, the counterfactual for Hong Kong is built as a combination of Austria, Italy, Korea, Mexico, Norway, and Singapore; i.e., those are the countries that best replicate the economy of Hong Kong in the period prior to the economic integration, according to the model by [Hsiao et al. \(2012\)](#).

If one wishes to fine-tune the process, the arguments specified in Table 2 can be passed to the function as well.⁵

For example, let us now try to replicate the results obtained by [Hsiao et al. \(2012\)](#) for the political integration of Hong Kong using the AICc criterion. If one were not to specify the set of possible controls as they do, the function would aim to use all controls. Since those are too many given the pre-treatment period and the user has not specified a custom nvmax, the default nvmax setting would switch to the length of the pre-treatment period minus four to allow for at least three degrees of freedom.

The treated unit remains the same, but the pre-treatment and treatments are different. If we call the function with the AICc criterion, and use the default ("All") for the controls:

```
> treated <- "HongKong"
> time.pretr <- 1:18 # 1993Q1-1997Q2
> time.tr <- 19:44 # 1997Q3-2003Q4
> pol.integ.all <- pampe(time.pretr = time.pretr, time.tr = time.tr,
+                           treated = treated, data = growth, select = "AICc")
```

⁵There is one last argument, placebos, that is not mentioned here but it will be discussed further in Section Placebo tests.

controls	The units used as controls to calculate the counterfactual, that have not received the treatment. By default, all the remaining (after removing the treated) columns in the data matrix are included as columns, but specific controls can be specified using their column name, e.g., <code>c("Australia", "Austria", "Canada")</code> , or their column index, e.g., <code>2:4</code> .
nbest	The original method by Hsiao et al. (2012) specifies to keep the best model in terms of R^2 for each $M(j)$, hence the default of this argument is one. However the user might choose to keep the best $2, 3, \dots$ before moving on to the second step of the method by changing the default of this argument.
nvmax	Indicates how many subsets of controls should the method check in the first step of the model. The original method by Hsiao et al. (2012) checks all subsets up to the biggest size, $M(j)^*$ up to $M(J)^*$ and hence the default (<code>nvmax = J</code>); but if the pre-treatment period is too short such that this might not be possible, the slight modification if checking subsets up to $T_0 - 4$ is proposed and this is the alternative default the method takes if J is too big for the pre-treatment period and the user has not specified an alternative <code>nvmax</code> . If the user-specified <code>nvmax</code> is too big, it will throw out an error indicating to change this argument or reduce the number of controls.
select	The model selection criterion for the second step of the method. In the original article they propose either <code>AICc</code> (default) or <code>AIC</code> . The user can choose between those two or <code>BIC</code> as well.

Table 2: Other non-required arguments for the `pampe` function.

We will obtain the following results:

```
Call:
lm(formula = fmla, data = data[time.pretr, ])

Residuals:
    Min      1Q  Median      3Q     Max 
-1.330e-03 -3.293e-04  9.270e-06  1.610e-04  1.621e-03 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 0.121549  0.003623 33.552 6.80e-10 ***
Canada       0.448293  0.027020 16.591 1.76e-07 ***
Germany     0.193300  0.040933  4.722  0.00150 ** 
Italy        0.657226  0.069384  9.472  1.27e-05 ***
Japan        -0.923212  0.038457 -24.007 9.66e-09 ***
Korea        -1.022964  0.040920 -24.999 7.01e-09 ***
UnitedKingdom 0.856127  0.054689 15.654 2.77e-07 ***
Philippines   -0.849836  0.042193 -20.142 3.85e-08 ***
Indonesia    -0.118165  0.016813 -7.028  0.00011 *** 
Thailand      0.221161  0.021389 10.340 6.61e-06 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.001186 on 8 degrees of freedom
Multiple R-squared:  0.9982, Adjusted R-squared:  0.9962 
F-statistic: 498.3 on 9 and 8 DF,  p-value: 5.029e-10
```

That is, the call does not throw out an error because of the slightly modified modeling strategy specified in the previous section, but this does not replicate the results obtained by [Hsiao et al. \(2012\)](#) and might not be what the user wants.

In this case, to replicate their results, the set of possible controls has to be specified first, and then we call the function:

```
> possible.ctrls <- c("China", "Indonesia", "Japan", "Korea", "Malaysia",
+                         "Philippines", "Singapore", "Taiwan", "UnitedStates", "Thailand")
> pol.integ <- pampe(time.pretr = time.pretr, time.tr = time.tr, treated = treated,
+                      controls = possible.ctrls, data = growth)

Call:
lm(formula = fmla, data = data[time.pretr, ])

Residuals:
    Min      1Q  Median      3Q     Max 
-0.007304 -0.004851  0.001140  0.004367  0.007178 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.02630   0.01705   1.543   0.1469    
Japan        -0.67596   0.11169  -6.052 4.08e-05 ***  
Korea        -0.43230   0.06338  -6.821 1.22e-05 ***  
UnitedStates 0.48603   0.21952   2.214   0.0453 *    
Taiwan        0.79259   0.30989   2.558   0.0238 *    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.005775 on 13 degrees of freedom
Multiple R-squared:  0.9314, Adjusted R-squared:  0.9103 
F-statistic: 44.15 on 4 and 13 DF,  p-value: 1.919e-07
```

This replicates their results. Note that this output suggests that out of the pre-selected potential controls (China, Indonesia, Japan, Korea, Malaysia, Philippines, Singapore, Taiwan, United States, and Thailand) the model suggests that a combination of Japan, Korea, United States, and Taiwan is the optimal one to replicate the economy of Hong Kong in the pre-economic integration period, while discarding the remaining countries.

If the user wants to replicate their results with AIC:

```
> pol.integ.aic <- pampe(time.pretr = time.pretr, time.tr = time.tr,
+                           treated = treated, controls = possible.ctrls,
+                           data = growth, select = "AIC")

Call:
lm(formula = fmla, data = data[time.pretr, ])

Residuals:
    Min      1Q  Median      3Q     Max 
-0.0068954 -0.0030066  0.0009741  0.0024680  0.0078690 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.03161   0.01639   1.928 0.077815 .  
Japan        -0.69002   0.10560  -6.534 2.79e-05 ***  
Korea        -0.37668   0.06884  -5.472 0.000143 ***  
UnitedStates 0.80994   0.28729   2.819 0.015480 *    
Philippines -0.16237   0.09993  -1.625 0.130163    
Taiwan        0.61889   0.31097   1.990 0.069850 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.005442 on 12 degrees of freedom
Multiple R-squared:  0.9438, Adjusted R-squared:  0.9204 
F-statistic: 40.3 on 5 and 12 DF,  p-value: 4.291e-07
```

Thus, the user can play around with the controls, nbest, nvmax and select arguments such that they better suit their needs and their particular dataset. As the bare minimum, time.pretr, time.tr, treated and data are required for the function to run.

controls	A named vector of the controls finally included in the model.
model	An object of class 'lm' with the optimal model. Usual methods such as fitted(), residuals(), summary(), etc. can be used on it.
counterfactual	A named matrix of the actual path together with the path of the estimated counterfactual for the time.pretr and time.tr periods.
data	The data used for the estimation, stored for later use in, for example, the robustness function also included in the package, which is explained later on.

Table 3: Results given by the pampe function.

Obtaining and transmitting results

Once the function has been correctly run and the user is satisfied with the model, the next step is to process the results obtained. The pampe() function returns an object of class 'pampe' with the objects specified in Table 3.⁶

Continuing with the example of the political integration of Hong Kong with the *AICc* criterion and the set of possible controls as specified by Hsiao et al. (2012):

```
> ## Setup
> treated <- "HongKong"
> time.pretr <- 1:18 # 1993Q1-1997Q2
> time.tr <- 19:44 # 1997Q3-2003Q4
> possible.ctrls <- c("China", "Indonesia", "Japan", "Korea", "Malaysia",
+                         "Philippines", "Singapore", "Taiwan", "UnitedStates", "Thailand")
> ## Call the function with AICc and the possible controls
> pol.integ <- pampe(time.pretr = time.pretr, time.tr = time.tr, treated = treated,
+                      controls = possible.ctrls, data = growth)

Call:
lm(formula = fmla, data = data[time.pretr, ])

Residuals:
    Min      1Q  Median      3Q     Max 
-0.007304 -0.004851  0.001140  0.004367  0.007178 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 0.02630   0.01705   1.543   0.1469    
Japan       -0.67596   0.11169  -6.052 4.08e-05 ***  
Korea       -0.43230   0.06338  -6.821 1.22e-05 ***  
UnitedStates 0.48603   0.21952   2.214   0.0453 *    
Taiwan       0.79259   0.30989   2.558   0.0238 *    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.005775 on 13 degrees of freedom
Multiple R-squared:  0.9314, Adjusted R-squared:  0.9103 
F-statistic: 44.15 on 4 and 13 DF,  p-value: 1.919e-07
```

Let us check the additional results:

```
> pol.integ
$controls
[1] "Japan"        "Korea"         "UnitedStates"  "Taiwan"
```

⁶If the placebos argument is not set to FALSE there are additional results, which will be explained in Section Placebo tests.

```
$model
Call:
lm(formula = fmla, data = data[time.pretr, ])
Coefficients:
(Intercept)      Japan       Korea UnitedStates      Taiwan
0.0263        -0.6760      -0.4323       0.4860        0.7926

$counterfactual
  1993Q1    1993Q2    1993Q3    1993Q4    1994Q1    1994Q2
0.05499950 0.06083154 0.06502118 0.06102413 0.07395480 0.06554503
  1994Q3    1994Q4    1995Q1    1995Q2    1995Q3    1995Q4
...
$data
...
```

A `summary()` method is included for objects of class ‘pampe’, with useful information for the researcher:

```
> summary(pol.integ)

Selected controls:
Japan, Korea, UnitedStates, and Taiwan.
```

```
Time-average estimated treatment effect:
-0.0396291
```

Optimal model estimation results:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.026300	0.017048	1.5427	0.14689
Japan	-0.675964	0.111688	-6.0522	4.084e-05 ***
Korea	-0.432298	0.063377	-6.8211	1.223e-05 ***
UnitedStates	0.486032	0.219521	2.2141	0.04531 *
Taiwan	0.792593	0.309892	2.5576	0.02385 *

Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'
	0.1	' '	1	

Residual standard error: 0.0058 on 13 degrees of freedom

Multiple R-squared: 0.931, Adjusted R-squared: 0.91

F-statistic: 44.15 on 4 and 13 DF, p-value: 1.919427e-07

Although not directly printed, the `summary()` method for objects of class ‘pampe’ also includes the actual and counterfactual paths which can be accessed via `summary(pol.integ)$res.table`. We can also manipulate `pol.integ$model` as we wish with the usual methods since it is an object of class ‘`lm`’. For example, `summary(pol.integ$model)` can be used to obtain a summary, `residuals(pol.integ$model)` for the residuals, or `fitted(pol.integ$model)` to recover the estimated values.

Another method included in the package is `plot()`. It works on objects of class ‘pampe’ to produce a plot of the actual evaluation of the treated unit together with the predicted counterfactual path. A simple plot call to our saved ‘pampe’ object, `plot(pol.integ)`, would produce Figure 2.

If, however, we want to produce and manipulate our own plot, it is just a matter of running a `matplot()` of the actual value together with the counterfactual saved in the results of the function (see Figure 3):

```
> ## A plot of the actual Hong Kong together with the predicted path
> matplot(c(time.pretr, time.tr), pol.integ$counterfactual, type = "l", xlab = "",
+           ylab = "GDP growth", ylim = c(-0.15, 0 .15), col = 1, lwd = 2, xaxt = "n")
> ## Axis labels & titles
> axis(1, at = c(time.pretr, time.tr)[c(seq(2, length(c(time.pretr, time.tr)),
+                                         by = 2))], labels = c(rownames(growth)[c(time.pretr, time.tr)
+                                         [c(seq(2, length(c(time.pretr, time.tr)), by = 2))]]], las = 3)
> title(xlab = "Quarter", mgp = c(3.6, 0.5, 0))
> ## Legend
```

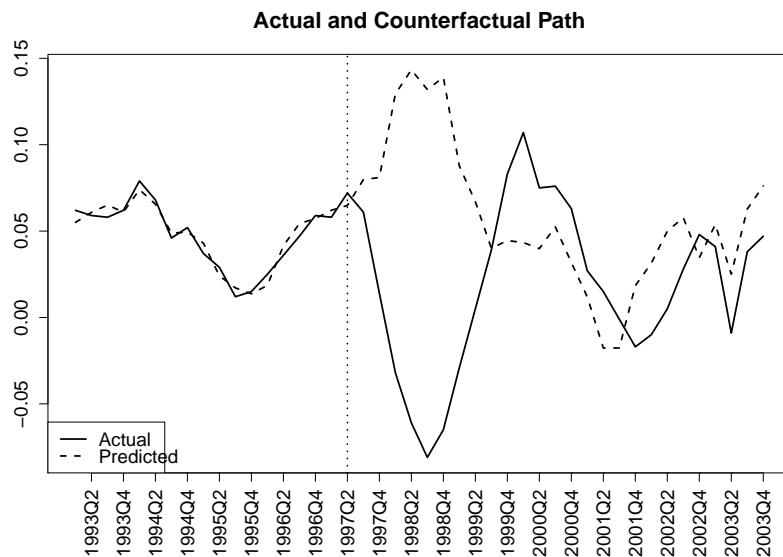


Figure 1: Output of plot(pol.integ).

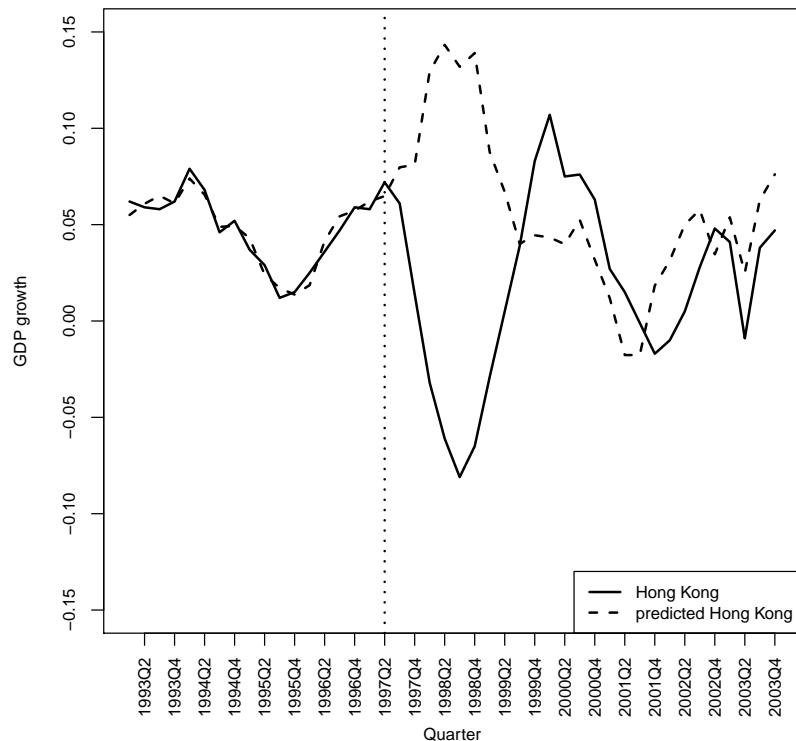


Figure 2: Output of using matplotlib().

```
> legend("bottomright", c("Hong Kong", "predicted Hong Kong"),
+       col = 1, lty = c(1, 2), lwd = 2)
> ## Add a vertical line when the tr starts
> abline(v = time.pretr[length(time.pretr)], lty = 3, lwd = 2)
```

To obtain a plot of the estimated treatment effect, we first calculate the treatment effect, which is the difference between the actual and predicted (counterfactual) path; then we plot it (see Figure 4). Note that if the method works well to replicate the economy in the pre-treatment period, the treatment effect should be around zero in the pre-treatment period.

```
> tr.effect <- pol.integ$counterfactual[, 1] - pol.integ$counterfactual[, 2]
> ## A plot of the estimated treatment effect
```

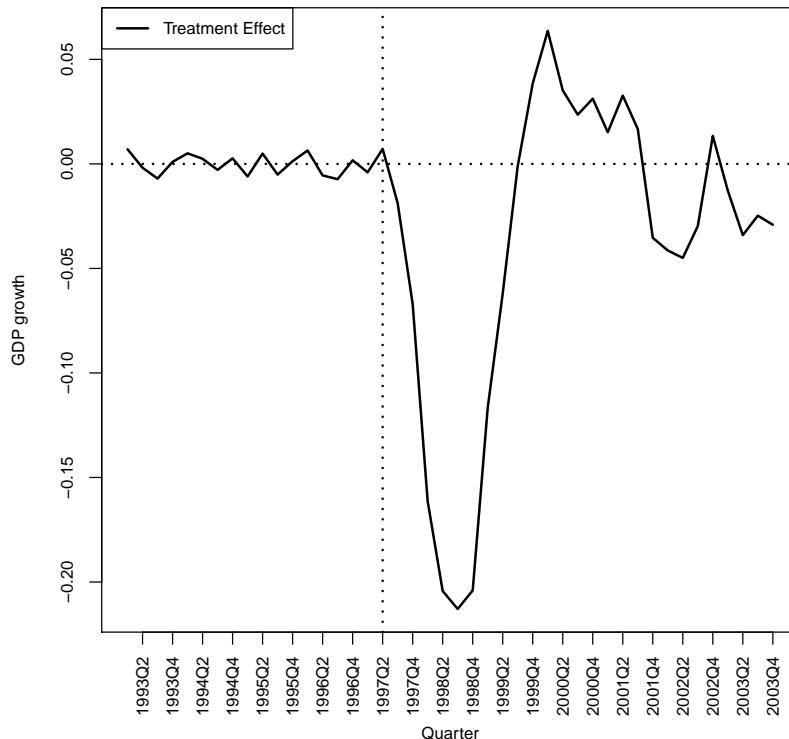


Figure 3: Plot of the estimated treatment effect.

```
> plot(c(time.pretr, time.tr), tr.effect, type = "l", ylab = "GDP growth",
+      xlab = "", col = 1, lwd = 2, xaxt = "n")
> ## Axis labels & titles
> axis(1, at = c(time.pretr, time.tr)[c(seq(2, length(c(time.pretr, time.tr)),
+      by = 2))], labels = c(rownames(growth)[c(time.pretr, time.tr)
+      [c(seq(2, length(c(time.pretr, time.tr)), by = 2))]]], las = 3)
> title(xlab = "Quarter", mgp = c(3.6, 0.5, 0))
> ## Legend
> legend("topleft", "Treatment Effect", col = 1, lty = 1, lwd = 2)
> ## Add a vertical line when the treatment starts
> abline(v = time.pretr[length(time.pretr)], lty = 3, lwd = 2)
> ## Horizontal line on zero
> abline(h = 0, lty = 3, lwd = 2)
```

The user might also be interested in exporting tables that show the results of the procedure, to be used in a L^AT_EX document. Simply manipulating the data and using `xtable` from the package of the same name `xtable` (Dahl, 2014) one can obtain the tables shown in Hsiao et al. (2012). An `xtable` method is also included for this purpose, which requires the output of the ‘pampe’ object and the user specifying which table type he or she wants: the table of the model or the treatment table, which includes the actual, predicted, and treatment paths.

```
> library("xtable")
> xtable(pol.integ, ttype = "model")
> xtable(pol.integ, ttype = "treatment")
```

Placebo tests

In order to perform inference on the results obtained, the package implements the so-called placebo studies procedure outlined in Abadie and Gardeazabal (2003); Abadie et al. (2010) and Abadie et al. (2015). The idea is to iterate the application of the panel data approach by reassigning the treatment to other non-treated units, i.e., to the controls in the donor pool; or by reassigning the treatment to other pre-intervention periods, when the treatment had yet to occur. The set of placebo effects can therefore be compared to the effect that was estimated for the “real” time and unit, in order to evaluate whether

placebo.ctrls	A list which includes another two objects: \$mspe and \$tr.effect. The first includes the mspe for the pre-treatment period (time.pretr) and the second is the estimated treatment effect for the treated unit in the first column and for the countries in the original donor pool (possible.ctrls) in the remaining columns.
placebo.time	The same as placebo.ctrl but with the reassignment of the treatment in time, to periods in the pre-treatment period (the reassignment is from half of the pre-treatment period until the period previous to the actual treatment).

Table 4: Additional results from the pampe function.

the effect estimated by the panel data approach when and where the treatment actually occurred is large relative to the placebo effects.

The function pampe() has both placebo studies (placebo-controls and placebos-in-time) built in. Thus the user can obtain the results from the placebo studies and perform this type of statistical inference simply by switching the last argument of the function pampe(), placebos, from the default FALSE to either "controls", "time", or "Both". Continuing the previous example, the call to the function is identical to the pol.integ one except that now we also ask for the placebos. The other arguments are inherited from before.

```
> pol.integ.placebos <- pampe(time.pretr = time.pretr, time.tr = time.tr,
+                                treated = treated, controls = possible.ctrls,
+                                data = growth, placebos = "Both")
```

Now the results obtained, besides pol.integ.placebos\$controls, \$model, and \$counterfactual like before, include the ones in Table 4.

For example, if we take a look at the first five rows and columns of pol.integ.placebos\$placebo.time\$tr.effect:

	1997Q3	1995Q3	1995Q4	1996Q1	1996Q2
1993Q1	0.007000500	-0.0084630055	0.0005825015	0.0005170915	3.642217e-05
1993Q2	-0.001831535	0.0046961110	0.0036438429	0.0009914089	-6.703595e-04
1993Q3	-0.007021179	0.0005439412	-0.0035785257	-0.0003308651	-2.804963e-03
1993Q4	0.000975869	-0.0065735180	-0.0041505788	-0.0021485760	3.376669e-03
1994Q1	0.005045203	0.0081985975	0.0036386090	0.0003446360	1.552006e-04

We can see that it is a table with the estimated treatment effects (difference between actual and predicted); the first column shows the actual treatment effect, whereas in the remaining columns we have the estimated treatment effect after having reassigned the treatment to other periods, specified in the column name. In this case, the second column has the treatment reassigned to 1995Q3.

Now these additional results can be used for plots and to check whether the treatment effect is significant. When the saved 'pampe' object has placebo studies stored inside, a plot() call to the 'pampe' object will produce the placebo plot(s) as well as the initial actual/predicted path plot. The placebo plot for the control reassignment is given in Figure 5.

Or again, the user might want to produce their own plot to adapt it to their required style:

```
> mspe <- pol.integ.placebos$placebo.ctrl$mspe
> linewidth <- matrix(2, 1, ncol(mspe) - 1)
> linewidth <- append(linewidth, 5, after = 0)
>
> matplot(c(time.pretr, time.tr), pol.integ.placebos$placebo.ctrl$tr.effect,
+          type = "l", xlab = "", ylab = "GDP growth gap",
+          col = c("red", matrix(1, 1, ncol(mspe) - 1)),
+          lty = c(1, matrix(2, 1, ncol(mspe) - 1)), lwd = linewidth,
+          ylim = c(-0.35, 0.2), xaxt = "n")
> ## Axis
> axis(1, at = c(time.pretr, time.tr)[c(seq(2, length(c(time.pretr, time.tr)),
+                                             by = 2))], labels = c(rownames(growth)[c(time.pretr, time.tr)
+                                             [c(seq(2, length(c(time.pretr, time.tr)), by = 2))]]], las = 3)
> title(xlab = "Quarter", mgp = c(3.6, 0.5, 0))
```

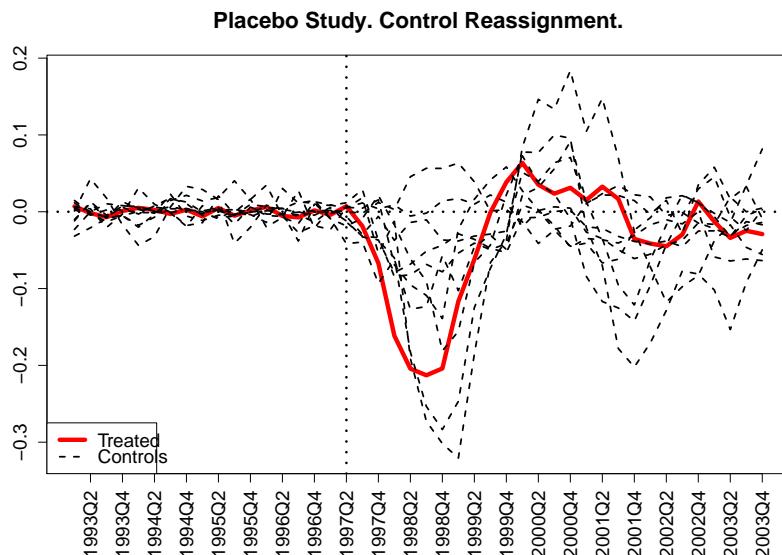


Figure 4: Placebo plot.

```
> ## Legend
> legend("bottomleft", c("Hong Kong", "Controls"), col = c("red", 1),
+       lty = c(1, 2), lwd = c(5, 2))
> ## Horizontal & vertical lines
> abline(h = 0, lty = 3, lwd = 2)
> abline(v = time.pretr[length(time.pretr)], lty = 3, lwd = 2)
```

Note that, contrary to what it could appear from the large size of the estimated treatment shown in the initial plots, the effect does not appear to be significant. This is because after re-applying the method to all other 8 countries from the set of possible controls, the effect for Hong Kong is not an outlier, i.e., the estimated effect for the controls – when there should be none – is similar to the result obtained for Hong Kong. For the effect of the political integration of Hong Kong to be significant it would have to be a true outlier, almost the only one with a non-zero gap. Also important is the fact that while this inference method is different from the one applied by Hsiao et al. (2012), which can be implemented by the user with R functions such as `acf()` and `arima()`, the conclusion is the same as theirs: the political integration of Hong Kong with mainland China does not have an effect on real GDP growth.

As an example of what a significant treatment effect would look like, we carry out the treatment-reassignment placebo tests for the economic integration of Hong Kong, which Hsiao et al. (2012) find to be significant.

```
> time.pretr <- c("1993Q1", "2003Q4")
> time.tr <- c("2004Q1", "2008Q1")
> ## Or if you know the row indexes use those directly, e.g.
> time.tr <- 45:61
> ## The treated unit
> treated <- "HongKong"
> econ.integ.placebos <- pampe(time.pretr = time.pretr, time.tr = time.tr,
+                                 treated = treated, data = growth,
+                                 placebos = "controls")
> plot(econ.integ.placebos)
```

The previous call will produce Figure 6. The reader will observe how, for the economic integration, the estimated treatment effect is in fact an outlier when compared to the controls. Since it is an outlier, together with another two or three of the units, out of 24 controls, we can say it is significant at least at a 5% level.

Let us now check the results obtained from the placebos-in-time. This tests for the causal nature of the effect. If by reassigning the treatment to a previous period we observe that the estimated path (in the pre-treatment period) does not appear to have an effect, but there is still an effect in the actual treatment, then one can assume that the estimated effect is indeed caused by the treatment (though in this case it turns out to be non-significant).

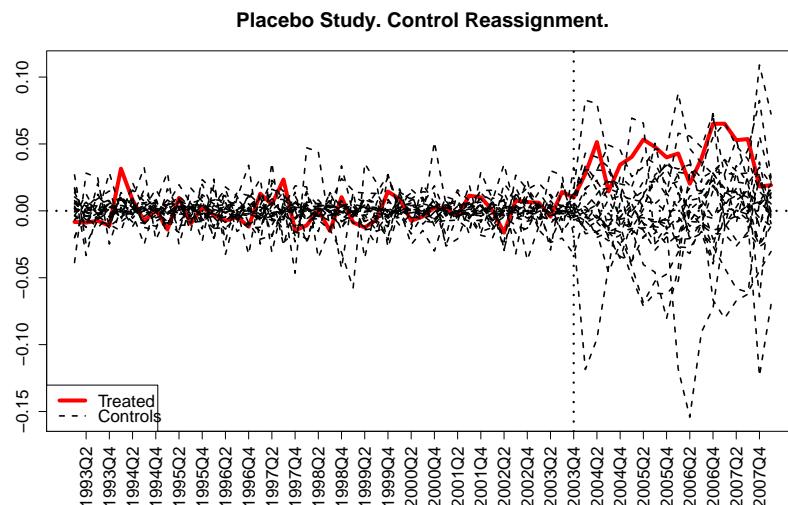


Figure 5: Placebo plot with control reassignment.

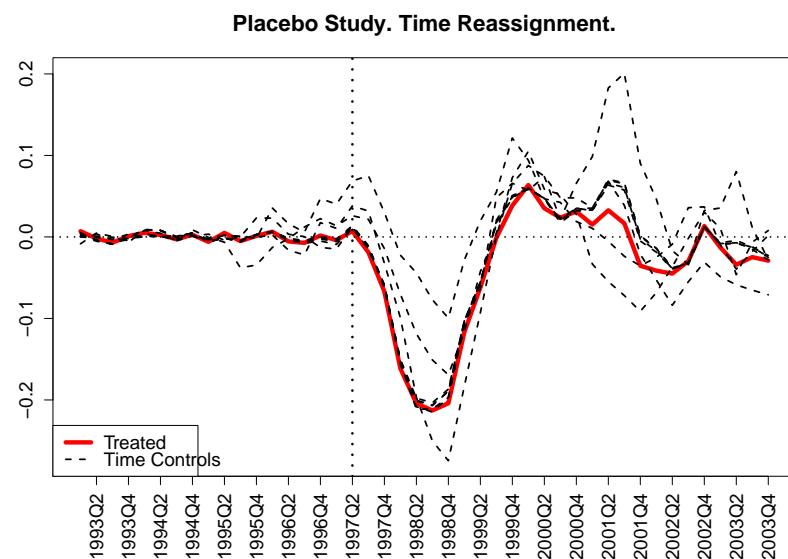


Figure 6: Placebo plot with time reassignment.

A basic plot can be produced by calling `plot(pol.integ)`, which would give Figure 7. Or the user can produce his or her own plot:

```
> ## Plot of the placebos-in-time
> ## For example let's plot the first time reassignment, to 1995Q3
> ## (2nd column)
> placebo.in.time1 <- pol.integ.placebos$placebo.time$tr.effect[, 2] +
+   growth[c(time.pretr, time.tr), 1]
> matplot(c(time.pretr, time.tr), cbind(growth[c(time.pretr, time.tr), 1],
+   pol.integ.placebos$counterfactual, placebo.in.time1), type = "l",
+   ylab = "GDP growth", xlab = "", ylim = c(-0.25, 0.2), col = 1,
+   lwd = 3, xaxt = "n")
> ## Axis
> axis(1, at = c(time.pretr, time.tr)[c(seq(2, length(c(time.pretr, time.tr)),
+   by = 2))], labels = c(rownames(growth)[c(time.pretr, time.tr),
+   [c(seq(2, length(c(time.pretr, time.tr)), by = 2))]]], las = 3)
> title(xlab = "Quarter", mgp = c(3.6, 0.5, 0))
> ## Legend
> legend("bottomleft", c("actual", "predicted", paste("placebo",
+   colnames(pol.integ.placebos$placebo.time$tr.effect)[2], sep = " "))),
```

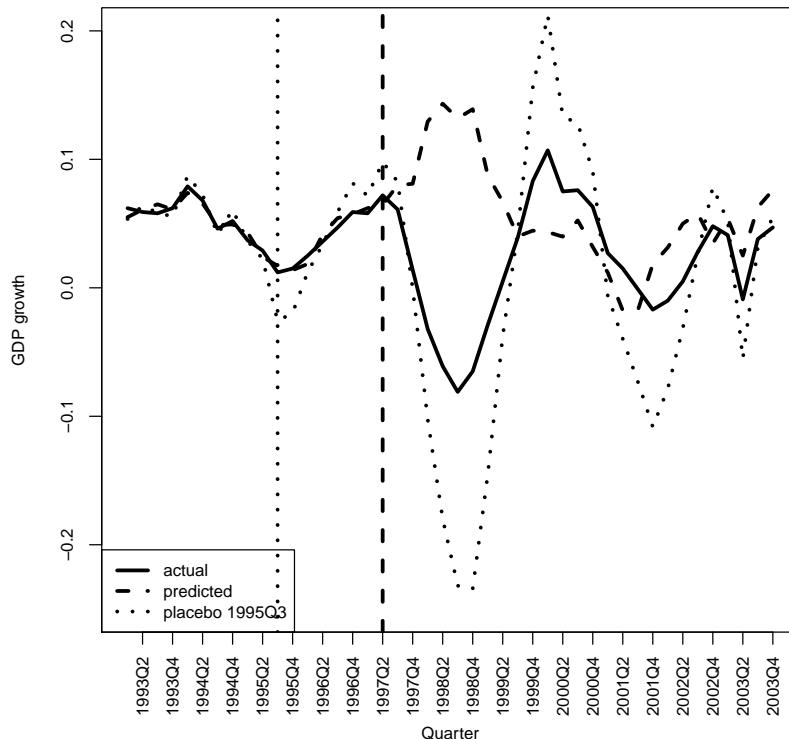


Figure 7: Plot of the placebos-in-time.

```

+      col = 1, lty = c(1, 2, 3), lwd = 3)
> ## Two vertical lines
> abline(v = time.pretr[length(time.pretr)], lty = 2, lwd = 3)
> abline(v = which(colnames(pol.integ.placebos$placebo.time$tr.effect)[2]
+ == rownames(growth)), lty = 3, lwd = 3)
> ## We can also plot the gaps all at the same time
> mspe <- pol.integ.placebos$placebo.time$mspe
> linewidth <- matrix(2, 1, ncol(mspe) - 1)
> linewidth <- append(linewidth, 5, after = 0)
>
> matplot(c(time.pretr, time.tr), pol.integ.placebos$placebo.time$tr.effect,
+ type = "l", xlab = "", ylab = "GDP growth gap",
+ col = c("red", matrix(1, 1, ncol(mspe) - 1)),
+ lty = c(1, matrix(2, 1, ncol(mspe) - 1)), lwd = linewidth,
+ ylim = c(-0.35, 0.2), xaxt = "n")
> ## Axis
> axis(1, at = c(time.pretr, time.tr)[c(seq(2, length(c(time.pretr, time.tr)),
+ by = 2))], labels = c(rownames(growth)[c(time.pretr, time.tr)
+ [c(seq(2, length(c(time.pretr, time.tr)), by = 2))]]], las = 3)
> title(xlab = "Quarter", mgp = c(3.6, 0.5, 0))
> ## Legend
> legend("topleft", c("Hong Kong", "Controls"), col = c("red", 1),
+ lty = c(1, 2), lwd = c(5, 2))
> ## Horizontal line
> abline(h = 0, lty = 3, lwd = 2)

```

The first example replicates the previous time reassignment plot. The second example would produce Figure 7.

Robustness checks

Besides placebo studies for inference tests, Abadie et al. (2015) show the importance of running robustness checks on the results obtained. This section demonstrates how to implement the so-called

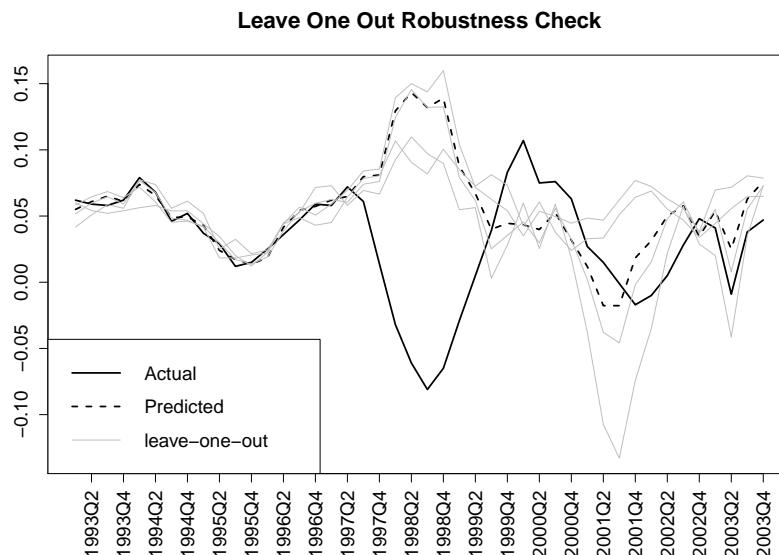


Figure 8: Leave-one-out robustness check.

leave-one-out robustness check, which iteratively removes one of the units in the control group of the final model, to check whether the results are driven by one unit in particular or, in contrast, the results are robust to removing one unit.

The leave-one-out robustness check can be applied using the other user-available function included in the package, `robustness()`. After applying the initial function `pampe()`, the user can carry out the robustness check simply by calling the function `robustness()` and specifying the name of the saved object from the `pampe` call.

```
> rob.check <- robustness(pol.integ)
```

This produces a matrix with the actual path, the initial predicted path, and each leave-one-out predicted path. Its first five rows are given by:

	Actual	Predict	w/ all	w/o Japan	w/o Korea	w/o UnitedStates	w/o Taiwan
1993Q1	0.062		0.05499950	0.05993785	0.04167956	0.058347659	0.05173061
1993Q2	0.059		0.06083154	0.05414176	0.05082512	0.064482544	0.05973967
1993Q3	0.058		0.06502118	0.05208924	0.05861153	0.068493568	0.06459402
1993Q4	0.062		0.06102413	0.05398689	0.05587109	0.063964428	0.05918035
1994Q1	0.079		0.07395480	0.05629484	0.07740008	0.071861773	0.07673810

The user can then plot this robustness check by calling the `plot()` method to the saved object. `xtable` and `summary` methods are also provided.

```
plot(rob.check)
```

The following plot (Figure 8) shows that the results obtained with four countries (Japan, Korea, US and Taiwan) are robust to the removal of one of them. That is, the results are not driven by one particular country.

If the user would prefer to reproduce the plot and manipulate the code to his or her liking, they should modify the following code, which replicates the above plot.

```
> linewidth <- matrix(1, 1, ncol(rob.check))
> linewidth <- append(linewidth, c(2, 2), after = 0)
>
> matplot(c(time.pretr, time.tr), cbind(apt.table[, 1:2], rob.check),
+           type = "l", xlab = "", ylab = "GDP growth gap",
+           col = c(1, 1, matrix("gray", 1, ncol(rob.check))),
+           lty = c(1, 2, matrix(1, 1, ncol(rob.check))), lwd = linewidth, xaxt = "n")
> ## Axis
> axis(1, at = c(time.pretr, time.tr)[c(seq(4, length(c(time.pretr, time.tr)),
+                                         by = 4))], labels = c(rownames(growth)[c(time.pretr, time.tr)
+                                         [c(seq(4, length(c(time.pretr, time.tr)), by = 4))]]], las = 3)
> title(xlab = "Quarter", mgp = c(3.6, 0.5, 0))
```

```

> ## Legend
> legend("bottomleft", c("Hong Kong", "Predicted", "leave-one-out"),
+        col = c(1, 1, "gray"), lty = c(1, 2, 1), lwd = c(2, 2, 1))
> ## Vertical line when treatment begins
> abline(v = max(time.pretr), lty = 3, lwd = 2)

```

Acknowledgments

I would like to thank Javier Gardeazabal and Yang Yang for their helpful comments and testing of the package, as well as the anonymous referees who suggested thoughtful improvements.

Funding for this research was provided by the Basque Government through grant EC-2013-1-53.

Bibliography

- A. Abadie and J. Gardeazabal. The economic costs of conflict: A case study of the Basque country. *American Economic Review*, 93(1):113–132, Mar. 2003. [p105, 107, 115]
- A. Abadie, A. Diamond, and J. Hainmueller. Control methods for comparative case studies: Estimating the effect of California’s tobacco control program. *Journal of the American Statistical Association*, 105(490):493–505, 2010. [p105, 107, 115]
- A. Abadie, A. Diamond, and J. Hainmueller. Comparative politics and the synthetic control method. *American Journal of Political Science*, 59(2):495–510, 2015. [p107, 115, 119]
- D. B. Dahl. *xtable: Export Tables to LaTeX or HTML*, 2014. URL <https://CRAN.R-project.org/package=xtable>. R package version 1.7-3. [p115]
- C. Hsiao, H. Steve Ching, and S. Ki Wan. A panel data approach for program evaluation: Measuring the benefits of political and economic integration of Hong Kong with Mainland China. *Journal of Applied Econometrics*, 27(5):705–740, 2012. doi: 10.1002/jae.1230. [p105, 106, 107, 108, 109, 110, 112, 115, 117]
- T. Lumley. *leaps: Regression Subset Selection*, 2014. URL <https://CRAN.R-project.org/package=leaps>. R package version 2.9. [p106]
- D. B. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688–701, 1974. [p106]

Ainhoa Vega-Bayo
Foundations of Economic Analysis II
University of the Basque Country, UPV/EHU
Avda. Lehendakari Aguirre 83, 48015 Bilbao
Spain
ainhoa.vega@ehu.eus

BSGS: Bayesian Sparse Group Selection

by Kuo-Jung Lee and Ray-Bing Chen

Abstract An R package **BSGS** is provided for the integration of Bayesian variable and sparse group selection separately proposed by [Chen et al. \(2011\)](#) and [Chen et al. \(in press\)](#) for variable selection problems, even in the cases of large p and small n . This package is designed for variable selection problems including the identification of the important groups of variables and the active variables within the important groups. This article introduces the functions in the **BSGS** package that can be used to perform sparse group selection as well as variable selection through simulation studies and real data.

Introduction

Variable selection is a fundamental problem in regression analysis, and one that has become even more relevant in current applications where the number of variables can be very large, but it is commonly assumed that only a small number of variables are important for explaining the response variable. This sparsity assumption enables us to select the important variables, even in situations where the number of candidate variables is much greater than the number of observations.

BSGS is an R package designed to carry out a variety of Markov chain Monte Carlo (MCMC) sampling approaches for variable selection problems in regression models based on a Bayesian framework. In this package, we consider two structures of variables and create functions for the corresponding MCMC sampling procedures. In the first case where the variables are treated individually without grouping structure, two functions, `CompWiseGibbsSimple` and `CompWiseGibbsSMP`, are provided to generate the samples from the corresponding posterior distribution. In the second case, it is assumed that the variables form certain group structures or patterns, and thus the variables can be partitioned into different disjoint groups. However, only a small number of groups are assumed to be important for explaining the response variable, i.e. the condition of the group sparsity, and we also assume that sparse assumption is held for the variables within the groups. This problem is thus termed a sparse group selection problem [Simon et al. \(2013\)](#); [Chen et al. \(in press\)](#), and the goal is to select the important groups and also identify the active variables within these important groups simultaneously. There are two functions to handle the sparse group selection problems, `BSGS.Simple` and `BSGS.Sample`, which are used to generate the corresponding posterior samples. Once the posterior samples are available, we then can determine the active groups and variables, estimate the parameters of interest and make other statistical inferences.

This paper is organized as follows. We first briefly introduce statistical models that are used to deal with the problems of variable selection in the **BSGS** package. We then describe the tuning parameters in the functions in the **BSGS** package. Two simulations are used to illustrate the details of the implementations of the functions. Finally we present a real economic example to demonstrate the **BSGS** package.

Framework of BSGS

We start with the introduction of individual variable selection problems, and then turn our attention to sparse group selection. For completeness, we describe the model and priors so that one may easily change the inputs of functions in the **BSGS** package for any purpose.

Variable selection

Consider a linear regression model given by

$$Y = \mathbf{X}\beta + \varepsilon, \quad (1)$$

where $Y = (Y_1, \dots, Y_n)'$ is the response vector of length n , $\mathbf{X} = [X_1, \dots, X_p]$ is an $n \times p$ design matrix, with X_i as the corresponding i -th variable (regressor) as a potential cause of the variation in the response, β is the corresponding unknown $p \times 1$ coefficient vector, and $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)'$ is the error term, which is assumed to have a normal distribution with a zero mean, and the covariance matrix $\sigma^2 I_n$, I_n is the $n \times n$ identity matrix. To achieve variable selection, we select a subset of X_1, \dots, X_p to explain the response Y . For this purpose, following the stochastic search variable selection method [George and McCulloch \(1993\)](#), a latent binary variable γ_i taking the value of 0 and 1 is introduced

to indicate whether or not the corresponding variable is selected. That is, if $\gamma_i = 1$, the variable X_i is selected, and otherwise it is not selected.

In this Bayesian framework we basically follow the prior assumption in Chen et al. (2011). We assume the prior distribution of γ_i is a Bernoulli distribution with probability $1 - \rho_i$, and then given γ_i , we assume the prior for β_i is a mixture of point mass at 0 denoted by δ_0 and a normal distribution as follows

$$\beta_i | \gamma_i \sim (1 - \gamma_i)\delta_0 + \gamma_i N(0, \tau_i^2), \quad (2)$$

where τ_i is a pre-specified constant. Moreover, (γ_i, β_i) are assumed to be independent for $i = 1, \dots, p$. Lastly, σ^2 is set to follow an inverse Gamma distribution,

$$\sigma^2 \sim \text{IG}(\nu/2, \nu\lambda/2). \quad (3)$$

Based on the model setting given above, there are two Gibbs samplers for this variable selection. The first procedure is the componentwise Gibbs sampler (CGS) which was introduced by Geweke (1996) and also mentioned in Chen et al. (2011). The other is the stochastic matching pursuit (SMP) algorithm Chen et al. (2011) in which the variables compete to be selected.

Sparse group selection

In traditional variable selection problems, each variable X_i is treated individually; however, in some real applications, a group of variables that behave similarly may be more meaningful in explaining the response. In other words, a group of variables potentially plays a more important role in explaining the response than a single variable can. The variable selection problem thus becomes a group selection one. In group selection problems, the variables within the selected groups are all treated as important. However, this assumption might not be held in practice such as the climate application in Chatterjee et al. (2012). Instead of the group selection problem, we thus consider approaches to sparse group selection, in which the sparse assumption is held for groups and the variables within groups.

Here the goal is not only to select the influential groups, but also to identify the important variables within these. To this end, a Bayesian group variable selection approach, the Group-wise Gibbs sampler (GWGS) Chen et al. (in press), is applied. Suppose that, in terms of expert or prior knowledge, the explanatory variables X_i 's are partitioned into g non-overlapping groups in a regression model. Each group l contains $j = 1, \dots, p_l$ variables with $\sum_{l=1}^g p_l = p$. Now the model is rewritten as

$$Y = \sum_{l=1}^g \mathbf{X}_l \beta_l + \varepsilon, \quad (4)$$

where $\mathbf{X}_l = [X_{l1}, \dots, X_{lp_l}]$ is the $n \times p_l$ sub-matrix of \mathbf{X} , $\beta_l = (\beta_{l1}, \dots, \beta_{lp_l})'$ is the $p_l \times 1$ coefficient vector of group l . The GWGS works by introducing two nested layers of binary variables to indicate whether a group or a variable is selected or not. At the group-selection level, we define a binary variable $\eta_l = 1$ if group l is selected, and $\eta_l = 0$ otherwise. At the variable-selection level, we define another binary variable $\gamma_{li} = 1$ if the variable i within group l , X_{li} , is selected, and $\gamma_{li} = 0$ otherwise.

We assume the group indicator, η_l , has the Bernoulli distribution with the probability $1 - \theta_l$. Within group l , the prior distribution of γ_{li} conditional on the indicator η_l is defined as

$$\gamma_{li} | \eta_l \sim (1 - \eta_l)\delta_0 + \eta_l \text{Ber}(1 - \rho_{li}), \quad (5)$$

where δ_0 is a point mass at 0 and $\text{Ber}(1 - \rho_{li})$ is Bernoulli distributed with the probability $1 - \rho_{li}$. Equation (5) implies that if the l -th group is not selected, it turns out that $\gamma_{li} = 0$ for all i . The prior distribution of the coefficient β_{li} given η_l and γ_{li} is given by

$$\beta_{li} | \eta_l, \gamma_{li} \sim (1 - \eta_l \gamma_{li})\delta_0 + \eta_l \gamma_{li} N(0, \tau_{li}^2),$$

where τ_{li} is a pre-specified value Chen et al. (2011). Finally, the variance σ^2 is assumed to have an inverse Gamma distribution, that is, $\sigma^2 \sim \text{IG}(\nu/2, \nu\lambda/2)$. We also assume $(\eta_l, \gamma_{l1}, \beta_{l1}, \dots, \gamma_{lp_l}, \beta_{lp_l})$, $l = 1, \dots, g$, are *a priori* independent.

Two sampling procedures are proposed in Chen et al. (in press) for Bayesian sparse group selection. The first is the GWGS. In the GWGS, simulating the indicator variable η_l from the posterior distribution would be computationally intensive, especially when the number of variables within the group is large. To address this issue, Chen et al. (in press) proposed a modified and approximation approach, a sample version of GWGS. In this a Metropolis-Hastings algorithm is adopted to replace the Gibbs sampling method in GWGS.

Implementation

In this section, we describe the default tuning parameters and some details in the implementation of the functions in **BSGS**.

Hyperparameter set-up

- **The tuning parameters, ν and λ :**

The parameters in the prior distribution of σ^2 are suggested by George and McCulloch (1993) setting the default values of $\nu = 0$ and λ being any positive value. A data-driven choice for this is proposed by Chipman et al. (1997), which sets the value of ν around 2 and the value of λ is set up to be the 99% quantile of the prior of σ^2 that is close to $\sqrt{\text{Var}(Y)}$. In addition, George and McCulloch (1997) simply set $\nu = 10$ and $\lambda = \sqrt{\text{Var}(Y)}$. In our experiences, a larger value of ν tends to result in a larger estimate of σ .

- **The parameter τ :**

Now we consider the assignment of the value of τ , the prior variance of the regression coefficient for the active variable. It was found that the larger the value of τ , the smaller the conditional probability of $\gamma = 1$ is. As a result, a large value of τ favors a more parsimonious mode. In contrast, a small value of τ would yield more complex models.

- **The parameters ρ and θ :**

The default of the prior inclusion probabilities of groups and variables is set equal to 0.5. In the case when p is much greater than n and only a small number of variables are considered active, we would assign a larger values to ρ and θ to reflect the prior belief of sparsity.

Stopping rule

The posterior distribution is not available in explicit form so we use the MCMC method, and specifically Gibbs sampling to simulate the parameters from this distribution Brooks et al. (2011). To implement the Gibbs sampler, the full conditional distributions of all parameters must be determined. A derivation of the full conditional distributions is provided in Chen et al. (in press). When these have been obtained, the parameters are then updated individually using a Gibbs sampler (where available), or a Metropolis-Hastings sampling algorithm. An MCMC sample will converge to the stationary distribution, i.e. the posterior distribution. We use the batch mean method to estimate the Monte Carlo standard error (MCSE) of the estimate of σ^2 and then decide to stop the simulation once the MCSE is less than a specified value, cf. Flegal et al. (2008). The default minimum number of iterations is 1000. If the MCSE does not achieve the prespecified value, an extra 100 iterations are run until the MCSE is less than the prespecified value. The sample can then be used for statistical inference.

Statistical inference

- **Variable and group selection criteria:**

In this package, we adopt the median probability criterion proposed by Barbieri and Berger (2004) for group and variable selections. Specifically, for the variable selection problem, we estimate the posterior inclusion probability $P(\gamma_i = 1|Y)$ from the posterior samples and then the i -th variable is selected into the model if the estimated posterior probability is larger than or equal to 1/2. Here instead of 1/2, this cut-off value is treated as a tuning parameter, α , which can be specified by users. For the sparse group selection problem, the estimated posterior probability of the l -th group is greater than or equal to α_g , i.e. $P(\eta_l = 1|Y) \geq \alpha_g$, we then include X_l into the model. Suppose the l -th group is selected, then the i -th variable within this group is selected if $P(\gamma_{li} = 1|\eta_l = 1, Y) \geq \alpha_i$. Here α_g and α_i are two pre-specified values between 0 and 1.

- **Posterior estimates of regression coefficients:**

We use the Rao-Blackwell method to estimate β by

$$\hat{\beta} = E(\beta|y) \approx \frac{1}{N_M} \sum_{m=1}^M \beta_m,$$

where β_m is the sample in m th iteration, M is the number of iterations, and N_M is the number of nonzero β_m .

Evaluation of model estimation

Regarding the stability of the estimation, we compare the accuracy of selection of the variables by the following measures in the simulation studies: the True Classification Rate (TCR), the True Positive Rate (TPR), and the False Positive Rate (FPR). These are defined as follows

$$\begin{aligned} \text{TCR} &= \frac{\text{number of correctly selected variables}}{\text{number of variables}}; \\ \text{TPR} &= \frac{\text{number of correctly selected variables}}{\text{number of active variables}}; \\ \text{FPR} &= \frac{\text{number of falsely selected variables}}{\text{number of inactive variables}}. \end{aligned}$$

TCR is an overall evaluation of accuracy in the identification of the active and inactive variables. TPR is the average rate of active variables correctly identified, and is used to measure the power of the method. FPR is the average rate of inactive variables that are included in the regression, and it can be considered as the type I error rate of the selection approach. In these three criteria, it is preferred to have a larger value of TCR or TPR, or a smaller value of FPR.

We also report the mean squared error (MSE),

$$\text{MSE} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 / n,$$

where \hat{y}_i is the prediction of y_i . This is used to evaluate whether the overall model estimation has a good fit with regard to the data set.

Examples

Two simulations and a real example are provided to demonstrate the use of functions in the **BSGS** package.

Simulation I

The traditional variable selection problem is illustrated in this simulation. We use an example to illustrate the functions `CompWiseGibbsSimple` and `CompWiseGibbsSMP` corresponding to the CGS and SMP sampling procedures to simulate the sample from the posterior distribution. Based on the samples, we can decide which variable is important in the regression model. In this simulation, the data Y of length $n = 50$ is generated from a normal distribution with a mean $X\beta$ and $\sigma^2 = 1$. We

assume $\beta = (3, -3.5, 4, -2.8, 3.2, \underbrace{0, \dots, 0}_{p-5}, p = 100)$, $p = 100$, and X is from a multivariate normal distribution with a mean 0 and the covariance matrix Σ as the identity matrix. We then generate the responses based on (1).

```
require(BSGS)
set.seed(1)

## Generate data
num.of.obs <- 50
num.of.covariates <- 100

beta.g <- matrix(c(3, -3.5, 4, -2.8, 3.2, rep(0, num.of.covariates-5)), ncol = 1)
r.true <- (beta.g != 0) * 1

pair.corr <- 0.0 ## pair correlations between covariates
Sigma <- matrix(pair.corr, num.of.covariates, num.of.covariates)
diag(Sigma) <- rep(1, num.of.covariates)
x <- mvrnorm(n = num.of.obs, rep(0, num.of.covariates), Sigma)

sigma2 <- 1
mu <- x %*% beta.g
y <- rnorm(num.of.obs, mu, sigma2)
```

Regarding to the hyperparameters, we simply set $\tau^2 = 10$, but one can use cross-validation to tune the parameter [Chen et al. \(2011\)](#). Following [George and McCulloch \(1997\)](#), we let $\nu = 10$ and

$\lambda = \sqrt{\text{Var}(Y)}$. Without any prior information on which variable is important, we let prior inclusion probability $\rho_i = 0.5$ for each variable. We use the rigid regression estimates as the initial values of regression coefficients β s.

```
## Specify the values of hyperparameters and initial values of parameters

tau2 <- 10      ## hyperparameter in Eq. (1)
nu0 <- 10       ## hyperparameter in Eq. (2)
lambda0 <- sd(y) ## hyperparameter in Eq. (2)

## Initial values for parameters
beta.initial <- t(solve(t(x) %*% x + diag(1/5, num.of.covariates)) %*% t(x) %*% y)
sigma2.initial<-1
r.initial <- rbinom(num.of.covariates, 1, 1)
```

Now two functions, `CompWiseGibbsSimple` and `CompWiseGibbsSMP`, are applied to simulate the samples from the posterior distribution. The minimum number of iterations is 1000. The simulation will stop when the MCSE of the estimate of σ^2 is less than 0.1 and otherwise an extra 100 iterations are run until the MCSE is less than this. For stability, we update `num.of.inner.iter.default = 10` times for β and γ and take the last ones as a sample before updating σ^2 .

```
num.of.iteration <- 1000
num.of.inner.iter.default <- 10
MCSE.Sigma2.Given <- 0.1

## Apply two sampling functions to generate the samples from the
## posterior distribution.

outputCGS <- CompWiseGibbsSimple(y, x, beta.initial, r.initial, tau2,
  rho, sigma2.initial, nu0, lambda0, num.of.inner.iter.default,
  num.of.iteration, MCSE.Sigma2.Given)

outputSMP <- CompWiseGibbsSMP(y, x, beta.initial, r.initial, tau2,
  rho, sigma2.initial, nu0, lambda0, num.of.inner.iter.default,
  num.of.iteration, MCSE.Sigma2.Given)
```

Once the simulation stops, the posterior samples are used to estimate the posterior quantities of interest. One can then check the number of iterations and computational times for both approaches.

```
## Output from the component-wise Gibbs sampling procedure
outputCGS$Iteration
[1] 1000
outputCGS$TimeElapsed
  user  system elapsed
 61.558   4.815  66.813

## Output from the component-wise Gibbs sampling procedure
outputSMP$Iteration
[1] 1000
outputSMP$TimeElapsed
  user  system elapsed
 45.970   5.907  52.383
```

One can then use the function `CGS.SMP.PE` to identify the important variables and to estimate the parameters. Due to the limitations of space, we do not include the estimates here. A variable i is considered important if the posterior probability of its indicator variable $\gamma_i = 1$ is greater than or equal to $\alpha_i = 1/2$. Once the critical point is decided, two functions `TCR.TPR.FPR.CGS.SMP` and `MSE.CGS.SMP` are carried out to evaluate the performance on the model estimations in terms of TCP, TPR, FTP and MSE.

```
## Output from the component-wise Gibbs sampling procedure
CGS.SMP.PE(outputCGS)

MSE.CGS.SMP(outputCGS, Y=y, X=x)
[1] 2.921087
```

```

TCR.TPR.FPR.CGS.SMP(outputCGS, r.true, 0.5)
$TCR
[1] 1

$TPR
[1] 1

$FPR
[1] 0

## Output from the component-wise Gibbs sampling procedure
CGS.SMP.PE(outputSMP)

MSE.CGS.SMP(outputSMP, Y=y, X=x)
[1] 3.751427

TCR.TPR.FPR.CGS.SMP(outputSMP, r.true, 0.5)
$TCR
[1] 1

$TPR
[1] 1

$FPR
[1] 0

```

Simulation II

We provide another stimulation to illustrate the use of functions for sparse group selection. In the following simulation, the data Y of length $n = 50$ is generated from a normal distribution with a mean $X\beta$ and $\sigma^2 = 1$, where X is from a multivariate distribution with mean 0 and covariance Σ with the pair correlation between variables equal to zero. There are 10 groups of variables. Each group contains 10 variables and some of them are active. More specifically, $X_l = X_{l1}, \dots, X_{lp_l}$, $k = 1, \dots, 10$ and $p_l = 10$ for all l . We assume the group $l = 1, 2, 5, 8$ are active. Variables $p_1 = 7, 8, 9$ in the group $l = 1$, $p_2 = 1, 2$ in the group $l = 2$, $p_5 = 3$ in the group $l = 5$, and $p_8 = 7$ in the group $l = 8$ are active. The response is generated via (4).

```

require(BSGS)

set.seed(1)

Num.Of.Iteration <- 1000
Num.of.Iter.Inside.CompWise <- 10
num.of.obs <- 50
num.of.covariates <- 100
num.of.group.var <- 10
Group.Index <- rep(1:10, each = 10)

nu <- 0
lambda <- 1
pair.corr <- 0.

Sigma <- matrix(pair.corr, num.of.covariates, num.of.covariates)
diag(Sigma) <- rep(1,num.of.covariates)

X <- mvrnorm(n = num.of.obs, rep(0, num.of.covariates), Sigma)

beta.true <- rep(0, num.of.covariates)
beta.true[c(7, 8, 9, 11, 12, 43, 77)] <- c(3.2, 3.2, 3.2, 1.5, 1.5, -1.5, -2)
beta.true <- cbind(beta.true)
r.true <- (beta.true != 0) * 1

sigma2.true <- 1

```

```
Y <- rnorm(num.of.obs, X %*% beta.true, sigma2.true)
```

Here we suppose that we have no prior information on the parameters. We let $\nu = 0$ and $\lambda = 1$ which corresponds to the non-informative prior for σ^2 . Also, we let $\rho_i = 0.5$ and $\theta = 0.5$ for prior inclusion probabilities of groups and variables. Finally, we let $\tau = 1$ for the variance of each regression coefficient.

```
## hyperparameters
tau2.value <- rep(1, num.of.covariates)
rho.value <- rep(0.5, num.of.covariates)
theta.value <- rep(0.5, num.of.group.var)
```

With the reasonable assignment of the initial values of parameters, we apply two functions, BSGS.Simple and BSGS.Sample, to estimate the posterior quantities of interest and in turn identify the important groups and variables. For illustration, we stop the simulation when the MCSE of estimate of σ^2 is less than 0.5.

```
## Initial values and stopping point
r.value <- rbinom(num.of.covariates, 1, 0.5)
eta.value <- rbinom(num.of.group.var, 1, 0.5)
beta.value <- cbind(c(t(solve(t(X) %*% X +
  diag(1/5, num.of.covariates)) %*% t(X) %*% Y) )) # beta.true
sigma2.value <- 1
MCSE.Sigma2.Given <- 0.5

## Apply two sampling approaches to generate samples
outputSimple <- BSGS.Simple(Y, X, Group.Index, r.value, eta.value, beta.value,
  tau2.value, rho.value, theta.value, sigma2.value, nu, lambda,
  Num.of.Iter.Inside.CompWise, Num.Of.Iteration, MCSE.Sigma2.Given)

outputSample <- BSGS.Sample(Y, X, Group.Index, r.value, eta.value, beta.value,
  tau2.value, rho.value, theta.value, sigma2.value, nu, lambda,
  Num.of.Iter.Inside.CompWise, Num.Of.Iteration, MCSE.Sigma2.Given)
```

One can easily use the function BSGS.PE to estimate the posterior probabilities of $\eta_l = 1$ and $\gamma_{li} = 1 | \eta_l = 1$ based on the samples generated from the posterior distribution. To investigate which sampling approach provides a better model estimation, one can calculate MSE by the function MSE.BSGS. Furthermore, the function TCR.TPR.FPR.BSGS is used to evaluate the performance on the accuracy of selection on variables. All functions are illustrated as follows. We take $\alpha_i = \alpha_g = 0.5$ in this example.

```
## The posterior quantities estimated by two sampling approaches respectively.
```

```
## Output from the simple version of BSGS
outputSimple$Iteration
[1] 1000
outputSimple$TimeElapsed
  user  system elapsed
238.755  0.007 239.037
BSGS.PE(outputSimple)$eta.est
  1    2    3    4    5    6    7    8    9   10
1.000 1.000 0.115 0.200 0.926 0.099 0.108 0.991 0.053 0.171
MSE.BSGS(outputSimple, Y=Y, X=X)
[1] 0.574

TCR.TPR.FPR.BSGS(outputSimple, r.true, 0.5)
$TCR
[1] 0.97

$TPR
[1] 1

$FPR
[1] 0.0323

## Output from the sample version of BSGS
```

```

outputSample$Iteration
[1] 3700
outputSample$TimeElapsed
  user  system elapsed
105.535   0.163 105.822
BSGS.PE(outputSample)$eta.est
    1     2     3     4     5     6     7     8     9     10
1.00000 0.90514 0.04189 0.02459 0.88000 0.01000 0.00486 0.91135 0.00486 0.00730
MSE.BSGS(outputSample, Y=Y, X=X)
[1] 2.16
TCR.TPR.FPR.BSGS(outputSample, r.true, 0.5)
$TCR
[1] 0.97

$TPR
[1] 1

$FPR
[1] 0.0323

```

One may be interested in how the different stopping points would affect the computational effort and model estimation. We thus compare the computational times and accuracy of parameter estimation in terms of TCR, TPR, FPR, and MSE for two different sampling approaches by using different MCSEs to stop the simulation and the results are shown in Tables 1 and 2. It has been found that for the case of $n = 50$ and $p = 100$ the simple version would converge faster than the sample version. However, the sample version would require less effort to produce the sample than the simple version.

MCSE	# of iterations	runtime (in sec)	MSE	TCR	TPR	FPR	$\hat{\sigma}^2$
0.5	1000	239	0.58	0.97	1	0.03	1.143
0.25	1000	239	0.58	0.97	1	0.03	1.143
0.1	1100	263	0.55	0.98	1	0.02	1.112

Table 1: Results based on simple version procedure for sparse group selection for different stopping points.

MCSE	# of iterations	runtime (in sec)	MSE	TCR	TPR	FPR	$\hat{\sigma}^2$
0.5	3700	105	2.16	0.97	1	0.03	1.418
0.25	9500	226	0.58	0.97	1	0.03	1.418
0.1	31200	4148	0.48	0.97	1	0.03	1.047

Table 2: Results based on sample version procedure for sparse group selection for different stopping points.

We further investigate the accuracy of parameter estimations and the selection of the variables in terms of MSE, TCR, TPR, and FPR for the two sampling approaches when the number of covariates increases but each group has 10 variables. We consider $p = 300, 500$, and 1000 and the simulation is terminated when the MCSE of the estimate of σ^2 is less than 0.5 for illustration. One may use different values of MCSE, but more computational time may be needed. Table 3 shows that the simulation stops with the same number of iterations and the parameter estimates and accuracy of variable selection show little difference. On the other hand, the results in Table 4 for the sample version GWGS show that more iterations are needed, under the same stopping rule. The two approaches thus perform equally well on the selection of variables, but by comparing MSEs it is evident that the simple version of GWGS outperforms with regard to the predictions. Although the sample version has less computational intensity, it needs more iterations to achieve the stopping point. If one is interested in selecting important variables, both approaches are effective. But if one is interested in choosing a model which fits the data well, it is thus suggested one uses the simple version approach.

Next, we compare the computational time when the number of variables in the group increases. We perform an experiment in which there are seven groups, each containing 15 variables. The assignments of hyperparameters are the same as those in Simulation II. Table 5 shows the computational time plus

# of covariates	# of iterations	runtime (in sec)	MSE	TCR	TPR	FPR	$\hat{\sigma}^2$
300	1000	212	0.56	0.95	0.86	0.05	1.94
500	1000	273	0.99	0.98	1	0.02	1.803
1000	1000	263	0.55	0.98	1	0.02	1.112

Table 3: Results based on the simple version procedure for sparse group selection for different numbers of covariates, with pair-correlation equal to 0.

# of covariates	# of iterations	runtime (in sec)	MSE	TCR	TPR	FPR	$\hat{\sigma}^2$
300	2100	127	1.63	0.99	1	0.01	1.819
500	14600	2258	4.73	0.97	1	0.03	1.601
1000	39900	39567	8.90	0.99	1	<0.01	2.158

Table 4: Results based on the sample version procedure for sparse group selection for different numbers of covariates, with pair-correlation equal to 0.

the model estimations. It can be seen that the sample version is strongly recommended when the number of variables within a group is greater than 15.

Sampling version	# of iterations	runtime (in sec)	MSE	TCR	TPR	FPR	$\hat{\sigma}^2$
Simple	1000	8330	0.11	0.93	1	0.08	0.90
Sample	2400	71	0.75	0.99	1	0.01	1.06

Table 5: Comparison between simple and sample versions for sparse group selection when the number of covariates within a group is 15.

A real economic example

This subsection further illustrates the functions in the **BSGS** based on an economic dataset from Rose and Spiegel [Rose and Spiegel \(2010, 2011, 2012\)](#) which is available at <http://faculty.haas.berkeley.edu/arose>. The response variable is the 2008-2009 growth rate for the crisis measure. Rose and Spiegel originally consider 119 explanatory factors for the crisis for as many as 107 countries, but there are missing data for a number of these.

```
require(BSGS)
## the whole data set
data(Crisis2008)
```

To maintain a balanced data set, we use 51 variables for a sample of 72 countries. For more information about the balanced data, please see the description of ‘Crisis2008’ in the **BSGS**. The balanced data is then analyzed to illustrate the main sampling function **BSGS.Simple** to simulate the sample from the posterior distribution. In the analysis, we demean the response so that it is not necessary to include the intercept into the design matrix. All variables are standardized except the dummy variables.

```
set.seed(1)
data(Crisis2008BalancedData)

var.names <- colnames(Crisis2008BalancedData)[-1]
country.all <- rownames(Crisis2008BalancedData)
cov.of.interest <- colnames(Crisis2008BalancedData)[-1]

Y <- Crisis2008BalancedData[, 1]
Y <- Y - mean(Y)
X <- Crisis2008BalancedData[, -1]

if (NORMALIZATION) {
  dummy.variable <- cov.of.interest[lapply(apply(X, 2, unique), length) == 2]
  non.dummy.X <- X[, !(colnames(X) %in% dummy.variable)]
```

```
X.normalized <- apply(non.dummy.X, 2, function(XX) (XX - mean(XX))/sd(XX))
X[, !(colnames(X) %in% dummy.variable)] <- X.normalized
}
```

As discussed in Ho (in press), these variables can be classified into the nine theoretical groups of the crisis' origin (the number in parentheses indicates the number of variables considered in the group): principal factors (10), financial policies (three), financial conditions (four), asset price appreciation (two), macroeconomic policies (four), institutions (11), geography (four), financial linkages (one), and trade linkages (12). Based on this information, we assign a group index to each variable.

```
Group.Index <- rep(1:9, c(10, 3, 4, 2, 4, 11, 4, 1, 12))
```

Since the number of covariates within the group is moderate, it is recommended that the simple version GWGS be applied to generate the samples. In this example, we have tested different values of τ^2 and finally we set $\tau^2 = 10$ due to the minimal MSE's. Here the stopping rule is when the MCSE of the estimate of σ^2 is less than or equal to 0.1 for the simple version. No prior information is provided to indicate which group or variable is more important, so we let $\theta = 0.5$ and $\eta = 0.5$. We let $\nu = 0$ and $\lambda = 1$ resulting in a non-informative prior for σ^2 . In each group, we will update parameters "Num.of.Iter.Inside.CompWise = 100" times within a group for stability.

```
Num.Of.Iteration <- 1000
Num.of.Iter.Inside.CompWise <- 100
num.of.obs <- nrow(X)
num.of.covariates <- ncol(X)
num.of.groups <- length(unique(Group.Index))
nu <- 0
lambda <- 1
beta.est <- lm(Y ~ X - 1)$coef
beta.est[is.na(beta.est)] <- 0
beta.value <- beta.est
tau2.value <- rep(1, num.of.covariates)

sigma2.value <- 1

r.value <- rep(0, num.of.covariates)
eta.value <- rep(0, num.of.groups)

tau2.value <- rep(10, num.of.covariates)

rho.value <- rep(0.5, num.of.covariates)
theta.value <- rep(0.5, num.of.groups)

MCSE.Sigma2.Given <- 0.5

outputCrisis2008 <- BSGS.Simple(Y, X, Group.Index, r.value, eta.value, beta.value,
tau2.value, rho.value, theta.value, sigma2.value, nu, lambda,
Num.of.Iter.Inside.CompWise, Num.Of.Iteration, MCSE.Sigma2.Given)
```

The posterior probabilities of η_l and γ_{li} are shown in Figure 1. Based on the median probability criterion, the group (or variable) whose posterior probability is larger than or equal to 0.5 is selected as an important group (or variable). It is found that only the groups, "Financial Policies" and "Trade Linkages" are considered to have an influence on the economic crisis. Moreover, within each important group, we also find that only some of the variables may make a contribution to explain the response.

Summary

This paper illustrated the usage of a new R package, **BSGS**, for identifying the important groups of variables and important variables in linear regression models. Furthermore, **BSGS** can be easily implemented with problems of a large p and small n . The grouping idea is also applicable to other regression and classification settings, for example, the multi-response regression and multi-class classification problems. We envision future additions to the package that will allow for extensions to these models.

We are confident that this package can be applied to many important real-world problems by keeping flexibility with regard to selecting variables within a group based on the hierarchical assignment of two layers of indicator variables. For instance, in the gene-set selection problem, a biological

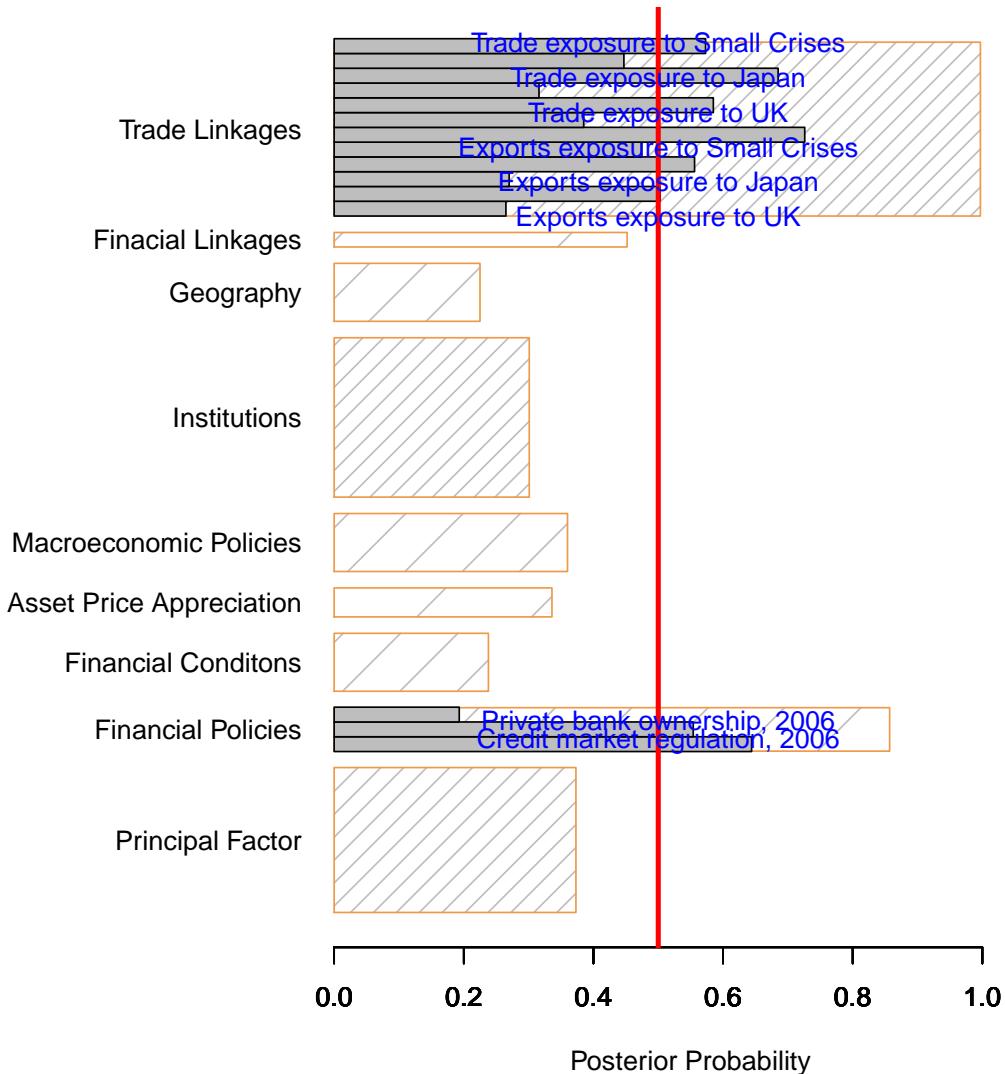


Figure 1: The group selection results for the cross-country severity of the crisis. The orange border indicates the posterior probability for the group selection, and the gray bar indicates the posterior probability for variable selection in the selected groups.

pathway may be related to a certain biological process, but it may not necessarily mean all the genes in the pathway are all related to the biological process. We may want not only to remove unimportant pathways effectively, but also identify important genes within important pathways.

Acknowledgment

This work is partially supported by the Ministry of Science and Technology under grant MOST 103-2633-M-006 -002 (Lee); the Ministry of Science and Technology under grant MOST 103-2118-M-006-002-MY2 (Chen), the Mathematics Division of the National Center for Theoretical Sciences in Taiwan.

Bibliography

- M. Barbieri and J. O. Berger. Optimal predictive model selection. *Annals of Statistics*, 32:870–897, 2004. [p124]
- S. Brooks, A. Gelman, G. L. Jones, and X. Meng. *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC, Boca Raton, 2011. [p124]

- S. Chatterjee, K. Steinhauer, A. Banerjee, S. Chatterjee, and A. Ganguly. Sparse group lasso: Consistency and climate applications. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, chapter 4, pages 47–58. 2012. [p123]
- R.-B. Chen, C.-H. Chu, T.-Y. Lai, and Y.-N. Wu. Stochastic matching pursuit for Bayesian variable selection. *Statistics and Computing*, 21:247–259, 2011. [p122, 123, 125]
- R.-B. Chen, C.-H. Chu, S. Yuan, and Y. N. Wu. Bayesian sparse group selection. *Journal of Computational and Graphical Statistics*, in press. doi: 10.1080/10618600.2015.1041636. [p122, 123, 124]
- H. Chipman, M. Hamada, and C. Wu. A Bayesian variable-selection approach for analyzing designed experiments with complex aliasing. *Technometrics*, 39:372–381, 1997. [p124]
- J. M. Flegal, M. Haran, and G. L. Jones. Markov chain Monte Carlo: Can we trust the third significant figure? *Statistical Science*, 23:250–260, 2008. [p124]
- E. I. George and R. E. McCulloch. Variable selection via Gibbs sampling. *Journal of the American Statistical Association*, 88:881–889, 1993. [p122, 124]
- E. I. George and R. E. McCulloch. Approaches for Bayesian variable selection. *Statistica Sinica*, 7: 339–373, 1997. [p124, 125]
- J. Geweke. Variable selection and model comparison in regression. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics*, volume 5, pages 609–620. Oxford University Press, Oxford, 1996. [p123]
- T.-K. Ho. Looking for a needle in a haystack: Revisiting the cross-country causes of the 2008–09 crisis by Bayesian model averaging. *Economica*, in press. [p131]
- A. K. Rose and M. M. Spiegel. Cross-country causes and consequences of the 2008 crisis: International linkages and American exposure. *Pacific Economic Review*, 15:340–363, 2010. [p130]
- A. K. Rose and M. M. Spiegel. Cross-country causes and consequences of the crisis: An update. *European Economic Review*, 55:309–324, 2011. [p130]
- A. K. Rose and M. M. Spiegel. Cross-country causes and consequences of the 2008 crisis: Early warning. *Japan and the World Economy*, 24:1–16, 2012. [p130]
- N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22:231–245, 2013. [p122]

Kuo-Jung Lee
Assistant Professor
Department of Statistics, National Cheng-Kung University
Tainan, Taiwan 70101
Taiwan
kuojunglee@mail.ncku.edu.tw

Ray-Bing Chen
Professor
Department of Statistics, National Cheng-Kung University
Tainan, Taiwan 70101
Taiwan
rbchen@mail.ncku.edu.tw

ClustVarLV: An R Package for the Clustering of Variables Around Latent Variables

by Evelyne Vigneau, Mingkun Chen and El Mostafa Qannari

Abstract The clustering of variables is a strategy for deciphering the underlying structure of a data set. Adopting an exploratory data analysis point of view, the Clustering of Variables around Latent Variables (CLV) approach has been proposed by [Vigneau and Qannari \(2003\)](#). Based on a family of optimization criteria, the CLV approach is adaptable to many situations. In particular, constraints may be introduced in order to take account of additional information about the observations and/or the variables. In this paper, the CLV method is depicted and the R package **ClustVarLV** including a set of functions developed so far within this framework is introduced. Considering successively different types of situations, the underlying CLV criteria are detailed and the various functions of the package are illustrated using real case studies.

Introduction

For the clustering of observations, a large number of packages and functions are available within the R environment. Besides the base package **stats** and the recommended package **cluster** ([Maechler et al., 2015](#)), about one hundred R packages have been listed in the CRAN Task View: Cluster Analysis and Finite Mixture Models ([Leisch and Grün, 2015](#)). This is representative of the huge number of applications in which the user is interested in making groups of similar cases, instances, subjects, ..., i.e., in clustering of the observations, in order to exhibit a typology within the population under study. The number of R packages, or R functions, specifically dedicated to the clustering of variables is much smaller.

As a matter of fact, clustering methods (e.g., hierarchical clustering or k-means clustering) are almost always introduced in standard text books using the Euclidean distance between a set of points or observations. Thus, it is not so easy to imagine situations in which defining clusters of variables makes sense. Would it be interesting, in an opinion survey, to identify, *a posteriori*, groups of questions, and not only clusters of people? The answer to this question is yes, particularly if the number of questions or items is large. Indeed, by merging connected questions, it is possible to identify latent traits, and, as a by-product, improve the interpretation of the outcomes of the subsequent analyses. In another domain, the recent progress in biotechnology enables us to acquire high-dimensional data on a few number of individuals. For instance, in proteomics or metabolomics, recent high-throughput technologies can gauge the abundance of thousands of proteins or metabolites simultaneously. In this context, identifying groups of redundant features appears to be a straightforward strategy in order to reduce the dimensionality of the data set. Based on DNA microarray data, gene clustering is not a new issue. It has usually been addressed using hierarchical clustering algorithms based on similarity indices between each pair of genes defined by their linear correlation coefficient, the absolute value or the squared value of the linear correlation coefficient (see, among others, [Eisen et al. 1998](#); [Hastie et al. 2000](#); [Park et al. 2007](#); [Tolosi and Lengauer 2011](#)). We can also mention some specific methods for gene clustering such as the diametrical clustering algorithm of [Dhillon et al. \(2003\)](#) or a clustering method based on canonical correlations proposed by [Bühlmann et al. \(2013\)](#). However, to the best of our knowledge, there is no implementation of these methods in R.

We introduce the **ClustVarLV** package ([Vigneau and Chen, 2015](#)) for variable clustering based on the Clustering of Variables around Latent Variables (CLV) approach ([Vigneau and Qannari, 2003](#)). The CLV approach shares several features with the already mentioned approaches of [Dhillon et al. \(2003\)](#) and [Bühlmann et al. \(2013\)](#), as well as with the clustering approach of [Enki et al. \(2013\)](#) for constructing interpretable principal components. It is also worth mentioning that the Valuer's procedure available in SAS ([Sarle, 1990](#)) has some common features with the CLV functions of the **ClustVarLV** package. All these methods are more or less connected to linear factor analysis. They could be viewed as empirical descriptive methods, unlike model-based approaches such as the likelihood linkage analysis proposed by [Kojadinovic \(2010\)](#) for the clustering of continuous variables. Let us note that there is a similar R package, **ClustOfVar** ([Chavent et al., 2013](#)), which proposed an implementation of some of the algorithms described in [Vigneau and Qannari \(2003\)](#). However the **ClustOfVar** package does not have the same functionalities as the **ClustVarLV** package. The comparison of these two related packages will be more detailed in a subsequent section.

Other interesting packages for clustering can also be cited: **clere** ([Yengo and Canoui, 2014](#)) for

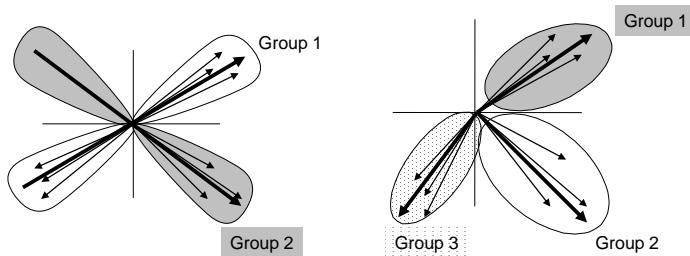


Figure 1: The two possible situations in CLV. On the left side: *directional groups* (positively and negatively highly correlated variables). On the right side: *local groups* (positively highly correlated variables). Arrows indicate variables and bold arrows indicate latent components associated with the various groups.

simultaneous variables clustering and regression; **biclust** (Kaiser et al., 2015) which provides several algorithms to find biclusters in two-dimensional data; **pvclust** (Suzuki and Shimodaira, 2014) which performs hierarchical cluster analysis and automatically computes p -values for all clusters in the hierarchy. This latter package considers the clustering of the columns of a data matrix (for instance, DNA microarray data) and computes (by default) the correlation coefficients between the columns to be clustered. Similarly, the function **varclus()** in the **Hmisc** (Harrell Jr et al., 2015) package can be used for performing a hierarchical cluster analysis of variables, using the Hoeffding D statistic, the squared Pearson or Spearman correlations, or the proportion of observations for which two variables are both positive as similarity measures. For **pvclust** and the function **varclus()** in package **Hmisc**, the clustering is done by the **hclust()** function.

In the following sections, the objective and principle of the CLV approach will be introduced in a comprehensive manner. The main functions of the **ClustVarLV** package for the implementation of the method will be listed. Next, different situations, associated with various forms of the CLV criterion, will be discussed and illustrated. The first setting will be the case of directional groups of variables for data dimension reduction and the identification of simple structures. Another one will be to identify clusters of variables taking account of an external information.

Synthetic presentation of the CLV method

In order to investigate the structure of a multivariate dataset, Principal Components Analysis (PCA) is usually used to find the main directions of variation. This can be followed by a rotation technique such as Varimax, Quartimax, ... (Jolliffe, 2002) in order to improve the interpretability to the principal components. The CLV approach is an alternative strategy of analysis whereby the correlated variables are lumped together and, within each cluster, a latent (synthetic) variable is exhibited. This latent variable is defined as a linear combination of only the variables belonging to the corresponding cluster. From this standpoint, CLV has the same objective as the Sparse Principal Component Analysis (Zou et al., 2006) which aims at producing modified principal components with sparse loadings.

The CLV approach (Vigneau and Qannari, 2003) is based on the maximization of a set of criteria which reflect the linear link, in each cluster, between the variables in this cluster and the associated latent variable. These criteria are related to the types of links between the observed and the latent variables that are of interest to the users, as illustrated in Figure 1.

- The first case (left hand panel in Figure 1) is to define directional groups, so that the observed variables that are merged together are as much as possible related to the group latent variable, no matter whether their correlation coefficients are positive or negative. In this case, the link between the observed and the latent variables are evaluated by means of the squared correlation coefficient between the variables, and the criterion considered for maximization is:

$$T = \sum_{k=1}^K \sum_{j=1}^p \delta_{kj} \operatorname{cov}^2 r(\mathbf{x}_j, \mathbf{c}_k) \text{ with } \operatorname{var}(\mathbf{c}_k) = 1 \quad (1)$$

where \mathbf{x}_j ($j = 1, \dots, p$) are the p variables to be clustered. These variables are assumed to be centered. In Equation (1), K is the number of clusters of variables, denoted G_1, G_2, \dots, G_K ; \mathbf{c}_k ($k = 1, \dots, K$) is the latent variable associated with cluster G_k and δ_{kj} reflects a crisp membership, with $\delta_{kj} = 1$ if the j th variable belongs to cluster G_k and $\delta_{kj} = 0$, otherwise.

- The second case (right hand panel in Figure 1) is to define local groups for which each variable shows a positive correlation with their associated latent variable. This case entails that negative correlation coefficients imply disagreement. Therefore, the CLV criterion is based on the correlation coefficient and the criterion to be maximized is:

$$S = \sum_{k=1}^K \sum_{j=1}^p \delta_{kj} \operatorname{cov}(\mathbf{x}_j, \mathbf{c}_k) \text{ with } \operatorname{var}(\mathbf{c}_k) = 1 \quad (2)$$

with the same notations as for Equation (1).

Moreover, as will be illustrated in Section “[Clustering of variables with external information](#)” the CLV criteria given in Equations (1) or (2) could be slightly modified, by introducing a constraint on the latent variables, in order to take account of additional information on the variables to be clustered.

It is worth noting that the well known VARCLUS procedure ([Sarle, 1990](#)), implemented in the SAS/STAT software, also offers these two options. However, in VARCLUS, no optimization criterion for the determination of the groups of variables is clearly set up. Moreover, this method of analysis consists of a rather complicated divisive hierarchical procedure.

From a practical standpoint, the CLV approach is based on a partitioning algorithm, described in [Vigneau and Qannari \(2003\)](#), akin to the k-means algorithm. However, this partitioning algorithm requires, on the one hand, the choice of the number K of clusters and, on the other hand, the initialization of the iterative process. To address these issues, our recommendation is to start by performing a hierarchical cluster analysis, with aggregating rules detailed in [Vigneau and Qannari \(2003\)](#). The first interest is to set up a dendrogram and a graph showing the evolution of the aggregation criterion between two successive partitions. This should help the user choosing the appropriate number of clusters. The second interest is that the clusters from the hierarchical analysis give reasonable initial partitions for performing the partitioning algorithm. This process of running a partitioning algorithm using the outcomes of the hierarchical clustering as starting point is called consolidation in the French literature ([Lebart et al. 2000; Warms-Petit et al. 2010](#)).

Overview of the functions in the **ClustVarLV** package

The list of the functions in the **ClustVarLV** package, that the users can call, is given in Table 1. The two main functions for the implementation of the CLV algorithms are **CLV()** and **CLV_kmeans()**.

The **CLV()** function performs an agglomerative hierarchical algorithm followed by a consolidation step performed on the highest levels of the hierarchy. The number of solutions considered for the consolidation can be chosen by the user (parameter `nmax`, equal to 20 by default). The consolidation is based on an alternated optimization algorithm, i.e., a k-means partitioning procedure, which is initialized by cutting the dendrogram at the required level. Alternatively, the user may choose to use the **CLV_kmeans()** function which is typically a partitioning algorithm for clustering the variables into a given number, K , of clusters. It involves either repeated random initializations or an initial partition of the variables supplied by the user. This second function may be useful when the number of variables is larger than a thousand because in this case the hierarchical procedure is likely to be time consuming (this point will be addressed in Section “[The CLV\(\) and CLV_kmeans\(\) functions](#)”). When the number of variables does not exceed several hundred, the dendrogram which can be drawn from the output of the **CLV()** function provides a useful tool for choosing an appropriate number, K , for the size of the partition of variables.

The two functions, **CLV()** and **CLV_kmeans()**, include a key parameter, which has to be provided by the user, with the data matrix. This parameter, called `method`, indicates the type of groups that are sought: `method = "directional"` or `method = 1` for directional groups and `method = "local"` or `method = 2` for local groups (Figure 1). These functions make it possible to cluster the variables of the data matrix (argument `X`) considered alone, or by taking account of external information available on the observations (argument `Xr`) or external information available for the variables themselves (argument `Xu`). A third “CLV” function has been included in the **ClustVarLV** package: It is the **LCLV** function which can be used when external information is available for both the observations and the variables (see Section “[Clustering of variables with directional groups](#)” for more details).

The other functions in the **ClustVarLV** package (version 1.4.1) are mainly utility and accessor functions providing additional outputs useful for the interpretation of the clustering results. Their usage will be illustrated with various case studies that will be discussed hereinafter.

Functions	Description
<i>"Clustering" functions</i>	
CLV	Hierarchical clustering of variables with consolidation
CLV_kmeans	Kmeans algorithm for the clustering of variables
LCLV	L-CLV for L-shaped data
<i>Methods for 'clv' objects</i>	
plot	Graphical representation of the CLV clustering stages
print	Print the CLV results
<i>Methods for 'lclv' objects</i>	
plot	Graphical representation of the LCLV clustering stages
print	Print the LCLV results
<i>Utility functions for the 'clv' and 'lclv' objects</i>	
summary	Method providing the description of the clusters of variables
plot_var	Representation of the variables and their group membership
get_partition	To get the clusters of variables
get_comp	To get the latent variables associated with each cluster
get_load	To get the loadings of the external variables in each cluster
<i>Miscellaneous</i>	
stand_quali	Standardization of the qualitative variables
data_biplot	Biplot for the dataset

Table 1: List of the functions in the **ClustVarLV** package.

Clustering of variables with directional groups

As indicated above, when the user chooses `method = "directional"` in the `CLV()` or `CLV_kmeans()` function, the criterion considered for optimization is the criterion T defined in Equation (1).

It can be shown (see for instance Vigneau and Qannari 2003) that when the maximum of the criterion T is reached, the latent variable c_k , in cluster G_k , is the first normalized principal component of matrix \mathbf{X}_k , the dataset formed of the variables belonging to G_k . Thus, the optimal value of $T^{(K)}$, for a partition into K groups, is the sum of the largest eigenvalues respectively associated with the variance-covariance matrices $\frac{1}{n}\mathbf{X}_k'\mathbf{X}_k$, with $k = 1, \dots, K$. The ratio between $T^{(K)}$ and $T^{(p)}$ provides the percentage of the total variance explained by the K CLV latent variables. Even if the K CLV latent variables, which are not necessarily orthogonal, cannot take account of as much total variance as the K first principal components, they may be more relevant for deciphering the underlying structure of the variables than the first principal components. Moreover, they are likely to be more easily interpretable. Enki et al. (2013) have also addressed the issue of identifying more interpretable principal components and proposed a procedure which bears some similarities with the CLV method.

First illustrative example: Identification of block structure, and underlying latent components, into a set of variables

We consider data from a French Research Project (AUPALESENS, 2010–2013) dealing with food-behavior and nutritional status of elderly people. More precisely, we selected the psychological behavior items, which are part of a large questionnaire submitted to 559 subjects. As a matter of fact, the 31 psychological items were organized into five blocks, each aiming to describe a given behavioral characteristic: emotional eating (E) with six items, external eating (X) with five items, restricted eating (R) with five items, pleasure for food (P) with five items, and self esteem (S) with ten items. Detailed description and analysis of the emotional, external and restricted eating items for this study are available in Bailly et al. (2012).

The `CLV()` function was performed on the data matrix, \mathbf{X} , which merges the 31 psychological items, using the following code:

```
R> library("ClustVarLV")
R> data("AUPA_psycho", package = "ClustVarLV")
```

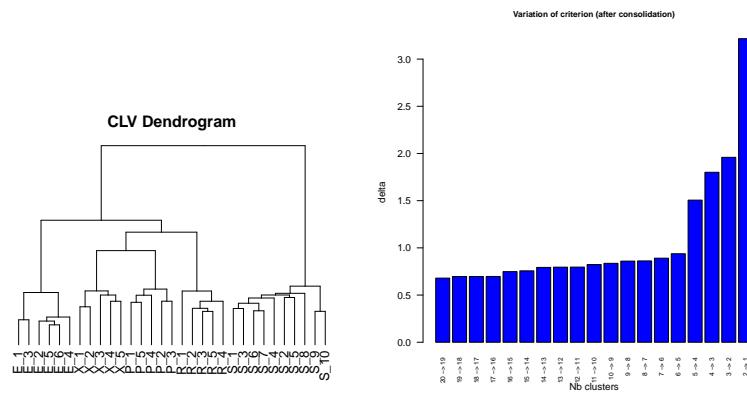


Figure 2: Graphs obtained by the clustering into directional groups of the psychological variables. On the left side, the dendrogram of the hierarchical clustering stage; on the right side, the variation of the clustering criterion after consolidation of the partitions by means of the partitioning algorithm.

```
R> res.clv <- CLV(AUPA_psycho, method = "directional", sX = TRUE)
R> plot(res.clv, type = "dendrogram")
R> plot(res.clv, type = "delta", cex = 0.7)
```

The dendrogram and the graph showing the variation of the clustering criterion when passing from a partition into K clusters to a partition into $(K - 1)$ clusters ($\Delta = T^{(K)} - T^{(K-1)}$) are shown in Figure 2. From the graph of Δ , it can be observed that the criterion clearly jumps when passing from five to four clusters. This means that the loss in homogeneity of the clusters is important with four clusters and that a partition into five clusters should be retained. The partition into $K = 5$ groups, available with `get_partition(res.clv, K = 5)`, perfectly retrieved the five blocks of psychological traits.

The summary method for 'clv' objects provides a description of the clusters:

```
R> summary(res.clv, K = 5)

      Group.1   Group.2   Group.3   Group.4   Group.5
nb:       6        5        5        5       10
prop_within: 0.6036  0.4077  0.4653  0.388   0.3614
prop_tot: 0.4368

Group1 cor in group |cor|next group
E5     0.85          0.25
E4     0.80          0.34
E6     0.80          0.25
E2     0.79          0.25
E3     0.73          0.31
E1     0.68          0.29

Group2 cor in group |cor|next group
X2     0.76          0.38
X4     0.67          0.30
X5     0.65          0.19
X1     0.58          0.17
X3     0.51          0.22

Group3 cor in group |cor|next group
R5     0.77          0.25
R3     0.76          0.21
R2     0.71          0.23
R4     0.66          0.11
R1     0.47          0.14

      Group4   Group5
cor in group |cor|next group
P1     0.72          0.18
P3     0.63          0.14
P2     0.61          0.10
P4     0.58          0.14
P5     0.57          0.19

      Group5
cor in group |cor|next group
S3     0.70          0.21
S1     -0.68         0.10
S6     -0.66         0.17
S7     -0.65         0.17
S10    0.65          0.07
S5     0.55          0.12
S4     -0.53         0.10
S9     0.53          0.10
S2     -0.51         0.14
S8     0.49          0.23
```

The homogeneity values within each cluster, assessed by the percentages of the total variance of the variables belonging to the cluster explained by the associated latent variable, are 60.4%, 40.8%, 46.5%, 38.8%, 36.1% respectively (the Cronbach's alphas are 0.87, 0.63, 0.71, 0.60, and 0.80, respectively). Furthermore, the five group latent variables make it possible to explain 43.7% of the total variance of all the $p = 31$ observed variables. For each variable in a cluster, its correlation coefficient with its own group latent variable and its correlation coefficient with the next nearest group latent variable are also given. Each item is highly correlated with its group latent variable.

Compared with the standardized PCA of X , five principal components (PCs) are required for retrieving 45.1% of the total variance, whereas four PCs account for 40.5% of the total variance. Moreover, it turned out that the interpretation of the first five PCs was rather difficult. If we consider

all the loadings larger than 0.3, in absolute value, the first PC, *PC1*, seems to be associated with all the items "E", X2, X3, R2 and S8; *PC2* is related to P1 and all the items "S" except S8, *PC3* to R1 only, *PC4* to X4, R3, R4, R5 and P3, and *PC5* to X1 and X5. It is known that rotation (by means of orthogonal or oblique transformations) may enhance the interpretation of the factors. In this case study, using a Varimax transformation, the five rotated PCs can be associated with one of the predefined blocks of items. However, the rotated principal components make it possible to retrieve the "true" structure if, and only if, the correct number of dimensions for the subspace of rotation is selected. This may be an impediment since the determination of the appropriate number of components is a tricky problem. In the case study at hand, various rules (Jolliffe, 2002) led to two, four or eight PCs. By contrast, the variation of the CLV criterion performs well for identifying the correct number of groups.

In another domain (i.e., the Health sector), Lovaglio (2011) pointed out that, within the Structural Equation Modeling framework, the first step which consists of building the measurement models could be based on the CLV technique. He showed that, considering a formative way, the subset of variables obtained by means of CLV() led to a better recovery of the original configuration, followed by VARCLUS based on PCA. This was far from being the case with the selection of variables on the basis of the outcomes of PCA or PCA with Varimax rotation.

Second illustrative example: Clustering of quantitative and qualitative variables

Chavent et al. (2012) proposed an R package, named **ClustOfVar**, which aims at clustering variables, with the benefit of allowing the introduction of quantitative variables, qualitative variables or a mix of those variables. The approach is based on a homogeneity criterion which extends the CLV criterion (Eq.1). More precisely, the correlation ratio (between groups variance to total variance ratio) of each qualitative variable and the latent variable in a cluster are included in the criterion in addition to the squared correlation coefficients used for the quantitative variables. In practice, for defining the partition of the variables and the latent variables within each cluster, the algorithms described in Chavent et al. (2012) are the same as those given in Vigneau and Qannari (2003) and Vigneau et al. (2006), with a small variation: The latent variables are derived from a PCAMIX model (Saporta, 1990; Kiers, 1991; Pagès, 2004) instead of a PCA model.

The strategy of clustering quantitative and qualitative variables raises the following question: Is it better to cluster qualitative variables along with the quantitative variables or to break down each qualitative variable into its categories and include these categories in a clustering approach such as CLV?

To answer this question, let us consider the dataset 'wine' provided in various packages (for instance, **ClustOfVar**, **FactoMineR**, Husson et al. (2015)). 21 french wines of Val of Loire are described by 29 sensory descriptors scored by wine professionals. Two nominal variables are also provided: the label of the origin (with three categories: "Saumur", "Bourgueil" and "Chinon") and the nature of the soil (with four categories: "Reference", "Env.1", "Env.2" and "Env.4"). The design of these two nominal variables is however not well-balanced. Chavent et al. (2012) considered only 27 quantitative variables (all the sensory descriptors except those regarding the global evaluation) and included the two qualitative variables. From the dendrogram obtained with the function hclustvar(), they retained six clusters. The summary of the partition into six clusters is shown below:

Cluster 1 :	squared loading	Cluster 4 :	squared loading
Odour.Intensity.before.shaking	0.76	Visual.intensity	0.86
Spice.before.shaking	0.62	Nuance	0.84
Odor.Intensity	0.67	Surface.feeling	0.90
Spice	0.54	Aroma.intensity	0.75
Bitterness	0.66	Aroma.persistency	0.86
Soil	0.78	Attack.intensity	0.77
		Astringency	0.79
Cluster 2 :	squared loading	Alcohol	0.68
Aroma.quality.before.shaking	0.78	Intensity	0.87
Fruity.before.shaking	0.85		
Quality.of.odour	0.79	Cluster 5 :	squared loading
Fruity	0.91	Plante	0.75
		Aroma.quality	0.84
Cluster 3 :	squared loading	Acidity	0.22
Flower.before.shaking	0.87	Balance	0.94
Flower	0.87	Smooth	0.92
		Harmony	0.87
Cluster 6 :	squared loading		
Phenolic	0.8		
Label	0.8		

The factor "Soil" was merged in the Cluster 1 with variables related to spicy sensation and the odor intensity. Its correlation ratio with the latent variable of this cluster is 0.78 (which corresponds to a *F*-ratio = 19.73 with a *p*-value = 9E-6). The factor "Label" was merged in the cluster 6 with the quantitative descriptor "Phenolic". The correlation ratio of "Label" with the latent variable of its

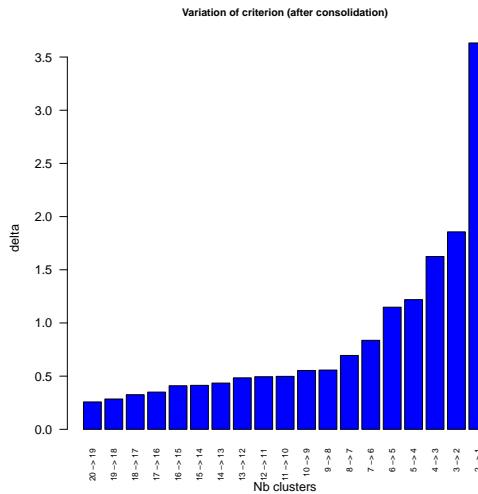


Figure 3: Graph showing the evolution of the aggregation criterion after consolidation.

cluster is 0.80 (F -ratio = 36.02, p -value = 5E-7).

In the **ClustVarLV** package, we propose to take account of the qualitative information, in addition to quantitative variables, by breaking down each qualitative variable into a matrix of indicators (G , say) of size $n \times M$, where M is the number of categories of the qualitative variable at hand. In the same vein as Multiple Correspondence Analysis (Saporta, 1990), we propose to standardize the matrix G . This leads us to the matrix $\tilde{G} = GD^{-1/2}$ where D is the diagonal matrix containing the relative frequency of each category. The utility function `stand_quali()` in **ClustVarLV** allows us to get the matrix \tilde{G} . Thereafter, the matrix submitted to the `CLV()` function is simply the concatenation of the standardized matrix of the quantitative variables and all the standardized blocks associated with each qualitative variables. The following code was used:

```
R> library("ClustVarLV")
R> data("wine", package = "FactoMineR")
R> X.quant <- wine[, 3:29]
R> X.quali <- wine[, 1:2]
R> Xbig <- cbind(scale(X.quant), stand_quali(X.quali))
R> resclv <- CLV(Xbig, method = "directional", sX = FALSE)
R> plot(resclv, "delta")
```

From the graph showing the evolution of the aggregation criterion (Figure 3), two, four, six or even eight clusters could be retained.

The partition into six clusters is described as follows:

```
R> summary(resclv, K = 6)

Group1          cor in group |cor|next group   Group4          cor in group |cor|next group
Odour.Intensity.before.shaking  0.87      0.63   Surface.feeling  0.95      0.80
Soil.Env.4       0.86      0.43   Intensity        0.94      0.82
Odour.Intensity  0.82      0.69   Visual.intensity 0.93      0.64
Spice.before.shaking  0.80      0.32   Aroma.persistency 0.93      0.76
Bitterness       0.80      0.49   Nuance          0.92      0.63
Spice            0.73      0.40   Astringency       0.89      0.70
                                         Attack.intensity 0.88      0.74
                                         Aroma.intensity  0.87      0.78
                                         Alcohol          0.83      0.59
Group2          cor in group |cor|next group   Group5          cor in group |cor|next group
Aroma.quality    0.93      0.64   Aroma.intensity  0.87      0.78
Balance          0.93      0.68   Alcohol          0.83      0.59
Smooth           0.92      0.77   Group5          cor in group |cor|next group
Quality.Odour    0.90      0.71   Phenolic         0.89      0.42
Harmony          0.90      0.87   Label.Bourgueuil -0.86      0.30
Aroma.quality.before.shaking 0.81      0.74   Label.Saumur     0.77      0.40
Plante            -0.78     0.42
Fruity.before.shaking  0.77      0.58   Group6          cor in group |cor|next group
Soil.Reference    0.70      0.46   Acidity          0.89      0.30
                                         Soil.Env.2      0.69      0.35
                                         Soil.Env.1      -0.68      0.37
                                         Label.Chinon    0.63      0.22
Group3          cor in group |cor|next group
Flower.before.shaking  0.93      0.44
Flower           0.93      0.35
```

It turns out that both functions, i.e., `hclustvar()` in **ClustOfVar** (hierarchical algorithm) and `CLV()` in **ClustVarLV** (hierarchical algorithm followed by a partitioning procedure), led to similar results for

the sensory descriptors.

The first group (Group 1) is related to the intensity of the odor with spicy notes, to which is associated the "Env.4" for the "Soil" factor, whereas it was globally "Soil" using `hclustvar()`. If we compare the correlation ratio of the qualitative variable "Soil" with its cluster latent variable using `hclustvar()` (i.e., 0.78), and the squared correlation coefficient of the category "Soil.Env.4" with its cluster latent variable using `CLV()` (i.e., 0.74), we can conclude that the contribution of the three other "Soil" categories to the correlation ratio is very small. This finding can easily be confirmed by means of a one-way ANOVA between the latent variable in the first cluster and the factor "Soil". Additionally, it can be shown that the correlation ratio (R^2) of a qualitative variable with respect to a quantitative variable (bfx , say) is equal to a weighted sum of the squared correlation coefficients of the indicators of its categories, given in \tilde{G} , with the quantitative variable, namely:

$$R^2 = \sum_{m=1}^M (1 - f_m) \text{cor}^2(\mathbf{g}_m, \mathbf{x}), \quad (3)$$

where \mathbf{g}_m is the indicator vector for the category m and f_m is the relative frequency. It follows that, the contribution of "Soil.Env.4" to the global R^2 of "Soil" in the first cluster found with `hclustvar()` is 85.4%. Thus, it appears that it is because of the specific nature of the soil in "Env.4" that the wines have a more intense odor and a more bitter flavor than the other wines.

The second group of attributes (Group 2) is related to the overall quality of the wines and it seems, from the results of `CLV()`, that the type "Reference" of the soil is likely to favor this quality. This was not observed with `hclustvar()` (see Cluster 5 in the summary of the partition into six clusters obtained with `hclustvar()`) because the qualitative variable "Soil" was globally associated with the Cluster 1.

Regarding the fifth groups of attributes (Group 5), the interpretation of the Phenolic flavor of some wines could be refined. If the "Label" was associated with the Phenolic attribute using `hclustvar()` (Cluster 6), the outputs of the `CLV()` function show that type "Saumur" was slightly more "Phenolic" than the type "Bourgeuil", whereas type "Chinon" (in Group 6) seems to have acid notes (but caution should be taken in this interpretation because of the small number of observations for "Chinon"). Nevertheless, it could be emphasized that the soil of "Env.2" is likely to give more acidity, unlike "Env.1". Finally let us notice that the acidity attribute was merged in the Cluster 5 obtained with `hclustvar()` but its squared loading to the latent variable of this cluster was relatively small.

Clustering of variables for local groups

In some specific situations, a negative correlation between two variables is considered as a disagreement. Therefore, these variables should not be lumped together in the same group.

Consider for instance the case of preference (or acceptability) studies in which consumers are asked to give a liking score for a set of products. For these data, the consumers play the role of variables whereas the products are the observations. The aim is to identify segments of consumers having similar preferences. This means positively correlated vectors of preference. In this situation, local groups are sought (illustrated in the right side of Figure 1) and the parameter `method = "local"` is to be used with the clustering functions of the `ClustVarLV` package. A case study developed in this context is available in [Vigneau et al. \(2001\)](#).

In other contexts, as in Near-Infrared spectroscopy or ^1H NMR spectroscopy, the CLV approach with local groups can be used for a first examination of the spectral data. [Jacob et al. \(2013\)](#) showed that this approach may help identifying spectral ranges and matching them with known compounds.

Technically, the identification of local groups of variables is performed in the CLV approach by the maximization of the criterion S given in Equation (2). As a result, it is easy to show that the maximal value is obtained, for a given number K of clusters, when each latent variable, \mathbf{c}_k , is proportional to the centroid variable $\bar{\mathbf{x}}_k$ of the variables in the cluster G_k .

Third illustrative example: Application to the segmentation of a panel of consumers

In order to illustrate the use of the `ClustVarLV` functions for the definition of local groups, let us consider the dataset 'apples_sh' available in the package ([Daillant-Spinnler et al., 1996](#)). Two types of information were collected: On the one hand, the sensory characterization, given by a trained panel, of 12 apple varieties from the Southern Hemisphere and, on the other hand, the liking scores, given by 60 consumers, for these varieties. We will consider the segmentation of the panel of consumers using the `CLV()` function with the option `method = "local"`:

```
R> library("ClustVarLV")
R> data("apples_sh", package = "ClustVarLV")
```

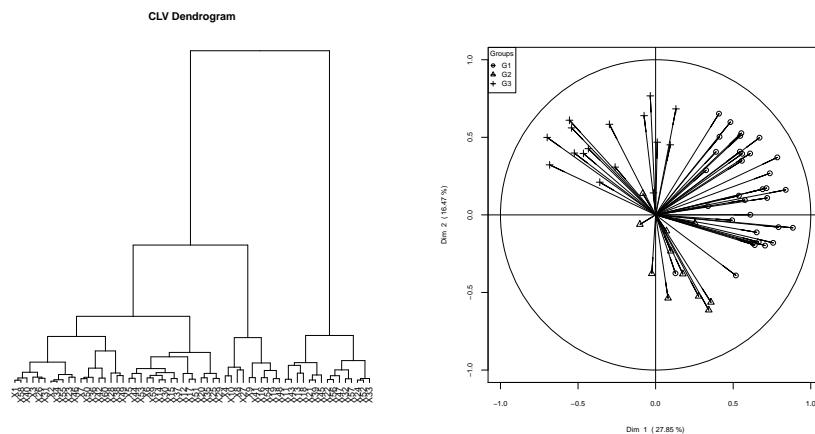


Figure 4: Segmentation of the panel of consumers for the apples case study. On the left side, the dendrogram of the hierarchical clustering; on the right side, the mapping of the consumers on the basis of the two first principal components, with group membership identification.

```
R> res.seg <- CLV(X = apples_sh$pref, method = "local")
R> plot(res.seg, "dendrogram")
R> table(get_partition(res.seg, K = 3))
R> plot_var(res.seg, K = 3, v_symbol = TRUE)
R> comp <- get_comp(res.seg, K = 3)
```

The dendrogram from `CLV()` given in the left side of Figure 4 suggests to retain three segments. These segments merged together 33, 11 and 16 consumers, respectively (after consolidation of the solution obtained by cutting the dendrogram at the chosen level). The `plot_var()` companion function makes it possible to show the group membership of each variable on a two dimensional subspace. The plot produced by this function (right side of Figure 4) is grounded on a PCA loading plot. By default, the two first principal components are considered, but the user may modify this option. In the previous code, the option '`v_symbol`' is set to `TRUE` in order to produce a figure readable in black and white. Without this option, color graphs will be produced, with or without the labels of the variables. In addition, the group latent variables may be extracted with the function `get_comp()`. They provide the preference profiles of the 12 apple varieties in the various consumer segments.

Clustering of variables with external information

The CLV approach has also been extended to the case where external information is available. The clustering of variables is achieved while constraining the group latent variables to be linear combinations of external variables.

Clustering with external information collected on the observations

Suppose that, in addition to the variables to be clustered, the observations are described by a second block of variables, \mathbf{Xr} (r stands for additional information collected on the rows of the core matrix \mathbf{X}) as in Figure 5. Both CLV criteria (Equations 1 and 2) can be used with the additional constraint that:

$$\mathbf{c}_k = \mathbf{Xr} \mathbf{a}_k \text{ with } \mathbf{a}_k' \mathbf{a}_k = 1 \quad (4)$$

for each latent variable \mathbf{c}_k with $k = 1, \dots, K$.

It can be shown (Vigneau and Qannari, 2003) that the solutions of the optimization problems are obtained when \mathbf{c}_k is the first component of a Partial Least Squares (PLS) regression of the group matrix \mathbf{X}_k on the external matrix \mathbf{Xr} , in the case of directional groups, or the first component of a PLS regression of the centroid variable $\bar{\mathbf{x}}_k$ on the external matrix \mathbf{Xr} , in the case of local groups.

External preference mapping is a domain in which the CLV approach with additional information on the observations has been successfully applied (Vigneau and Qannari, 2002). In addition to clustering the consumers according to the similarity of their preference scores as it was illustrated in the third illustrative example, the aim is also to segment the consumers while explaining their preferences by means of the sensory characteristics of the products. Thus, the segmentation and the modeling of the main directions of preference may be achieved simultaneously. If we consider again

the ‘apples_sh’ dataset, two matrices are available: ‘apples_sh\$pref’, the preference scores of the consumers, and ‘apples_sh\$senso’, the sensory characterization of the 12 apple varieties using 43 sensory attributes. The CLV() function includes parameters for taking account of such external block of information. Namely:

```
R> res.segext <- CLV(X = apples_sh$pref, Xr = apples_sh$senso, method = "local",
+                      sX = TRUE, sXr = TRUE)
R> table(get_partition(res.seg, K = 3), get_partition(res.segext, K = 3))
R> load3G <- get_load(res.segext, K = 3)
```

For a solution with three clusters, it turns out that the segments previously defined have been rearranged in order to take account of the sensory attributes of the apples. The loadings \mathbf{a}_k (for $k = 1, 2, 3$) of the sensory descriptors, which can be extracted using the utility function get_load(), made it possible to explain the difference in preference in each segment.

Clustering with additional information on the variables

When additional information is available on the variables, the CLV approach has also been adapted in order to take this information into account in the clustering process.

For instance, let us consider the problem of the clustering of spectral variables. Typically, a spectrometer (Near Infrared or a Nuclear Magnetic Resonance spectrometer) makes it possible to collect thousands of measurements at different spectral variables (wavelengths or chemical shifts). This leads to a large amount of information with high level of redundancy since close spectral points convey more or less the same information. Instead of trimming off close spectral points, the clustering of variables is a more effective way of identifying automatically spectral ranges associated with the same functional chemical groups (Vigneau et al., 2005). However, the fact that the variables correspond to successive wavelengths was not taken into account with the previous criteria given in Equation 1, or Equation 2. One can expect that adding information on the spectral structure of the variables can improve the quality of the clusters of variables, in the sense that variables within the same spectral range are more likely to be lumped together. The additional information to be considered in such a situation is related to the spectral proximity between the variables.

We denote by \mathbf{Z} , the matrix of the additional information on the variables. The rows in \mathbf{Z} are matched with the columns of the matrix \mathbf{X} . The CLV approach is performed by combining, in each cluster of variables, the \mathbf{X} - and the \mathbf{Z} -information. Namely, for a given cluster G_k , a new matrix \mathbf{P}_k is defined by:

$$\mathbf{P}_k = \mathbf{X}_k \mathbf{Z}_k, \quad (5)$$

where \mathbf{X}_k is the sub-matrix of \mathbf{X} formed by the p_k variables belonging to G_k , and similarly, \mathbf{Z}_k is a sub-matrix of \mathbf{Z} which involves only these p_k variables. Thus, \mathbf{P}_k can be viewed as a weighted version of \mathbf{X}_k , or as an interaction matrix between the \mathbf{X} - and \mathbf{Z} -information estimated within G_k . The nature of \mathbf{Z} , as well as the pretreatment applied, lead to one or the other point of view. The CLV criteria have been modified so that the latent variable in a cluster is a linear combination of the associated \mathbf{P}_k matrix. If we denote by \mathbf{t}_k the latent variable in the cluster G_k , the objective is either to maximize

$$T^Z = \sum_{k=1}^K \sum_{j=1}^{p_k} \delta_{kj} \text{cov}^2(\mathbf{x}_j, \mathbf{t}_k) \quad (6)$$

or

$$S^Z = \sum_{k=1}^K \sum_{j=1}^{p_k} \delta_{kj} \text{cov}(\mathbf{x}_j, \mathbf{t}_k) \quad (7)$$

with the constraints that:

$$\mathbf{t}_k = \mathbf{P}_k \mathbf{u}_k / \text{trace}(\mathbf{P}_k' \mathbf{P}_k)$$

and

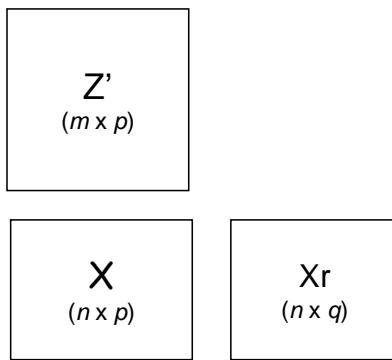
$$\mathbf{u}_k' \mathbf{u}_k = 1$$

The parameter \mathbf{Xu} in the CLV() function makes it possible to take account of the external information on the variables. A typical line of code in this case may be as:

```
R> resclv <- CLV(X = X, Xu = Z, method = "local", sX = FALSE)
```

Clustering with additional information on the observations and the variables

When external informations on observations and variables are available, \mathbf{X} , \mathbf{Xr} and \mathbf{Z} are associated either by their rows or by their columns, so that the three blocks of data may be arranged in the form

**Figure 5:** L-shaped data.

of an L (Figure 5). Therefore, the acronym *L-CLV* has been adopted and the `LCLV()` function included in the package **ClustVarLV** has been developed for this case.

The L-CLV approach directly stems from the previous extensions of the CLV approach. It consists in the maximization, in each cluster k (with $k = 1, \dots, K$) of the covariance between a pair of latent variables, \mathbf{c}_k and \mathbf{t}_k . \mathbf{c}_k is a linear combination of the co-variables measured on the observations, \mathbf{X} , and \mathbf{t}_k is a linear combination of the \mathbf{P}_k variables (defined in the previous section). The criterion to be maximized is:

$$\tilde{T} = \sum_{k=1}^K \text{cov}(\mathbf{c}_k, \mathbf{t}_k) \text{ with } \mathbf{c}_k = \mathbf{Xr} \mathbf{a}_k, \mathbf{t}_k = \mathbf{P}_k \mathbf{u}_k = \mathbf{X}_k \mathbf{Z}_k \mathbf{u}_k \text{ and } \mathbf{a}_k' \mathbf{a}_k = 1, \mathbf{u}_k' \mathbf{u}_k = 1 \quad (8)$$

or alternatively,

$$\tilde{T} = \sum_{k=1}^K \mathbf{u}_k' \mathbf{Z}_k' \mathbf{X}_k' \mathbf{Xr} \mathbf{a}_k \quad (9)$$

From the expression in Equation 9, it turns out that L-CLV bears strong similarities with the so-called L-PLS method (Martens et al., 2005). The main difference lies in the fact that L-CLV involves a clustering process and that a specific matrix mixing the \mathbf{X} , \mathbf{Xr} and \mathbf{Z} informations is considered and updated in each cluster.

Interested readers are referred to Vigneau et al. (2011) and Vigneau et al. (2014) for further details and an illustration of the procedure for the segmentation of a panel of consumers, according to their likings (\mathbf{X}), interpretable in terms of socio-demographic and behavioral parameters (given in \mathbf{Z}) and in relation with the sensory key-drivers (in \mathbf{Xr}). For such case studies the `LCLV()` function has been used with the following code (default options used):

```
R> resL <- LCLV(X = X, Xr = Xr, Xu = Z)
R> ak <- get_load(resL, K = 4)$loading_v
R> uk <- get_load(resL, K = 4)$loading_u
R> ck <- get_comp(resL, K = 4)$compc
R> tk <- get_comp(resL, K = 4)$compt
R> parti4G <- get_partition(resL, K = 4)
```

The function `get_load()` allows one to extract, for a given number of clusters K , the loadings \mathbf{a}_k and the loadings \mathbf{u}_k . This makes it possible to interpret the results in the light of the external informations. The latent variables \mathbf{c}_k and \mathbf{t}_k (for $k = 1, \dots, K$) are also available using the function `get_comp()` and the cluster membership of the variables are provided with the function `get_partition()`.

Technical considerations

The `CLV()` and `CLV_kmeans()` functions

The `CLV()` function was described for the clustering of variables, for local or directional groups, when external information is taken into account or not. This function involves two stages, a hierarchical algorithm followed by a non-hierarchical (or partitioning) algorithm. As a matter of fact, the hierarchical algorithm provides, at a given level, h , an optimal partition conditionally on the partition obtained at the previous level, $h - 1$. The partitioning algorithm starts with the partition obtained by cutting

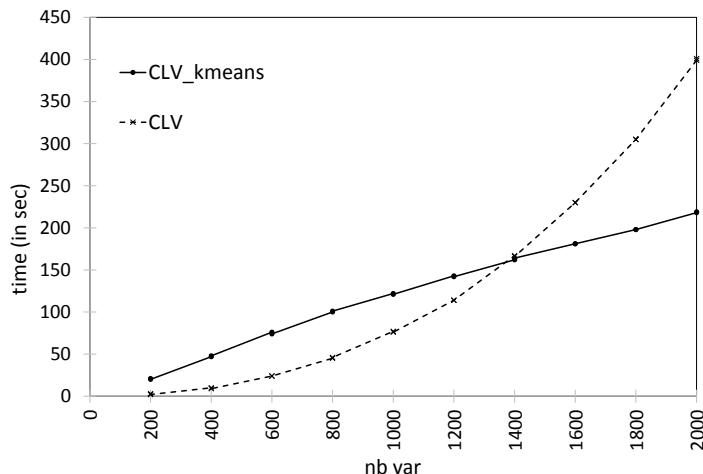


Figure 6: Comparison of the processing time with `CLV()` and `CLV_kmeans()` as a function of the number of variables (the other parameters of the experiment being fixed).

the dendrogram at a given level (say, h) and an alternating optimization scheme is used until the convergence of the criterion to be maximized. The number of iterations before convergence is given in the list of the results (e.g., `resclv$tabres[, "iter"]`). This second stage is called consolidation stage. By default, the consolidation is performed for the last twenty levels of the hierarchy, i.e., for $K = 1$ to $K = 20$.

However, when the number of variables is large, the hierarchical algorithm may be time consuming. For this reason, the `CLV_kmeans()` function was added to the package **ClustVarLV**. This function has the same parameters and options as the `CLV()` function, but performs only the partitioning stage. In this case, the number of clusters, K , should be given as an input parameter. For the initialization of the iterative algorithm, the user may suggest a partition used as a starting point, or, may ask that random initializations of the algorithm are repeatedly performed. The number of repetitions in case of random initializations is stated by the user (argument `nstart`).

Figure 6 shows that the time required for the `CLV_kmeans()` function increases approximately linearly with the number of variables. Let us notice that in this experiment, there were twenty observations, the `nstart` parameter was fixed to 50, and the `CLV_kmeans()` function was used iteratively twenty times, by varying the number of clusters from $K = 1$ to $K = 20$. In comparison, the relationship between the time required for the `CLV()` function (consolidation done for $K = 1$ to $K = 20$) and the number of variables looks like a power function. As can be observed (Figure 6), when the number of variables was about 1400, the processing time was comparable for both procedures. When the number of variables was larger, as it is often the case when dealing with -omics data, the `CLV_kmeans()` function (used for partitions into one cluster until twenty clusters) provides a faster implementation. However, for reasonable number of variables to cluster, the `CLV()` function appears preferable. This is not only because `CLV()` is relatively fast in this case, but also because it provides a graph of the evolution of the aggregation criterion which is helpful for choosing the number of clusters.

The ClustOfVar and ClustVarLV packages

As stated above, both packages, **ClustOfVar** and **ClustVarLV**, are devoted to the cluster analysis of variables. They both draw from the same theoretical background (Vigneau and Qannari, 2003). We emphasize hereinafter some differences of these two packages.

In the first place, it seems that **ClustVarLV** is less time consuming than **ClustOfVar**. To illustrate this aspect, we considered a large dataset, named “Colon”, which is available in the **plsgenomics** package (Boulesteix et al., 2015). It concerns the gene expression of 2000 genes for 62 samples from the microarray experiments of Colon tissue samples of Alon et al. (1999). As shown below, the running time was less than 7 minutes for the `CLV()` function, whereas the `hclustvar()` of the **ClustOfVar** required more than an hour and a half. The performance of `CLV()` over `hclustvar()` can be partly explained by the fact that **ClustVarLV** is interfaced with C++ blocks of code thanks to the **Rcpp** package (Eddelbuettel and François, 2011; Eddelbuettel, 2013).

```
R> data("Colon", package = "plsgenomics")
R> library("ClustVarLV")
R> system.time(CLV(Colon$x, method = "directional", sX = TRUE, nmax = 1))
```

```

user      system    elapsed
 385.30    7.60     392.95

R> library("ClustOfVar")
R> system.time(hclustvar(Colon$X))

user      system    elapsed
4926.37   15.57    4942.44

```

We also indicated that the feature in **ClustOfVar** that is generally put forward is the possibility to cluster both quantitative and qualitative variables. We have stressed through the ‘wine’ dataset the limitation of clustering together quantitative and qualitative variables and we advocated breaking down the qualitative variables into the indicator variables associated with its categories. It is also worth mentioning that **ClustVarLV** covers a much wider scope than **ClustOfVar** as it makes it possible:

- (i) to cluster variables according to local (`method = "local"`) or directional groups (`method = "directional"`), this latter option being the only possibility offered by **ClustOfVar**;
- (ii) to perform a cluster analysis on non standardized (`sX = FALSE`) or standardized variables (`sX = TRUE`), whereas **ClustOfVar** systematically standardizes the variables;
- (iii) to cluster the variables taking into account external information on the observations and/or the variables.

Concluding remarks

The R package **ClustVarLV** contains the functions `CLV`, `CLV_kmeans` and `LCLV` related to the CLV approach, which can be used with or without external information. Additional functions have also been included in order to extract different types of results or to enhance the interpretation of the outcomes. A vignette is included in the package documentation (web link: [ClustVarLV](#)) and provides some basic examples for running the main functions of the **ClustVarLV** package.

Several developments of the CLV approach are under investigation and will be implemented in the forthcoming updates of the **ClustVarLV** package. The “cleaning up” of the variables which do not have a clear assignment to their current cluster (noise variables, for instance) is one of the issues that we are investigating. Another interesting topic is the clustering of variables with the aim of explaining a given response variable as described in [Chen and Vigneau \(in press\)](#).

Bibliography

- U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, and A. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences of the United States of America*, 96(12):6745–6750, 1999. [[p145](#)]
- N. Bailly, I. Maitre, M. Amand, C. Hervé, and D. Alaphilippe. The Dutch eating behaviour questionnaire (DEBQ). Assessment of eating behaviour in an aging French population. *Appetite*, 59(3):853–858, 2012. [[p137](#)]
- A.-L. Boulesteix, G. Durif, S. Lambert-Lacroix, J. Peyre, and K. Strimmer. *plsgenomics: PLS Analyses for Genomics*, 2015. URL <https://CRAN.R-project.org/package=plsgenomics>. R package version 1.3-1. [[p145](#)]
- P. Bühlmann, P. Rütimann, S. van de Geer, and C.-H. Zhang. Correlated variables in regression: Clustering and sparse estimation. *Journal of Statistical Planning and Inference*, 143(11):1835—1858, 2013. [[p134](#)]
- M. Chavent, V. Kuentz-Simonet, B. Liquet, and J. Saracco. ClustOfVar: An R package for the clustering of variables. *Journal of Statistical Software*, 50(13):1–16, 2012. URL <http://www.jstatsoft.org/v50/i13/>. [[p139](#)]
- M. Chavent, V. Kuentz, B. Liquet, and J. Saracco. *ClustOfVar: Clustering of Variables*, 2013. URL <https://CRAN.R-project.org/package=ClustOfVar>. R package version 0.8. [[p134](#)]
- M. Chen and E. Vigneau. Supervised clustering of variables. *Advanced in Data Analysis and Classification*, in press. doi: 10.1007/s11634-014-0191-5. [[p146](#)]

- B. Daillant-Spinnler, H. MacFie, P. Beyts, and D. Hedderley. Relationships between perceived sensory properties and major preference directions of 12 varieties of apples from the Southern Hemisphere. *Food Quality and Preference*, 7(2):113–126, 1996. [p141]
- I. S. Dhillon, E. M. Marcotte, and U. Roshan. Diametrical clustering for identifying anti-correlated gene clusters. *Bioinformatics*, 19(13):1612–1619, 2003. [p134]
- D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, New York, 2013. [p145]
- D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL <http://www.jstatsoft.org/v40/i08/>. [p145]
- M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences of the United States of America*, 95(25):14863–14868, 1998. [p134]
- D. G. Enki, N. T. Trendafilov, and I. T. Jolliffe. A clustering approach to interpretable principal components. *Journal of Applied Statistics*, 40(3):583–599, 2013. [p134, 137]
- F. E. Harrell Jr, C. Dupont, et al. *Hmisc: Harrell Miscellaneous*, 2015. URL <https://CRAN.R-project.org/package=Hmisc>. R package version 3.17-0. [p135]
- T. Hastie, R. Tibshirani, M. B. Eisen, A. Alizadeh, R. Levy, L. Staudt, W. C. Chan, D. Botstein, and P. Brown. ‘Gene shaving’ as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology*, 1(2):1–21, 2000. [p134]
- F. Husson, J. Josse, S. Le, and J. Mazet. *FactoMineR: Multivariate Exploratory Data Analysis and Data Mining*, 2015. URL <https://CRAN.R-project.org/package=FactoMineR>. R package version 1.31.4. [p139]
- D. Jacob, C. Deborde, and A. Moing. An efficient spectra processing method for metabolite identification from 1H-NMR metabolomics data. *Analytical and Bioanalytical Chemistry*, 405(15):5049–5061, 2013. [p141]
- I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 2nd edition, 2002. [p135, 139]
- S. Kaiser, R. Santamaría, T. Khamiakova, M. Sill, R. Theron, L. Quintales, F. Leisch, and E. De Troyer. *biclust: BiCluster Algorithms*, 2015. URL <https://CRAN.R-project.org/package=biclust>. R package version 1.2.0. [p135]
- H. Kiers. Simple structure in component analysis techniques for mixtures of qualitative and quantitative variables. *Psychometrika*, 56(2):197–212, 1991. [p139]
- I. Kojadinovic. Hierarchical clustering of continuous variables based on the empirical copula process and permutation linkages. *Computational Statistic and Data Analysis*, 54(1):90–108, 2010. [p134]
- L. Lebart, A. Morineau, and M. Piron. *Statistique exploratoire multidimensionnelle*. Dunod, Paris, 3ieme edition, 2000. [p136]
- F. Leisch and B. Grün. CRAN task view: Cluster analysis & finite mixture models, 2015. URL <https://CRAN.R-project.org/view=Cluster>. Version 2015-07-24. [p134]
- P. G. Lovaglio. Model building and estimation strategies for implementing the balanced scorecard in health sector. *Quality & Quantity*, 45(1):199–212, 2011. [p139]
- M. Maechler, P. Rousseeuw, A. Struyf, and M. Hubert. *cluster: “Finding Groups in Data”: Cluster Analysis Extended Rousseeuw et al.*, 2015. URL <https://CRAN.R-project.org/package=cluster>. R package version 2.0.3. [p134, 228]
- H. Martens, E. Anderssen, A. Flatberg, L. H. Gidskehaug, M. Hoy, F. Westad, A. Thybo, and M. Martens. Regression of a matrix on descriptors of both its rows and its columns via latent variables: L-PLSR. *Computational Statistics and Data Analysis*, 48(1):103–123, 2005. [p144]
- J. Pagès. Analyse factorielle de données mixtes. *Revue de Statistique Appliquée*, 52(4):93–111, 2004. [p139]
- M. Y. Park, T. Hastie, and R. Tibshirani. Averaged gene expressions for regression. *Biostatistics*, 8(2):212–227, 2007. [p134]
- G. Saporta. Simultaneous analysis of qualitative and quantitative data. In Societa Italiana di Statistica, editor, *Atti Della XXXV Riunione Scientifica*, pages 63–72, 1990. [p139, 140]

- W. Sarle. *SAS/STAT User's Guide: The Varclus Procedure*. SAS Institute, Inc., Cary, NC, USA, 4th edition, 1990. [p134, 136]
- R. Suzuki and H. Shimodaira. *pvcclus: Hierarchical Clustering with P-Values via Multiscale Bootstrap Resampling*, 2014. URL <https://CRAN.R-project.org/package=pvcclus>. R package version 1.3-2. [p135]
- L. Tolosi and T. Lengauer. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*, 27(14):1986–1994, 2011. [p134]
- E. Vigneau and M. Chen. *ClustVarLV: Clustering of Variables Around Latent Variables*, 2015. URL <https://CRAN.R-project.org/package=ClustVarLV>. R package version 1.4.1. [p134]
- E. Vigneau and E. Qannari. Clustering of variables around latent components. *Communications in Statistics – Simulation and Computation*, 32(4):1131–1150, 2003. [p134, 135, 136, 137, 139, 142, 145]
- E. Vigneau and E. M. Qannari. Segmentation of consumers taking account of external data: A clustering of variables approach. *Food Quality and Preference*, 13(7–8):515–521, 2002. [p142]
- E. Vigneau, E. M. Qannari, P. H. Punter, and S. Knoops. Segmentation of a panel of consumers using clustering of variables around latent directions of preference. *Food Quality and Preference*, 12(5–7):359–363, 2001. [p141]
- E. Vigneau, K. Sahmer, E. M. Qannari, and D. Bertrand. Clustering of variables to analyze spectral data. *Journal of Chemometrics*, 19(3):122–128, 2005. [p143]
- E. Vigneau, E. M. Qannari, K. Sahmer, and D. Ladiray. Classification de variables autour de composantes latentes. *Revue de Statistique Appliquée*, 54(1):27–45, 2006. [p139]
- E. Vigneau, I. Endrizzi, and E. Qannari. Finding and explaining clusters of consumers using the CLV approach. *Food Quality and Preference*, 22(4):705–713, 2011. [p144]
- E. Vigneau, M. Charles, and M. Chen. External preference segmentation with additional information on consumers: A case study on apples. *Food Quality and Preference*, 22(4):83–92, 2014. [p144]
- E. Warms-Petit, E. Morignat, M. Artois, and D. Calavas. Unsupervised clustering of wildlife necropsy data for syndromic surveillance. *BMC Veterinary Research*, 6:56, 2010. URL <http://www.biomedcentral.com:1746-6148/6/56>. [p136]
- L. Yengo and M. Canoui. *clere: Simultaneous Variables Clustering and Regression*, 2014. URL <https://CRAN.R-project.org/package=clere>. R package version 1.1.2. [p134]
- H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):262–286, 2006. [p135]

Evelyne Vigneau
Sensometrics and Chemometrics Laboratory
National College of Veterinary Medicine, Food Science and Engineering (Oniris)
Rue de la Geraudiere, CS 82225
44322 Nantes Cedex 03
France
evelyne.vigneau@oniris-nantes.fr

Mingkun Chen
Sensometrics and Chemometrics Laboratory
National College of Veterinary Medicine, Food Science and Engineering (Oniris)
Rue de la Geraudiere, CS 82225
44322 Nantes Cedex 03
France

El Mostafa Qannari
Sensometrics and Chemometrics Laboratory
National College of Veterinary Medicine, Food Science and Engineering (Oniris)
Rue de la Geraudiere, CS 82225
44322 Nantes Cedex 03
France

Working with Multilabel Datasets in R: The **mldr** Package

by Francisco Charte and David Charte

Abstract Most classification algorithms deal with datasets which have a set of input features, the variables to be used as predictors, and only one output class, the variable to be predicted. However, in late years many scenarios in which the classifier has to work with several outputs have come to life. Automatic labeling of text documents, image annotation or protein classification are among them. Multilabel datasets are the product of these new needs, and they have many specific traits. The **mldr** package allows the user to load datasets of this kind, obtain their characteristics, produce specialized plots, and manipulate them. The goal is to provide the exploratory tools needed to analyze multilabel datasets, as well as the transformation and manipulation functions that will make possible to apply binary and multiclass classification models to this data or the development of new multilabel classifiers. Thanks to its integrated user interface, the exploratory functions will be available even to non-specialized R users.

Introduction

Pattern classification is an important task nowadays and is in use everywhere, from our e-mail client, which is able to separate spam from legit messages, to credit institutions, that rely on it to detect fraud and grant or deny loans. All these cases operate with binary datasets, since a message is either spam or legit, and multiclass datasets, the loan is safe, medium, risky or highly risky, for instance. In both cases the user expects only one output.

The huge growth of the amount of information stored in late years on the web, such as blog posts, pictures taken from cameras and phones, videos hosted on YouTube, and messages on social networks, has led to more complex classification work. A blog post can be classified into several non-exclusive categories, for instance news, economy and politics simultaneously. A picture can be assigned a set of labels, such as landscape, sky and forest. A video can be labeled into several music genres at once, etc. All of these are examples of problems in need of multilabel classification.

Binary and multiclass datasets can be managed in R by using data frames. Usually the last attribute (column of the “`data.frame`”) is the output class, which might contain only TRUE/FALSE values or values belonging to a finite set (a factor). Multilabel datasets (MLDs) can also be stored in an R “`data.frame`”, but an additional structure to know which attributes are output labels is needed. Moreover, this kind of datasets have many specific characteristics that do not exist in the traditional ones. The average number of labels per instance, the imbalance ratio for each label, the number of labelsets (sets of labels assigned to each row) and their frequencies, and the level of concurrence among imbalanced labels are some of the traits that differentiate an MLD from the others.

Until now, most of the software to work with MLDs has been written in Java. The two best known frameworks are MULAN ([Tsoumakas et al., 2011](#)) and MEKA ([Read and Reutemann, 2012](#)). Both implementations rely on WEKA ([Hall et al., 2009](#)), which offers a large variety of binary and multiclass classifiers, as well as functions needed to deal with ARFF (*Attribute-Relation File Format*) files. Most of the existing MLDs are stored in ARFF format. MULAN and MEKA provide the specialized tools needed to deal with multilabel ARFFs, and the infrastructure to build multilabel classifiers (MLCs). Although R can access WEKA functionality through the **RWeka** ([Hornik et al., 2009](#)) package, handling MLDs is far from an easy task in R. This has been the main motivation behind the development of the **mldr** package. To the best of our knowledge, **mldr** is the first R package aimed to ease the work with multilabel data.

The **mldr** package aims to provide the user with the functions needed to perform exploratory analysis of MLDs, determining their main traits both statistically and visually. Moreover, it also brings the proper tools to manipulate this kind of datasets, including the application of the most common transformation methods, BR (*Binary Relevance*) and LP (*Label Powerset*), that will be described in the following section. These would be the foundation for processing MLDs with traditional classifiers, as well as for developing new multilabel algorithms.

The **mldr** package does not depend on the **RWeka** package, and it is not linked to MULAN nor MEKA. It has been designed to allow reading both MULAN and MEKA MLDs, but without any external dependencies. In fact, it would be possible to load MLDs stored in other file formats, as well as creating them from scratch. When loaded, MLDs are wrapped in an S3 type object with class “`mldr`”, which allows for the use of methods. The object will contain the data in the MLD and also a large set of measures obtained from it. The functions provided by the package ease the access to

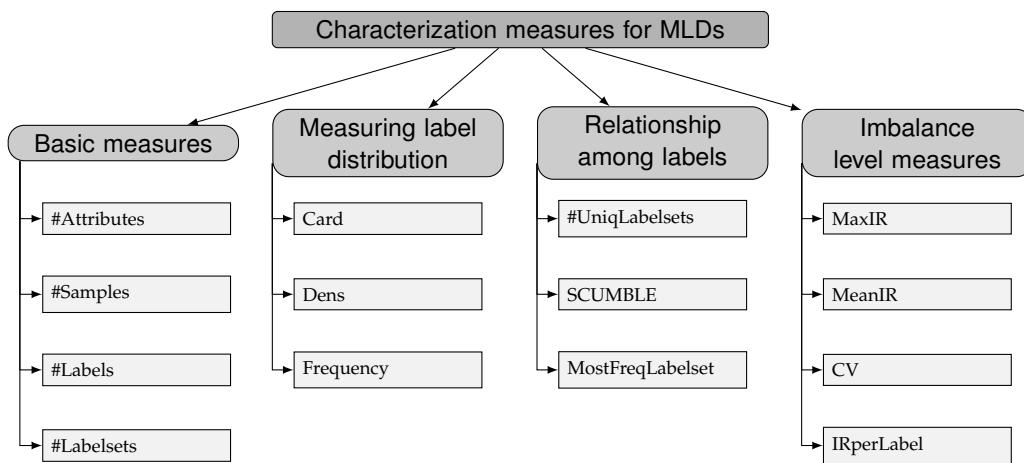


Figure 1: Characterization measures taxonomy.

this information, produce some specific plots, and make possible the manipulation of its content. A web-based graphical user interface, developed using the `shiny` (Chang et al., 2015) package, puts the exploratory analysis tools of the `mldr` package at the fingertips of all users, even those who have little experience using R.

In the following section the foundations related to MLDs and MLC will be briefly introduced. After that, the structure of the `mldr` package, and the operations it provides will be explained. Finally, the user interface provided by `mldr` to ease exploratory analysis tasks over MLDs will be shown. All code displayed in this paper is available in a vignette, accessible by entering `vignette("mldr", package = "mldr")`.

Working with multilabel datasets

MLDs are generated from text documents (Klimt and Yang, 2004), sets of images (Duygulu et al., 2002), music collections, and protein attributes (Diplaris et al., 2005), among other sources. For each sample a set of features (input attributes) is collected, and a set of labels (the output labelset) is assigned. Usually there are several hundreds or even thousands of attributes, and it is not rare that a MLD has more labels than features. Some MLDs have only a few labels per instance, while others have dozens of them. In some MLDs the number of label combinations (labelsets) is quite short, whereas in others it can be very large. Most MLDs are imbalanced, which means that some labels are very frequent while others are scarcely represented. The labels in an MLD can be correlated or not. Moreover, frequent labels and rare labels can appear together in the same instances.

As can be seen, a lot of different scenarios can be found depending on the MLD characteristics. This is the reason why several specific measures have been designed to assess MLD traits (Tsoumakas et al., 2010), since they can have a serious impact on the MLC's performance. The following two subsections introduce several of these measures and some of the approaches pursued to face multilabel classification.

Multilabel dataset traits

The most common characterization measures for MLDs can be grouped into four categories, as depicted in Figure 1.

The most basic information that can be obtained from an MLD is the number of instances, attributes and labels. For any MLD containing $|D|$ instances, any instance $D_i, i \in \{1..|D|\}$ will be the union of a set of attributes and a set of labels $(X_i, Y_i), X_i \in X^1 \times X^2 \times \dots \times X^f, Y_i \subseteq L$, where f is the number of input features and X^j is the space of possible values for the j -th attribute, $j \in \{1..f\}$. L being the full set of labels used in D , Y_i could be any subset of items in L . Therefore, theoretically the number of potential labelsets could be $2^{|L|}$. In practice this number tends to be limited by $|D|$.

Each instance D_i has an associated labelset, whose length (number of active labels) can be in the range $\{0..|L|\}$. The average number of active labels per instance is the most basic measure of any MLD, usually known as *Card* (standing for cardinality). It is calculated as shown in Equation 1. Dividing this measure by the number of labels in L , as shown in Equation 2, results in a dimension-less measure,

known as *Dens* (standing for label density).

$$\text{Card}(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} |Y_i|, \quad (1)$$

$$\text{Dens}(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i|}{|L|}. \quad (2)$$

Most multilabel datasets are imbalanced, meaning that some of the labels are very frequent whereas others are quite rare. The level of imbalance of a label can be measured by the imbalance ratio, *IRlbl*, defined in Equation 3. To know how much imbalance there is in D , the *MeanIR* measure (Charte et al., 2015) is calculated as the mean imbalance ratio among all labels, as shown in Equation 4. In order to know the significance of this last measure, the standard CV (Coefficient of Variation, Equation 5) can be used.

$$\text{IRlbl}(y) = \frac{\max_{y' \in L} \left(\sum_{i=1}^{|D|} h(y', Y_i) \right)}{\sum_{i=1}^{|D|} h(y, Y_i)} \quad h(y, Y_i) = \begin{cases} 1 & y \in Y_i \\ 0 & y \notin Y_i \end{cases}, \quad (3)$$

$$\text{MeanIR} = \frac{1}{|L|} \sum_{y \in L} \text{IRlbl}(y), \quad (4)$$

$$CV = \frac{\text{IRlbl}\sigma}{\text{MeanIR}} \quad \text{IRlbl}\sigma = \sqrt{\sum_{y \in L} \frac{(\text{IRlbl}(y) - \text{MeanIR})^2}{|L| - 1}}. \quad (5)$$

The number of different labelsets, as well as the amount of them being unique labelsets (i.e., appearing only once in D), give us an idea on how sparsely the labels are distributed. The labelsets by themselves indicate how the labels in L are related. A very frequent labelset implies that the labels in it tend to appear jointly in D . The *SCUMBLE* measure, introduced in Charte et al. (2014) and shown in Equation 7, is used to assess the concurrence level among frequent and infrequent labels.

$$\text{SCUMBLE}_{ins}(i) = 1 - \frac{1}{\text{IRlbl}_i} \left(\prod_{l=1}^{|L|} \text{IRlbl}_{il} \right)^{(1/|L|)}, \quad (6)$$

$$\text{SCUMBLE}(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \text{SCUMBLE}_{ins}(i). \quad (7)$$

Besides the aforementioned insights, there are some other interesting traits that can be indirectly obtained from the previous measures, such as the ratio between input features and output labels, the maximum *IRlbl*, or the coefficient of variation in the imbalance of levels, among others.

Although the raw numbers given by these calculations describe the nature of any multilabel dataset to a good level, in general a visualization of its characteristics is desirable to ease its interpretation by researchers.

The information obtained from the previous measures depicts the characteristics of the dataset. These insights, along with other factors such as the loss function used by the classifier, help in choosing the most proper algorithm to learn from it and, in the future, make predictions on new data. Traditional classification models, such as trees and support vector machines, are designed to give only one output as result. Multilabel classification can mainly be faced through two different approaches discussed in the following.

Multilabel classification

- **Algorithm adaptation:** The goal is to modify existing algorithms taking into account the multilabel nature of the samples, for instance hosting more than one class in the leaves of a tree instead of only one.
- **Problem transformation:** This approach transforms the original data to make it suitable to traditional classification algorithms, then combines the obtained predictions to build the labelsets given as output result.

Although several transformation methods have been defined in the specialized literature, there are two among them that stand out because they are the foundation for many others:

- **Binary Relevance (BR):** Introduced by [Godbole and Sarawagi \(2004\)](#) as an adaptation of OVA (*One-vs-All*) to the multilabel scenario, this method transforms the original multilabel dataset into several binary datasets, as many as there are different labels. In this way any binary classifier can be used by joining the individual predictions to generate the final output.
- **Label Powerset (LP):** Introduced by [Boutell et al. \(2004\)](#), this method transforms the multilabel dataset into a multiclass dataset by using the labelset of each instance as class identifier. Any multiclass classifier can be used, transforming back the predicted class into a labelset.

BR and LP have been used not only as a direct technique to implement multilabel classifiers, but also as a base method to build more sophisticated algorithms. Several ensembles of binary classifiers relying on BR have been proposed, such as CC (*Classifier Chains*) or ECC (*Ensemble of Classifier Chains*), both by [Read et al. \(2011\)](#). The same is applicable to the LP transformation, the foundation of ensemble multilabel classifiers such as RAkEL (Random k-Labelsets for Multi-Label Classification, [Tsoumakas and Vlahavas 2007](#)) and EPS (Ensemble of Pruned Sets, [Read et al. 2008](#)).

For the readers interested in more details, a recent review on multilabel classification has been published by [Zhang and Zhou \(2014\)](#).

The **mldr** package

R is among the most used tools when it comes to performing data mining tasks, including binary and multiclass classification. However, the work with MLDs in R is not as easy as it is with classic datasets. This is the main motivation behind the development of the **mldr** package, whose goals and functionality are described in this section.

Main goals of the **mldr** package

When we planned the development of this package, our main objective was to ease the exploration of MLDs in R. This included loading existing MLDs in different formats, as well as obtaining from them all the available information. These functions should be accessible to everyone, even to users not used to the R command line but to GUIs (*Graphical User Interfaces*) such as those provided by packages **Rcmdr** (aka *R Commander*, [Fox 2005](#)) or **rattle** ([Williams, 2011](#)).

At the same time, we aimed to include the tools needed to manipulate the MLDs, to apply filters and transformations, as well as to create MLDs from scratch. This functionality, directed to more experienced R users, opens the doors to implement other algorithms on top of **mldr**, for instance preprocessing methods or multilabel classifiers.

Installing and loading the **mldr** package

The **mldr** package is available from the Comprehensive R Archive Network (CRAN), therefore it can be installed as any other package, by simply typing:

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

mldr depends on three R packages: **XML** ([Lang and the CRAN Team, 2015](#)), **circlize** ([Gu et al., 2014](#)) and **shiny**. The first one allows reading XML (*eXtensible Markup Language*) files, the second one is used to generate a specific type of plot (described below), and the third one is the base of its user interface.

Older releases of **mldr**, as well as the development version, are available at <http://github.com/fcharte/mldr>. It is possible to install the development version using the `install_github()` function from **devtools** ([Wickham and Chang, 2015](#)).

Once installed, the package has to be loaded before it can be used. This can be done through the `library()` or `require()` functions, as usual. After loading the package three sample MLDs will be available: `birds`, `emotions` and `genbase`. These are contained in the '`birds.rda`', '`emotions.rda`' and '`genbase.rda`' files, which are lazily loaded along with the package.

The **mldr** package uses its own internal representation for MLDs, which are assigned the "mldr" class. Inside an "mldr" object, such as the previous mentioned `emotions` or `birds`, both the data in the MLD and all the information obtained from this data can be found.

Loading and creating MLDs

Besides the three sample MLDs included in the package, the `mldr()` function allows to load any MLD stored in MULAN or MEKA file formats. Assuming that the files '`corel5k.arff`' and '`corel5k.xml`', which

hold the Corel5k (Duygulu et al., 2002) MLD in MULAN format, are in the current directory, the loading is done as follows:

```
> core15k <- mldr("core15k")
```

If the XML file is not available, it is possible to indicate just the number of labels in the MLD instead. In this case, the function assumes that the labels are at the end of the list of features. For instance:

```
> core15k <- mldr("core15k", label_amount = 374)
```

Loading an MLD in MEKA file format is equally easy. In this case there is no XML file with label information used, but a special header inside the ARFF file, a fact that will be indicated to `mldr()` with the `use_xml` argument:

```
> imdb <- mldr("imdb", use_xml = FALSE)
```

In all cases the result, as long as the MLD can be correctly loaded and parsed, will be a new “`mldr`” object ready to use.

If the MLD we are interested in is not in MULAN or MEKA format, firstly it will have to be loaded into a “`data.frame`”, for instance using functions such as `read.csv()`, `read.table()` or a more specialized reader, and secondly this “`data.frame`” and an integer vector stating the indices of the labels inside it are given to the `mldr_from_dataframe()` function. This is a general function for creating an “`mldr`” object from any “`data.frame`”, so it can also be used to generate new MLDs on the fly, as shown in the following example:

```
> df <- data.frame(matrix(rnorm(1000), ncol = 10))
> df$Label1 <- c(sample(c(0,1), 100, replace = TRUE))
> df$Label2 <- c(sample(c(0,1), 100, replace = TRUE))
> mymldr <- mldr_from_dataframe(df, labelIndices = c(11, 12), name = "testMLDR")
```

This will assign to `mymldr` an MLD, named `testMLDR`, with 10 input attributes and 2 labels.

Obtaining information from an MLD

After loading any MLD, a quick summary of its main characteristics can be obtained by means of the usual `summary()` function, as shown below:

```
> summary(birds)
num.attributes num.instances num.labels num.labelsets num.single.labelsets
                279           645          19            133             73
max.frequency cardinality   density   meanIR    scumble num.inputs
                294     1.013953 0.05336597 5.406996 0.03302765        260
```

Any of these measures can be individually obtained through the `measures` element of the “`mldr`” class, like this:

```
> emotions$measures$num.attributes
[1] 78

> genbase$measures$scumble
[1] 0.0287591
```

Full information about the labels in the MLD, including the number of times they appear, their `IRlbl` and `SCUMBLE` measures, can be retrieved by using the `labels` element of the “`mldr`” class:

```
> birds$labels
      index count      freq      IRLbl      SCUMBLE
Brown Creeper       261     14 0.021705426 7.357143 0.12484341
Pacific Wren        262     81 0.125581395 1.271605 0.05232609
Pacific-slope Flycatcher 263     46 0.071317829 2.239130 0.06361470
Red-breasted Nuthatch 264      9 0.013953488 11.444444 0.15744451
Dark-eyed Junco     265     20 0.031007752 5.150000 0.10248336
Olive-sided Flycatcher 266     14 0.021705426 7.357143 0.18493760
Hermit Thrush       267     47 0.072868217 2.191489 0.06777263
Chestnut-backed Chickadee 268     40 0.062015504 2.575000 0.06807452
Varied Thrush        269     61 0.094573643 1.688525 0.07940806
Hermit Warbler       270     53 0.082170543 1.943396 0.07999006
```

Swainson's Thrush	271	103	0.159689922	1.000000	0.11214301
Hammond's Flycatcher	272	28	0.043410853	3.678571	0.06129884
Western Tanager	273	33	0.051162791	3.121212	0.07273988
Black-headed Grosbeak	274	9	0.013953488	11.444444	0.20916487
Golden Crowned Kinglet	275	37	0.057364341	2.783784	0.09509474
Warbling Vireo	276	17	0.026356589	6.058824	0.14333613
MacGillivray's Warbler	277	6	0.009302326	17.166667	0.24337605
Stellar's Jay	278	10	0.015503876	10.300000	0.12151527
Common Nighthawk	279	26	0.040310078	3.961538	0.06520272

The same is applicable for labelsets and attributes, by means of the labelsets and attributes elements of the class.

To access the MLD content, attributes and label values, the `print()` function can be used, as well as the `dataset` element of the “`mldr`” object.

Plotting functions

Exploratory analysis of MLDs can be tedious, since most of them have thousands of attributes and hundreds of labels. The `mldr` package provides a `plot()` function specific for dealing with “`mldr`” objects, allowing the generation of several specific types of plots. The first argument given to `plot()` must be an “`mldr`” object, while the second one specifies the type of plot to be produced.

```
> plot(emotions, type = "LH")
```

There are seven different types of plots available: three histograms showing relations between instances and labels, two bar plots with similar purpose, a circular plot indicating types of attributes and a concurrence plot for labels. All of them are shown in Figure 2, generated by the following code:

```
> layout(matrix(c(1, 1, 7, 1, 1, 4, 5, 5, 4, 2, 6, 3), 4, 3, byrow = TRUE))
> plot(emotions, type = c("LC", "LH", "LSH", "LB", "LSB", "CH", "AT"))
```

The concurrence plot is the default one, with type “`LC`”, and responds to the need of exploring interactions among labels, and specifically between majority and minority ones. This plot has a circular shape, with the circumference partitioned into several disjoint arcs representing labels. Each arc has length proportional to the number of instances where the label is present. These arcs are in turn divided into bands that join two of them, showing the relation between the corresponding labels. The width of each band indicates the strength of the relation, since it is proportional to the number of instances in which both labels appear simultaneously. In this manner, a concurrence plot can show whether imbalanced labels appear frequently together, a situation which could limit the possible improvement of a preprocessing technique (Charte et al., 2014).

Since drawing interactions among a lot of labels can produce a confusing result, this last type of plot accepts more arguments: `labelCount`, which accepts an integer that will be used to generate the plot with that number of labels chosen at random; and `labelIndices`, which allows to indicate exactly the indices of the labels to be displayed in the plot. For example, in order to plot the first ten labels of `genbase`:

```
> plot(genbase, labelIndices = genbase$labels$index[1:10])
```

The label histogram (type “`LH`”) relates labels and instances in a way that shows how well-represented labels are in general. The X axis represents the number of instances and the Y axis the amount of labels. This means that if a large number of labels are appearing in very few instances, all data will be concentrated on the left side of the plot. On the contrary, if labels are generally present in many instances, data will tend to accumulate on the right side. This plot shows imbalance of labels when there is data accumulated on both sides of the plot, which implies that many labels are underrepresented, and a large amount are overrepresented as well.

The labelset histogram (named “`LSH`”) is similar to the former. However, instead of representing the number of instances in which each label appears, it shows the amount of labelsets. This indicates quantitatively whether labelsets repeat consistently or not among instances.

The label and labelset bar plots display exactly the number of instances for each one of the labels and labelsets, respectively. Their codes are “`LB`” for the label bar plot and “`LSB`” for the labelset one.

The cardinality histogram (type “`CH`”) represents the amount of labels instances have in general. Therefore data accumulating on the right side of the plot indicates that instances do have a notable amount of labels, whereas data concentrating on the left side shows the opposite situation.

The attribute types plot (named “`AT`”) is a pie chart displaying the number of labels, numeric attributes and finite set (character) attributes, thus showing the proportions between these types of attributes to ease the understanding of the amount of input information and that of output data.

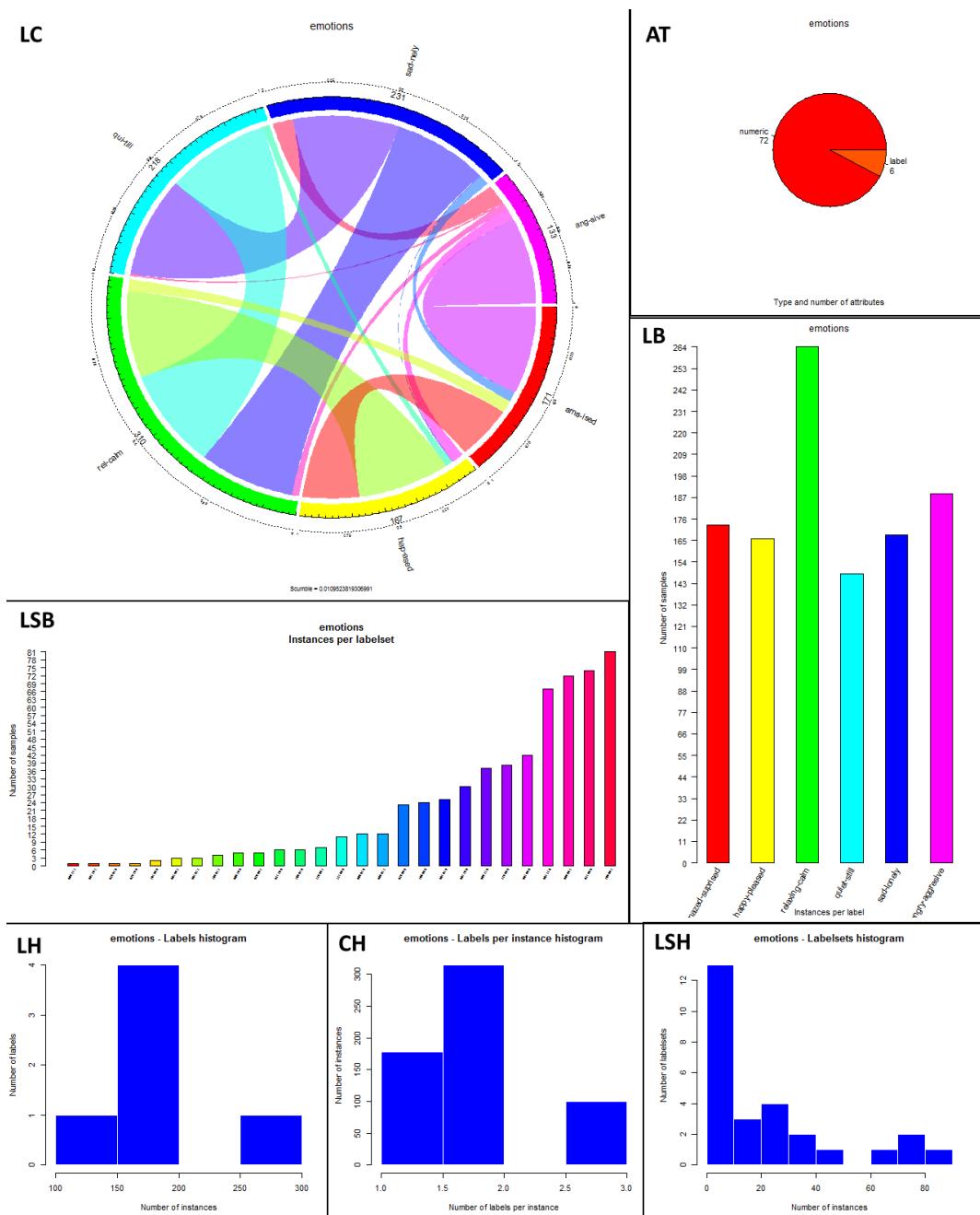


Figure 2: Plots generated by `mldr`'s `plot()` function. The type of plot is indicated at the top-left corner of each rectangle.

Additionally, `plot()` accepts coloring arguments, `col` and `color.function`. The former can be used on all plot types except for the label concurrence plot, and must be a vector of colors. The latter is only used on the label concurrence plot and accepts a coloring function, such as `rainbow` or `heat.colors`, as can be seen in the following example:

```
> plot(emotions, type = "LC", color.function = heat.colors)
> plot(emotions, type = "LB", col = terrain.colors(emotions$measures$num.labels))
```

Transforming and filtering functions

Manipulation of datasets is a crucial task in multilabel classification. Since transformation is one of the main approaches to tackle the problem, both BR and LP transformations are implemented in package `mldr`. They can be obtained using the `mldr_transform` function, which accepts an “`mldr`” object as first argument, the type of transformation, “`BR`” or “`LP`”, as second, and an optional vector of label

indices to be included in the transformation as last argument:

```
> emotionsbr <- mldr_transform(emotions, type = "BR")
> emotionslp <- mldr_transform(emotions, type = "LP", emotions$labels$index[1:4])
```

The BR transformation will return a list of “data.frame” objects, each one of them using one of the labels as class, whereas the LP transformation will return a single “data.frame” representing a multiclass dataset using each labelset as a class. Both of these transformations can be directly used in order to apply binary and multiclass classification algorithms, or even implement new ones.

```
> emo_lp <- mldr_transform(emotions, "LP")
> library(RWeka)
> classifier <- IBk(classLabel ~ ., data = emo_lp, control = Weka_control(K = 10))
> evaluate_Weka_classifier(classifier, numFolds = 5)
```

==== 5 Fold Cross Validation ====

==== Summary ====

Correctly Classified Instances	205	34.57	%
Incorrectly Classified Instances	388	65.43	%
Kappa statistic	0.2695		
Mean absolute error	0.057		
Root mean squared error	0.1748		
Relative absolute error	83.7024 %		
Root relative squared error	94.9069 %		
Coverage of cases (0.95 level)	75.3794 %		
Mean rel. region size (0.95 level)	19.574 %		
Total Number of Instances	593		

A filtering utility is included in the package as well. Using it is intuitive, since it can be called with the square bracket operator [. This allows to partition an MLD or filter it according to a logical condition.

```
> emotions$measures$num.instances
[1] 593

> emotions[emotions$dataset$.SCUMBLE > 0.01]$measures$num.instances
[1] 222
```

Combined with the joining operator, +, this enables users to implement new preprocessing techniques that modify information in the MLD in order to improve classification results. For example, the following would be an implementation of an algorithm disabling majority labels on instances with highly imbalanced labels:

```
> mldbase <- mld[.SCUMBLE <= mld$measures$scumble]
> # Samples with cooccurrence of highly imbalanced labels
> mldhigh <- mld[.SCUMBLE > mld$measures$scumble]
> majIndexes <- mld$labels[mld$labels$IRLbl < mld$measures$meanIR, "index"]
> # Deactivate majority labels
> mldhigh$dataset[, majIndexes] <- 0
> mldbase + mldhigh # Join the instances without changes with the filtered ones
```

In this last example, the first two commands filter the MLD, separating instances with their SCUMBLE lower than the mean and those with it higher. Then, the third line obtains the indices of the labels with lower IRLbl than the mean, thus these are the majority labels of the dataset. Finally, these labels are set to 0 in the instances with high SCUMBLE, and then the two partitions are joined again.

Lastly, another useful feature included in the **mldr** package is the MLD comparison with the == operator. This indicates whether both MLDs in comparison share the same structure, which would mean they have the same attributes, and these would have the same type.

```
> emotions[1:10] == emotions[20:30]
[1] TRUE

> emotions == birds
[1] FALSE
```

Assessing multilabel predictive performance

Assuming that a set of predictions has been obtained for a MLD, e.g., through a set of binary classifiers, a multiclass classifier or any other algorithm, the next step would be to evaluate the classification performance. In the literature there exist more than 20 metrics for this task, and some of them are quite complex to calculate. The `mldr` package provides the `mldr_evaluate` function to accomplish this task, supplying both example based and label based metrics.

Multilabel evaluation metrics are grouped into two main categories: example based and label based metrics. Example based metrics are computed individually for each instance, then averaged to obtain the final value. Label based metrics are computed per label, instead of per instance. There are two approaches called *micro-averaging* and *macro-averaging* (described below). The output of the classifier can be a bipartition (i.e., a set of 0s and 1s denoting the predicted labels) or a ranking (i.e., a set of real values denoting the relevance of each label). For this reason, there are bipartition based and ranking based evaluation metrics for each one of the two previous categories.

D being the MLD, L the full set of labels used in D , Y_i the subset of predicted labels for the i -th instance, and Z_i the true subset of labels, the example/bipartition based metrics returned by `mldr_evaluate` are the following:

- **Accuracy:** It is defined (see Equation 8) as the proportion of correctly predicted labels with respect to the total number of labels for each instance.

$$\text{Accuracy} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}. \quad (8)$$

- **Precision:** This metric is computed as indicated in Equation 9, giving as result the ratio of relevant labels predicted by the classifier.

$$\text{Precision} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \cap Z_i|}{|Z_i|}. \quad (9)$$

- **Recall:** It is a metric (see Equation 10) commonly used along with the previous one, measuring the proportion of predicted labels which are relevant.

$$\text{Recall} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \cap Z_i|}{|Y_i|}. \quad (10)$$

- **F-Measure:** As can be seen in Equation 11, this metric is the harmonic mean between *Precision* (see Equation 9) and *Recall* (see Equation 10), providing a balanced assessment between precision and sensitivity.

$$\text{FMeasure} = 2 * \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (11)$$

- **Hamming Loss:** It is the most common evaluation metric in the multilabel literature, computed (see Equation 12) as the symmetric difference between predicted and true labels and divided by the total number of labels in the MLD.

$$\text{HammingLoss} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \Delta Z_i|}{|L|}. \quad (12)$$

- **Subset Accuracy:** This metric is also known as *0/1 Subset Accuracy* and *Classification Accuracy*, and it is the most strict evaluation metric. The `expr` operator (see Equation 13) returns 1 when *expr* is true and 0 otherwise. In this case its value is 1 only if the predicted set of labels equals the true one.

$$\text{SubsetAccuracy} = \frac{1}{|D|} \sum_{i=1}^{|D|} [\![Y_i = Z_i]\!]. \quad (13)$$

Let $\text{rank}(x_i, y)$ be a function returning the position of y , a certain label, in the x_i instance. The example/ranking based evaluation metrics returned by the `mldr_evaluate` function are the following ones:

- **Average Precision:** This metric (see Equation 14) computes the proportion of labels ranked ahead of a certain relevant label. The goal is to establish how many positions have to be

traversed until this label is found.

$$\text{AveragePrecision} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{1}{|Y_i|} \sum_{y \in Y_i} \frac{|\{y' \in Y_i : \text{rank}(x_i, y') \leq \text{rank}(x_i, y)\}|}{\text{rank}(x_i, y)}. \quad (14)$$

- **Coverage:** Defined as indicated in Equation 15, this metric calculates the extent to which it is necessary to go up in the ranking to cover all relevant labels.

$$\text{Coverage} = \frac{1}{|D|} \sum_{i=1}^{|D|} \max_{y \in Y_i} \text{rank}(x_i, y) - 1. \quad (15)$$

- **One Error:** It is a metric (see Equation 16) which determines how many times the best ranked label given by the classifier is not part of the true label set of the instance.

$$\text{OneError} = \frac{1}{|D|} \sum_{i=1}^{|D|} \left[\underset{y \in Z_i}{\text{argmax}} \text{rank}(x_i, y) \notin Y_i \right]. \quad (16)$$

- **Ranking Loss:** This metric (see Equation 17) compares each pair of labels in L , computing how many times a relevant label (member of the true labelset) appears ranked lower than a non-relevant label. In the equation, \bar{Y}_i denotes $L \setminus Y_i$.

$$\text{RankingLoss} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{1}{|Y_i| |\bar{Y}_i|} |\{(y_a, y_b) \in Y_i \times \bar{Y}_i : \text{rank}(x_i, y_a) > \text{rank}(x_i, y_b)\}|. \quad (17)$$

Regarding the label based metrics, there are two different ways to aggregate the values of the labels. The macro-averaging approach (see Equation 18) computes the metric independently for each label and then averages the obtained values to get the final measure. On the contrary, the micro-averaging approach (see Equation 19) first aggregates the counters for all the labels and then computes the metric only once. In the following equations TP , FP , TN and FN stand for *True Positives*, *False Positives*, *True Negatives* and *False Negatives*, respectively.

$$\text{MacroMetric} = \frac{1}{|L|} \sum_{l=1}^{|L|} \text{evalMetric}(TP_l, FP_l, TN_l, FN_l). \quad (18)$$

$$\text{MicroMetric} = \text{evalMetric} \left(\sum_{l=1}^{|L|} TP_l, \sum_{l=1}^{|L|} FP_l, \sum_{l=1}^{|L|} TN_l, \sum_{l=1}^{|L|} FN_l \right). \quad (19)$$

All the bipartition based metrics, such as *Precision*, *Recall* or *FMeasure*, can be computed as label based measures following these two approaches. In this category, there are as well as some ranking based metrics, such as *MacroAUC* (see Equation 20) and *MicroAUC* (see Equation 21).

$$\text{MacroAUC} = \frac{1}{|L|} \sum_{l=1}^{|L|} \frac{|\{x', x'' : \text{rank}(x', y_l) \geq \text{rank}(x'', y_l), (x', x'') \in X_l \times \bar{X}_l\}|}{|X_l| |\bar{X}_l|}, \quad (20)$$

$X_l = \{x_i : y_l \in Y_i\}$, $\bar{X}_l = \{x_i : y_l \notin Y_i\}$.

$$\text{MicroAUC} = \frac{|\{x', x'', y', y'' : \text{rank}(x', y') \geq \text{rank}(x'', y''), (x', y') \in S^+, (x'', y'') \in S^-\}|}{|S^+| |S^-|}, \quad (21)$$

$S^+ = \{(x_i, y) : y \in Y_i\}$, $S^- = \{(x_i, y) : y \notin Y_i\}$.

When the partition of the MLD for which the predictions have been obtained, and the predictions themselves are given to the `mldr_evaluate` function, a list of 20 measures is returned. For instance:

```
> # Get the true labels in emotions
> predictions <- as.matrix(emotions$dataset[, emotions$labels$index])
> # and introduce some noise
> predictions[sample(1:593, 100), sample(1:6, 100, replace = TRUE)] <-
+   sample(0:1, 100, replace = TRUE)
> # then evaluate the predictive performance
> res <- mldr_evaluate(emotions, predictions)
> str(res)
```

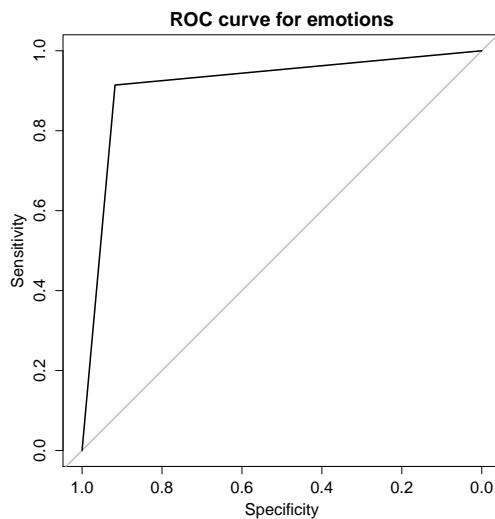


Figure 3: ROC curve plot for the data returned by `mldr_evaluate`.

```
List of 20
$ Accuracy      : num 0.917
$ AUC           : num 0.916
$ AveragePrecision: num 0.673
$ Coverage       : num 2.71
$ FMeasure        : num 0.952
$ HammingLoss     : num 0.0835
$ MacroAUC        : num 0.916
$ MacroFMeasure   : num 0.87
$ MacroPrecision   : num 0.829
$ MacroRecall      : num 0.915
$ MicroAUC        : num 0.916
$ MicroFMeasure   : num 0.872
$ MicroPrecision   : num 0.834
$ MicroRecall      : num 0.914
$ OneError         : num 0.116
$ Precision        : num 0.938
$ RankingLoss      : num 0.518
$ Recall          : num 0.914
$ SubsetAccuracy   : num 0.831
$ ROC             :List of 15
...
> plot(res$ROC, main = "ROC curve for emotions") # Plot ROC curve
```

If the `pROC` (Robin et al., 2011) package is available, this list will include non-null AUC (*Area Under the ROC Curve*) measures and also an element called `ROC`. The latter holds the information needed to plot the ROC (*Receiver Operating Characteristic*) curve, as shown in the last line of the previous example. The result would be a plot similar to that in Figure 3.

The `mldr` user interface

This package provides the user with a web-based graphical user interface on top of the `shiny` package, allowing to interactively manipulate measurements and obtain graphics and other results. Once `mldr` is loaded, this GUI can be launched from the R console with a single command:

```
> mldrGUI()
```

This will cause the user's default browser to start or open a new tab in which the GUI will be displayed, organized into a tab bar and a content pane. The tab bar allows the change of section so that different information is shown in the pane.

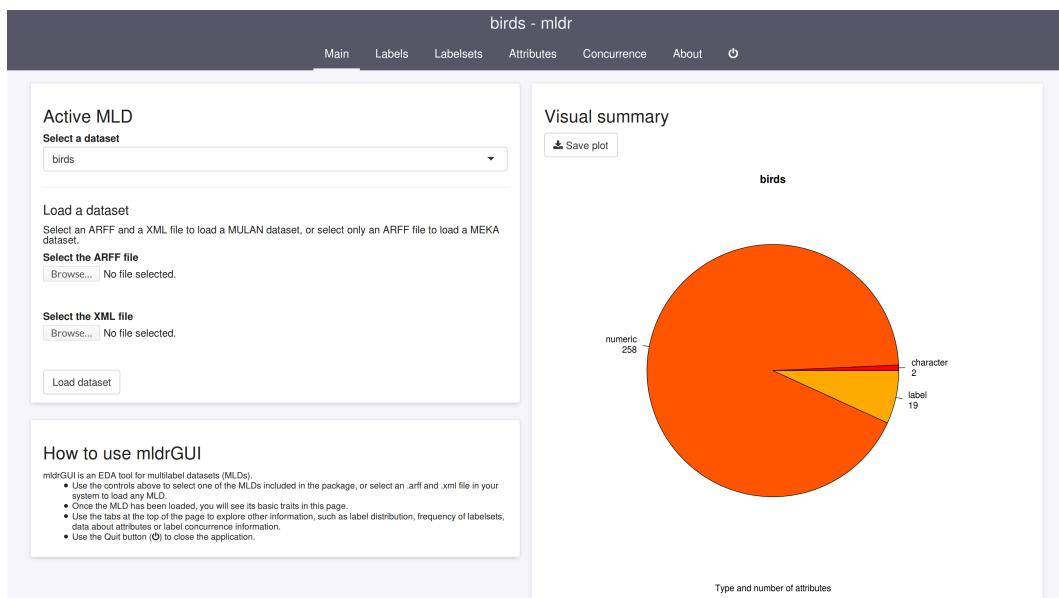


Figure 4: Main page of the **shiny** based graphical user interface.

The GUI will initially display the Main section, as shown in Figure 4. It contains options able to select an MLD from those available, and to load a new one by uploading its ARFF and XML files onto the application. On the right side, several plots are stacked. These show the amount of attributes of each type (numeric, character or label), the amount of labels per instance, the amount of instances corresponding to labels and the number of instances related to labelsets. Each plot can be saved as an image on the file system. Right below these graphics, some tables containing basic measures are shown. The first one lists generic measures related to the entire MLD, and is followed by measures specific to labels, such as *Card* or *Dens*. The last table shows a summary of measures for labelsets.

The Labels section contains a table enumerating each label of the MLD with its relevant details and measures: its index in the attribute list, its count and frequency, its *IRlbl* and its *SCUMBLE*. Labels in this table can be reordered using the headers, and filtered by the Search field. Furthermore, if the list is longer than the number specified in the Show field, it will be split into several pages. The data shown in all tables can be exported to files in several formats. On the right side, a plot shows the amount of instances that have each label. This is an interactive plot, and allows the range of labels to be manipulated.

Since relations between labels can determine the behavior of new data, studying labelsets is important in multilabel classification. Thus, the section named Labelsets provides information about them, listing each labelset along with its count. This list can be filtered and split into pages as well, and is accompanied by a bar plot showing the count of instances per labelset.

In order to obtain statistical measures about input attributes, the Attributes section organizes all of them into a paged table, displaying their type and some data or measures according to it. If the attribute is numeric, then there will be a table containing its minimum and maximum values, its quartiles and its mean. On the contrary, if the attribute takes values from a finite set, each possible value will be shown along with its count in the MLD.

Lastly, concurrence among labels is provenly a factor to take into account when applying pre-processing techniques to MLDs. For this reason, the Concurrence section attempts to create an easy way of visualizing concurrence among labels (see Figure 5), with a label concurrence plot displaying the selected labels in the left-side table and their cooccurrences represented by bands in the circle. By default, the ten labels with highest *SCUMBLE* are selected. The user is able to select and deselect other labels by clicking their corresponding row on the table.

Summary

In this paper the **mldr** package, aimed to provide exploratory analysis and manipulation tools for MLDs, has been introduced. The functions supplied by this package allow both loading existing MLDs and generating new ones. Several characterization measures and specific plots can be obtained for any MLD, and the content of an MLD can be extracted, filtered and joined, producing new MLDs. Any MLD can be transformed into a set of binary datasets or a multiclass dataset by means of the

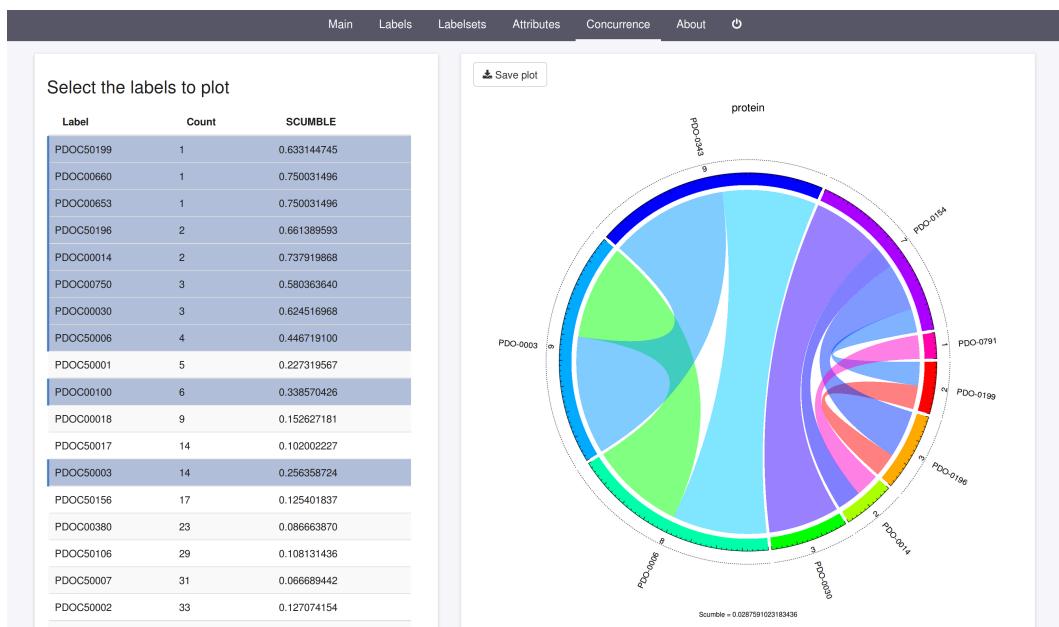


Figure 5: The plots can be customized and saved.

transformation functions of package **mldr**. Finally, a web-based graphical user interface eases the access to most of this functionality for everyone.

In its current version, package **mldr** is a strong base to develop any preprocessing method for MLDs, as has been shown. The development of the **mldr** package will continue in the near future by including the tools needed to implement and evaluate multilabel classifiers. With this foundation, we aim to encourage other developers to incorporate their own algorithms into **mldr**, as we will do in forthcoming releases.

Acknowledgment

This paper is partially supported by the project TIN2012-33856 of the Spanish Ministry of Science and Technology.

Bibliography

- M. Boutell, J. Luo, X. Shen, and C. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004. [p152]
- W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2015. URL <https://CRAN.R-project.org/package=shiny>. R package version 0.12.2. [p150, 250, 279]
- F. Charte, A. Rivera, M. J. Jesus, and F. Herrera. Concurrence among imbalanced labels and its influence on multilabel resampling algorithms. In *Proceedings of the 9th International Conference on Hybrid Artificial Intelligent Systems, Salamanca, Spain, HAIS'14*, volume 8480, 2014. [p151, 154]
- F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera. Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, 163(0):3–16, 2015. [p151]
- S. Diplaris, G. Tsoumakas, P. Mitkas, and I. Vlahavas. Protein classification with multiple algorithms. In *Proceedings of the 10th Panhellenic Conference on Informatics, Volos, Greece, PCI'05*, pages 448–456, 2005. [p150]
- P. Duygulu, K. Barnard, J. de Freitas, and D. Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *Proceedings of the 7th European Conference on Computer Vision-Part IV, Copenhagen, Denmark, ECCV'02*, pages 97–112, 2002. [p150, 153]
- J. Fox. The R Commander: A basic statistics graphical user interface to R. *Journal of Statistical Software*, 14(9):1–42, 2005. URL <http://www.jstatsoft.org/v14/i09>. [p152]

- S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data Mining*, volume 3056, pages 22–30, 2004. [p152]
- Z. Gu, L. Gu, R. Eils, M. Schlesner, and B. Brors. circlize implements and enhances circular visualization in R. *Bioinformatics*, 30:2811–2812, 2014. [p152]
- M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11, 2009. [p149]
- K. Hornik, C. Buchta, and A. Zeileis. Open-source machine learning: R meets Weka. *Computational Statistics*, 24(2):225–232, 2009. [p149]
- B. Klimt and Y. Yang. The Enron corpus: A new dataset for email classification research. In *Proceedings of the 5th European Conference on Machine Learning, Pisa, Italy, ECML'04*, pages 217–226. 2004. [p150]
- D. T. Lang and the CRAN Team. *XML: Tools for Parsing and Generating XML within R and S-Plus*, 2015. URL <https://CRAN.R-project.org/package=XML>. R package version 3.98-1.3. [p152]
- J. Read and P. Reutemann. MEKA: A multi-label extension to WEKA, 2012. URL <http://meka.sourceforge.net/>. [p149]
- J. Read, B. Pfahringer, and G. Holmes. Multi-label classification using ensembles of pruned sets. In *Proceedings of the 8th IEEE International Conference on Data Mining, 2008. ICDM'08*, pages 995–1000. IEEE, 2008. [p152]
- J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85:333–359, 2011. [p152]
- X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. pROC: An open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12, 2011. [p159]
- G. Tsoumakas and I. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *Proceedings of the 18th European Conference on Machine Learning, Warsaw, Poland, ECML'07*, volume 4701, pages 406–417, 2007. [p152]
- G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, chapter 34, pages 667–685. Springer, Boston, MA, 2010. [p150]
- G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. MULAN: A Java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. [p149]
- H. Wickham and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2015. URL <https://CRAN.R-project.org/package=devtools>. R package version 1.8.0. [p152, 279]
- G. J. Williams. *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*. Use R! Springer, 2011. [p152]
- M. Zhang and Z. Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, Aug. 2014. [p152]

Francisco Charte
Department of Computer Science and Artificial Intelligence
University of Granada
Granada
Spain
fcharte@ugr.es

David Charte
University of Granada
Granada
Spain
fdavidcl@correo.ugr.es

PracTools: Computations for Design of Finite Population Samples

by Richard Valliant, Jill A. Dever, and Frauke Kreuter

Abstract PracTools is an R package with functions that compute sample sizes for various types of finite population sampling designs when totals or means are estimated. One-, two-, and three-stage designs are covered as well as allocations for stratified sampling and probability proportional to size sampling. Sample allocations can be computed that minimize the variance of an estimator subject to a budget constraint or that minimize cost subject to a precision constraint. The package also contains some specialized functions for estimating variance components and design effects. Several finite populations are included that are useful for classroom instruction.

Introduction

Samples from finite populations are one of the mainstays of research in demographics, economics, and public health. In the U.S., for example, the Consumer Price Index is based on samples of business establishments and households (Bureau of Labor Statistics, 2013, chap. 17); the unemployment rate is estimated from the Current Population Survey, which is a sample of households (Bureau of Labor Statistics, 2013, chap. 1); and various health characteristics of the population are estimated from the National Health Interview Survey (Center for Disease Control and Prevention, 2013b) and the National Health and Nutrition Examination Survey (Center for Disease Control and Prevention, 2013a) both of which are household surveys. Smaller scale academic and marketing research surveys are also typically done using finite population samples.

Standard techniques used in sample design are stratification, clustering, and selection with varying probabilities. Depending on the units to be surveyed (e.g., persons, schools, businesses, institutions) and the method of data collection (e.g., telephone, personal interview, mail survey), the samples may be selected in one or several stages. There are several packages in R that can select samples and analyze survey data. Among the packages for sample selection are `pps` (Gambino, 2012), `sampling` (Tillé and Matei, 2013), `samplingbook` (Manitz, 2013), and `simFrame` (Alfons et al., 2010). The `survey` package (Lumley, 2010, 2014) has an extensive set of features for creating weights, generating descriptive statistics, and fitting models to survey data.

A basic issue in sample design is how many units should be selected at each stage in order to efficiently estimate population values. If strata are used, the number of units to allocate to each stratum must be determined. In this article, we review some basic techniques for sample size determination in complex samples and the package `PracTools` (Valliant et al., 2015) that contains specialized routines to facilitate the calculations, most of which are not found in the packages noted above. We briefly summarize some of selection methods and associated formulas used in designing samples and describe the capabilities of `PracTools`. The penultimate section presents a few examples using the `PracTools` functions and the final section is a conclusion.

Designing survey samples

Complex samples can involve any or all of stratification, clustering, multistage sampling, and sampling with varying probabilities. This section discusses these techniques, why they are used, and formulas that are needed for determining sample allocations. Many texts cover these topics, including Cochran (1977), Lohr (1999), Särndal et al. (1992), and Valliant et al. (2013).

Simple random sampling

Simple random sampling without replacement (*srswor*) is a method of probability sampling in which all samples of a given size n have the same probability of selection. The function `sample` in the `base` package in R can be used to select simple random samples either with or without replacement. One way of determining an *srswor* sample size is to specify that a population value θ be estimated with a certain coefficient of variation (CV) which is defined as the ratio of the standard error of the estimator, $\hat{\theta}$, to the value of the parameter: $CV(\hat{\theta}) = \sqrt{Var(\hat{\theta})}/\theta$. For example, suppose that y_k is a value associated with element k , U denotes the set of all elements in the universe, N is the number of elements in the population, and the population parameter to be estimated is the mean, $\bar{y}_U = \sum_{k \in U} y_k/N$. With a simple random sample, this can be estimated by the sample mean, $\bar{y}_s = \sum_{k \in s} y_k/n$, where s is the set

of sample elements and n is the sample size. Setting the required CV of \bar{y}_s to some desired value CV_0 in an *srswor* leads to a sample size of

$$n = \frac{\frac{S_U^2}{\bar{y}_U^2}}{CV_0^2 + \frac{S_U^2}{N\bar{y}_U^2}}, \quad (1)$$

where S_U^2 is the population variance of the y_k 's. The term S_U^2/\bar{y}_U^2 is referred to as the *unit relvariance*. If y_k is a 0/1 variable identifying whether an element has a characteristic or not, then $S_U^2 = N(N - 1)^{-1}p_U(1 - p_U)$ where p_U is the proportion in the population with the characteristic. The function `nCont` in **PracTools** will make the computation in (1). In a real application, the population values in (1) must be estimated from a sample or guessed based on prior knowledge.

Another way of determining a sample size is to set a tolerance for how close the estimate should be to the population value. If the tolerance (sometimes called the *margin of error*) is e_0 and the goal is to be within e_0 of the population mean with probability $1 - \alpha$, this translates to requiring $Pr(|\bar{y}_s - \bar{y}_U| \leq e_0) = 1 - \alpha$. This is equivalent to setting the half-width of a $100(1 - \alpha)\%$ normal approximation, two-sided confidence interval (CI) to $e_0 = z_{1-\alpha/2} \sqrt{V(\bar{y}_s)}$. The notation z_ϵ denotes the 100ϵ percentile of the standard normal distribution. The sample size required to accomplish this is

$$n = \frac{z_{1-\alpha/2}^2 S_U^2}{e_0^2 + z_{1-\alpha/2}^2 S_U^2 / N}. \quad (2)$$

One could also require that the relative absolute error, $|(\bar{y}_s - \bar{y}_U)/\bar{y}_U|$, be less than e_0 with a specified probability. In that case, (2) is modified by replacing S_U^2 with the unit relvariance, S_U^2/\bar{y}_U^2 . Both calculations can be made using the function `nContMoe` in **PracTools**. When estimating a proportion, there are options other than a normal approximation confidence interval on p_U for setting a margin of error. Two are to work with the log-odds, $p_U/(1 - p_U)$, or to use the method due to [Wilson \(1927\)](#), which are both available in **PracTools**.

Another estimand in a survey might be the difference in means or proportions. The difference could be between two disjoint groups or between the estimates for the same group at two different time periods. The standard approach in such a case would be to find a sample size that will yield a specified power for detecting a particular size of the difference. The functions `power.t.test` and `power.prop.test` in the R package `stats` will do this for independent simple random samples.

The case of partially overlapping samples can also be handled (e.g., see [Woodward 1992](#)). For example, persons may be surveyed at some baseline date and then followed-up at a later time. An estimate of the difference in population means may be desired, but the samples do not overlap completely because of dropouts, planned sample rotation, or nonresponse. Such non-overlap would be common in panel surveys. Suppose that s_1 and s_2 are the sets of sample units with data collected only at times 1 and 2, and that s_{12} denotes the overlap. Thus, the full samples at times 1 and 2 are $s_1 \cup s_{12}$ and $s_2 \cup s_{12}$. Also, suppose that the samples at the two time periods are simple random samples. Assume that the samples at times 1 and 2 are not necessarily the same size, so that $n_1 = rn_2$ for some positive number r . The samples might be of different sizes because of other survey goals or because the budget for data collection is different for the two times. A case that is covered by the analysis below is one where an initial sample of size n_1 is selected, a portion of these respond at time 2, and additional units are selected to obtain a total sample of size n_2 for time 2. Taking the case of simple random sampling, the difference in means at the first and second time points can be written as

$$\hat{d} = \hat{x} - \hat{y} = \frac{1}{n_1} \sum_{s_1} x_i - \frac{1}{n_2} \sum_{s_2} y_i + \sum_{s_{12}} \left(\frac{x_i}{n_1} - \frac{y_i}{n_2} \right).$$

The variance can be expressed as

$$Var(\hat{d}) = \frac{\sigma_x^2}{n_1} + \frac{\sigma_y^2}{n_2} - 2\sigma_{xy} \frac{n_{12}}{n_1 n_2}, \quad (3)$$

where σ_x^2 and σ_y^2 are the population variances at the two time periods, σ_{xy} is the element-level covariance, and n_{12} is the number of units in s_{12} . Writing $n_{12} = \gamma n_1$ and $r = n_1/n_2$, the variance becomes $Var(\hat{d}) = \frac{1}{n_1} [\sigma_x^2 + r\sigma_y^2 - 2\gamma r\sigma_{xy}]$. For a one-sided test of $H_0 : \mu_D = 0$ versus $H_A : \mu_D = \delta$ to be done with power β , the required sample size n_1 is

$$n_1 = \frac{1}{\delta^2} [\sigma_x^2 + r\sigma_y^2 - 2\gamma r\sigma_{xy}] (z_{1-\alpha} - z_\beta)^2. \quad (4)$$

The value of n_2 is then determined from $r = n_1/n_2$. The function `nDep2sam` in **PracTools** will perform

this calculation. `nProp2sam` will do a similar calculation for testing the difference in proportions with overlapping samples.

Probability proportional to size sampling

Probability proportional to size (*pps*) sampling can be very efficient if the measure of size (*mos*) used for sampling is correlated with the quantities measured in a survey. For example, enrollment in an elementary school may be related to the number of children receiving a government assistance program. In an establishment survey, the total number of employees is often correlated with other employment counts in the establishment, like the number who participate in a retirement plan. In a hospital survey, the number of inpatient beds is usually related to numbers of patients discharged in a month's time. Household samples are often selected using several stages, the first of which is a sample of geographic areas. An effective *mos* is typically the number of persons or housing units in each geographic area.

If a variable follows a certain regression structure in the population, then an optimal measure of size can be estimated. The key finding is due to [Godambe and Joshi \(1965\)](#). [Isaki and Fuller \(1982\)](#) extended this to a linear model M where $E_M(y_i) = \mathbf{x}_i^T \beta$ and $\text{Var}_M(y_i) = v_i$ with \mathbf{x}_i defined as a vector of x 's (auxiliary variables), and β is a vector of regression slopes of the same dimension as \mathbf{x}_i . Assume that a population total is estimated and a regression estimator is used that is approximately unbiased when averaging over the model and a probability sampling design. In that case, $\sqrt{v_i}$ is the best *mos* for *pps* sampling.

A model that may fit some establishment or institutional populations reasonably well has a variance with the form, $\text{Var}_M(y_i) = \sigma^2 x_i^\gamma$, where x_i is a *mos* and γ is a power. Typical values of γ are in the interval $[0, 2]$. The function `gammaFit` in **PracTools** returns an estimate of γ using an iterative algorithm. The algorithm is based on initially running an ordinary least squares (OLS) regression of y_i on \mathbf{x}_i . The OLS residuals, e_i , are then used to regress $\log(e_i^2)$ on $\log(x_i)$ with an intercept. The slope in this regression is an estimate of γ . This procedure iterates by using the latest estimate of γ in a weighted least squares regression of y_i on \mathbf{x}_i . The parameter γ is then re-estimated in the logarithmic regression. The algorithm proceeds until some user-controllable convergence criteria are met.

The variance formulas for *pps* without replacement sampling are difficult or impossible to use for sample size determination because they involve joint selection probabilities of units and the sample size is not readily accessible. One practical approach is to use a variance formula appropriate for *pps* with replacement (*ppswr*) sampling. The simplest estimator of the mean that is usually studied with *ppswr* sampling is called “ p -expanded with replacement” (*pwr*) ([Särndal et al., 1992](#), chap. 2) and is defined as

$$\hat{y}_{pwr} = \frac{1}{Nn} \sum_s \frac{y_i}{p_i}, \quad (5)$$

where p_i is the probability that element i would be selected in a sample of size 1. A unit is included in the sum as many times as it is sampled. The variance of \hat{y}_{pwr} in *ppswr* sampling is

$$\text{Var}(\hat{y}_{pwr}) = \frac{1}{N^2 n} \sum_U p_i \left(\frac{y_i}{p_i} - t_U \right)^2 \equiv \frac{V_1}{N^2 n}, \quad (6)$$

where t_U is the population total of y . If the desired coefficient of variation is CV_0 , Equation (6) can be solved to give the sample size as

$$n = \frac{V_1}{N^2} \frac{1}{\bar{y}_U^2 CV_0^2}. \quad (7)$$

We later give an example of how (7) may be evaluated using **PracTools**.

Stratified sampling

Stratified sampling is a useful way of restricting the dispersion of a sample across groups in a population. It can also lead to improvements in precision of overall estimates if an efficient allocation to the strata is used. For example, establishments can be stratified by type of business (retail, wholesale, manufacturing, etc.). Other methods of creating strata are provided by the R package `stratification` ([Baillargeon and Rivest, 2014](#)). Given that strata have been created, there are various ways of efficiently allocating a sample to strata: (i) minimize the variance of an estimator given a fixed total sample size (Neyman allocation), (ii) minimize the variance of an estimator for a fixed total budget, (iii) minimize the total cost for a target CV or variance of an estimator, or (iv) allocate the sample subject to several CV or cost criteria subject to a set of constraints on stratum sample sizes or other desiderata. The last is referred to as *multicriteria optimization*.

The standard texts noted earlier give closed form solutions for (i) and (ii). For example, suppose a mean is estimated, an *srswor* is to be selected within each stratum ($h = 1, \dots, H$), and that the total cost can be written as $C = C_0 + \sum_{h=1}^H c_h n_h$ where C_0 denotes fixed costs that do not vary with the sample size, c_h is the cost per-element in stratum h , and n_h is the number of elements sampled from stratum h . The allocation to strata that minimizes the variance of the estimated mean subject to a fixed total budget C is

$$n_h = (C - C_0) \frac{W_h S_h / \sqrt{c_h}}{\sum_{h=1}^H (W_h S_h \sqrt{c_h})}, \quad (8)$$

where W_h is the proportion in the population in stratum h and S_h is the population standard deviation in stratum h of the variable whose mean is estimated. This and the allocations for (i) and (iii) above can be found using `strAlloc` in **PracTools**.

Multicriteria optimization and allocations with constraints are more realistic for multipurpose surveys. In some cases, solutions to particular allocation problems are available as in [Gabler et al. \(2012\)](#). More generally, the `alabama` ([Varadhan, 2015](#)) and `Rsolnp` ([Ghalanos and Theussl, 2014](#)) packages will solve nonlinear optimization problems with constraints and can be very useful for complicated sample allocations. Among the constraints that are used in practical work are ones on minimum and maximum stratum sample sizes and relvariances of overall and individual stratum estimates. [Theussl and Borchers \(2015\)](#) present a CRAN Task View on optimization and mathematical programming in R. An R package that will form strata for multipurpose samples is `SamplingStrata` ([Barcaroli, 2014](#)). Also, the Solver add-on to Microsoft *Excel*® ([Fylstra et al., 1998](#)) will handle allocation problems that are quite complex and is easy to use. Use of these tools for sample allocation is covered in some detail in [Valliant et al. \(2013, chap. 5\)](#).

Two- and three-stage sampling

Two- and three-stage sampling is commonplace in household surveys but can be also used in other situations. For example, the U.S. National Compensation Survey selects a three-stage sample—geographic areas, establishments, and occupations—to collect compensation data ([Bureau of Labor Statistics, 2013](#), chap. 8). Allocating the sample efficiently requires estimates of the contribution to the variance of an estimate by each stage of sampling.

As an example, consider a two-stage design in which the primary sampling units (PSUs) are selected with varying probabilities and with replacement and elements are selected at the second-stage by *srswor*. As noted earlier, determining sample sizes as if the PSUs are selected with-replacement is a standard workaround in applied sampling to deal with the fact that without-replacement variance formulas for *pps* samples are too complex to use for finding allocations. (Other selection methods along with three-stage designs are reviewed in [Valliant et al. \(2013, chap.9.\)](#).) Let m be the number of sample PSUs, N_i be the number of elements in the population for PSU i , and suppose that the same number of elements, \bar{n} , is selected from each PSU. The *pwr*-estimator of a total is

$$\hat{t}_{pwr} = \frac{1}{m} \sum_{i \in s} \frac{\hat{t}_i}{p_i},$$

where $\hat{t}_i = \frac{N_i}{\bar{n}} \sum_{k \in s_i} y_{ik}$ is the estimated total for PSU i from a simple random sample and p_i is the 1-draw selection probability of PSU i , i.e., the probability in a sample of size one. The variance of \hat{t}_{pwr} is

$$V(\hat{t}_{pwr}) = \frac{S_{U1(pwr)}^2}{m} + \frac{1}{m\bar{n}} \sum_{i \in U} \left(1 - \frac{\bar{n}}{N_i}\right) \frac{N_i^2 S_{U2i}^2}{p_i},$$

where U is the universe of PSUs, $S_{U1(pwr)}^2 = \sum_{i \in U} p_i \left(\frac{t_i}{p_i} - t_U \right)^2$, t_i is the total of y for PSU i , and S_{U2i}^2 is the population variance of y within PSU i . Dividing this by t_U^2 and assuming that the within-PSU sampling fraction, \bar{n}/N_i , is negligible, we obtain the relative variance (relvariance) of \hat{t}_{pwr} as, approximately,

$$\frac{V(\hat{t}_{pwr})}{t_U^2} = \frac{B^2}{m} + \frac{W^2}{m\bar{n}} = \frac{\tilde{V}}{m\bar{n}} k [1 + \delta(\bar{n} - 1)], \quad (9)$$

with $\tilde{V} = S_U^2/\bar{y}_U^2$, \bar{y}_U is the population mean per element, $k = (B^2 + W^2)/\tilde{V}$, $B^2 = S_{U1(pwr)}^2/t_U^2$, $W^2 = t_U^{-2} \sum_{i \in U} N_i^2 S_{U2i}^2/p_i$, and $\delta = B^2/(B^2 + W^2)$.

A simple cost function for two-stage sampling assumes that there is a cost per sample PSU and a cost per sample element of collecting and processing data. We model the total cost as

$$C = C_0 + C_1 m + C_2 m \bar{n},$$

where

- C_0 = costs that do not depend on the number of sample PSUs and elements;
- C_1 = cost per sample PSU; and
- C_2 = cost per element within PSU.

The optimal number of units to select per PSU, i.e., the number that minimizes the approximate relvariance, is

$$\bar{n}_{opt} = \sqrt{\frac{C_1}{C_2} \frac{1-\delta}{\delta}}. \quad (10)$$

Only the ratio of the unit costs needs to be known in order to compute \bar{n}_{opt} . To find the optimal m for a fixed total cost, we substitute \bar{n}_{opt} into the cost function to obtain

$$m_{opt} = \frac{C - C_0}{C_1 + C_2 \bar{n}_{opt}}. \quad (11)$$

Alternatively, to find the optimal m for a fixed relvariance, CV_0^2 , \bar{n}_{opt} is substituted into the relvariance formula (9). **clus0pt2** in **PracTools** will do either of these calculations.

For three-stage sampling, suppose that m PSUs are selected with varying probabilities and with-replacement, \bar{n} secondary sampling units (SSUs) are selected within each PSU by *srswor*, and \bar{q} elements are sampled by *srswor* within each sample SSU. This design is referred to as *ppswr/srs/srs* below. The relvariance of the *pwr*-estimator of a total in such a three-stage sample (with a negligible sampling fraction in the second and third stages) can be written as, e.g., see Hansen et al. (1953) and Valliant et al. (2013, chap.9):

$$\begin{aligned} \frac{V(\hat{t}_{pwr})}{t_U^2} &\doteq \frac{B^2}{m} + \frac{W_2^2}{m\bar{n}} + \frac{W_3^2}{m\bar{n}\bar{q}} \\ &= \frac{\tilde{V}}{m\bar{n}\bar{q}} \{k_1\delta_1\bar{n}\bar{q} + k_2[1 + \delta_2(\bar{q} - 1)]\}, \end{aligned} \quad (12)$$

where $B^2 = M^2 S_{U1}^2 / t_U^2$, $W_2^2 = M \sum_{i \in U} N_i^2 S_{U2i}^2 / t_U^2$, and $W_3^2 = M \sum_{i \in U} N_i \sum_{j \in U_i} Q_{ij}^2 S_{U3ij}^2 / t_U^2$. The variance components and other terms in Equation (12) are defined as:

$\tilde{V} = \frac{1}{Q-1} \sum_{i \in U} \sum_{j \in U_i} \sum_{k \in U_{ij}} (y_k - \bar{y}_U)^2 / \bar{y}_U^2$ is the unit relvariance of y in the population with Q being the total number of elements;

$S_{U1}^2 = \frac{\sum_{i \in U} (t_{ij} - \bar{t}_{U1})^2}{M-1}$, the variance among the M PSU totals;

$S_{U2i}^2 = \frac{1}{N_i-1} \sum_{j \in U_i} (t_{ij} - \bar{t}_{Ui})^2$ is the unit variance of the N_i SSU totals in PSU i with $t_{ij} = \sum_{k \in U_{ij}} y_k$ being the population total for PSU/SSU ij , $\bar{t}_{Ui} = \sum_{j \in U_i} t_{ij} / N_i$ is the average total per SSU in PSU i ;

$S_{U3ij}^2 = \frac{1}{Q_{ij}-1} \sum_{k \in U_{ij}} (y_k - \bar{y}_{Uij})^2$ is the unit variance among the Q_{ij} elements in PSU/SSU ij with $\bar{y}_{Uij} = \sum_{k \in U_{ij}} y_k / Q_{ij}$.

$k_1 = (B^2 + W^2) / \tilde{V}$;

$W^2 = \frac{1}{t_U^2} \sum_{i \in U} Q_i^2 S_{U3i}^2 / p_i$ with Q_i being the number of elements in PSU i ,

$S_{U3i}^2 = \frac{1}{Q_i-1} \sum_{j \in U_i} \sum_{k \in U_{ij}} (y_k - \bar{y}_{Ui})^2$ and $\bar{y}_{Ui} = \sum_{j \in U_i} \sum_{k \in U_{ij}} y_k / Q_i$; i.e., S_{U3i}^2 is the element-level variance among all elements in PSU i ; and

$k_2 = (W_2^2 + W_3^2) / \tilde{V}$;

$\delta_1 = B^2 / (B^2 + W^2)$;

$\delta_2 = W_2^2 / (W_2^2 + W_3^2)$.

The terms δ_1 and δ_2 are referred to as *measures of homogeneity*, as is δ for two-stage sampling. Equation (12) is useful for sample allocation because the measures of homogeneity are in $[0, 1]$, k_1 and k_2 are usually near 1, and \tilde{V} can usually be estimated.

To arrive at an optimal allocation, costs need to be considered. A cost function for three-stage sampling, analogous to the one for two-stage sampling, is

$$C = C_0 + C_1 m + C_2 m \bar{n} + C_3 m \bar{n} \bar{q}. \quad (13)$$

The term C_0 is again costs that do not depend on the sample sizes at different stages; C_1 is the cost per PSU; C_2 is the cost per SSU; and C_3 is the cost per element within each SSU. Minimizing the $ppswr/srs/srs$ relvariance in Equation (12) subject to a fixed total cost gives the following optima:

$$\bar{q}_{opt} = \sqrt{\frac{1 - \delta_2}{\delta_2} \frac{C_2}{C_3}}, \quad (14)$$

$$\bar{n}_{opt} = \frac{1}{\bar{q}} \sqrt{\frac{1 - \delta_2}{\delta_1} \frac{C_1 k_2}{C_3 k_1}}, \quad (15)$$

$$m_{opt} = \frac{C - C_0}{C_1 + C_2 \bar{n} + C_3 \bar{n} \bar{q}}. \quad (16)$$

If a target relvariance is set at CV_0^2 , then the equations for finding the optima \bar{q}_{opt} and \bar{n}_{opt} are the same. The optimum number of PSUs is found by substituting \bar{n}_{opt} and \bar{q}_{opt} into the relvariance in Equation (12). `clus0pt3` will do these computations for three-stage samples.

Two-phase sampling

In finite population sampling, a distinction is drawn between *multistage* sampling and *multiphase* sampling. In a multiphase sample, an initial sample is selected, some characteristics of the units are observed, and a decision is made about how to select a subsample from the initial sample based on what has been observed. In multistage sampling, the same design is used in later stages regardless of what was found in the first-stage units. There is a more technical definition of the difference between multistage and multiphase, but it is unimportant for this discussion. An example of two-phase sampling is to select a subsample of nonrespondents to the initial phase to attempt to get them to cooperate. This is known as a *nonresponse follow-up study* (NRFU).

Another type of two-phase design is *double sampling for stratification*. In this design, information is collected in the first phase which is then used to stratify elements for second phase sampling. For example, researchers working to develop a case definition for undiagnosed medical symptoms in U.S. personnel serving in the 1991 Persian Gulf War surveyed a stratified simple random sample of Gulf War-era veterans (Iannacchione et al., 2011). Based on survey responses in the first phase, respondents were classified as likely having or not having a certain type of illness. Blood specimens were requested from randomly sampled phase-1 respondents within the illness strata and analyzed using expensive tests.

As an illustration, take the case of double sampling for stratification. Cochran (1977) and Neyman (1938) give the two-phase variance of an estimated mean or proportion when phase-1 is a simple random sample of $n_{(1)}$ elements, phase-2 is a stratified simple random sample (*stsrs*) of $n_{(2)}$ elements, and an optimal allocation to strata is used in the second phase. The sampling fractions at both stages are assumed to be negligible. The optimal proportion of the phase-2 sample to assign to stratum h for estimating the population mean is $n_{(2)h}/n_{(2)} = W_h S_h / \sum_h W_h S_h$. The formula for the variance of an estimated stratified mean with this allocation is

$$V_{opt} = \frac{\sum_h W_h (\bar{y}_{Uh} - \bar{y}_U)^2}{n_{(1)}} + \frac{(\sum_h W_h S_h)^2}{n_{(2)}} \equiv \frac{V_{(1)}}{n_{(1)}} + \frac{V_{(2)}}{n_{(2)}}, \quad (17)$$

where \bar{y}_{Uh} is the stratum h population mean. The phase-2 subsampling rate from the phase-1 sample units that minimizes Equation (17) is

$$\frac{n_{(2)}}{n_{(1)}} = \sqrt{\frac{V_{(2)}}{V_{(1)}} \left/ \frac{c_{(2)}}{c_{(1)}}\right.},$$

where $c_{(1)}$ and $c_{(2)}$ are the per-unit costs in the first and second-phases, respectively. The formulas for the phase-1 and phase-2 sample sizes that minimize V_{opt} subject to a fixed total cost C are

$$n_{(1)} = \frac{C}{c_{(1)} + c_{(2)} \sqrt{K}}, \quad n_{(2)} = n_{(1)} \sqrt{K},$$

where

$$K = \left(V_{(2)} / V_{(1)} \right) \left/ \left(c_{(2)} / c_{(1)} \right) \right..$$

The function `dub` in **PracTools** will calculate the optimal second-phase sampling fraction and the phase-1 and phase-2 sample sizes that will minimize the variance in Equation (17). The function also computes the size of an *srs* that would cost the same as the two-phase sample and the ratio of the

two-phase stratified variance to the *srs* variance.

Extension to nonlinear estimators and limitations

PracTools will calculate sample sizes for estimators whose variance can be written in one of the forms given in Section [Designing survey samples](#). This directly covers linear estimators of means and totals. A nonlinear estimator that is a differentiable function of a vector of estimated totals is also covered. But, a user must do some work to linearize the estimator and determine the inputs that are required for a **PracTools** function. Suppose that the estimator is $\hat{\theta} = f(\hat{t}_1, \dots, \hat{t}_p)$ where f is a differentiable function and \hat{t}_j is a linear estimator of a population total, t_j ($1, 2, \dots, p$). The linear approximation to $\hat{\theta} - \theta$ is

$$\hat{\theta} - \theta \doteq \sum_{j=1}^p \frac{\partial f}{\partial t_j} (\hat{t}_j - t_j), \quad (18)$$

where $\theta = f(t_1, \dots, t_p)$, and the partials are evaluated at the population values. The repeated sampling variance of this approximation is then the same as the variance of $\sum_{j=1}^p \frac{\partial f}{\partial t_j} \hat{t}_j$ since θ and t_j are treated as constants. Taking the case of two-stage sampling, suppose that the estimator of the total for variable j is $\hat{t}_j = \sum_{i \in s} \sum_{k \in s_i} w_{ik} y_{ik}(j)$ where w_{ik} is a weight for element k in PSU i , $y_{ik}(j)$ is its data value, s is the set of sample PSUs, and s_i is the sample of elements within SSU i . Substituting this into (18) and reversing the order of summation between PSUs and variables leads to the expression

$$\hat{z} = \underbrace{\sum_{i \in s} \sum_{j=1}^p \frac{\partial f}{\partial t_j} \hat{t}_i(j)}_{\hat{z}_i}. \quad (19)$$

where $\hat{t}_i(j) = \sum_{k \in s_i} w_{ik} y_{ik}(j)$. One then computes the design-variance of \hat{z} based on the particular sample design used and the form of the derivatives. This general approach is known as the *linear substitute* method and is described in detail in [Wolter \(2007\)](#).

In a two-stage sample where m PSUs are selected with replacement and with varying probabilities, and \bar{n} elements are selected by simple random sampling from each PSU, (9) applies. The weight is defined as $w_{ik} = (mp_i)^{-1} (N_i/\bar{n})$ where p_i is the 1-draw selection probability, as before. If the estimator is the mean computed as $\hat{\theta} = \hat{t}_y/\hat{M}$ with $\hat{M} = \sum_{i \in s} \sum_{k \in s_i} w_{ik}$, then $\hat{z}_i = (Mmp_i)^{-1} N_i e_i$ where $e_i = \bar{y}_i - \bar{y}_U$ with y_i being the sample mean of y in PSU i and \bar{y}_U the population mean. Expression (19) becomes

$$\hat{z} = \frac{1}{Mm} \sum_{i \in s} \frac{\hat{t}_{zi}}{p_i},$$

with $\hat{t}_{zi} = (N_i/\bar{n} \sum_{k \in s_i} e_{ik})$ and $e_{ik} = y_{ik} - \bar{y}_U$. Expression (9) would then be evaluated with e_{ik} replacing y_{ik} . With the linear substitute method, residuals typically appear in the linear approximation and are the basis for a variance estimator. Population quantities, like M and \bar{y}_U , are replaced by sample estimates in a variance estimator. The Section [Examples](#) gives an illustration of this method using one of the datasets in **PracTools**.

Nonetheless, there are types of estimators that our package does not cover. Quantile estimators require a special approximation and variance formula ([Francisco and Fuller, 1991](#)) that does not come from the standard linearization approach. The Gini coefficient, used as a measure of income inequality (e.g., [Deaton, 1997](#)), is another example of an estimator that is too complicated to linearize using the methods above.

The R package PracTools

PracTools is a collection of specialized functions written in R along with several example finite populations that can be used for teaching. The package is available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=PracTools>. Because the function code is visible, the routines can be modified (and improved) by any user. A brief description of the functions is given in Table 1. The use of the functions, their input parameters, and the values they return are described in the help files. The package also contains nine example populations of different types. These are listed in Table 2.

Function	Purpose
BW2stagePPS, BW2stageSRS	Variance components in two-stage samples from a population frame
BW2stagePPSe	Estimated variance components in two-stage samples from a sample
BW3stagePPS	Variance components in three-stage samples from a population frame
BW3stagePPSe	Estimated variance components in three-stage samples from a sample
clusOpt2	Optimal allocation in a two-stage sample
clusOpt2fixedPSU	Optimal second-stage sample size in a two-stage sample when the PSU sample is fixed
clusOpt3	Optimal allocation in a three-stage sample
clusOpt3fixedPSU	Optimal second and third-stage sample sizes in a three-stage sample when the PSU sample is fixed
CVcalc2, CVcalc3	Compute the coefficient of variation of an estimated total in two- and three-stage designs
deffH, deffK, deffS	Henry, Kish, and Spencer design effects
dub	Allocation of double sample for stratification
gammaFit	Estimate variance power in a linear model
nCont, nContMoe	Sample size to meet CV, variance, or margin of error targets (continuous variable)
nDep2sam	Sample sizes for two-sample comparison of means with overlapping samples (continuous variable)
nLogOdds	Sample size calculation for a proportion using log-odds method
nProp	Sample size calculation for a proportion using target CV or variance
nProp2sam	Sample sizes for two-sample comparison of proportions with overlapping samples (continuous variable)
nPropMOE	Sample size calculation for a proportion using margin of error target
NRFUopt	Sample sizes for two-phase nonresponse follow-up study
nWilson	Sample size calculation for a proportion using Wilson method
pclass	Form nonresponse adjustment classes based on propensity scores
strAlloc	Sample allocation in stratified samples

Table 1: Functions in **PracTools**.

Population	Description
HMT	Generate population that follows the model in Hansen et al. (1983)
hospital	Population of 393 short-stay hospitals with fewer than 1000 beds
labor	Clustered population of 478 persons extracted from Sept. 1976 Current Population Survey
MDarea.pop	Artificial population of 403,997 persons arrayed in census tracts and block groups
nhis, nhispart	Datasets of persons with demographic and socioeconomic variables
nhis.large	21,588 persons with 18 demographic and health-related variables
smho.N874	874 mental health organizations with 6 financial variables
smho98	875 mental health organizations with 8 financial and patient-count variables

Table 2: Finite populations in **PracTools**.

Examples

This section gives some examples for computing sample sizes for estimating proportions and differences of proportions, an allocation to strata, and the optimal numbers of PSUs, secondary units, and elements in a three-stage sample.

Proportions and differences in proportions

The function `nProp` will return the sample size required for estimating a proportion with a specified CV or variance. For a CV target, Equation (1) is used; the formula for a variance target is similar. The function takes the following parameters:

<code>CV0</code>	target value of coefficient of variation of the estimated proportion
<code>V0</code>	target value of variance of the estimated proportion
<code>pU</code>	population proportion
<code>N</code>	number of units in finite population; default is <code>Inf</code>

A single numeric value, the sample size, is returned. An advance guess is needed for the value of the population proportion, p_U . By default, the population is assumed to be very large ($N = \infty$), but specifying a finite value of N results in a finite population correction being used in calculating the sample size. To estimate a proportion anticipated to be $p_U = 0.1$ with a CV_0 of 0.05, the function call and resulting output is:

```
> nProp(CV0 = 0.05, N = Inf, pU = 0.1)
[1] 3600
```

If the population has only 500 elements, then the necessary sample size is much smaller:

```
> nProp(CV0 = 0.05, N = 500, pU = 0.1)
[1] 439.1315
```

In this function and others in the package, sample sizes are not rounded in case the exact value is of interest to a user. To obtain sample sizes for two overlapping groups, `nDep2sam` is the appropriate function, which uses Equation (4). The function takes these inputs:

<code>S2x</code>	unit variance of analysis variable x in sample 1
<code>S2y</code>	unit variance of analysis variable y in sample 2
<code>g</code>	proportion of sample 1 that is in the overlap with sample 2
<code>r</code>	ratio of the size of sample 1 to that of sample 2
<code>rho</code>	unit-level correlation between x and y
<code>alt</code>	should the test be 1-sided or 2-sided; allowed values are "one.sided" or "two.sided"
<code>del</code>	size of the difference between the means to be detected
<code>sig.level</code>	significance level of the hypothesis test
<code>pow</code>	desired power of the test

Among other things, the user must specify the unit (or population) standard deviations in the two populations from which the samples are selected, the proportion of the first sample that is in the second (i.e., a measure of overlap), and the unit-level correlation between the variables being measured in the two samples. If there is no overlap in the samples, it would be natural to set `rho = 0`. The size of the difference in the means that is to be detected and the power of the test on the difference in means must also be declared. The code below computes the sample size needed to detect a difference of 5 in the means with a power of 0.8 when the unit variances in both groups are 200, 75 percent of the first sample is in the second, the samples from the two groups are to be the same size, and the unit-level correlation is 0.9. This function and several others in the package use the class '`power.htest`' as a convenient way of returning the output.

```
> nDep2sam(S2x = 200, S2y = 200, g = 0.75, r = 1, rho = 0.9,
+           alt = "one.sided", del = 5, sig.level = 0.05, pow = 0.80)
```

Two-sample comparison of means
Sample size calculation for overlapping samples

```
n1 = 33
n2 = 33
S2x.S2y = 200, 200
delta = 5
gamma = 0.75
r = 1
rho = 0.9
alt = one.sided
sig.level = 0.05
power = 0.8
```

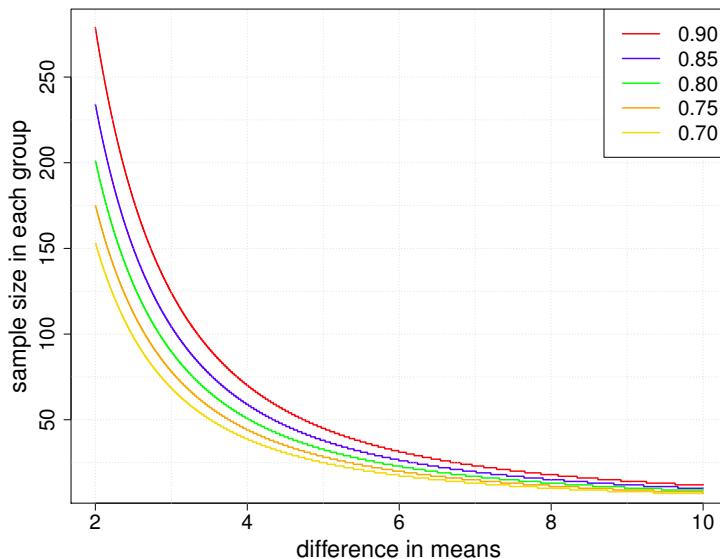


Figure 1: Sample size in each group for detecting a range of differences in means with several levels of power.

`nDep2sam` will also accept a vector of differences as input, e.g., `del <- seq(2, 10, 0.001)`. This makes it easy to generate a plot like in Figure 1, where the sample size in each group is plotted against the difference in means for several levels of power. This is a useful way to present options to users.

Probability proportional to size sampling

The function `gammaFit` will estimate the variance parameter in the model $E_M(y_i) = \mathbf{x}_i^T \beta$, $Var_M(y_i) = \sigma^2 x_i^\gamma$. The code below uses the hospital population bundled with **PracTools** to estimate the variance power in the model, $E_M(y) = \beta_1 \sqrt{x} + \beta_2 x$, $V_M(y) = \sigma^2 x^\gamma$, returning $\hat{\gamma} = 1.612009$. Using the function `UPrandomsystematic` from the package **sampling**, a sample of size 30 is then selected with probability proportional to $\sqrt{x^\gamma}$, which is the optimal measure of size for estimating the population total of y .

```
> data("hospital")
> x <- hospital$x
> y <- hospital$y
> X <- cbind(sqrt(x), x)
> (res <- gammaFit(X = X, x = x, y = y, maxiter = 100, tol = 0.001))
Convergence attained in 47 steps.
g.hat = 1.612009
$g.hat
      X
1.612009
> require(sampling)
> n <- 30
> pik <- n * sqrt(x^res$g.hat) / sum(sqrt(x^res$g.hat))
> sam <- UPrandomsystematic(pik)
> hosp.sam <- hospital[sam == 1, ]
```

To determine a sample size for *ppswr* sampling, the function `nCont` can be used. This function is designed to evaluate Equation (1) for simple random samples. However, Equation (7) has the same form if we equate $V_1/(N^2 \bar{y}_U^2)$ in Equation (7) to S^2/\bar{y}_U^2 in Equation (1) and set $N = \infty$. If $V_1/(N^2 \bar{y}_U^2) = 2$ and the CV target is 0.05, the call to `nCont` is

```
> nCont(CV0 = 0.05, N = Inf, CVpop = sqrt(2))
[1] 800
```

Allocations to strata

A sample can be allocated to strata using `strAlloc`. The function takes a number of parameters which are described in the help file. A standard problem in applied sampling is to find an allocation that will

minimize the variance of an estimated mean subject to a fixed total budget. To solve this problem, the stratum population sizes, standard deviations, stratum per-unit costs, total budget, and the type of allocation (`alloc = "totcost"`) are specified; partial output is:

```
> Nh <- c(215, 65, 252, 50, 149, 144)
> Sh <- c(267, 106, 69, 110, 98, 445)
> ch <- c(1400, 200, 300, 600, 450, 1000)
> strAlloc(Nh = Nh, Sh = Sh, cost = 100000, ch = ch, alloc = "totcost")

allocation = totcost
Nh = 215, 65, 252, 50, 149, 144
Sh = 267, 106, 69, 110, 98, 445
nh = 30.578027, 9.710196, 20.008418, 4.475183, 13.719233, 40.387433
nh/n = 0.25722085, 0.08168169, 0.16830983, 0.03764502, 0.11540551, 0.33973710
```

The function returns a list with the type of allocation, population stratum counts and standard deviations, sample sizes for each stratum, the proportions of the sample allocated to each stratum, and the anticipated standard error of the mean. Other options for the allocation types are proportional to stratum population sizes ("prop"), Neyman ("neyman"), and minimization of the total cost subject to a specified variance or CV target ("totvar"). The `nh` component of the list can then be used in, e.g., the `strata` function in **sampling**, to select the sample. Either the `round` or `ceiling` function could be applied to `nh` to create integer sample sizes. (If non-integers are supplied to `strata`, they will be truncated to the integer floor.)

Allocations in two- and three-stage samples

When designing multistage samples, decisions must be made about how many units to select at each stage. To illustrate this, we consider a three-stage sample. A considerable amount of data is needed to estimate realistic ingredients required for `clusOpt3`. The function, `BW3stagePPSe`, will estimate B^2 , W^2 , W_2^2 , W_3^2 , δ_1 , and δ_2 from a three-stage where the first-stage is selected `ppswr` and the last two stages are selected by `srswor`. `BW2stagePPSe` does similar calculations for two-stage sampling. Variance component estimation is, of course, a difficult area where a number of alternatives have been developed in the model-based literature. The forms used in `BW2stagePPSe` and `BW3stagePPSe` are fairly simple ANOVA-type estimates. These estimates have known defects, like occasionally being negative.

The following example computes a three-stage allocation that minimizes the variance of the *pwr*-estimator assuming that the budget for variable cost is 100,000; the PSU, SSU, and per-element costs are 500, 100, and 120, respectively; $\delta_1 = 0.01$, $\delta_2 = 0.10$; the unit relvariance is $\tilde{V} = 1$; and the ratios, k_1 and k_2 , are both 1. `cal.sw = 1` specifies that the optima be found for a fixed total budget. The full description of the input parameters can be found in the help file for `clusOpt3`.

```
> clusOpt3(unit.cost = c(500, 100, 120), delta1 = 0.01, delta2 = 0.10, unit.rv = 1,
+           k1 = 1, k2 = 1, tot.cost = 100000, cal.sw = 1)

C1 = 500
C2 = 100
C3 = 120
delta1 = 0.01
delta2 = 0.1
unit relvar = 1
k1 = 1
k2 = 1
cost = 1e+05
m.opt = 28.3
n.opt = 7.1
q.opt = 2.7
CV = 0.0499
```

Along with the inputs, the output includes the optimal sample size for each stage and the CV that is anticipated for the *pwr*-estimator given that design. The sample sizes above can be rounded and then used in the **sampling** package to select units at each of the stages. For example, suppose that 28 PSUs and 7 SSUs will be selected with probabilities proportional to a *mos*. Within each sample SSU, a sample of 3 elements will be selected via `srswor`. The PSUs can be selected using the function `cluster` and the `data` extracted. Then, a cluster sample of 7 SSUs can be selected from each of those 28 units in a loop, again using `cluster` and the `data` for those sample SSUs extracted. The sample of SSUs would then be treated as strata and the `strata` function used to select 3 elements from each SSU using `srswor`.

Double sampling for stratification

The function `dub` will compute the allocation to strata for a double sampling design in which phase-1 is used to assign units to strata. The function takes these input parameters:

<code>c1</code>	cost per unit in phase-1
<code>c2</code>	cost per unit in phase-2
<code>Ctot</code>	total variable cost
<code>Nh</code>	vector of stratum population counts or proportions
<code>Sh</code>	vector of stratum population standard deviations
<code>Yh.bar</code>	vector of stratum population means

The inputs, N_h , S_h , and \bar{Y}_h , will typically have to be estimated from the first-phase sample. The example below computes the allocation to four strata assuming a total cost of 20,000 and unit costs of $c_{(1)} = 10$ and $c_{(2)} = 50$. A proportion is being estimated.

```
> Wh <- rep(0.25, 4)
> Ph <- c(0.02, 0.12, 0.37, 0.54)
> Sh <- sqrt(Ph * (1 - Ph))
> c1 <- 10; c2 <- 50; Ctot <- 20000
> dub(c1, c2, Ctot, Nh = Wh, Sh, Yh.bar = Ph)

V1 = 0.04191875
V2 = 0.1307118
n1 = 404.1584
n2 = 319.1683
n2/n1 = 0.789711
ney.alloc = 30.89801, 71.71903, 106.55494, 109.99634
Vopt = 0.0005132573
nsrs = 400
Vsrs = 0.0004839844
Vratio = 1.06
```

The function also computes the size of an *srs*, *nsrs*, that would cost the same total amount, assuming that the per-unit cost is $c_{(2)}$; the anticipated variances with the optimal two-phase allocation and the *srs* of size *nsrs*; and the ratio of the two variances. Often, the two-phase design has very little gain, and sometimes a loss as in this example, compared to simple random sampling. However, double sampling for stratification is usually undertaken to control the sample sizes in the strata whose members are not known in advance.

Sample size for a nonlinear estimator

To illustrate a calculation for a nonlinear estimator, consider the proportion of Hispanics with insurance coverage in the `MDarea.pop`, which is part of the package. Define y_{2k} to be 1 if a person is Hispanic and 0 if not; $\alpha_{1k} = 1$ if a person has insurance coverage. Then, $y_{1k} = \alpha_{1k}y_{2k}$ is 1 if person k has insurance and is Hispanic and is zero otherwise. The linear substitute is $z_k = y_{1k} - \theta y_{2k}$ where θ is the proportion of Hispanics with insurance coverage. In this case, z_k can take only three values: $-\theta$, 0, and $1 - \theta$. If a simple random sample of clusters and persons within clusters is selected, `BW2stageSRS` can be used to compute B^2 , W^2 , and δ using the linear substitutes as inputs. Assuming that the full population is available, the R code is the following. We do the calculation for clusters defined as tracts (a small geographic area with about 4000 persons defined for census-taking).

```
> # recode Hispanic to be 1 = Hispanic, 0 if not
> y2 <- abs(MDarea.pop$Hispanic - 2)
> y1 <- y2 * MDarea.pop$ins.cov
> # proportion of Hispanics with insurance
> p <- sum(y1) / sum(y2)
> # linear sub
> z <- y1 - p * y2
> BW2stageSRS(z, psuID = MDarea.pop$TRACT)
```

The result is $\delta = 0.00088$. Thus, the effect of clustering on this estimated proportion is inconsequential—a two-stage sample will estimate the proportion almost as precisely as an *srs* would. In contrast, if the estimate is the total number of Hispanics with insurance, then we call `BW2stageSRS` this way:

```
> BW2stageSRS(y1, psuID = MDarea.pop$TRACT)
```

which returns $\delta = 0.02251$.

Summary

Finite population sampling is one of the more important areas in statistics since many key economic and social measures are derived from surveys. R through its packages is gradually accumulating capabilities for selecting and analyzing samples from finite populations. Pieces that have been missing are sample size computations for the kinds of complex designs that are used in practice. **PracTools** contributes to filling that gap by providing a suite of sample size calculation routines for one-, two-, and three-stage samples. We also include features for stratified allocations, for probability proportional to size sampling, and for incorporating costs into the computations. Several realistic example populations, that should be useful for classroom instruction, are also part of the package.

The package is limited in the sense that it covers only some of the sample selection schemes that we have found are most useful and prevalent in the practice of survey sampling. There are many other selection algorithms that have their own, specialized variance formulas. Tillé (2006) covers many of these. **PracTools** also does not select samples, but there are a number of other R packages, mentioned in this paper that do. One of the great advantages of R is that users can readily access different packages for specialized tasks like sample size calculation and sample selection.

Bibliography

- A. Alfons, M. Templ, and P. Filzmoser. An object-oriented framework for statistical simulation: The R package simFrame. *Journal of Statistical Software*, 37(3):1–36, 2010. URL <http://www.jstatsoft.org/v37/i03/>. [p163]
- S. Baillargeon and L.-P. Rivest. *stratification: Univariate Stratification of Survey Populations*, 2014. URL <https://CRAN.R-project.org/package=stratification>. R package version 2.2-5. [p165]
- G. Barcaroli. SamplingStrata: An R package for the optimization of stratified sampling. *Journal of Statistical Software*, 61(4):1–24, 2014. URL <http://www.jstatsoft.org/v61/i04/>. [p166]
- Bureau of Labor Statistics. BLS handbook of methods, 2013. URL <http://www.bls.gov/opub/hom/>. [p163, 166]
- Center for Disease Control and Prevention. National health and nutrition examination survey, 2013a. URL <http://www.cdc.gov/nchs/nhanes.htm>. [p163]
- Center for Disease Control and Prevention. National health interview survey, 2013b. URL <http://www.cdc.gov/nchs/nhis.htm>. [p163]
- W. Cochran. *Sampling Techniques*. John Wiley & Sons, Inc., New York, 1977. [p163, 168]
- A. Deaton. *Analysis of Household Surveys*. Johns Hopkins University Press, Baltimore, 1997. [p169]
- C. Francisco and W. A. Fuller. Quantile estimation with a complex survey design. *The Annals of Statistics*, 19:454–469, 1991. [p169]
- D. Fylstra, L. Lasdon, J. Watson, and A. Waren. Design and use of the Microsoft Excel solver. *INFORMS Interfaces*, 28:29–55, 1998. [p166]
- S. Gabler, M. Ganninger, and R. Münnich. Optimal allocation of the sample size to strata under box constraints. *Metrika*, 75:151–161, 2012. [p166]
- J. G. Gambino. *pps: Functions for PPS Sampling*, 2012. URL <https://CRAN.R-project.org/package=pps>. R package version 0.94. [p163]
- A. Ghalanos and S. Theussl. *Rsolnp: General Non-linear Optimization Using Augmented Lagrange Multiplier Method*, 2014. URL <https://CRAN.R-project.org/package=Rsolnp>. R package version 1.15. [p166]
- V. P. Godambe and V. M. Joshi. Admissibility and Bayes estimation in sampling finite populations – I. *The Annals of Mathematical Statistics*, 36:1707–1723, 1965. [p165]
- M. H. Hansen, W. H. Hurwitz, and W. G. Madow. *Sample Survey Methods and Theory*, volume I. John Wiley & Sons, Inc., New York, 1953. [p167]
- M. H. Hansen, W. G. Madow, and B. J. Tepping. An evaluation of model-dependent and probability sampling inferences in sample surveys. *Journal of the American Statistical Association*, 78:776–793, 1983. [p170]

- V. G. Iannacchione, J. A. Dever, C. M. Bann, K. A. Considine, D. Creel, C. P. Carson, H. L. Best, and R. W. Haley. Validation of a research case definition of Gulf War illness in the 1991 U.S. military population. *Neuroepidemiology*, 37(2):129–140, 2011. [p168]
- C. T. Isaki and W. A. Fuller. Survey design under the regression superpopulation model. *Journal of the American Statistical Association*, 77(377):89–96, 1982. [p165]
- S. L. Lohr. *Sampling: Design and Analysis*. Duxbury Press, Pacific Grove CA, 1999. [p163]
- T. Lumley. *Complex Surveys*. John Wiley & Sons, Inc., New York, 2010. [p163]
- T. Lumley. *survey: Analysis of Complex Survey Samples*, 2014. URL <https://CRAN.R-project.org/package=survey>. R package version 3.30-3. [p163]
- J. Manitz. *samplingbook: Survey Sampling Procedures*, 2013. URL <https://CRAN.R-project.org/package=samplingbook>. R package version 1.2.0, with contributions by M. Hempelmann, G. Kauermann, H. Kuechenhoff, S. Shao, C. Oberhauser, N. Westerheide, M. Wiesenfarth. [p163]
- J. Neyman. Contribution to the theory of sampling human populations. *Journal of the American Statistical Association*, 33(201):101–116, 1938. [p168]
- C. Särndal, B. Swensson, and J. Wretman. *Model Assisted Survey Sampling*. Springer-Verlag, New York, 1992. [p163, 165]
- S. Theussl and H. Borchers. CRAN task view: Optimization and mathematical programming, 2015. URL <https://CRAN.R-project.org/view=Optimization>. [p166]
- Y. Tillé. *Sampling Algorithms*. Springer-Verlag, New York, 2006. [p175]
- Y. Tillé and A. Matei. *sampling: Survey Sampling*, 2013. URL <https://CRAN.R-project.org/package=sampling>. R package version 2.6. [p163]
- R. Valliant, J. Dever, and F. Kreuter. *Practical Tools for Designing and Weighting Survey Samples*. Springer-Verlag, New York, 2013. [p163, 166, 167]
- R. Valliant, J. Dever, and F. Kreuter. *PracTools: Tools for Designing and Weighting Survey Samples*, 2015. URL <https://CRAN.R-project.org/package=PracTools>. R package version 0.2. [p163]
- R. Varadhan. *alabama: Constrained Nonlinear Optimization*, 2015. URL <https://CRAN.R-project.org/package=alabama>. R package version 2015.3-1, with contributions from G. Grothendieck. [p166]
- E. B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22:209–212, 1927. [p164]
- K. M. Wolter. *Introduction to Variance Estimation*. Springer-Verlag, New York, 2nd edition, 2007. [p169]
- M. Woodward. Formulas for sample size, power, and minimum detectable relative risk in medical studies. *The Statistician*, 41:185–196, 1992. [p164]

Richard Valliant

*Joint Program in Survey Methodology
Universities of Michigan and Maryland
1218 Lefrak Hall, College Park MD 20742 USA*
rvallian@umd.edu

Jill A. Dever

*RTI International
701 13th Street NW, Suite 750, Washington, DC 20005-3967 USA*
jdever@rti.org

Frauke Kreuter

*Joint Program in Survey Methodology, University of Maryland
1218 Lefrak Hall, College Park MD 20742 USA*
fkreuter@umd.edu

ALTopt: An R Package for Optimal Experimental Design of Accelerated Life Testing

by Kangwon Seo and Rong Pan

Abstract The R package **ALTopt** has been developed with the aim of creating and evaluating optimal experimental designs of censored accelerated life tests (ALTs). This package takes the generalized linear model approach to ALT planning, because this approach can easily handle censoring plans and derive information matrices for evaluating designs. Three types of optimality criteria are considered: *D*-optimality for model parameter estimation, *U*-optimality for reliability prediction at a single use condition, and *I*-optimality for reliability prediction over a region of use conditions. The Weibull distribution is assumed for failure time data and more than one stress factor can be specified in the package. Several graphical evaluation tools are also provided for the comparison of different ALT test plans.

Introduction

Accelerated life testing (ALT) is commonly used for obtaining a product's failure time data by subjecting it to elevated stress conditions, such as temperature, humidity, and voltage. As a result, the product fails in a shorter time period than would be expected under normal stress conditions. The failure data obtained from ALTs can then be extrapolated to the normal use stress level to estimate the product's lifetime distribution. Nelson (2005a,b) provides a comprehensive review of ALT papers up to 2005.

To avoid poor experimental results and to obtain more accurate inference on the acceleration model and on reliability prediction, it is necessary to have an effective ALT test plan. A well-designed ALT test plan often aims to achieve some statistical optimality. However, conventional experimental designs (e.g., factorial designs) are not effective as ALT test plans because of the following features of ALTs:

- Extrapolation – Test stress levels are typically higher than the normal use stress levels. As failure time data will be collected at these higher stress levels, extrapolating them to the normal use stress level is needed for reliability prediction. Nonlinear relationships between failure time and stress levels are expected.
- Non-normal distributions of failure times – The failure time distribution is typically positively skewed, e.g., the Weibull distribution.
- Censoring of failure time data – Censoring occurs when the exact failure times of test units are not observed. There are several reasons for censoring. In some cases, test units do not fail by the end of the test period, in which case the data becomes right-censored. In other cases, test units are periodically inspected, so the only information available is the time interval of failure, while the exact failure time is unknown. The latter case is called interval-censoring.

In this article, we introduce an R package, **ALTopt** (Seo and Pan, 2015), that constructs optimal test plans for ALTs with right- and interval-censored data. This package is based on the work done by Monroe et al. (2011) and Yang and Pan (2013), where generalized linear models (GLMs) were used to model censored ALT data.

Optimal designs of ALT

Optimality criteria

The **ALTopt** package accommodates three optimality criteria: *D*-optimality, *U*-optimality and *I*-optimality. A *D*-optimal design minimizes the generalized variance of parameter estimates, while a *U*-optimal or *I*-optimal design minimizes, respectively, the (average) variance of response prediction at a single use condition or over a region of use conditions.

In the GLM context, the *D*-optimal design is defined by

$$\xi^* := \arg \max_{\xi} |\mathbf{X}(\xi)' \mathbf{W} \mathbf{X}(\xi)|.$$

Let n be the number of test units and p be the number of model parameters. Matrix $\mathbf{X}(\xi)$ is the $n \times p$ model matrix constructed by expanding a design matrix to include all regression terms in the chosen model form, and matrix \mathbf{W} is the $n \times n$ diagonal matrix of weights that depends on the GLM formulation used.

Using the same notation, the U -optimal and I -optimal designs can be defined, respectively, as

$$\xi^* := \arg \min_{\xi} \mathbf{x}'_{use} \cdot (\mathbf{X}(\xi)' \mathbf{W} \mathbf{X}(\xi))^{-1} \cdot \mathbf{x}_{use},$$

and

$$\xi^* := \arg \min_{\xi} \frac{\int_{\Omega} \mathbf{x}'_{use} \cdot (\mathbf{X}(\xi)' \mathbf{W} \mathbf{X}(\xi))^{-1} \cdot \mathbf{x}_{use} d\mathbf{x}_{use}}{S_{\Omega}},$$

where \mathbf{x}_{use} is the single use condition, Ω is the region of use conditions, and S_{Ω} is the area of use region. For GLMs, the weights \mathbf{W} are functions of the regression coefficients in the linear predictor. Therefore, the information matrix contains unknown model parameter values, implying that the choice of these unknown values also affects the optimal design (see [Johnson and Montgomery \(2009\)](#) for more details). In this article, we assume that these parameter values are pre-specified. They are referred to as the planning values by [Meeker and Escobar \(1998\)](#).

GLMs for ALT

A function that links failure time and stress variables is needed in order to extrapolate the results obtained in the test region to the use region. The GLM formulation for ALT is built upon the Cox's proportional hazard (PH) assumption. This section provides the derivation of these formulations for right-censored and interval-censored ALT data.

The Cox's proportional hazard model

The PH model assumes that, given the vector of explanatory variables \mathbf{x} , the hazard function of failure time is given by

$$h(t, \mathbf{x}; \beta) = h_0(t) e^{\mathbf{x}' \beta}, \quad (1)$$

where $h_0(t)$ is called the baseline hazard function and β is a vector of regression coefficients.

Note that the baseline hazard function is a function of time only. From Eq. (1) we can derive that

$$H(t, \mathbf{x}) = H_0(t) e^{\mathbf{x}' \beta}, \quad (2)$$

where $H(t, \mathbf{x})$ is the cumulative hazard function and $H_0(t)$ is the baseline cumulative hazard function. It is also easy to show that a reliability function is given by

$$R(t, \mathbf{x}) = (R_0(t))^{e^{\mathbf{x}' \beta}}, \quad (3)$$

where $R(t, \mathbf{x})$ is the reliability function and $R_0(t) = \exp(-H_0(t))$ is the baseline reliability function.

The baseline hazard function of a Weibull distribution is given by $h_0(t) = \lambda_0 \alpha t^{\alpha-1}$, where λ_0 is called the intrinsic failure rate and α is the shape parameter of Weibull distribution. By Eq. (1), the hazard function of Weibull distribution can be expressed as $h(t, \mathbf{x}; \beta) = \lambda_0 \alpha t^{\alpha-1} e^{\mathbf{x}' \beta}$ and, by Eq. (2), its cumulative hazard function is as $H(t, \mathbf{x}) = \lambda_0 t^{\alpha} e^{\mathbf{x}' \beta}$.

GLM for right-censored failure time data

With the proportional hazard assumption, the failure density function is given by

$$f(t) = h(t) R(t) = h_0(t) e^{\mathbf{x}' \beta} (R_0(t))^{e^{\mathbf{x}' \beta}}.$$

For a failure time data set that includes right-censored survival times, each observation can be expressed as a pair (t_i, c_i) , $i = 1, 2, \dots, n$, where t_i is either a failure time or censoring time and c_i is an indicator variable, which is 1 if the i^{th} unit failed and 0 if it has not failed. Thus, the likelihood function is given by

$$L = \prod_{i=1}^n (f(t_i))^{c_i} (R(t_i))^{1-c_i} = \prod_{i=1}^n (h(t_i))^{c_i} R(t_i).$$

From Eqs. (1) and (3) the log-likelihood function can be written as

$$\ln L = \sum_{i=1}^n [c_i \ln h(t_i) + \ln R(t_i)] = \sum_{i=1}^n \left[c_i (\ln h_0(t_i) + \mathbf{x}_i' \boldsymbol{\beta}) + e^{\mathbf{x}_i' \boldsymbol{\beta}} \ln R_0(t_i) \right].$$

Let $\mu_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})(-\ln R_0(t_i))$, we have

$$\ln L = \sum_{i=1}^n [c_i \ln h_0(t_i) - c_i \ln(-\ln R_0(t_i)) + c_i \ln \mu_i - \mu_i]. \quad (4)$$

Note that the last two terms of sum on the right-hand side of Eq. (4) are the same as the kernel of the log-likelihood function of n independent Poisson distributed random variables with mean μ_i . The first two terms do not depend on the parameter $\boldsymbol{\beta}$. Therefore, the maximum likelihood estimator $\hat{\boldsymbol{\beta}}$ of (4) is similar to the estimator that maximizes the log-likelihood function of Poisson distributions. If the indicator variable c_i is treated as from a Poisson distribution with mean μ_i , then the GLM formulation becomes

- The response variables, c_i 's, are independently sampled from $Poisson(\mu_i)$;
- The linear predictor is $\eta_i = \mathbf{x}_i' \boldsymbol{\beta}$;
- The link function is given by $\ln \mu_i = \eta_i + \text{an offset term}$.

This offset term in the link function is $\ln H_0(t_i)$, the log transformation of the baseline cumulative hazard function. This GLM formulation is applicable for any failure time distribution with right-censored data, as long as the PH assumption holds.

Since the log link function is the canonical link function for the Poisson distribution, the asymptotic variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ is given by

$$Var(\hat{\boldsymbol{\beta}}) = (\mathbf{X}(\xi)' \mathbf{W} \mathbf{X}(\xi))^{-1},$$

where $\mathbf{W} = diag\{\sigma_i^2\}$ and σ_i^2 is the variance of Poisson distribution; i.e., $\sigma_i^2 = \mu_i = e^{\mathbf{x}_i' \boldsymbol{\beta}} H_0(t_i)$. We replace t_i with its expectation, which is given by

$$E[t_i] = P(t < t_c) \cdot E[t_i | t < t_c] + P(t \geq t_c) \cdot E[t_i | t \geq t_c].$$

Monroe et al. (2011) has shown that

$$\mu_i = \left[1 - e^{-H(t_c, \mathbf{x}_i)} \right] = \Phi(t_c, \mathbf{x}_i),$$

where Φ is the failure time distribution and t_c is the censoring time.

For a Weibull distribution with the known shape parameter α , it follows that

$$\mu_i = \left[1 - \exp(-\lambda_0 t_c^\alpha e^{\mathbf{x}_i' \boldsymbol{\beta}}) \right] = \left[1 - \exp(-e^{\beta_0 + \mathbf{x}_i' \boldsymbol{\beta}} t_c^\alpha) \right],$$

where $\beta_0 = \ln \lambda_0$ which plays a role of the intercept term in the linear predictor.

GLM for interval-censored failure time data

For interval-censored data, the whole testing period is divided into multiple time intervals such as $[0, t_1), [t_1, t_2), \dots, [t_{k-1}, t_k), [t_k, \infty)$, and failures are expected to occur within one of these intervals. Define the failure probability of the i^{th} test unit within the j^{th} interval to be

$$p_{ij} = P(t_{j-1} \leq T_i < t_j),$$

and the conditional probability of surviving at the beginning of the j^{th} interval but failing within the j^{th} interval as

$$\pi_{ij} = P(t_{j-1} \leq T_i < t_j | T_i \geq t_{j-1}), \quad j = 1, 2, \dots, k+1.$$

It can be shown that $p_{i1} = \pi_{i1}$ and $p_{ij} = (1 - \pi_{i1})(1 - \pi_{i2}) \cdots (1 - \pi_{i,j-1}) \pi_{ij}$ for $j = 2, 3, \dots, k+1$.

Define an indicator variable, r , such that $r_{ij} = 0$ if the i^{th} test unit does not fail within the j^{th} interval and $r_{ij} = 1$ if the i^{th} test unit does fail. Suppose that there are n items, then the number of observations is $n \times (k+1)$. For example,

$$(0, t_1, r_{i1}), (t_1, t_2, r_{i2}), \dots, (t_k, \infty, r_{ik+1}), \quad i = 1, 2, \dots, n$$

The likelihood function can be expressed as

$$L = \prod_{i=1}^n \prod_{j=1}^{k+1} p_{ij}^{r_{ij}},$$

which is equivalent to

$$L = \prod_{i=1}^n \prod_{j=1}^{k+1} \pi_{ij}^{r_{ij}} (1 - \pi_{ij})^{s_{ij}}, \quad (5)$$

where $s_{ij} = r_{i,j+1} + r_{i,j+2} + \dots + r_{i,k+1}$. Therefore, $s_{ij} = 1$ if the failure of the i^{th} test unit occurs at a time after the j^{th} interval and $s_{ij} = 0$ if the failure of the i^{th} test unit occurs within or before the j^{th} interval.

The likelihood function of Eq. (5) has the same form as the likelihood function of independent binomial random variables. We treat r_{ij} as a binomial random variable with probability π_{ij} and sample size $m_{ij} = r_{ij} + s_{ij}$. The data set can be presented as a series of quadruplets:

$$(0, t_1, r_{i1}, m_{i1}), (t_1, t_2, r_{i2}, m_{i1}), \dots, (t_k, \infty, r_{ik+1}, m_{ik+1}), i = 1, 2, \dots, n.$$

Now, we examine the probability π_{ij} . Notice that

$$1 - \pi_{ij} = P(T_i \geq t_j | T_i \geq t_{j-1}) = \frac{R(t_j)}{R(t_{j-1})}. \quad (6)$$

By Eq. (3) it becomes

$$1 - \pi_{ij} = \left[\frac{R_0(t_j)}{R_0(t_{j-1})} \right]^{x_i' \beta}.$$

Applying the natural logarithm function twice yields

$$\ln \left[-\ln(1 - \pi_{ij}) \right] = x_i' \beta + \ln \left[\ln R_0(t_{j-1}) / R_0(t_j) \right]. \quad (7)$$

The second term of the right hand side of Eq. (7) does not depend on the regression coefficient β , thus Eq. (7) is a complementary log-log link function with an offset term. We can treat r_{ij} 's as independent random variables that follow a binomial distribution with the probability parameter π_{ij} and sample size m_{ij} , and the GLM formulation is written as

- The response variables, r_{ij} 's, are distributed as independent $\text{Binomial}(m_{ij}, \pi_{ij})$;
- The linear predictor is $\eta_i = x_i' \beta$;
- The link function is given by $\ln \left[-\ln(1 - \pi_{ij}) \right] = \eta_i + \text{an offset term}$.

Since the log-log link is not a canonical link for the binomial distribution, we need to introduce $\Delta = \text{diag} \{ d\theta_i / d\eta_i \}$ in the weight matrix where θ_i is the natural location parameter of the binomial distribution; i.e.,

$$\begin{aligned} \Delta &= \text{diag} \left\{ \frac{d\theta_i}{d\eta_i} \right\} \\ &= \text{diag} \left\{ \frac{d \left(\ln \frac{\pi_{ij}}{1 - \pi_{ij}} \right)}{d \left(\ln(-\ln(1 - \pi_{ij})) \right)} \right\} \\ &= \text{diag} \left\{ -\frac{\ln(1 - \pi_{ij})}{\pi_{ij}} \right\}. \end{aligned}$$

Then, the asymptotic variance-covariance matrix of $\hat{\beta}$ is given by

$$\begin{aligned} \text{Var}(\hat{\beta}) &= (\mathbf{X}^*(\xi)' \mathbf{W} \mathbf{X}^*(\xi))^{-1} \\ &= (\mathbf{X}^*(\xi)' \Delta \mathbf{V} \Delta \mathbf{X}^*(\xi))^{-1}, \end{aligned}$$

where $\mathbf{X}^*(\xi) = \mathbf{X}(\xi) \otimes \mathbf{1}_{k+1}$ and $\mathbf{V} = \text{diag}\{\sigma_{ij}^2\}$. Note that, instead of using $\mathbf{X}(\xi)$, the original model matrix, $\mathbf{X}^*(\xi)$, which is a matrix of size $n(k+1) \times p$, is used. Each row of $\mathbf{X}(\xi)$ is repeated $(k+1)$ times in $\mathbf{X}^*(\xi)$ because each test unit has $(k+1)$ intervals.

In a binomial distribution, $\sigma_{ij}^2 = m_{ij}\pi_{ij}(1 - \pi_{ij})$, which includes the random variable m_{ij} . Replacing m_{ij} with its expectation, the weight matrix becomes

$$\begin{aligned} \mathbf{W} &= \Delta \mathbf{V} \Delta \\ &= \text{diag} \left\{ \left(-\frac{\ln(1 - \pi_{ij})}{\pi_{ij}} \right) E(m_{ij}) \pi_{ij} (1 - \pi_{ij}) \left(-\frac{\ln(1 - \pi_{ij})}{\pi_{ij}} \right) \right\} \\ &= \text{diag} \left\{ \frac{\{\ln(1 - \pi_{ij})\}^2 (1 - \pi_{ij})}{\pi_{ij}} E(m_{ij}) \right\}. \end{aligned} \quad (8)$$

Assuming a Weibull distribution for a product's lifetime, we have $R(t, \mathbf{x}) = \exp(-H(t, \mathbf{x})) = \exp(-\lambda_0 t^\alpha e^{\mathbf{x}' \beta}) = e^{-t^\alpha e^{\beta_0 + \mathbf{x}' \beta}}$. Substituting it into (6) yields

$$1 - \pi_{ij} = \frac{e^{-t_j^\alpha e^{\beta_0 + \mathbf{x}' \beta}}}{e^{-t_{j-1}^\alpha e^{\beta_0 + \mathbf{x}' \beta}}}.$$

Assume all time intervals have the same length, Δt . Then,

$$1 - \pi_{ij} = \frac{e^{-(j\Delta t)^\alpha e^{\beta_0 + \mathbf{x}' \beta}}}{e^{(-(j-1)\Delta t)^\alpha e^{\beta_0 + \mathbf{x}' \beta}}} = e^{((j-1)^\alpha - j^\alpha) \Delta t^\alpha e^{\beta_0 + \mathbf{x}' \beta}}. \quad (9)$$

We also have

$$\begin{aligned} E(m_{ij}) &= 0 \times P(T_i < t_{j-1}) + 1 \times P(T_i \geq t_{j-1}) \\ &= P(T_i \geq t_{j-1}) \\ &= R(t_{j-1}, \mathbf{x}_i) \\ &= e^{-(j-1)\Delta t)^\alpha e^{\beta_0 + \mathbf{x}_i' \beta}}. \end{aligned} \quad (10)$$

Substituting Eq. (9) and Eq. (10) into Eq. (8) yields the weight matrix for the interval-censored Weibull failure time data:

$$\begin{aligned} \mathbf{W} &= \text{diag} \left\{ \frac{\{(j-1)^\alpha - j^\alpha\} \Delta t^\alpha e^{\beta_0 + \mathbf{x}_i' \beta} 2 e^{((j-1)^\alpha - j^\alpha) \Delta t^\alpha e^{\beta_0 + \mathbf{x}_i' \beta}}}{1 - e^{((j-1)^\alpha - j^\alpha) \Delta t^\alpha e^{\beta_0 + \mathbf{x}_i' \beta}}} e^{-(j-1)\Delta t)^\alpha e^{\beta_0 + \mathbf{x}_i' \beta}} \right\} \\ &= \text{diag} \left\{ \frac{(j-1)^\alpha - j^\alpha)^2 \Delta t^{2\alpha} e^{2(\beta_0 + \mathbf{x}_i' \beta) - j^\alpha \Delta t^\alpha e^{\beta_0 + \mathbf{x}_i' \beta}}}{1 - e^{((j-1)^\alpha - j^\alpha) \Delta t^\alpha e^{\beta_0 + \mathbf{x}_i' \beta}}} \right\}. \end{aligned}$$

Introduction to the package **ALTop**

The main purpose of the **ALTop** package is to construct D -, U -, and I -optimal ALT test plans. Two main functions, `altopt.rc` and `altopt.ic`, are developed respectively for the right-censoring and interval-censoring cases. The following assumptions are required for using this package:

- Failure time data follows the Weibull distribution where the shape parameter is specified by the user.
- Log-linear functions are used to model the relationship between a failure time distribution parameter and the stress factors.
- For interval-censored data, all the intervals have the same length.

Lastly, the package can accommodate many stress factors, but 5 or fewer is recommended for computational efficiency.

This package also provides two functions for evaluating existing test plans – `alteval.rc` and `alteval.ic`. These functions can be used for comparing test plans generated by this **ALTop** package or any other methods. Graphical displays of prediction variance are made available. These plotting features enhance the usefulness of this package for comparing and selecting test plans.

Creating optimal ALT test plans

The syntax of `altopt.rc` and `altopt.ic` functions are as follows:

```

altopt.rc(optType, N, tc, nf, alpha, formula, coef,
  useCond, useLower, useUpper, nOpt = 1, nKM = 30, nCls = NULL)

altopt.ic(optType, N, t, k, nf, alpha, formula, coef,
  useCond, useLower, useUpper, nOpt = 1, nKM = 30, nCls = NULL)

```

The arguments within these functions are:

- optType – Choice of "D", "U", and "I" optimality.
- N – The number of test units.
- tc (altopt.rc only) – The planned right censoring time.
- t (altopt.ic only) – The planned total testing time (i.e., the end point of the last interval).
- k (altopt.ic only) – The number of time intervals.
- nf – The number of stress factors.
- alpha – The value of the shape parameter of the Weibull distribution.
- formula – The object of class "formula" expressing the linear predictor model.
- coef – The numeric vector containing the coefficients of each term in formula.
- useCond – The numeric vector of use condition. It should be provided when optType is "U". The length of the vector should be same as the number of stress factors.
- useLower – The numeric vector of lower bound of use region in coded units. It should be provided when optType is "I". The length of the vector should be the same as the number of stress factors.
- useUpper – The numeric vector of upper bound of use region in coded units. It should be provided when optType is "I". The length of the vector should be the same as the number of stress factors.
- nOpt – The number of repetitions of optimization processes. The default value is 1. If nOpt is larger than 1, each optimization process starts from randomly chosen design points in the design region and each solution may slightly differ. The output shows the best solution overall.
- nKM – The number of repetitions of k-means clustering, which is used to generate the optimal design clustered by kmeans. The default value is 30.
- nCls – The number of clusters used for k-means clustering. If not specified, it is set as the number of parameters in the linear predictor model.

We use the function `stats::optim` with the "L-BFGS-B" method to perform optimization. This function allows box constraints on design variables. In our case, we have a cuboidal design region where the levels of each stress factor are coded to be between 0 and 1. More details about the "L-BFGS-B" method are available in [Byrd et al. \(1995\)](#).

The output of these functions are given as a list with the following components:

- call – The matched call.
- opt.design.rounded – The optimal design clustered by rounding.
- opt.value.rounded – The objective function value of opt.design.rounded.
- opt.design.kmeans – The optimal design clustered by k-means algorithm.
- opt.value.kmeans – the objective function value of opt.design.kmeans.

The procedure begins by generating an initial test plan with N design points, which are randomly selected from possible points in the design region. For example, if we have 100 test units and 2 stress factors the optimization process begins from 100 randomly chosen initial points, which spread out over the design region. Throughout the optimization procedure, each of these 100 points converges to its own optimal location. To create a practical test plan, it is sometimes necessary to enforce some clustering procedure to reduce the number of distinct design points. Two clustering methods are implemented in the package. When the design points are very close, the simple method of rounding (to the 3rd decimal place) the stress values is effective and straightforward. When there are still too many design points, the alternative is to use k-means clustering, which requires the specification of the number of clusters, nCls. By carefully selecting the number of clusters, it is possible to reduce the number of distinct design points without significantly affecting the value of the objective function. The final recommended test plans are provided by the elements, opt.design.rounded, or opt.design.kmeans, presented by a table containing each design point location and the number of test units allocated at each design point. The corresponding values of the objective function of these plans are also stored in opt.value.rounded and opt.value.kmeans.

The `design.plot` function displays the recommended test plan as a bubble plot on a two-dimensional design region of user-specified stress factors. The size of each bubble represents the relative size of the test unit allocations. The arguments of `design.plot` are as follows:

```
design.plot(design, xAxis, yAxis)
```

- `design` – A data frame containing the coordinates and the test unit allocation at each design point. The components, `opt.design.rounded` or `opt.design.kmeans`, of an output created by `altopt.rc` or `altopt.ic` can be given for this argument directly and any other design with the same form of those can also be given.
- `xAxis` – The name of the factor to be displayed on the x axis.
- `yAxis` – The name of the factor to be displayed on the y axis.

Evaluating ALT test plans

This package provides several methods to evaluate an ALT test plan. The first method is the numerical evaluation of a given test plan using `alteval.rc` or `alteval.ic`. These functions return the value of the objective function of the test plan. The arguments are as follows:

```
alteval.rc(designTable, optType, tc, nf, alpha, formula, coef,
           useCond, useLower, useUpper)
```

```
alteval.ic(designTable, optType, t, k, nf, alpha, formula, coef,
           useCond, useLower, useUpper)
```

The existing test plan is specified in the argument `designTable`. The other arguments of `alteval.rc` and `alteval.ic` are similar to the arguments in `altopt.rc` and `altopt.ic`.

ALTopt also provides three different graphs for evaluating a test plan – the prediction variance (PV) contour plot, the fraction of use space (FUS) plot, and the variance dispersion of use space (VDUS) plot. These graphical tools are useful when visualizing the prediction variance throughout the entire use-space region (Myers et al., 2009, chap. 8). The PV contour plot displays the contours of the estimated prediction variance from the design region to the use region of a two-dimensional user-specified stress factor space. Functions `pv.contour.rc` and `pv.contour.ic` generate the PV contour plot of an ALT test plan with right and interval censoring, respectively. The FUS plot is an extension of the fraction of design space (FDS) proposed by Zahran et al. (2003). The vertical axis of a FUS plot is the fraction of the use space region that has prediction variance less than or equal to the given values in the horizontal axis. Functions `pv.fus.rc` and `pv.fus.ic` create the FUS plot of right and interval censoring ALT plans, respectively. In addition, the FUS curves of multiple designs can be overlaid on one graph by using `compare.fus`, so these designs can be compared graphically. The VDUS plot is an extension of the variance dispersion graphs (VDGs) of Giovannitti-Jensen and Myers (1989) to the cuboidal use space region. It shows plots of minimum, average and maximum prediction variance from the center to the boundary of the use region. The comparison of multiple VDUS is also available through `compare.vdus`. The arguments of these functions are omitted here, because they are similar to previously described functions.

An example with two stress factors and right censoring

In this section, we demonstrate the use of **ALTopt** using the right-censored ALT data set from Yang and Pan (2013). In this experiment, an ALT of 100 test units is conducted with two stress factors – temperature and humidity. The lowest and highest stress levels in the test region are (60 °C, 60 %) and (110 °C, 90 %), respectively. The normal use condition is (30 °C, 25 %), while the typical use region has the range from (20 °C, 20 %) to (40 °C, 30 %). The natural stress variables of these two factors are defined by $S_1 = 11605/T$, where T is the temperature in degrees Kelvin (i.e., temp °C + 273.15), and $S_2 = \ln(h)$, where h is the relative humidity. These values are assigned to the following variables:

```
R> NuseCond <- c(11605 / (30 + 273.15), log(25))
R> NuseLow <- c(11605 / (20 + 273.15), log(20))
R> NuseHigh <- c(11605 / (40 + 273.15), log(30))
R> NdesLow <- c(11605 / (60 + 273.15), log(60))
R> NdesHigh <- c(11605 / (110 + 273.15), log(90))
```

Next, we apply a coding scheme on these natural variables so that the highest stress level becomes (0, 0) and the lowest stress level becomes (1, 1).

$$x_1 = \frac{S_1 - S_1^H}{S_1^L - S_1^H}, \quad x_2 = \frac{S_2 - S_2^H}{S_2^L - S_2^H}. \quad (11)$$

Here, x_1 and x_2 are the coded stress variables of S_1 and S_2 , respectively. The **ALTopt** package provides a utility function, `convert.stress.level`, to convert the natural stress level to the coded stress level, and vice versa. The use condition and the use stress region are accordingly coded as follows:

```
R> library(ALTopt)
R> (useCond <- as.numeric(convert.stress.level(NdesLow, NdesHigh,
+      actual = NuseCond))) # Coded use condition

[1] 1.758337 3.159172

R> (useLower <- as.numeric(convert.stress.level(NdesLow, NdesHigh,
+      actual = NuseHigh))) # Coded use region's lower bound

[1] 1.489414 2.709511

R> (useUpper <- as.numeric(convert.stress.level(NdesLow, NdesHigh,
+      actual = NuseLow))) # Coded use region's upper bound

[1] 2.045608 3.709511
```

We assume that the failure times follow an exponential distribution, i.e., $\alpha = 1$, and the pre-specified linear predictor is given by

$$\eta_i = -4.086x_1 - 1.476x_2 + 0.01x_1x_2. \quad (12)$$

Suppose the total testing time is 30 time units. The *D-optimal* test plan is generated by the following lines of code:

```
R> set.seed(10)
R> DR <- altopt.rc("D", N = 100, tc = 30, nf = 2, alpha = 1,
+      formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01))
R> DR

$call
altopt.rc(optType = "D", N = 100, tc = 30, nf = 2, alpha = 1,
      formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01))

$opt.design.rounded
  x1 x2 allocation
1 0.000 0        21
2 0.835 0        28
3 0.000 1        26
4 0.639 1        25

$opt.value.rounded
[1] 27153.91

$opt.design.kmeans
  x1 x2 allocation
1 0.8353075 0        28
2 0.0000000 0        21
3 0.6390136 1        25
4 0.0000000 1        26

$opt.value.kmeans
[1] 27153.92
```

While the formula does not include the intercept term explicitly, the value of the intercept parameter still needs to be specified (in this case, it is 0). From the final design output, we noticed that the designs generated by rounding and clustering are almost the same.

We can also generate the *U-optimal* and *I-optimal* designs using the following lines of code:

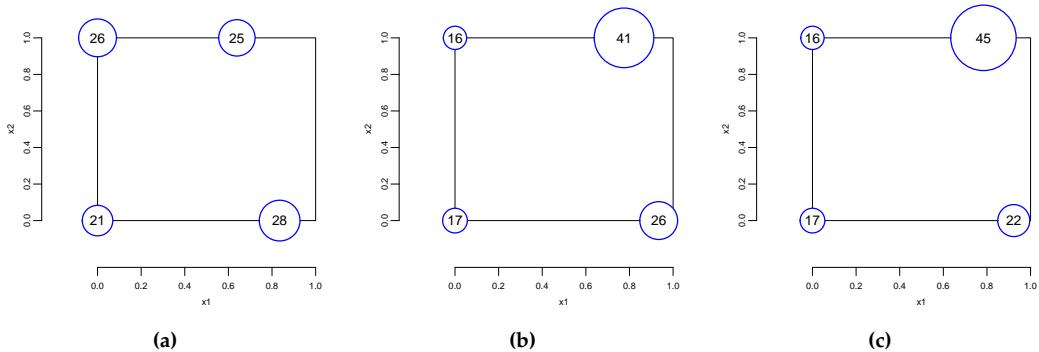


Figure 1: Design plots of (a) *D*-optimal, (b) *U*-optimal and (c) *I*-optimal designs with right censoring drawn by `design.plot` function.

```
R> set.seed(50)
R> UR <- altopt.rc("U", N = 100, tc = 30, nf = 2, alpha = 1,
+   formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01),
+   useCond = useCond)

R> set.seed(100)
R> IR <- altopt.rc("I", N = 100, tc = 30, nf = 2, alpha = 1,
+   formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01),
+   useLower = useLower, useUpper = useUpper)
```

Using `design.plot` function, we can draw the bubble plots of these test plans.

```
R> design.plot(DR$opt.design.rounded, xAxis = x1, yAxis = x2)
R> design.plot(UR$opt.design.rounded, xAxis = x1, yAxis = x2)
R> design.plot(IR$opt.design.rounded, xAxis = x1, yAxis = x2)
```

From Figure 1, one can see that the *U*- and *I*-optimal test plans resemble each other, while the *D*-optimal test plan is very different from the other two. This is expected because the objective functions of *U*- and *I*-optimal designs involve the variance of reliability prediction, while the *D*-optimal design involves the variance of parameter estimation. From the *U*- and *I*-optimal test plans, it is noticeable that a large number of test units is allocated at the lowest stress level. This type of test unit allocation scheme is common in ALTs (e.g., Meeker and Nelson, 1975).

To compare the *D*-optimal test plan and the *U*-optimal test plan, the `pv.contour.rc` function generates the contour plot of prediction variance using the following lines of code:

```
R> pv.contour.rc(DR$opt.design.rounded, xAxis = x1, yAxis = x2,
+   tc = 30, nf = 2, alpha = 1, formula = ~ x1 + x2 + x1:x2,
+   coef = c(0, -4.086, -1.476, 0.01), useCond = useCond)
R> pv.contour.rc(UR$opt.design.rounded, xAxis = x1, yAxis = x2,
+   tc = 30, nf = 2, alpha = 1, formula = ~ x1 + x2 + x1:x2,
+   coef = c(0, -4.086, -1.476, 0.01), useCond = useCond)
```

Figure 2 shows that the *U*-optimal test plan has lower prediction variance than the *D*-optimal test plan at the normal use condition. The FUS and VDUS plots can also be used for further comparison of these test plans. These plots are shown in Figure 3.

```
R> fusDR <- pv.fus.rc(DR$opt.design.rounded, tc = 30, nf = 2, alpha = 1,
+   formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01),
+   useLower = useLower, useUpper = useUpper)
R> fusUR <- pv.fus.rc(UR$opt.design.rounded, tc = 30, nf = 2, alpha = 1,
+   formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01),
+   useLower = useLower, useUpper = useUpper)
R> compare.fus(fusDR, fusUR)

R> vdusDR <- pv.vdus.rc(DR$opt.design.rounded, tc = 30, nf = 2, alpha = 1,
+   formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01),
+   useLower = useLower, useUpper = useUpper)
R> vdusUR <- pv.vdus.rc(UR$opt.design.rounded, tc = 30, nf = 2, alpha = 1,
```

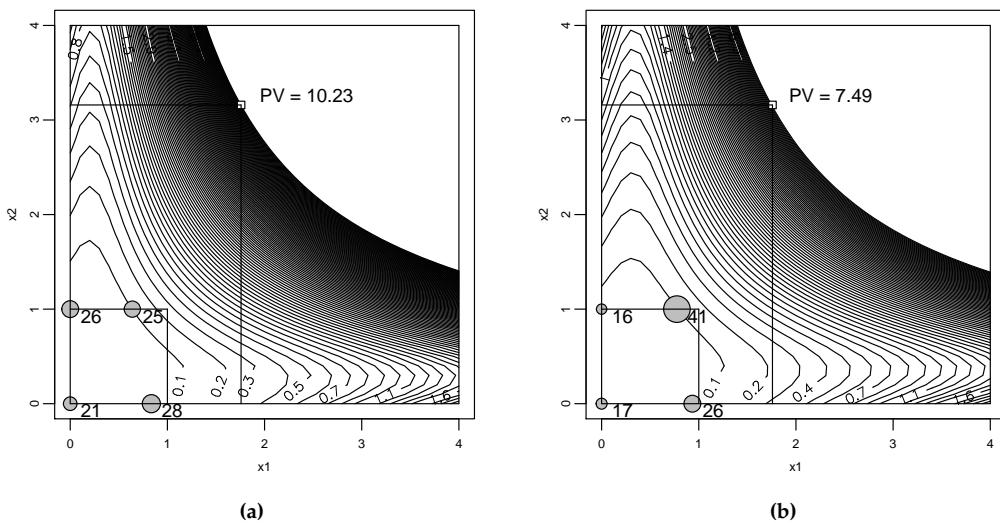


Figure 2: Prediction variance contour plots of (a) *D-optimal* and (b) *U-optimal* designs with right censoring drawn by `pv.contour.rc` function.

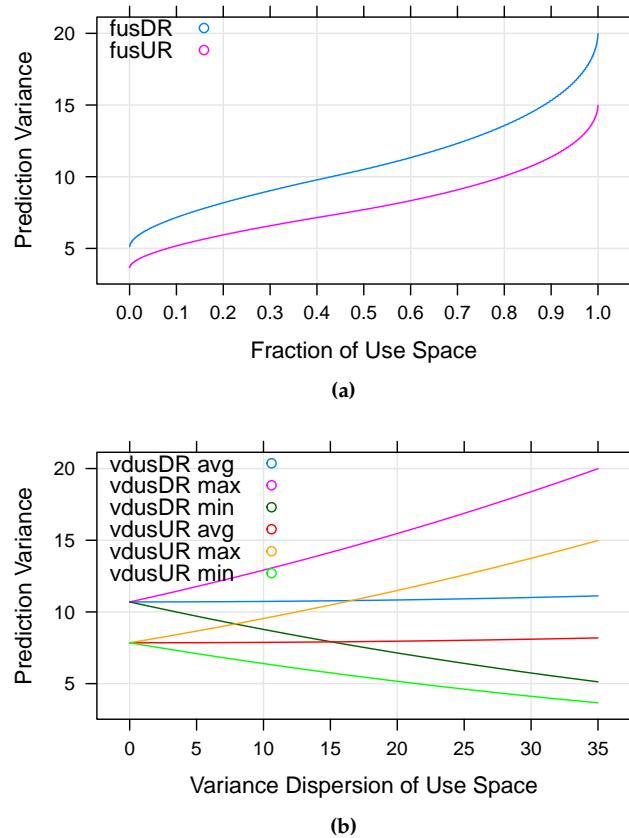


Figure 3: Comparison of *D-optimal* and *U-optimal* designs with right censoring using (a) FUS plot and (b) VDUS plot.

```
+   formula = ~ x1 + x2 + x1:x2, coef = c(0, -4.086, -1.476, 0.01),
+   useLower = useLower, useUpper = useUpper)
R> compare.vdus(vdusDR, vdusUR)
```

Figure 3 shows that the U -optimal test plan performs better in majority of the use-space region with respect to the prediction variance.

Finally, the `convert.stress.level` function is useful for converting the U -optimal test plan to the natural stress level conditions.

```
R> convert.stress.level(NdesLow, NdesHigh, stand = UR$opt.design.rounded)
```

	x1	x2	allocation
1	30.28840	4.499810	17
2	34.53414	4.499810	26
3	30.28840	4.094345	16
4	33.81136	4.094345	41

Summary

This paper describes the **ALTopt** package in R for constructing optimal ALT test plans for right- and interval-censored data. The package accommodates three statistical optimality criteria – D -optimal, U -optimal and I -optimal. It applies the GLM approach to the modeling of failure/censoring times and the derivation of the asymptotic variance-covariance matrix of regression coefficients. Failure times are assumed to follow a Weibull distribution. To use the package effectively, users are required to specify the linear predictor of the GLM and the shape parameter of the Weibull distribution. An example demonstrated the construction of optimal test plans for an ALT with two stress factors and right-censored data. This package also provides graphical functions for evaluating and comparing various test plans.

Bibliography

- R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995. [p182]
- A. Giovannitti-Jensen and R. H. Myers. Graphical assessment of the prediction capability of response surface designs. *Technometrics*, 31(2):159–171, 1989. [p183]
- R. T. Johnson and D. C. Montgomery. Choice of second-order response surface designs for logistic and Poisson regression models. *International Journal of Experimental Design and Process Optimisation*, 1(1):2–23, 2009. [p178]
- W. Q. Meeker and L. A. Escobar. *Statistical Methods for Reliability Data*, volume 314. John Wiley & Sons, 1998. [p178]
- W. Q. Meeker and W. Nelson. Optimum accelerated life-tests for the Weibull and extreme value distributions. *IEEE Transactions on Reliability*, 24(5):321–332, 1975. [p185]
- E. M. Monroe, R. Pan, C. M. Anderson-Cook, D. C. Montgomery, and C. M. Borror. A generalized linear model approach to designing accelerated life test experiments. *Quality and Reliability Engineering International*, 27(4):595–607, 2011. [p177, 179]
- R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, volume 705. John Wiley & Sons, 2009. [p183]
- W. B. Nelson. A bibliography of accelerated test plans. *IEEE Transactions on Reliability*, 54(2):194–197, 2005a. [p177]
- W. B. Nelson. A bibliography of accelerated test plans Part II—references. *IEEE Transactions on Reliability*, 54(3):370–373, 2005b. [p177]
- K. Seo and R. Pan. *ALTopt: Optimal Experimental Designs for Accelerated Life Testing*, 2015. URL <https://CRAN.R-project.org/package=ALTopt>. R package version 0.1.1. [p177]
- T. Yang and R. Pan. A novel approach to optimal accelerated life test planning with interval censoring. *IEEE Transactions on Reliability*, 62(2):527–536, 2013. [p177, 183]

A. Zahran, C. M. Anderson-Cook, and R. H. Myers. Fraction of design space to assess prediction capability of response surface designs. *Journal of Quality Technology*, 35(4):377–386, 2003. [p183]

Kangwon Seo

School of Computing, Informatics, Decision Systems Engineering

Arizona State University

699 S Mill Ave, Tempe, AZ 85281

USA

kseo7@asu.edu

Rong Pan

School of Computing, Informatics, Decision Systems Engineering

Arizona State University

699 S Mill Ave, Tempe, AZ 85281

USA

rong.pan@asu.edu

abctools: An R Package for Tuning Approximate Bayesian Computation Analyses

by Matthew A. Nunes and Dennis Prangle

Abstract Approximate Bayesian computation (ABC) is a popular family of algorithms which perform approximate parameter inference when numerical evaluation of the likelihood function is not possible but data can be simulated from the model. They return a sample of parameter values which produce simulations close to the observed dataset. A standard approach is to reduce the simulated and observed datasets to vectors of *summary statistics* and accept when the difference between these is below a specified threshold. ABC can also be adapted to perform model choice.

In this article, we present a new software package for R, **abctools** which provides methods for tuning ABC algorithms. This includes recent dimension reduction algorithms to tune the choice of summary statistics, and coverage methods to tune the choice of threshold. We provide several illustrations of these routines on applications taken from the ABC literature.

Introduction

Approximate Bayesian computation (ABC) refers to a family of statistical techniques for inference in cases where numerical evaluation of the likelihood is difficult or intractable, ruling out standard maximum likelihood and Bayesian techniques. It has been successfully applied in a wide range of scientific fields which encounter complex data and models, such as population genetics (Fagundes et al., 2007; Beaumont, 2010), ecology (Csilléry et al., 2010), infectious disease modelling (Luciani et al., 2009; Brooks-Pollock et al., 2014), systems biology (Ratmann et al., 2007; Toni et al., 2009) and astronomy (Cameron and Pettitt, 2012; Weyant et al., 2013).

ABC performs inference based on simulation of datasets rather than likelihood evaluation. For this reason it is known as a *likelihood-free* method. The simplest ABC algorithm is rejection-ABC. This simulates parameter values from the prior and corresponding datasets from the model of interest. Parameters are accepted if the distance between *summary statistics* of the simulated and the observed data is below a *threshold*, ϵ . A similar approach can be used to choose between several models with intractable likelihoods. In all cases two key tuning choices for ABC are ϵ and which summary statistics are used. **abctools** provides various tools to assist these choices. It has been designed to complement existing software for performing ABC algorithms, especially the **abc** package (Csilléry et al., 2012). The examples in this paper use version 1.0.3 of **abctools**. Note that all the methods provided require access to at least some of the datasets simulated by ABC. In this sense they are post-processing tools.

The remainder of the article is organised as follows. First a review of relevant ABC algorithms, theory and software is given. Then two data examples are introduced which will be used for illustration throughout the paper. The following section describes the summary statistic selection methods provided by **abctools**. The final section considers choice of ϵ using the coverage property (Prangle et al., 2014).

Review of ABC

The following algorithms perform ABC for parameter inference or model choice. This is done in a Bayesian framework. Observed data is represented by x_{obs} . One or several probability densities $p(x|\theta, m)$ are available as models for the data. Here θ is a vector of parameters and m is a model indicator. Prior model weights $p(m)$ and parameter densities for each model $p(\theta|m)$ must also be specified. (Note that there is no requirement for the length of θ to be the same in all models.) If there is only one model of interest (the parameter inference case) the model can be written as $p(x|\theta)$ and then only a single parameter prior $p(\theta)$ is needed.

The ABC algorithms require that it is possible to sample from the priors and models. They also require various tuning choices: a distance function $d(\cdot, \cdot)$ (Euclidean distance is a common choice), a threshold $\epsilon \geq 0$ and a mapping $s(\cdot)$ from data to a vector of summary statistics.

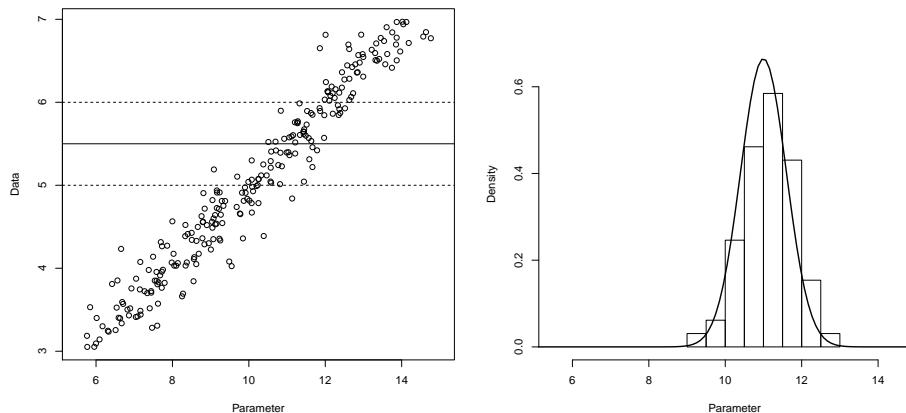


Figure 1: A pictorial illustration of the rejection-ABC algorithm for inference in a single parameter model (details omitted). The left panel shows simulated parameters and datasets. The solid horizontal line shows the observed data, and the dashed lines show acceptance boundaries. The right panel is a histogram and kernel density estimate of the accepted parameter values. These approximate the posterior distribution.

Rejection-ABC for parameter inference

Initialisation: For the observed dataset x_{obs} , compute a vector of summary statistics $s_{\text{obs}} = s(x_{\text{obs}})$.

Main loop:

1. Draw a parameter vector $\theta' \sim p(\theta)$ from the prior distribution;
2. Generate data from the model $x' \sim p(x|\theta')$, and compute summaries $s' = s(x')$;
3. If $d(s', s_{\text{obs}}) \leq \varepsilon$, accept θ' .

Rejection-ABC for model choice

Initialisation: For the observed dataset x_{obs} , compute a vector of summary statistics $s_{\text{obs}} = s(x_{\text{obs}})$.

Main loop:

1. Draw a model $m' \sim p(m)$ from the prior distribution on models;
2. Draw a parameter vector $\theta' \sim p(\theta|m')$ from the prior distribution on parameters for model m' ;
3. Generate data from the model $x' \sim p(x|\theta', m')$, and compute summaries $s' = s(x')$;
4. If $d(s', s_{\text{obs}}) \leq \varepsilon$, accept (m', θ') .

Both algorithms output a sample from an approximation to the posterior distribution. That is, for parameter inference the output is $\theta_1, \theta_2, \dots$ from an approximation to $p(\theta|x_{\text{obs}})$, and for model choice the output is $(m_1, \theta_1), (m_2, \theta_2), \dots$ from an approximation to $p(\theta, m|x_{\text{obs}})$.

If $\varepsilon = 0$ then only exact matches $x' = x_{\text{obs}}$ are accepted. It can easily be shown that in this case the output sample follows the exact posterior distribution of interest. However this is rarely practical as the probability of an exact match is typically very low for discrete data or zero for continuous data. Hence a tolerance $\varepsilon > 0$ is used, producing a sample from an approximation to the posterior (see Figure 1). An obvious acceptance criterion is $d(x', x_{\text{obs}}) \leq \varepsilon$, but this has been found to produce a poor approximation unless the data is low dimensional. Intuitively this is because close matches to the data become increasingly unlikely as the number of numerical components which must be matched increases. This *curse of dimensionality* problem motivates the use of low-dimensional summary statistics, which have greatly improved results in practice. See Beaumont (2010); Csilléry et al. (2010); Marin et al. (2012) for a more detailed discussion of this issue, and general background on ABC.

Two crucial tuning choices in rejection-ABC are the tolerance ε and the summary statistics $s(\cdot)$. Several approaches have been proposed in the literature to address these choices. **abctools** implements a range of such summary statistic selection methods and a method for choosing ε based on Prangle et al. (2014). Note that there are many other methods: for example see Blum et al. (2013) for a recent review of ABC summary statistic selection methods and Csilléry et al. (2012) for choice of ε by cross-validation.

Name	References	Stand-alone	Platform	Models
abc	Csilléry et al. (2012)	No (R package)	All	General
ABCreg	Thornton (2009)	Yes	Linux, OS X	General
easyABC	Jabot et al. (2013)	No (R package)	All	General
ABCtoolbox	Wegmann et al. (2010)	Yes	Linux, Windows	Genetics
Bayes-SSC	Anderson et al. (2005)	Yes	All	Genetics
DIY-ABC	Cornuet et al. (2008, 2010, 2014)	Yes	All	Genetics
msBayes	Hickerson et al. (2007)	Yes	Linux, OS X	Genetics
MTML-msBayes	Huang et al. (2011)	Yes	Linux, OS X	Genetics
onesamp	Tallmon et al. (2008)	Yes (web interface)	All	Genetics
PopABC	Lopes et al. (2009)	Yes	All	Genetics
REJECTOR	Jobin and Mountain (2008)	Yes	All	Genetics
EP-ABC	Barthélémy and Chopin (2014)	No (MATLAB toolbox)	All	State space models (and related)
ABC-SDE	Picchini (2013)	No (MATLAB toolbox)	All	Stochastic differential equations
ABC-SysBio	Liepe et al. (2010)	Yes (Python scripts)	All	Systems biology

Table 1: Software for ABC. “All” regarding platform refers to Linux, OS X (Mac) and Windows.

There are several ABC algorithms which are more efficient than rejection-ABC. These concentrate on simulating from models and parameter values close to previously successful values. These include Markov chain Monte Carlo (Marjoram et al., 2003; Sisson and Fan, 2011) and sequential Monte Carlo (SMC) techniques (Sisson et al., 2007; Toni et al., 2009; Beaumont et al., 2009; Del Moral et al., 2012). A complementary approach is to post-process ABC output to reduce the approximation in using $\epsilon > 0$ by adjusting accepted parameter values via regression onto the observed summary statistics (Beaumont et al., 2002; Blum and François, 2010). In both cases tuning ϵ and $s(\cdot)$ remains of crucial importance. All of the **abctools** methods can be used with post-processing. Also, all of the summary statistic selection methods can be adapted for use with other ABC algorithms and the details of this are discussed below. However the approach to tuning ϵ is applicable to rejection-ABC only. The reason is that ABC must be repeated under many different observations, and this is only computationally feasible under rejection-ABC as the same simulations can be reused each time. Some alternative methods have been proposed for the choice of ϵ in ABC-SMC algorithms, see for example Drovandi and Pettitt (2011); Del Moral et al. (2012); Lenormand et al. (2013).

Existing software

This section details existing software available for ABC, then outlines how **abctools** provides previously unavailable methodology and how it can be used alongside other software. Existing software is detailed in Table 1.

The software varies widely in which ABC algorithms are implemented. Of the two R packages, **abc** implements ABC-rejection with many methods of regression post-processing, while **easyABC** implements a wider suite of ABC algorithms but not post-processing. For full details of the other software see the references in Table 1.

Some of the available software packages provide methods for selecting summary statistics. A projection method based on partial least squares (Wegmann et al., 2009) is available in *ABCtoolbox*, and one for model choice based on linear discriminant analysis (Estoup et al., 2012) in *DIY-ABC*. Another category of methods is *regularisation techniques*, for example via ridge regression (Blum and François, 2010; Blum et al., 2013). Ridge regression regularisation is implemented in the R package **abc**; see Csilléry et al. (2012) for more details. The **abc** package also provide a method to choose ϵ by cross-validation.

The **abctools** package has been designed to complement the existing software provision of ABC algorithms by focusing on tools for tuning them. It implements many previously unavailable methods from the literature and makes them easily available to the research community. The software has been structured to work easily in conjunction with the **abc** package, but the package also has the flexibility to be used with other ABC software. This is discussed below (under “Using other ABC algorithms with **abctools**”), along with details of how the package framework can be used to implement further emerging methodology for summary statistic selection and construction.

Data examples

Summaries of genetic variation

The first dataset represents data generated from a commonly used model in population genetics. Specifically, the **abctools** package contains the two datasets `coal` and `coalobs`. The dataset `coal` is a matrix of dimension 100000×9 , representing parameters and summaries generated from an infinite-sites coalescent model for genetic variation (see [Nordborg 2007](#) for more details). In particular, the parameters of interest are the scaled mutation rate, $\bar{\theta}$, and the scaled recombination rate, ρ ; columns 3–9 are data summaries, namely the number of segregating sites (C_1); the pairwise mean number of nucleotidic differences (C_3); the mean R^2 across pairs separated by < 10% of the simulated genomic regions (C_4); the number of distinct haplotypes (C_5); the frequency of the most common haplotype (C_6) and the number of singleton haplotypes (C_7). The summary C_2 (column 4) is a spurious statistic, namely a standard uniform random deviate.

The data `coalobs` is a matrix of dimension 100×9 , representing similar instances of summary statistics from the model and associated parameters; these can be treated as observed data. Similar data were analysed in simulations in [Joyce and Marjoram \(2008\)](#) and [Nunes and Balding \(2010\)](#). The datasets can be loaded with `data(coal)` and `data(coalobs)` respectively.

A bigger dataset with 10^6 rows of similar summaries can be loaded using the code:

```
> mycon <- url("http://www.maths.lancs.ac.uk/~nunes/ABC/coalaracle.rda")
> load(mycon)
> close(mycon)
```

g-and-k distribution

The g-and-k distribution, used in various applications such as finance and environmental modelling, is a family of distributions which is specified by its quantile distribution, but does not have a closed form expression for its density ([Rayner and MacGillivray, 2002](#)). Data can easily be simulated by the inversion method. The dataset included in the **abctools** package is a matrix of dimension 100000×11 consisting of $n = 100000$ simulations of 4 parameters (A , B , g and k), together with 7 summary statistics representing the octiles of 1000 independent draws given the corresponding parameters. Such quantiles have been used for inference in an ABC context by [Drovandi and Pettitt \(2011\)](#) and [Fearnhead and Prangle \(2012\)](#), amongst others.

The dataset can be loaded using the code:

```
> mycon <- url("http://www.maths.lancs.ac.uk/~nunes/ABC/gkdata.rda")
> load(mycon)
> close(mycon)
```

The code used to generate these simulations is available at <http://www.maths.lancs.ac.uk/~nunes/ABC/gksim.R>.

Summary statistics selection

Identifying an informative and low-dimensional set of summaries to represent high dimensional data for use in ABC methods is of high importance for meaningful posterior inference; a number of methods to achieve this have been proposed in the statistical literature. We assume there is a prespecified set of *input statistics* of the data $z(x) = \{z_1, \dots, z_k\}$. This may be the raw data, or some transformations believed to be informative. Techniques for choosing ABC summary statistics fall into several categories, including: methods that select a *best subset* of z ([Joyce and Marjoram, 2008](#); [Nunes and Balding, 2010](#)) and secondly, *projection techniques* that project z onto a lower dimensional space ([Wegmann et al., 2009](#); [Blum and François, 2010](#); [Fearnhead and Prangle, 2012](#)). A review of methods for choosing summary statistics, including those mentioned above, can be found in [Blum et al. \(2013\)](#). This study found that when k was relatively small, best subset methods were generally preferable, and otherwise projection techniques performed better.

In what follows we describe the implementations of a number of methods for choosing summary statistics in the **abctools** package, namely the *approximate sufficiency* algorithm of [Joyce and Marjoram \(2008\)](#); the entropy criterion and two-stage methods of [Nunes and Balding \(2010\)](#), and the *semi-automatic ABC* projection technique of [Fearnhead and Prangle \(2012\)](#). For summary statistics selection the user must simulate parameters and data and supply these to the package. The resulting summary

statistics can then be passed to another package to perform ABC. This form of operation makes **abctools** particularly suited to rejection-ABC. Note however, that many of the main routines in this section have similar arguments, indicative of the flexible and modular nature of the package. Indeed, the final part of this section discusses the `selectsumm` wrapper function which can be used to implement any of the methods, as well as using **abctools** with other user-defined ABC routines.

Best subset methods

As outlined above, the principle of summary subset selection methods is to select a subset of informative statistics $s_A \subseteq z$ for use in ABC inference, such as the rejection-ABC algorithm described above. In this section we outline the implementations of some of these “best subset” algorithms for summary selection.

Subset selection via approximate sufficiency. Joyce and Marjoram (2008) introduced a method of summary selection based on a measure of approximate sufficiency. The idea of the sufficiency criterion is that, if a (sub)set of summaries is sufficient for θ , then adding an extra statistic won’t affect the posterior distribution for θ . Motivated by this observation, the algorithm of Joyce and Marjoram (2008) sequentially tests the potential inclusion of individual statistics into the set s_A , accepting them if the change in the corresponding posterior density approximation exceeds a threshold. The change in the posterior is deemed sufficient if

$$\left| \frac{p_{ABC}(\theta|z_1, \dots, z_{k-1}, z_k)}{p_{ABC}(\theta|z_1, \dots, z_{k-1})} - 1 \right| > T(\theta), \quad (1)$$

where p_{ABC} denotes a histogram estimator approximation of the posterior density. See Section 5 of Joyce and Marjoram (2008) for details of how the threshold $T(\theta)$ is defined. Note that due to the form of the criterion (1), the test is at present only suitable for single parameter inference.

The hypothesis test is performed by the **abctools** function `AS.test`. The function has inputs `x1` and `x2`, representing approximate posterior samples for the density *without* or *including* the statistic being tested, respectively. The test returns a Boolean variable (TRUE or FALSE) indicating whether the second posterior sample (as represented by `x2`) is sufficiently different from the first posterior sample `x1`.

As an example of this, running the code

```
> unif.sample <- runif(10000); norm.sample <- rnorm(10000)
> AS.test(x1 = unif.sample, x2 = norm.sample)
[1] TRUE
```

results in a statement that the two posterior samples `x1` and `x2` are judged to be statistically different.

To decide on the final set of summaries, the test is performed as a sequential search, testing candidate statistics from z in turn. The final subset s_A is dependent on the order in which statistics from z are tested for inclusion; in practice, this order is random. The sequential testing procedure is implemented in the **abctools** function `AS.select`. The main arguments of the function are:

obs Input statistics corresponding to observed data, $z(x_{\text{obs}})$. This is a matrix of dimension `ndatasets` x `k`.
param Simulated parameters (drawn from a prior) which were used to generate simulated data under the model; a matrix of dimension `nsims` x `p`.
sumstats Input statistics $z(x)$ generated using the model with the parameters `param`; a matrix of dimension `nsims` x `k`.

After performing the summary search procedure, the `AS.select` function returns the final subset of statistics s_A in the best component of the output. If the optional `trace` argument is set to TRUE (the default), the function will print messages to inform the user about the summary statistics search.

An example of using the `AS.select` function using the coalescent data described above is shown below.

```
> data(coal); data(coalobs)
> param <- coal[, 2]
> simstats <- coal[, 3:9]
> obsstats <- matrix(coalobs[1, 3:9], nrow = 1)
> set.seed(1)
> ASchoice <- AS.select(obsstats, param, simstats)
Sumstat order for testing is: 2 3 6 4 1 7 5
Current subset is: empty Test adding: 2
```

```

Empty subset not allowed - add

Current subset is: 2 Test adding: 3
No significant change to ABC posterior - don't add

Current subset is: 2 Test adding: 6
No significant change to ABC posterior - don't add

Current subset is: 2 Test adding: 4
No significant change to ABC posterior - don't add

Current subset is: 2 Test adding: 1
No significant change to ABC posterior - don't add

Current subset is: 2 Test adding: 7
Significant change to ABC posterior - add

Consider removing previous summaries
Current subset is: 2 7 Test removing: 2
No significant change to ABC posterior - remove

Current subset is: 7 Test adding: 5
No significant change to ABC posterior - don't add

Selected summaries: 7

> ASchoice$best
[1] 7

```

The result of the sequential search is that out of the summary subsets tested, the single summary subset $\{C_7\}$ is judged to be the most informative.

Subset selection via minimising an information criterion. Another sequential search algorithm for summary statistics in the **abctools** package is the flexible *minimum criterion* function `mincrit`. Essentially, this function cycles through each subset of summaries in turn, and computes a specified criterion on the ABC posterior sample produced with that particular set of summaries. The best subset s_A is judged to be that which minimises the criterion over all possible subsets of statistics. The search proposed in Nunes and Balding (2010) suggests minimising the κ -nearest neighbour entropy, E of the posterior sample

$$\hat{E} = \log \left[\frac{\pi^{p/2}}{\Gamma(p/2+1)} \right] - \psi(\kappa) + \log n + \frac{p}{n} \sum_{i=1}^n \log R_{i,\kappa}, \quad (2)$$

where p is the dimension of the parameter vector θ , $\psi(\cdot)$ denotes the digamma function, and where $R_{i,\kappa}$ denotes the Euclidean distance from θ^i to its κ -th closest neighbour in the posterior sample (Singh et al., 2003). Nunes and Balding (2010) follow Singh et al. (2003) in using $\kappa = 4$ for reasons of numerical stability. Blum et al. (2013) extend this entropy expression for weighted posterior samples. This entropy calculation in (2) is computed in **abctools** using the `nn.ent` function. For example, for the 4th nearest neighbour entropy calculation for a posterior sample `psample`, one would use the command

```
> nn.ent(psample, k = 4)
```

The `mincrit` function has many of the same arguments as the `AS.select` function above, including `obs`, `param` and `sumstats`, see the `mincrit` function documentation in the package **abctools** for a full list. Other function arguments include `crit`, which specifies the criterion to minimise. The default for this is `nn.ent`. The heuristic for this criterion as suggested by Nunes and Balding (2010) is that the entropy measures how concentrated the posterior is, and thus how much information is contained within the sample. However, other measures of spread or informativeness could be used in the `crit` argument instead of `nn.ent`.

Since `mincrit` performs an exhaustive search of all subsets of z , which can potentially be computationally intensive, the function has been designed to allow the user to decrease the number of computations by restricting the search to particular subsets of interest. In particular, as with the `AS.select` function, the user can limit the search to subsets of a maximum size, using the `limit` argument. Internally, this calls the function `compmat` to produce subsets on which to perform the criterion. For example `compmat(4)` produces a matrix of all subsets of size 4, whereas the code `compmat(4, limit`

= 2) computes a matrix of all 10 subsets of size 2 and below from 4 statistics, each row of the matrix indicating which of the 4 statistics are included in the subset:

```
C1 C2 C3 C4
[1,] 1 0 0 0
[2,] 0 1 0 0
[3,] 0 0 1 0
[4,] 0 0 0 1
[5,] 1 1 0 0
[6,] 1 0 1 0
[7,] 1 0 0 1
[8,] 0 1 1 0
[9,] 0 1 0 1
[10,] 0 0 1 1
```

In addition, the search can be limited by setting the argument `sumsubs` to a particular subset of initial summaries. This has the effect of only considering subsets containing those statistics. Alternatively, with the argument `do.only`, the user can specify certain summary subsets to consider. This can either be in matrix format like the output from `compmat`, or a vector of indices indicating rows of `compmat(k)` for which to compute the `crit` criterion.

To run the minimum criterion search algorithm, one could do:

```
> entchoice <- mincrit(obsstats, param, simstats, crit = nn.ent,
+                         do.only = 1:30)
```

This would only consider the first 30 subsets as specified in `compmat(ncol(obsstats))`.

The `mincrit` function returns a list object with the following components:

critvals If `do.crit = TRUE`, a matrix representing the computed `crit` criterion values.

best A matrix representing the best subset (which minimises `crit`).

posssubs A matrix (or vector) of subsets considered by the search algorithm. This component reflects the choice of input `do.only`.

sumsubs The index of the initial pool of statistics considered in the search. By default, this is set to `1:ncol(obsstats)`.

The best subset is judged to be the 20th subset in the search, {C₃, C₅}, as seen from the best component of the output:

```
> entchoice$best
 [,1] [,2]
20     3     5
```

Two stage procedure. As a refinement of the entropy-based summary selection procedure, Nunes and Balding (2010) propose running a second summary search based on the best subset found by minimum entropy. The closest *simulated* datasets to x_{obs} are identified using the summaries chosen in the first stage. The number of these close datasets is controlled by the argument `dsets`. The second stage selects a subset of summaries which minimises a measure of true posterior loss when ABC is performed on these datasets. This is done by comparing the ABC output to the true generating parameter values by some criterion. The default is calculating relative sum of squares error (RSSE). Since this second stage is effectively a search similar in form to that performed by `mincrit`, the functionality of `mincrit` is exploited by calling it internally within `stage2`. By default, the posterior loss minimisation is computed with the function `rsse`. The argument `init.best` specifies which subset to use as a basis to perform the second ABC analysis, e.g., the best subset chosen by the minimum entropy criterion. Other arguments to this function mimic those of `mincrit`.

An example call for this function is

```
> twostchoice <- stage2(obsstats, param, simstats, dsets = 25,
+                         init.best = 20, do.only = 1:30)
> twostchoice$best
 [,1] [,2]
21     3     6
```

The output object is the same as that of `mincrit`, with the exception that in addition, `stage2` also returns the `dsets` simulated datasets deemed closest to the observed data z_{obs} .

Semi-automatic ABC

When the set of input statistics $z(x) = (z_1, z_2, \dots, z_k)$ is large, it is computationally inefficient to search all possible subsets. Furthermore, good summary statistics for ABC may not be individual z_i s but combinations e.g., their mean. *Semi-automatic ABC* (Fearnhead and Prangle, 2012) is a projection method which attempts to find linear combinations which are informative about θ by fitting a regression. This produces a low dimensional vector of summaries as there is one for each parameter, i.e., $\hat{\theta}_i(z) = \beta_{i0} + \sum_{j=1}^k \beta_{ij} z_j$ for $1 \leq i \leq p$ where p is the dimension of θ . The summaries are estimators of the conditional posterior parameter mean $\mathbb{E}(\theta|x)$. As theoretical support, Fearnhead and Prangle prove that ABC using $s(x) = \mathbb{E}(\theta|x)$ (i.e., perfect estimators) and $\varepsilon = 0$ would minimise a posterior loss function reflecting the quality of point estimators.

Linear regression is a crude tool to estimate $\mathbb{E}(\theta|x)$ so some further steps are proposed. These require some user input, which is why the method is referred to as *semi-automatic*. Firstly the set of input statistics z must be carefully chosen. For this method it should be composed of many potentially informative data features. These could include the raw data and various non-linear transformations for example. Secondly it is recommended to only fit the regression locally to the main posterior mass by using the following steps.

1. Perform an ABC pilot run using summary statistics chosen subjectively or using another method. Use this to determine the region of main posterior mass, referred to as the *training region*.
2. Simulate parameters θ_{train}^j from the prior truncated to the training region and corresponding datasets x_{train}^j for $1 \leq j \leq N$.
3. Fit regressions as detailed above for various choices of $z = z(x)$.
4. Choose the best fitting regression (e.g., using BIC) and run ABC using the corresponding summaries. For robustness it is necessary to truncate the prior to the training region; our experience is that without such truncation artefact posterior modes may appear outside the training region.

Note that in rejection-ABC the same simulations can be used for the pilot ABC, training and main ABC steps, if desired. Also, step 1 can be omitted and the entire parameter space used as the training region. This is simpler, but empirical evidence shows that in some situations the training step is crucial to good performance (Fearnhead and Prangle, 2012; Blum et al., 2013).

abctools provides two functions for semi-automatic ABC. To facilitate a quick analysis, `semiauto.abc` performs a simple complete analysis; this uses rejection-ABC, avoids selecting a training region (i.e., it uses the full parameter space instead), and uses a single prespecified choice of z . To allow the user to implement the full method, `saABC` implements step 3 only. We describe only the former here as the latter is a very straightforward function. The main arguments of `semiauto.abc` are:

obs Input statistics corresponding to observed data. This is a matrix of dimension `ndatasets` \times `k`. In fact only a subset $z'(x_{obs})$ need be supplied. The full vector $z(x_{obs})$ consists of deterministic transformations of these specified by `satr`.

param Simulated parameters (drawn from a prior) which were used to generate simulated data under the model; a matrix of dimension `nsims` \times `p`.

sumstats Input statistics $z'(x)$ generated using the model with the parameters `param`; a matrix of dimension `nsims` \times `k`.

satr A list of *functions*, representing the vector of transformations to perform on the features `sumstats`, with which to estimate the relationship to the parameters θ . For more details, see the examples below.

Other arguments to the function are the same as `mincrit`; see the `saABC` documentation for more details.

To perform semi-automatic ABC using the vector of elementwise transformations (z', z'^2, z'^3, z'^4) , one could use the function call:

```
> saabc <- semiauto.abc(obsstats, param, simstats,
+                         satr = list(function(x) {
+                           outer(x, Y = 1:4, "^"))))
```

Alternatively, the same transformations could be specified by setting `satr` to `list(function(x) cbind(x, x^2, x^3, x^4))`. This alternative way of choosing this argument uses a single function which outputs all four transformations as a vector.

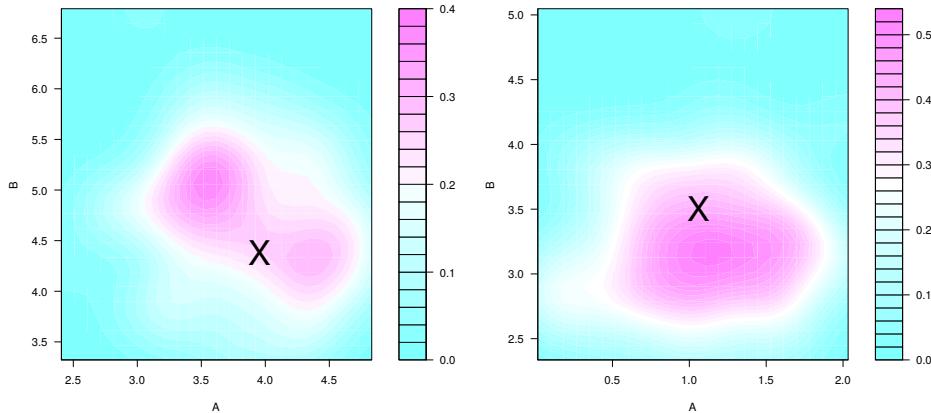


Figure 2: Joint posterior densities for two datasets for the (A, B) g-and-k distribution parameters, based on summary statistics chosen by semi-automatic ABC. The true parameter values are indicated by crosses.

The output from the `semiauto.abc` function is similar to that of `mincrit`, except that the output object also has a component `sainfo`, containing relevant choices of arguments pertaining to the ABC runs in steps 1 and 4 above. More specifically, the `sainfo` component is a list with information about the simulations used to perform each of the ABC runs, as well as the vector of transformations `satr`.

An example of `semiauto.abc` on the g-and-k dataset is as follows. The corresponding results and an analysis on another dataset are shown in Figure 2.

```
> mycon <- url("http://www.maths.lancs.ac.uk/~nunes/ABC/gkdata.rda")
> load(mycon)
> close(mycon)
> params <- gkdata[, 1:4]
> octiles <- gkdata[, 5:11]
> obs <- octiles[9, ]
> tfs <- list(function(x){cbind(x, x^2, x^3, x^4)})
> saabc <- semiauto.abc(obs = obs, param = params, sumstats = octiles,
+                         satr = tfs, overlap = TRUE, saprop = 1,
+                         abcprop = 1, tol = 0.001, method = "rejection",
+                         final.dens = TRUE)
> dens <- kde2d(saabc$post.sample[, 1], saabc$post.sample[, 2])
> filled.contour(dens, xlab = "A", ylab = "B")
```

An example on the coal data is as follows. Results are shown in Figure 3.

```
> data(coal)
> data(coalobs)
> coalparams <- coal[, 1:2]
> coaldatas <- coal[, 3:9]
> coalobs <- coal[1, 3:9]
> mytf <- list(function(x){cbind(x, x^2, x^3, x^4)})
> saabc.coal <- semiauto.abc(obs = coalobs, param = coalparams,
+                               sumstats = coaldatas, satr = mytf,
+                               tol = 0.001, overlap = TRUE, saprop = 1,
+                               abcprop = 1, method = "rejection",
+                               final.dens = TRUE)
> dens.coal <- kde2d(saabc.coal$post.sample[, 1],
+                       saabc.coal$post.sample[, 2])
> filled.contour(dens.coal, xlab = "theta", ylab = "rho")
```

The `selectsumm` convenience wrapper

The summary selection methods described in this section can be used with the individual functions as described above. Alternatively, the `abctools` package contains a convenient generic function `selectsumm`, with which any of the summary statistics choice algorithms can be performed. The

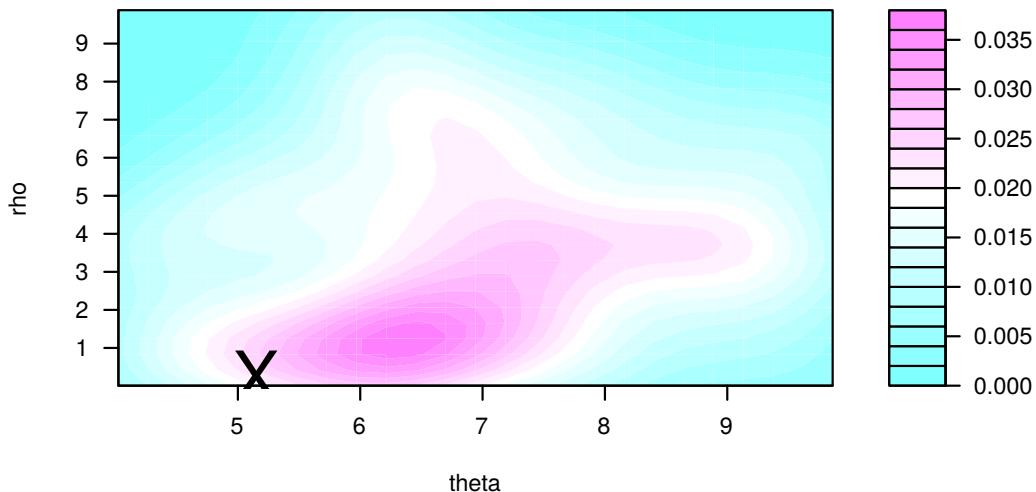


Figure 3: Joint posterior densities for the coal example, based on summary statistics chosen by semi-automatic ABC. The true parameter values are indicated by crosses.

argument `ssmethod` can be any of the functions described above, for example `mincrit`. Note that any other arguments to the `ssmethod` function can be passed to `selectsumm` easily. In particular, many of the summary selection routines have common optional arguments, for example

obspar An optional matrix of true parameters corresponding to the observed summaries `obs`. This is useful if the function is used to test summary selection techniques on fake observed data (for which you know the generating parameters).

abcmethod A function which performs an ABC algorithm, for example the `abc` function from the **abc** R package. Other user-defined functions can also be supplied; see below for more details. By default, the `ssmethod` function uses the abc rejection-ABC algorithm, with a tolerance of `tol = 0.01`.

limit An (optional) integer value indicating whether to limit the search to subsets of a particular maximum size. For example, `limit = 3` would only consider potential subsets of statistics s_A with $|s_A| = 3$, see the subset selection section for more details.

do.err A logical variable indicating whether the simulation error should be computed to assess the performance of the selection algorithm. This is only relevant if `obspar` is supplied.

final.dens A logical variable. If `final.dens = TRUE`, then the final approximate posterior sample is returned, resulting from the ABC algorithm (`abcmethod`) using the final subset of summaries s_A .

errfn A function used to compute the simulation error between the posterior sample and the generating parameter values `obspar`. An example of such a function included in the **abctools** package is the relative sum of squares error (RSSE), computed using the function `rsse`.

Note that the `selectsumm` function can perform summary selection for any number of observed summary vectors; the function implements the `ssmethod` on each row of the `obsstats` argument. Examples of the `selectsumm` function call are

```
> ASchoice <- selectsumm(obsstats, param, simstats, ssmethod = AS.select)
or
> mycon <- url("http://www.maths.lancs.ac.uk/~nunes/ABC/gkdata.rda")
> load(mycon)
> close(mycon)
> param <- gkdata[, 1:2] # A and B parameters
> simstats <- gkdata[, 5:11]
> obsstats <- gkdata[9:10, 5:11] # treated as real data
> entchoicegk <- selectsumm(obsstats, param, simstats, ssmethod = mincrit,
+ crit = nn.ent, limit = 3, final.dens = TRUE,
+ do.err = TRUE, obspars = gkdata[9:10, 1:2])
> entchoicegk$best
S1 S2 S3 S4 S5 S6 S7
```

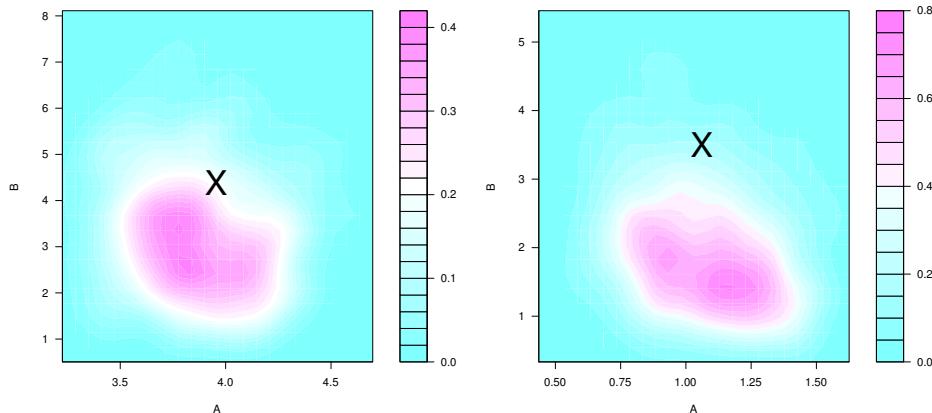


Figure 4: Joint posterior densities for two datasets for the (A, B) g-and-k distribution parameters, based on the $\{s_4, s_5\}$ statistics, as chosen by the minimum entropy subset selection method. The true parameter values are indicated by crosses.

```
23 0 0 0 1 1 0 0
23 0 0 0 1 1 0 0
```

If `do.err = TRUE`, then the inference error (as compared with the truth in `obspar`) is computed using the `errfn` function and is also returned in the `err` component of the function output. In addition, if `final.dens = TRUE` the output list element `post.sample` will contain the approximate posterior sample from the ABC inference corresponding to using s_A in the `abcmethod` ABC inference function. For example, for the `entchoicegk` object, the approximate posterior sample corresponds to the algorithm `abcmethod` using the subset (of size ≤ 3) with the lowest entropy. The resulting bivariate posterior density can then be seen by using the command `kde2d` from package **MASS** (Venables and Ripley, 2002):

```
> dens1 <- kde2d(entchoicegk$post.sample[, 1, 1],
+                  entchoicegk$post.sample[, 2, 1])
> dens2 <- kde2d(entchoicegk$post.sample[, 1, 2],
+                  entchoicegk$post.sample[, 2, 2])
> filled.contour(dens1, xlab = "A", ylab = "B")
> filled.contour(dens2, xlab = "A", ylab = "B")
```

The resulting posterior densities are shown in Figure 4.

Any other arguments to be passed to the function specified by the `abcmethod` argument can also be included. For more details on the optional arguments for the `abc` function see Csilléry et al. (2012).

Using other ABC algorithms with **abctools**

The flexibility of the **abctools** package can be exploited by using user-defined ABC algorithm implementations through the `abcmethod` argument to all of the ABC summary choice methods, namely `AS.select`, `mincrit`, `stage2` and `semiauto.abc`, or the convenience wrapper `selectsumm`, described above. The only constraint on the user's code for the ABC method is that it must return an object with a component named either `adj.values` or `unadj.values` containing the approximate posterior sample, to (minimally) mimic a return object of class "abc". For example, if one had written a function `likefreemcmc` to perform likelihood-free Markov chain Monte Carlo, one could use this in combination with a minimal criterion computed on the resulting (MCMC) posterior samples using the code:

```
> mcmcabc <- mincrit(obssstats, param, simstats, abcmethod = likefreemcmc)
```

To use **abctools** within ABC inference methods implemented by generic software, simply supply an appropriate R wrapper function to the `abcmethod` argument.

User-defined ABC summary selection methods can be accommodated with the **abctools** package. A new *projection* method, `projABC` say, could be implemented using the wrapper `selectsumm` as follows:

```
> projchoice <- selectsumm(obssstats, param, simstats, ssmethod = projABC)
```

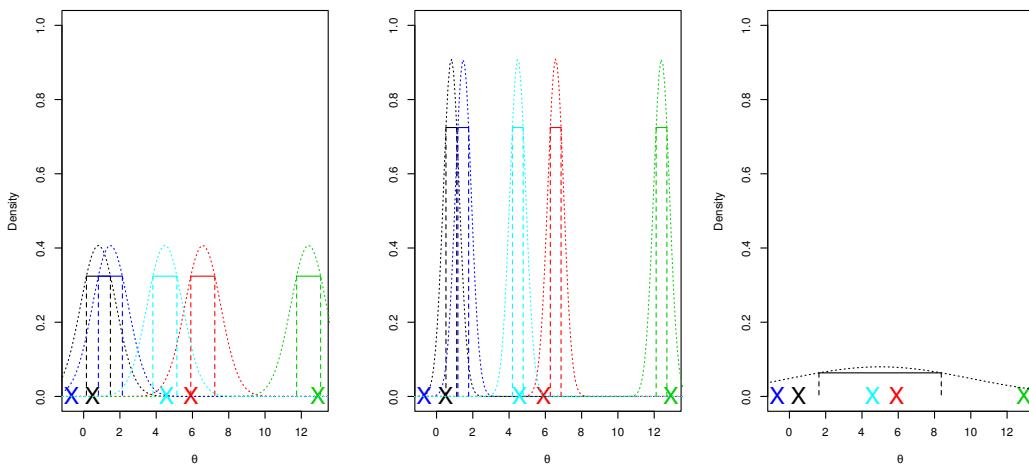


Figure 5: Illustration of the coverage property. The crosses represent simulated θ_0 values. The dotted curves on the left graph show the posterior densities based on noisy data about θ . On the middle graph they show approximate posterior densities which are over-precise, and the right-hand graph shows the prior density. All graphs have the same θ_0 and data values. The horizontal lines show 50% credible intervals. In the case shown on the left roughly half of these will contain the corresponding θ_0 value, which is consistent with the coverage property. For the middle graph case the proportion is generally smaller, illustrating that the coverage property does not hold. The right-hand graph shows that the prior credible interval also contains roughly half the θ_0 values, illustrating that coverage also holds here.

For the implementation to work, the summary choice function must have arguments named `obsstats`, `param`, `simstats` for the observed data, simulated parameters and simulated summaries respectively, as well as the logical argument `final.dens` indicating whether the approximate posterior sample is to be returned. Optional arguments could also be passed to `projABC` through the `selectsumm` wrapper.

Coverage

Theory

The **abctools** package can also test the accuracy of an ABC analysis, in particular to help choose the ϵ tuning parameter. This is done by testing whether it satisfies the *coverage property* (Prangle et al., 2014). As a simple example, consider (exact) Bayesian inference for the scalar parameter θ given data x . A standard summary of this is an $\alpha\%$ credible interval: an interval I such that $\Pr(\theta \in I|x) = \alpha/100$. Suppose a dataset is simulated from a parameter value θ_0 drawn from the prior and an $\alpha\%$ credible interval is calculated. It is easy to show that the probability the interval contains θ_0 is $\alpha/100$. When this is true for all α , an inference method is said to satisfy the coverage property. This is illustrated by Figure 5. Note that the probability in question relates to a *random choice* of θ_0 . The stricter requirement of frequentist coverage requires a similar condition holds for every θ_0 .

Monahan and Boos (1992) and Cook et al. (2006) showed that an equivalent condition to the coverage property is that the distribution of p_0 , the posterior quantile of θ_0 , must be $U(0, 1)$. This property is much easier to test numerically, as shown in Figure 6. Prangle et al. (2014) discuss how such a test can be implemented efficiently in a rejection-ABC context. This involves performing ABC analyses under many data sets simulated from known θ_0 values. A manageable computational cost is achieved by reusing the same ABC simulations in each analysis and exploiting multicore processing. Prangle et al. (2014) also show that when θ_0 values are drawn from the prior, the coverage property is a necessary condition for an inference procedure to give the correct posterior but not a sufficient condition: the coverage property also holds for an inference procedure that always returns the prior distribution (see Figure 5). Recommendations are given for how this problem can be avoided, involving drawing θ_0 values from a non-prior distribution, but are not discussed here for reasons of brevity. In addition, they discuss testing the coverage property in ABC model choice analyses. The idea is to test that amongst analyses giving model \mathcal{M} weight of roughly α , the proportion of being truly from model \mathcal{M} is close to α . Several test statistics are proposed which can be calculated with **abctools**.

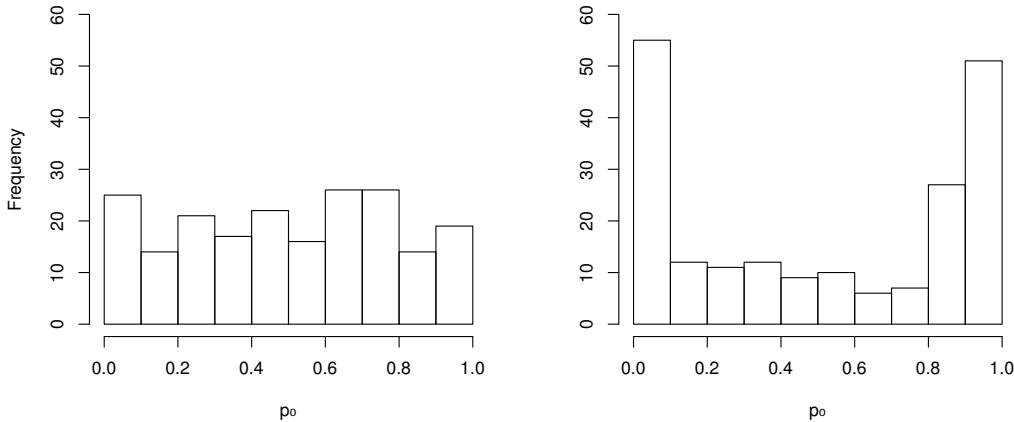


Figure 6: Illustration of testing the coverage property via the distribution of p_0 values, extending the example of Figure 5. The left-hand histogram shows the case where 200 p_0 values are calculated from the posterior distribution. On the right an over-precise estimate of the posterior is used instead. Clearly the left-hand histogram is consistent with $p_0 \sim U(0, 1)$ approximately and the right-hand one is not. This is confirmed by using the Kolmogorov-Smirnov test, which gives p -values of 0.82 (left) and 10^{-7} (right).

Package usage

The following code illustrates a typical analysis using the `cov.pi` (parameter inference) and `cov.mc` (model choice) functions. For the choice of tolerance ϵ , the user must supply simulated parameters, summary statistics and, for model choice, model indicators. Figure 7 shows typical output (n.b. some code to improve the appearance of this figure has been omitted.)

```
> library(abctools); library(ggplot2)
> data(human)
> ## Summary statistics for bottleneck model:
> stat.italy.sim <- subset(stat.3pops.sim, subset = (models == "bott"))
> ## Interesting epsilon values:
> myeps <- exp(seq(log(0.5), log(10), length.out = 15))
> set.seed(1)
> mytestsets <- sample(1:nrow(stat.italy.sim), 200)
> covout.pi <- cov.pi(param = par.italy.sim, sumstat = stat.italy.sim,
+                      testsets = mytestsets, eps = myeps,
+                      multicore = TRUE, diagnostics = c("KS", "CGR"),
+                      cores = 4)
> qplot(x = eps, y = pvalue, colour = test,
+        data = subset(cabc.out$diag, parameter == "Ne"), log = "y")
> mytestsets <- sample(nrow(stat.3pops.sim), 200)
> covout.mc <- cov.mc(index = models, sumstat = stat.3pops.sim,
+                      testsets = mytestsets, eps = myeps,
+                      diagnostics = c("freq", "loglik.binary"),
+                      multicore = TRUE, cores = 4)
> qplot(x = eps, y = pvalue, colour = test, data = covout.mc$diag,
+        log = "y")
```

The code analyses the `human` dataset supplied in the `abc` package (Csilléry et al., 2012), which contains simulated parameter values and summary statistics for a population genetic model. The `cov.pi` function estimates 200 p_0 values for each parameter. To do this, 200 of the simulated datasets are randomly sampled to be used as *pseudo-observed data* in leave-one-out style ABC analyses. The p -values of various diagnostic test statistics are returned in the `diag` component of the output. The left panel of Figure 7 plots p -values of uniformity tests – Kolmogorov-Smirnov and that of Cook et al. (2006) – as ϵ varies for one particular parameter. These show typical behaviour; coverage is supported for large ϵ when the ABC output is approximately drawn from the prior and also for ϵ small enough that ABC output is approximately drawn from the posterior. The right panel shows p -values for tests of whether output for the bottleneck model satisfies coverage. Again, coverage holds for large and small ϵ , but there is disagreement in between.

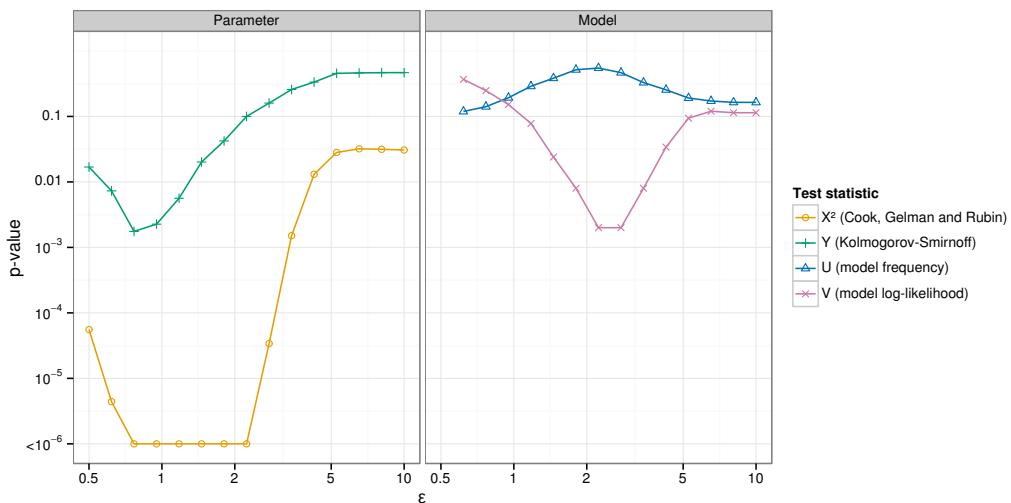


Figure 7: p -values testing coverage in the human dataset example. The left-hand graph is for the N_e parameter in the bottleneck model. The right-hand graph is for model choice considering the adequacy of the bottleneck model predictions.

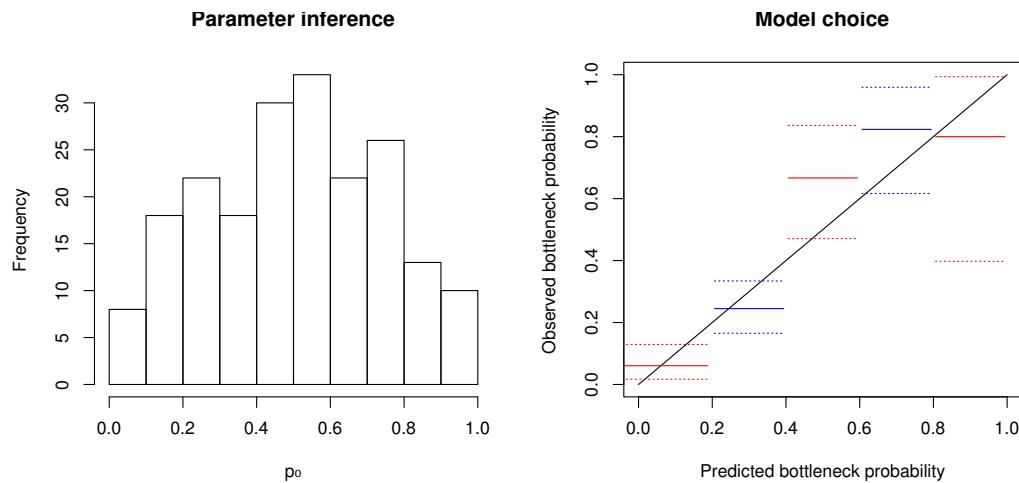


Figure 8: Detailed coverage diagnostic plots for the human dataset example with $\varepsilon = 2.2$.

Prangle et al. (2014) argue that p -values of test statistics only investigate certain aspects of coverage. A fuller investigation of interesting ε values, for example where test statistics disagree, can be found by diagnostic plots. For parameter inference histograms of underlying p_0 values are recommended, and for model choice plots of estimated against observed model probabilities, after some aggregation. This information is returned in the raw component of the output and can be plotted as follows, giving Figure 8. The `mc.ci` command is part of **abctools**.

```
> par(mfrow = c(1, 2))
> ## nb myeps[8] is 2.2
> hist(subset(covout.pi$raw, eps == myeps[8])$Ne, xlab = "p0",
+       main = "Parameter inference")
> mc.ci(covout.mc$raw, eps = myeps[8], modname = "bott",
+       modtrue = models, main = "Model choice")
```

The left-hand side of Figure 8 shows that for $\varepsilon = 2.2$, coverage clearly does not hold for the parameter of interest. The right-hand side shows no evidence to reject coverage for the bottleneck model.

Summary

This article has described the R package **abctools**. This implements several techniques for tuning approximate Bayesian inference algorithms. In particular, the package contains summary statistic

selection routines for the approximate sufficiency method of Joyce and Marjoram (2008); the entropy minimisation and two-stage error algorithm proposed by Nunes and Balding (2010); and the regression method of Fearnhead and Prangle (2012). It also contains methods to choose the acceptance threshold ϵ by assessing the coverage property of Prangle et al. (2014).

Acknowledgements

The authors would like to thank Scott Sisson and Michael Blum for some helpful suggestions when preparing the **abctools** R package accompanying this manuscript.

Bibliography

- C. N. K. Anderson, U. Ramakrishnan, Y. L. Chan, and E. A. Hadly. Serial SimCoal: A population genetics model for data from multiple populations and points in time. *Bioinformatics*, 21(8):1733–1734, 2005. [p191]
- S. Barthélémy and N. Chopin. Expectation propagation for likelihood-free inference. *Journal of the American Statistical Association*, 109(505):315–333, 2014. [p191]
- M. A. Beaumont. Approximate Bayesian computation in evolution and ecology. *Annual Review of Ecology, Evolution and Systematics*, 41:379–406, 2010. [p189, 190]
- M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate Bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002. [p191]
- M. A. Beaumont, J.-M. Marin, J.-M. Cornuet, and C. P. Robert. Adaptive approximate Bayesian computation. *Biometrika*, 96(4):983–990, 2009. [p191]
- M. G. B. Blum and O. François. Non-linear regression models for approximate Bayesian computation. *Statistics and Computing*, 20(1):63–73, 2010. [p191, 192]
- M. G. B. Blum, M. A. Nunes, D. Prangle, and S. A. Sisson. A comparative review of dimension reduction methods in approximate Bayesian computation. *Statistical Science*, 28(2):189–208, 2013. [p190, 191, 192, 194, 196]
- E. Brooks-Pollock, G. O. Roberts, and M. J. Keeling. A dynamic model of bovine tuberculosis spread and control in Great Britain. *Nature*, 511(7508):228–231, 2014. [p189]
- E. Cameron and A. N. Pettitt. Approximate Bayesian computation for astronomical model analysis: A case study in galaxy demographics and morphological transformation at high redshift. *Monthly Notices of the Royal Astronomical Society*, 425(1):44–65, 2012. [p189]
- S. R. Cook, A. Gelman, and D. B. Rubin. Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, 15(3):675–692, 2006. [p200, 201]
- J.-M. Cornuet, F. Santos, M. A. Beaumont, C. P. Robert, J.-M. Marin, D. J. Balding, T. Guillemaud, and A. Estoup. Inferring population history with DIYABC: A user-friendly approach to approximate Bayesian computation. *Bioinformatics*, 24(23):2713–2719, 2008. [p191]
- J.-M. Cornuet, V. Ravigné, and A. Estoup. Inference on population history and model checking using DNA sequence and microsatellite data with the software DIYABC (v1. 0). *BMC Bioinformatics*, 11(401), 2010. [p191]
- J.-M. Cornuet, P. Pudlo, J. Veyssier, A. Dehne-Garcia, M. Gautier, R. Leblois, J.-M. Marin, and A. Estoup. DIYABC v2.0: A software to make approximate Bayesian computation inferences about population history using single nucleotide polymorphism, DNA sequence and microsatellite data. *Bioinformatics*, 30(8):1187–1189, 2014. [p191]
- K. Csilléry, M. G. B. Blum, O. E. Gaggiotti, and O. François. Approximate Bayesian computation (ABC) in practice. *Trends in Ecology & Evolution*, 25(7):410–418, 2010. [p189, 190]
- K. Csilléry, O. François, and M. G. B. Blum. abc: An R package for approximate Bayesian computation (ABC). *Methods in Ecology and Evolution*, 2012. [p189, 190, 191, 199, 201]
- P. Del Moral, A. Doucet, and A. Jasra. An adaptive sequential Monte Carlo method for approximate Bayesian computation. *Statistics and Computing*, 22(5):1009–1020, 2012. [p191]

- C. C. Drovandi and A. N. Pettitt. Likelihood-free Bayesian estimation of multivariate quantile distributions. *Computational Statistics and Data Analysis*, 55(9):2541–2556, 2011. [p191, 192]
- A. Estoup, E. Lombaert, J.-M. Marin, T. Guillemaud, P. Pudlo, C. P. Robert, and J.-M. Cornuet. Estimation of demo-genetic model probabilities with approximate Bayesian computation using linear discriminant analysis on summary statistics. *Molecular Ecology Resources*, 12(5):846–855, 2012. [p191]
- N. J. R. Fagundes, N. Ray, M. A. Beaumont, S. Neuenschwander, F. M. Salzano, S. L. Bonatto, and L. Excoffier. Statistical evaluation of alternative models of human evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 104(45):17614–17619, 2007. [p189]
- P. Fearnhead and D. Prangle. Constructing summary statistics for approximate Bayesian computation: Semi-automatic ABC (with discussion). *Journal of the Royal Statistical Society: Series B*, 74(3):419–474, 2012. [p192, 196, 203]
- M. J. Hickerson, E. Stahl, and N. Takebayashi. msBayes: Pipeline for testing comparative phyogeographic histories using hierarchical approximate Bayesian computation. *BMC Bioinformatics*, 8(268), 2007. [p191]
- W. Huang, N. Takebayashi, Y. Qi, and M. J. Hickerson. MTML-msBayes: Approximate Bayesian comparative phylogeographic inference from multiple taxa and multiple loci with rate heterogeneity. *BMC Bioinformatics*, 12(1), 2011. [p191]
- F. Jabot, T. Faure, and N. Dumoulin. EasyABC: Performing efficient approximate Bayesian computation sampling schemes using R. *Methods in Ecology and Evolution*, 4(7):684–687, 2013. [p191]
- M. J. Jobin and J. L. Mountain. REJECTOR: Software for population history inference from genetic data via a rejection algorithm. *Bioinformatics*, 24(24):2936–2937, 2008. [p191]
- P. Joyce and P. Marjoram. Approximately sufficient statistics and Bayesian computation. *Statistical Applications in Genetics and Molecular Biology*, 7(1), 2008. Article 26. [p192, 193, 203]
- M. Lenormand, F. Jabot, and G. Deffuant. Adaptive approximate Bayesian computation for complex models. *Computational Statistics*, 28(6):2777–2796, 2013. [p191]
- J. Liepe, C. Barnes, E. Cule, K. Erguler, P. Kirk, T. Toni, and M. P. H. Stumpf. ABC-SysBio – approximate Bayesian computation in python with GPU support. *Bioinformatics*, 26(14):1797–1799, 2010. [p191]
- J. S. Lopes, D. Balding, and M. A. Beaumont. PopABC: A program to infer historical demographic parameters. *Bioinformatics*, 25(20):2747–2749, 2009. [p191]
- F. Luciani, S. A. Sisson, H. Jiang, A. R. Francis, and M. M. Tanaka. The epidemiological fitness cost of drug resistance in *Mycobacterium tuberculosis*. *Proceedings of the National Academy of Sciences of the United States of America*, 106(34):14711–14715, 2009. [p189]
- J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012. [p190]
- P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, 100(26):15324–15328, 2003. [p191]
- J. F. Monahan and D. D. Boos. Proper likelihoods for Bayesian analysis. *Biometrika*, 79(2):271–278, 1992. [p200]
- M. Nordborg. Coalescent theory. In D. J. Balding, M. J. Bishop, and C. Cannings, editors, *Handbook of Statistical Genetics*, pages 179–208. Wiley, Chichester, third edition, 2007. [p192]
- M. A. Nunes and D. J. Balding. On optimal selection of summary statistics for approximate Bayesian computation. *Statistical Applications in Genetics and Molecular Biology*, 9(1), 2010. [p192, 194, 195, 203]
- U. Picchini. abc-sde: Approximate Bayesian computation for stochastic differential equations, 2013. URL <http://sourceforge.net/projects/abc-sde/>. [p191]
- D. Prangle, M. G. B. Blum, G. Popovic, and S. A. Sisson. Diagnostic tools for approximate Bayesian computation using the coverage property. *Australia and New Zealand Journal of Statistics*, 56(4):309–329, 2014. [p189, 190, 200, 201, 203]
- O. Ratmann, O. Jørgensen, T. Hinkley, M. P. H. Stumpf, S. Richardson, and C. Wiuf. Using likelihood free inference to compare evolutionary dynamics of the protein networks of *H. pylori* and *P. falciparum*. *PLoS Computational Biology*, 3(11):2266–2278, 2007. [p189]

- G. D. Rayner and H. L. MacGillivray. Numerical maximum likelihood estimation for the g-and-k and generalized g-and-h distributions. *Statistics and Computing*, 12(1):57–75, 2002. [p192]
- H. Singh, V. Misra, N. and Hnizdo, A. Fedorowicz, and E. Demchuk. Nearest neighbor estimates of entropy. *American Journal of Mathematical and Management Sciences*, 23(3–4):301–321, 2003. [p194]
- S. A. Sisson and Y. Fan. Likelihood-free Markov chain Monte Carlo. In S. P. Brooks, A. Gelman, G. Jones, and X.-L. Meng, editors, *Handbook of Markov Chain Monte Carlo*, pages 319–341. Chapman and Hall/CRC Press, 2011. [p191]
- S. A. Sisson, Y. Fan, and M. Tanaka. Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, 104:1760–1765, 2007. Errata (2009), 106, 16889. [p191]
- D. A. Tallmon, A. Koyuk, G. Luikart, and M. A. Beaumont. onesamp: A program to estimate effective population size using approximate Bayesian computation. *Molecular Ecology Resources*, 8(2):299–301, 2008. [p191]
- K. R. Thornton. Automating approximate Bayesian computation by local linear regression. *BMC Genetics*, 10(35), 2009. [p191]
- T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. H. Stumpf. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society and its Interface*, 6(31):187–202, 2009. [p189, 191]
- W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL <http://www.stats.ox.ac.uk/pub/MASS4>. [p199]
- D. Wegmann, C. Leuenberger, and L. Excoffier. Efficient approximate Bayesian computation coupled with Markov chain Monte Carlo without likelihood. *Genetics*, 182(4):1207–1218, 2009. [p191, 192]
- D. Wegmann, C. Leuenberger, S. Neuenschwander, and L. Excoffier. ABCtoolbox: A versatile toolkit for approximate Bayesian computations. *BMC Bioinformatics*, 11(116), 2010. [p191]
- A. Weyant, C. Schafer, and W. M. Wood-Vasey. Likelihood-free cosmological inference with type Ia supernovae: Approximate Bayesian computation for a complete treatment of uncertainty. *The Astrophysical Journal*, 764(2):116, 2013. [p189]

Matthew A. Nunes
Lancaster University
Lancaster
UK
m.nunes@lancaster.ac.uk

Dennis Prangle
University of Reading
Reading
UK
d.b.prangle@reading.ac.uk

mtk: A General-Purpose and Extensible R Environment for Uncertainty and Sensitivity Analyses of Numerical Experiments

by Juhui Wang, Robert Faivre, Hervé Richard and Hervé Monod

Abstract Along with increased complexity of the models used for scientific activities and engineering come diverse and greater uncertainties. Today, effectively quantifying the uncertainties contained in a model appears to be more important than ever. Scientific fellows know how serious it is to calibrate their model in a robust way, and decision-makers describe how critical it is to keep the best effort to reduce the uncertainties about the model. Effectively assessing the uncertainties about the model requires mastering all the tasks involved in the numerical experiments, from optimizing the experimental design to managing the very time consuming aspect of model simulation and choosing the adequate indicators and analysis methods.

In this paper, we present an open framework for organizing the complexity associated with numerical model simulation and analyses. Named **mtk** (Mexico Toolkit), the developed system aims at providing practitioners from different disciplines with a systematic and easy way to compare and to find the best method to effectively uncover and quantify the uncertainties contained in the model and further to evaluate their impact on the performance of the model. Such requirements imply that the system must be generic, universal, homogeneous, and extensible. This paper discusses such an implementation using the R scientific computing platform and demonstrates its functionalities with examples from agricultural modeling.

The package **mtk** is of general purpose and easy to extend. Numerous methods are already available in the actual release version, including Fast, Sobol, Morris, Basic Monte-Carlo, Regression, LHS (Latin Hypercube Sampling), PLMM (Polynomial Linear metamodel). Most of them are compiled from available R packages with extension tools delivered by package **mtk**.

Introduction

Nowadays, computational modeling has become a common practice for scientific experiments and discoveries. Global climate models have been used for both short-term weather forecast (Lynch, 2008) and long-term climate change (Risbey et al., 2014). Environmental models have been developed for assessing the impact of a waste water treatment plant on a river flow (Brock et al., 1992). Epidemic models have been elaborated to investigate the mechanism by which diseases spread and to evaluate strategies to control their outbreaks (Papaix et al., 2014), etc. Most of them become more and more complex, with many parameters, state-variables and non-linear relationships, etc. Overloading the model to better mimic observed real data does not seem to be a passing practice but a continuing trend. Along with increased complexity of the models come diverse and greater uncertainties. Although computational modeling may improve our understanding of how an evidence emerges, and helps to get insight into how the elements of the system come together and interplay, one is usually left with the feeling that another model might produce different results and that some uncertainties have still remained somewhere in the system. “*Under the best circumstances, such models have many degrees of freedom and, with judicious fiddling, can be made to produce virtually any desired behavior, often with both plausible structure and parameter values*” (Hornberger and Spear, 1981). Although we admit that such a statement is exaggerating, it is greatly necessary to look into this issue and to try our best effort to get insight into the sources of such doubts.

Uncertainty and sensitivity analyses, when correctly applied, may help to gain an understanding of the impact of the various sources of uncertainties and to further assess the system performance and set up strategies for getting better control over the behavior of the model (Helton et al., 2006; Cariboni et al., 2007; Marino et al., 2008). Numerous methods and software have been developed (Adams et al., 2013; Saltelli et al., 2005; Pujol et al., 2015). Most of them are context-specific, domain-specific or theory-oriented. For example, Dakota was developed in the context of mechanics and large-scale engineering simulation (Adams et al., 2013). It is a closed complete software package which provides an efficient implementation of the iterative analysis model for parameter estimation, cost-based optimization, and sensitivity analysis. Implemented as a C++ library, methods developed in the project Dakota were widely used in a variety of large scale engineering projects relative to chemical (Salinger et al., 2004) and mechanical industries (Weirs et al., 2012). SimLab, on the other side, is a representative theory-

oriented package which covers the global sensitivity analysis techniques derived from Monte Carlo analysis (Saltelli et al., 2005; Joint Research Centre, 2006). Methods developed from SimLab are widely used in a large number of fields (Ciuffo et al., 2012). Although the software package offers a module to link for external model simulation and allows complex models beyond mathematical functions to be executed, such flexibility seems limited merely to the model implementation. The package obviously suffers from the lack of generality in the sense that it is difficult to include new methods especially those not based on Monte Carlo analysis. More ambitious, OpenTURNS (Baudin et al., 2015) builds on the global methodology promoted by an ESREDA group (de Rocquigny et al., 2008). It provides a great number of features for quantifying, prioritizing, and propagating uncertainties in computational models, but its extension requires programming skills that theoreticians and domain experts might not possess. More flexible and generic, Promethee provides a grid computing environment for numerical engineering and an interface for integrating R packages (Richet et al., 2009, 2010). When it comes to the software packages available for the R computing environment, there exists a great number of packages (Pujol et al., 2015; Dupuy et al., 2015; Monod et al., 2015; Lamboni et al., 2015). We do not aim at providing an exhaustive review of all the packages here, but we would like to point out package `sensitivity` (Pujol et al., 2015) which implements both the sampling and global analysis methods such as Sobol, FAST and Morris (Saltelli et al., 2005), package `spartan` (Alden et al., 2013, 2015) which compiles four widely used techniques for numerical experiments (the consistency analysis (Read et al., 2012), the robustness analysis (Read et al., 2012), the latin-hypercube sampling technique (Saltelli et al., 2000) and the eFAST technique (Marino et al., 2008)) and demonstrates their effectiveness for biological systems, and packages `diceDesign` (Dupuy et al., 2015) and `planor` (Monod et al., 2015) which implement the space-filling sampling technique (Pronzato and Müller, 2012) and the techniques for regular factorial designs (Monod et al., 2015), respectively.

Although these tools are very useful and greatly contribute to the development and the popularity of uncertainty and sensitivity analyses, they present some drawbacks. Most of them offer no possibility to evolve or to integrate methods developed in other contexts. However, uncertainty and sensitivity analyses are intrinsically trial-and-error processes because of the lack of reliable knowledge and data about the causes of the uncertainties contained in the model. There is no method which is universal and suitable for all contexts. Practitioners must repeat, undertake numerous tests, and vary the parameters and methods until finding the best one fitting to the situation. Sticking to a method which is inappropriate for the circumstances leads inevitably to a wrong way and to misinterpret the results. Thus, it is necessary to develop a simple to use, but powerful software package allowing practitioners to test and compare different methods for their own data. Such an application needs to be easy to set up, and yet unifying in its ability to include a wide range of methods and powerful to objectively analyze and rapidly report the results.

Inspired from these issues, we tried to compile the available methods into a general purpose open platform and make them become accessible to researchers and practitioners from different disciplines. Named `mtk` (Mexico Toolkit), the package we present here builds on an object-oriented framework using the R scientific computing platform. It provides facilities to interplay with external simulation platforms and to share data and knowledge with external applications in a seamless manner. It is easy to use, homogeneous, and offers a unique syntax and semantics for computing and data management. It is extensible in the sense that it tries to cover a large variety of *factor* types, and can easily integrate methods developed in the future without any major effort of reprogramming, even those developed by researchers not involved in the `mtk` initiative. It is self-contained and provides efficient tools to control all the processing tasks involved in the numerical experiments, from experimental design and model simulation to sensitivity computing and data reporting. Moreover, it is scalable to small or big projects, suitable for collaborative work in which the domain experts build the model and run the simulation, and the statisticians take charge of the different tasks of analyses and reporting.

We must note here that although the `mtk` package is designed to study any type of numerical simulation, one should not apply any method to any model on any occasion. How to match the methods to the problems is a difficult issue, which should not be accounted for only by software engineering but also by advice from domain experts and specialists on model exploration.

Methodology

Based on the computation of specific quantitative measures that allow, in particular, assessment of variability in output variables and importance of input variables, both uncertainty and sensitivity analyses are relevant methods for exploring numerical experiments (Saltelli et al., 2005; Fairre, 2013). Nevertheless, uncertainty and sensitivity analyses meet with different issues. Uncertainty analysis seeks to assess the impacts of the uncertainties contained in the inputs of the model on the outputs. It deals with the question of what level of uncertainty might be induced by the uncertainties contained in the inputs, and focuses on describing the probability distribution of the outputs as a function of

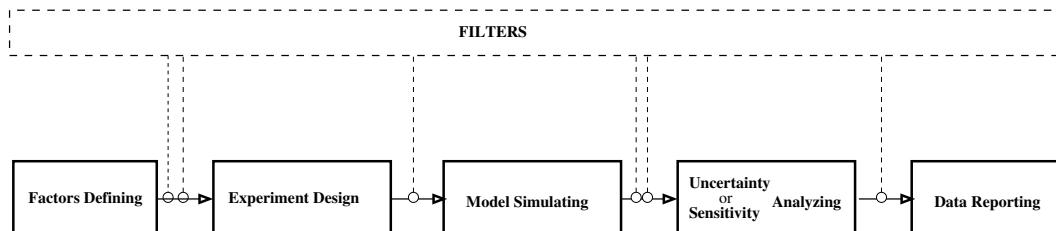


Figure 1: By decomposing the activities involved in uncertainty and sensitivity analyses into elementary tasks and using filters to connect them, we can cope with the heterogeneity of the approaches used in uncertainty and sensitivity analyses and unify them within a consistent and easily extensible framework.

the uncertainties contained in the inputs. In the simplest case, this probability distribution might be computed analytically from the characteristics of the uncertainties contained in the inputs. But in practice, the models are usually too complex to allow for any analytical solutions, and modern statistical methods must be used to estimate or approximate the probability distribution of the outputs. Various methods are already available. Among them, simulation methods seem to be the most representative and universal.

On the other hand, sensitivity analysis aims to identify the sources of the uncertainties and quantify their relative contributions. It deals with the question of which inputs exhibit the most important uncertain behaviors against the model, and allows us to focus on the ones that matter and ignore those that are less significant. Sensitivity analysis can be used to meet various objectives and goals such as identifying and prioritizing the most influential inputs, identifying non-influential inputs in order to fix them to nominal values, mapping the output behavior as a function of the inputs by focusing on a specific domain of inputs if necessary, calibrating model inputs using available information, etc. Terms such as influence, importance, ranking by importance, and dominance are all related to sensitivity analysis.

Sensitivity and uncertainty analyses rely on large and heterogeneous collection of approaches and tools. In this study, we try to find a consistent framework to unify the different approaches and tools. Our framework is a workflow-based one, which consists in decomposing the procedures of the uncertainty and sensitivity analyses into a series of elementary and generic tasks that can be manipulated and presented in a standard and homogeneous way. Each activity involved in the uncertainty and sensitivity analyses can be considered either as an elementary task or a combination of the elementary tasks. Appropriate combination and scheduling of the tasks allow to handle situations of any complexity. This workflow-based approach results in a unified way to cope with the heterogeneity of the activities involved in uncertainty and sensitivity analyses, and leads to a generic and extensible design.

The resulting workflow builds on five main tasks: i) choosing the input factors and their distribution uncertainties; ii) building the experimental design by factor sampling; iii) managing the model simulation; iv) analyzing the results obtained from the simulation; v) preparing to present and report the results. Thus, all approaches can be considered as a partial or complete combination of the main tasks. Moreover, filters are available and can be added to cope with atypical and complex situations. They are often used to convert or import data in order to connect the main tasks.

Architecture and design

Building on an object-oriented framework, the **mtk** package follows the recommendation for S4 classes and methods available in R (Chambers, 2008). As shown in Figure 2, it comprises three mandatory components: the factor unit, the workflow unit, and the data import and export unit. Each unit is part of a service mission and manages the exchange of data and services with other units via interfaces. Thus, a unit knows other units and communicates with them only through the interfaces. This practice promotes efficient software engineering when multiple teams are involved, and makes the long-term software maintenance easier (Chambers, 2014).

The factor unit

The factor unit manages data and services with regard to the parameters and inputs of the model. It also ensures efficient support to manage the uncertainty behaviors that we know about the model. When running an uncertainty or sensitivity analysis, the first thing to do is to determine what the

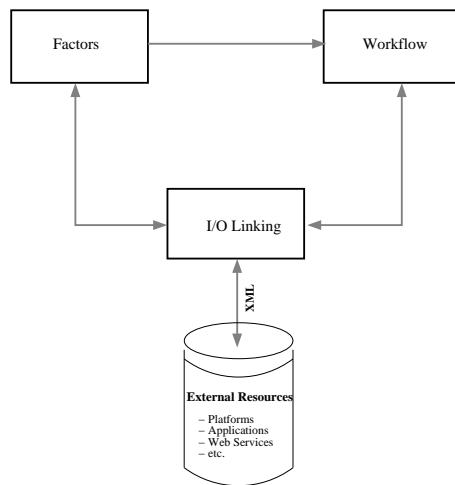


Figure 2: A general view of the system. The system is mainly composed of three components: a unit to manage the *factors* and their uncertainties, a unit to manage the processes and workflow, and a unit to manage the collaboration with external resources.

Classes	Definition	Nature
'mtkFactor'	Entity used to represent a <i>factor</i>	Instantiable
'mtkExpFactors'	Entity used to collect all the <i>factors</i> involved in the experiment	Instantiable
'mtkDomain'	Entity used to define the uncertain domain of a <i>factor</i>	Instantiable
'mtkLevels'	Entity used to define a discrete probability distribution	Instantiable
'mtkValue'	Triplet used to define a typed variable	Virtual
'mtkParameter'	Entity used to define a parameter	Instantiable
'mtkFeature'	Entity used to represent complex relationships among <i>factors</i>	Instantiable
'mtkProcess'	Entity used to manage a process	Virtual
'mtkExpWorkflow'	Entity used to manage the workflow	Instantiable
'mtkExperiment'	Entity to manage a simplified version of workflow	Instantiable
'mtkParser'	Entity used to parse XML files	Instantiable
'mtkReporter'	Entity responsible for advanced data reporting	Virtual
'mtkResults'	Entity used to hold results produced by a process	Virtual
'mtkDesigner'	Entity used to manage <i>design</i> process	Instantiable
'mtkSystemDesigner'	Entity used to manage <i>design</i> method implemented as a system application	Instantiable
'mtkNativeDesigner'	Entity used to manage methods <i>design</i> implemented as an R function	Instantiable
'mtkMorrisDesigner'	Entity used to manage the process implementing the method Morris	Instantiable

Table 1: The principal classes used in the **mtk** package to manage the *factors* and the processes involved in numerical experiments.

parameters and inputs to the model are and, among them, which parameters and inputs exhibit uncertainties. Such kinds of parameters and inputs are referred to as *factors*. Since the uncertainty of the *factor* is restricted within a domain, we usually set it up with a probability distribution function. The factor unit is the component which is responsible for managing the information about the *factors* and their uncertainty domains such as the arguments to the probability distribution function, whether the *factors* are correlated and how they correlate, and so on. An important feature of the **mtk** package is its capability to manage *factors* with complex characteristics. They might be qualitative or quantitative

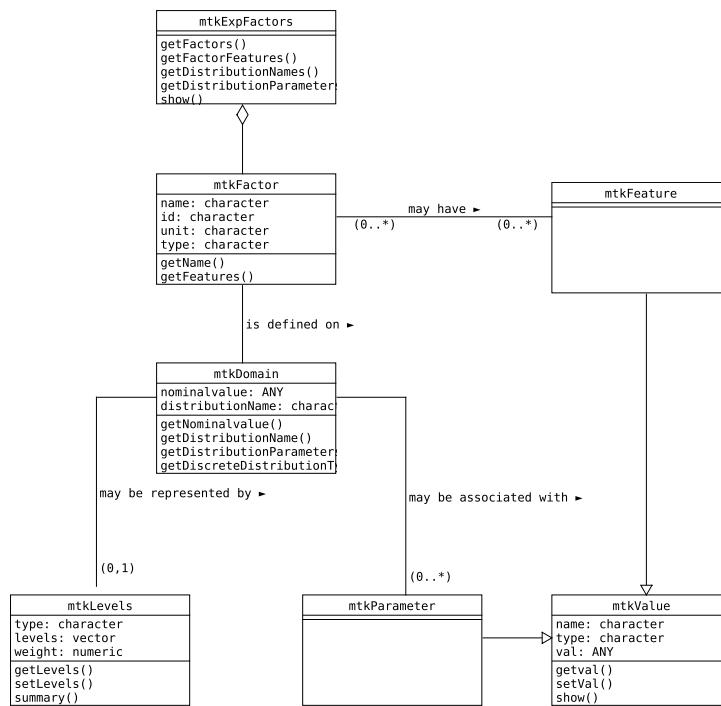


Figure 3: Data model used to manage the *factors* and their uncertainties. The model is represented using the UML notation (Fowler, 2003), and the referred classes are presented in Table 1.

as well as continuous or discrete.

Table 1 defines the classes used in the **mtk** package to manage the *factors* and their uncertainties, and Figure 3 shows the data model in UML notation (Fowler, 2003). It provides a consistent framework for both representing and handling information useful for describing the *factors* and their uncertainty domains precisely. We have focused on describing the relevant data and their relationships and sought to give a consistent data framework which can be considered as general as possible and easily extensible to integrate new methods developed in the future even by researchers not involved in the **mtk** initiative (for more discussion, please refer to Section [Representing the *factors* and their uncertainties in an homogeneous and extensible way](#)).

The workflow unit

This component manages and orchestrates the execution and progress of the processes involved in the numerical experiments. In this task, a process takes much more than calling a more or less sophisticated function within the software package. This is able to analyze information from the context, to define a strategy taking into account the availability of data and services (which might be local or remote, and if locally available, might be implemented as an independent system application, an R function, or an internal element of the **mtk** package), and finally to select the appropriate processing to launch, to formulate the produced results, and to make them available to other components of the system or independent applications outside the system.

In the current version, the **mtk** package supports four types of processes: the parser for XML files, the experimental design, the model simulation, and the computation of sensitivity indices. Each process possesses descriptors to inform about its state and progress: whether the process is ready to run or it is running or it has already run and produced the results that we expected. The workflow manager has the control over the launching and evolution of all the processes involved. Before invoking a process, the workflow manager makes sure that all required resources are available and that they are coherent with the state of the process. After the execution of a process, it checks the consistency of the results and makes them available for other processes.

As shown in Table 1 and Figure 4, processes are organized into a hierarchical structure by inheritance. The common components of the processes are summarized within an abstract class named '**mtkProcess**'. The child classes inherit the components from their parent classes higher in the hierarchical structure. For example, '**mtkDesigner**' is a process which inherits the common components defined within the '**mtkProcess**' and adds new features specific to the experimental design. The process

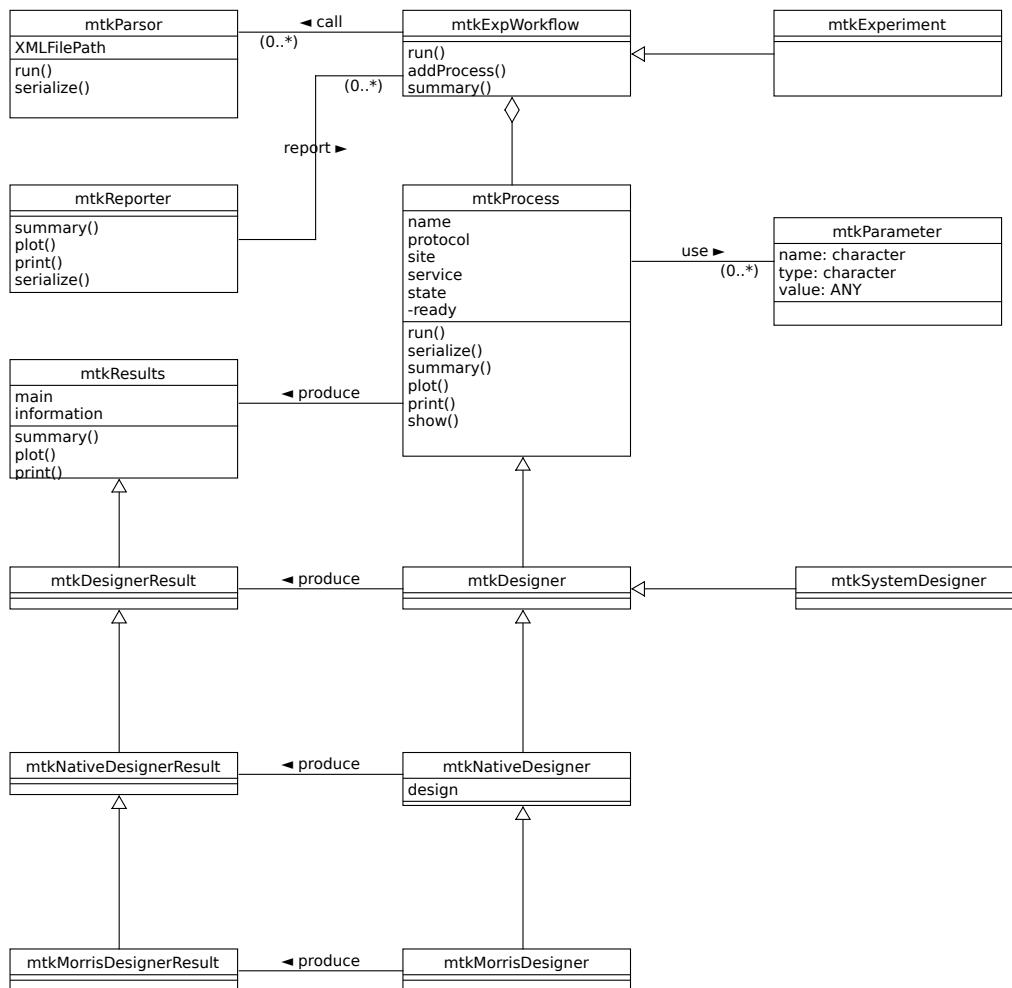


Figure 4: UML data model for organizing the processes and workflow implemented in the **mtk** package. The referred classes are presented in Table 1. To make the schema readable, only the elements related to the Morris method for experimental design are shown. The prefix Native is used to refer to the process implemented as an R function, and the prefix System is used when the process is implemented as an independent system application.

'mtkMorrisDesigner' shares common components with other methods from experimental design and further appends new features specific to the Morris method (Saltelli et al., 2005). From Figure 4, we can also note that each process is associated with a specific class for managing the results produced by the process. In fact, results produced by methods from uncertainty and sensitivity analyses are very different both in terms of contents and in terms of structures. Some methods produce data which could be represented within a data frame, and some others produce results that can be formulated only in the format of a list. Classic approaches require that we must always be concerned with the structure used to hold the data, and develop bespoke functions for each type of the results (Chambers, 2014). The object-oriented approach allows us to be released from such constraints. The classes that hold the data are not only responsible for data storage but also for the services to transform and report the data. For example, the experimental design is no longer treated only as a function to produce data in some specific format, but considered as a process which is an active element not only capable of generating and reporting the data but also capable of checking their consistency and setting up relationships with other components of the system.

The data import and export unit

Based on widely used open standards such as XML, URI (Uniform Resource Identifier), Web services, etc., the data import and export unit allows elements of the package to communicate and to be extended with external resources such as independent platforms or applications for model simulation, Web services that supply data or methods for experimental design and sensitivity analysis, etc. For instance,

the workflow for a sensitivity analysis might be generated from independent external platforms and coded into an XML file; the **mtk** package can import the XML file, set up the workflow, and run it automatically. This can be referred to as one of the methods that the **mtk** package uses to integrate external platforms as a collaborative component.

The package also provides serializing functions allowing to save the internal status of the workflow into an XML file so that independent external applications could collect information about the processes and data involved in the workflow, and further wrap them as an internal component (see Section [Conclusions and perspectives](#)).

Features and functions

Representing the *factors* and their uncertainties in an homogeneous and extensible way

In the **mtk** package, we sought to provide a consistent and easily extensible framework for both representing and handling information useful for describing the *factors* and their uncertainty domains precisely. For example, the concept *feature* was introduced to address the issue about the diversity of the *factors* in their types and relationships. Mathematically speaking, a *feature* is a simple triplet that can be used to make come together a variable name, a data type and a numerical value. The *feature* concept is simple but fundamental to both the scalability of the implemented methods and the extensibility of the package. In fact, *factors* used in uncertainty and sensitivity analyses may have different formats and be defined on various domains. In addition, they may be qualitative or quantitative, continuous or discrete, ordered or unordered, etc. Furthermore, several *factors* may be linked by constraints in space or time, and they may be either correlated or hierarchical as well. Therefore, we need a unified framework to cope with such a variety of *factors*. The concept of a *feature* has accomplished this goal. It allows to set up a universal framework for depicting the *factors* and their relationships and leads to a consistent and unified schema to manage the heterogeneity of the *factors*: an ordered list of *features* associated with a *factor* can be used to describe the spatial and temporal location of a *factor*, and one or more *features* can be applied to a group of *factors* to catch the relationship of any complexity among them, etc.

The *factors* and their uncertainties are represented with the class ‘**mtkFactor**’. This class has four primary attributes and two associative attributes:

name The name of the *factor* used in the workflow.

id The name of the *factor* used in the simulation model if it is different.

type The type of the values associated with the *factor*.

unit A unit of measurement associated with the values of the *factor* if it exists.

domain An object of the class ‘**mtkDomain**’ to describe the uncertainty of the *factor*.

featureList A list of objects from the class ‘**mtkFeature**’ to provide additional information about the underlying *factor* such as spatial or temporal location, relationships among a group of *factors*, etc.

The attributes **name**, **id**, **type** and **unit** are simple strings, and the associative attributes **domain** and **featureList** are objects of the classes ‘**mtkDomain**’ and ‘**mtkFeature**’ respectively.

There are two ways that can be used to define the *factors*: interactively within an R session or by parsing an XML file. The interactive definition of the *factors* within an R session is implemented with the function **make.mtkFactor()**, and the following examples demonstrate this function. The definition of the *factors* from an XML file might be considered as a component of the data import and export unit which will be discussed in the next section.

```
# Load mtk package:
library(mtk)

# Define a continuous factor:
make.mtkFactor('A', distribName = 'unif', distribPara = list(min = 0, max = 1))

# Define a new discrete factor:
make.mtkFactor('D', distribName = 'discrete',
               distribPara = list(type = 'categorical',
                                   levels = c('a', 'b', 'c'), weights = rep(1/3, 3)))
```

The first example shows how to define a *factor* named *A*, whose uncertainty is defined by a continuous uniform distribution over the interval [0, 1]. The second example demonstrates the definition of a categorical *factor* which is named *D*, and takes values from the set {*a*, *b*, *c*} and each with a probability equal to 1/3.

Formalizing the data and services for collaborative work

The language XML has been used to overcome the heterogeneity of data and services exchanged between the **mtk** package and external resources. Indeed, one of the main difficulties that we met in building the **mtk** package was the variety of data and services that need to be managed in the context of uncertainty and sensitivity analyses. Data and services might be local or remote, and implemented in R or other programming languages. They might also have different structures and various formats. Instead of putting emphasis on the data and services themselves, we have chosen to focus on the structure of the data and services that the package exchanges with external resources, and to formalize them according to the XML standards. XML schemas were elaborated and used to tackle the issue of numerical experiments in the open collaborative framework (Richard et al., 2013). By reformalizing the elements involved in the experimental design, the model simulation control, the workflow management and the data reuse, the XML schemas lead to a unified representation of the data and services that an open collaborative framework needs to produce or consume. Thus, managing the heterogeneity of data and services involved in the uncertainty and sensitivity analyses is greatly simplified and highly standardized. From the point of view of programming, it is reduced to the development of a class which is able to parse the XML files following the defined XML schemas.

The XML parsing has been realized with the class ‘**mtkParser**’, and its use is very simple. We just need to specify the path to access the XML file and the workflow into which the extracted information will be directed. The following code shows how to construct a parser from an XML file, which is delivered with the distribution package. Note that the XML file is usually produced by an external platform, and thus the XML parsing mechanism can also be used as a way to integrate the **mtk** package with external applications.

```
# Create a parser to parse the XML file : './WWDM.xml'.
# Note that the XML file is delivered within the package "mtk"
parser <- mtkParser('./WWDM.xml')
```

Organizing the implementation of the methods hierarchically and presenting them with a unified syntax

To collect all the available methods into a unique framework and to present them under a unified syntax, we adopted a workflow-based approach which consists in decomposing the procedures of sensitivity analysis into a series of elementary and generic processes, and organizing them into a hierarchical structure. Each activity involved in sensitivity analysis can be considered either as an elementary process or a combination of elementary processes. Appropriate combination of the processes allows the handling of situations of any complexity.

Taking the experimental design as an example, this is part of the mission services provided by the workflow management unit. The **mtk** package offers a generic and easily extensible implementation of a Web-based open framework, and such an implementation needs to be carefully thought and designed. In fact, the methods used to generate the experimental design might be complex and complicated. Besides the inherent variety of the contexts where the methods were developed, they might be implemented locally or remotely and in different programming languages and according to various protocols. Furthermore, the experimental design might be generated on-line or off-line. To provide the package with an architecture easy to extend and the ability to cope with different situations, a general purpose class ‘**mtkDesigner**’ is derived from the abstract class ‘**mtkProcess**’ so that users can extend the framework to fit to specific circumstances (please see the class organization presented in Figure 4). The ‘**mtkDesigner**’ class inherits the following slots from the class ‘**mtkProcess**’, which enable the Web-based computing:

protocol The protocol used to run the process. It may take on values such as “**mtk**”, “**R**”, “**system**” and “**http**”, where the value “**mtk**” indicates that the process is implemented as an internal element of the **mtk** package, the value “**R**” that the process is implemented as a native R function, the value “**system**” that the process is implemented as an independent application, and the value “**http**” that the process is implemented with Web service technologies.

site The site where the processing is implemented.

service The name of service which realizes the underlying tasks.

To make importing methods implemented locally as an independent R function easier, a class ‘**mtkNativeDesigner**’ derived from ‘**mtkDesigner**’ is provided together with its constructor as follows:

```
mtkNativeDesigner(design = NULL, X = NULL, information = NULL)
```

This class can deal with two scenarios, whereby either the method of experimental design is implemented as an independent function in R or the experimental design was generated off-line. The

first example below shows how to construct an experimental design from a method implemented as an independent R function, and the second shows how to import an experimental design generated off-line.

```
# Set up an experimental design with a method implemented
# by an R function named 'mc04()'
sampler <- mtkNativeDesigner(design = mc04(factors,
                                             distribution, parameters, size = 20))
# Set up an experimental design by importing the design produced
# off-line which is stored as a data.frame named 'plan'.
sampler <- mtkNativeDesigner(X = plan,
                             information = list(method = 'Morris', size = 20))
```

Note that the technical details mentioned before are just intended for importing in live external elements into package **mtk**. When it comes to the elements already integrated in the package, one does not need to care about how the methods are physically implemented (locally or remotely, as an R function or through a Web service, etc.). To use a method, it is only necessary to instantiate an object from the underlying class. For instance, to set up an experimental design with the method Morris wherever it is implemented physically, we just need to instantiate an object of the class '**mtkMorrisDesigner**'.

```
# Set up an experimental design with the method 'Morris' with parameters.
sampler <- mtkMorrisDesigner(listParameters = list(size = 20))
```

Currently, the **mtk** package supports three kinds of elementary processes: *designer*, *evaluator* and *analyser*. Each manages one of the principal activities involved in uncertainty and sensitivity analyses, and is associated respectively with the experimental design, the model simulation and the sensitivity computing.

We should point out again that within the **mtk** package, all processes involved in the sensitivity analysis are managed in the same way just as the designer is managed. For instance, the common properties involved in the model simulation are put into the class '**mtkEvaluator**' which is itself derived from the class '**mtkProcess**'. Also, the models might be implemented locally or remotely, written in R or in another programming language, and the simulation might be produced on-line or off-line. If the model is implemented locally, the specific class proposed is the class '**mtkNativeEvaluator**' with the associated constructor as follows:

```
mtkNativeEvaluator(model = NULL, Y = NULL, information = NULL)
```

This class has the same syntax as the class '**mtkNativeDesigner**', and this is one of the biggest advantages of using the **mtk** package. It provides a homogeneous way and mechanism to manipulate all the methods and functions managed by the package. For example, if we want to simulate the Ishigami model ([Ishigami and Homma, 1990](#)) which describes the dynamics of a non-linear function with three *factors*, it does not take more than to set up a model evaluator (or simulator) with the code as follows:

```
# Simulate the 'Ishigami' model which has no parameter.
simulator <- mtkIshigamiEvaluator()
```

As well, to use the Morris method to compute the sensitivity indices is not harder than to write the following code:

```
# Set up a process to compute the sensitivity indices with the Morris method
analyzer <- mtkMorrisAnalyser(listParameters = list(nboot = 20))
```

Managing efficiently the activities with a workflow-based approach

A workflow is an orchestrated and repeatable sequence of activities that are responsible to transform data and to provide services. The **mtk** package organizes the activities into standardized and elementary processes. Before invoking a process, the workflow ensures that the process is ready to run and the needed data are available and consistent with the state of the process. After running the process, the workflow manages the results, makes them available, and ensures that they can be successfully reused. Indeed, some processes are very time-consuming, and they require enormous computing power to produce results. This is especially true for complex model simulations which may take days or even weeks on a cluster before making the simulated data available. Therefore, it is important to avoid restarting a process if no new data has been produced even if the workflow needs to be restarted to incorporate new elements. For instance, suppose that an experiment was designed with the Monte Carlo method and analyzed with the multiple regression method, and one wishes to analyze the same simulated data with another method. In this case, it would be possible to reuse the experimental

design and the simulated data already obtained. The workflow implemented in the **mtk** package manages these kinds of constraints and enables to maximize the reuse of resources. An example of this approach will be presented in Section A *case study*.

The workflow management has been implemented with the class ‘**mtkExpWorkflow**’, which can be created in two ways: either interactively within an R session or automatically through an XML file.

The interactive method is the most common procedure used by R users, and it consists of four steps: i) defining the *factors* and their uncertainties; ii) specifying the processes involved in the sensitivity analysis; iii) forming a workflow; and iv) running the workflow and reporting the results. The example below presents the construction and execution of a workflow to analyze the Ishigami model with the Basic Monte Carlo method for the experimental design and the regression method for sensitivity computing.

```
# Load the mtk package:
library(mtk)

# Specify the factors and their uncertainty domains:
x1 <- make.mtkFactor(name = 'x1', distribName = 'unif',
                      distribPara = list(min = -pi, max = pi))
x2 <- make.mtkFactor(name = 'x2', distribName = 'unif',
                      distribPara = list(min = -pi, max = pi))
x3 <- make.mtkFactor(name = 'x3', distribName = 'unif',
                      distribPara = list(min = -pi, max = pi))
ishi.factors <- mtkExpFactors(list(x1, x2, x3))

# Specify the processes involved:
designer <- mtkBasicMonteCarloDesigner(listParameters = list(size = 20))
simulator <- mtkIshigamiEvaluator()
analyser <- mtkRegressionAnalyser(listParameters = list(nboot = 20))

# Form the workflow:
experiment <- mtkExpWorkflow(expFactors = ishi.factors,
                               processesVector = c(design = designer,
                                                   evaluate = simulator, analyze = analyser))

# Run the workflow and report the results:
run(experiment)
summary(experiment)
```

The automatic method consists of controlling the workflow through an XML file in which all the information necessary for the definition and execution of the workflow is specified. The XML files can be created manually by users or even more often by external platforms. The latter allows to manage the **mtk** workflow from an external platform and offers a way to carry out uncertainty and sensitivity analyses without having to get out of the modeling or simulation platform. Once the XML file is formed, the **mtk** package takes control over the XML file and provides facilities for information extraction, and workflow initialization and control.

The example below shows how to build a workflow from an XML file. Note that the XML file used here can be found in the supplementary material provided with the distributed package.

```
# Load the mtk package:
library(mtk)

# Create a workflow from the XML file:  './WWDM.xml'
expXML <- mtkExpWorkflow(xmlFilePath = './WWDM.xml')

# Run the workflow and report the results
run(expXML)
summary(expXML)
```

Extending the package with new or existing methods

In order to encourage researchers to publish their methods through the **mtk** framework, we provide facilities to easily import available methods directly into the system. The **mtk** package comes with three tools: **mtk.designerAddons()**, **mtk.evaluatorAddons()**, and **mtk.analyserAddons()**.

The tool **mtk.designerAddons()** is a function that allows users to turn new or existing methods for experimental designs developed as R functions into classes compliant with the **mtk** package.

This function has the following prototype:

```
mtk.designerAddons(where, library, authors, name, main, summary = NULL,
                   print = NULL, plot = NULL)
```

where `NULL` or a string to denote the file containing the R function to convert.

library NULL or a string to denote the name of a package containing the R function to convert if it is provided via a package.

authors NULL or the copyright information about the authors of the R function.

name A string to name the method when used with the **mtk** package.

main The name of the R function implementing the method.

summary NULL or a special version of the summary method provided in the file *where* or in the package *library*.

plot NULL or a special version of the plot method provided in the file *where* or in the package *library*.

print NULL or a special version of the print method provided in the file *where* or in the package *library*.

No constraints are imposed on the function to convert except for the format of its inputs and outputs. The R function implementing the method must have at least the three arguments: factors, distribNames, and distribParameters. The argument factors takes as values either a number or a list of names for enumerating the *factors* to analyze. The arguments distribNames and distribParameters are both lists, whose elements are used to specify the uncertainty domains of the *factors*.

The output produced by the function must be formatted as a named list with two elements: main and information. The element main is a data.frame containing the produced experimental design and the element information is a named list whose elements are used to provide optional information about the method used.

If the summary(), print() and plot() methods provided within the package **mtk** are not concise enough to describe the underlying experimental design, or the method developers wish to report it in a specific way, they can replace these methods by new ones.

The example below shows how to use the function mtk.designerAddons() to convert an existing method into **mtk** compliant classes so that the method can be seamlessly used with the package. In order to demonstrate the potential of the package, we have chosen to import an existing method implemented in an independent package: the method "Morris" of the package **sensitivity** (Pujol et al., 2015). The file morris_sampler.R contains the program codes used to wrap the original function so that the inputs and outputs meet the requirement of the tool mtk.designerAddons(). In this example, the wrapped function is renamed sampler.morris(). In order to better outline the produced experimental design, a new method of the function plot() for 'morris' objects has been provided via the function plot.morris().

```
# Load the mtk package:
library(mtk)

# Convert the file 'morris_sampler.R' to a mtk compliant class 'mtkMorrisDesigner':
mtk.designerAddons(where = 'morris_sampler.R',
                     authors = 'G. Pujol, B. Ioos, and A. Janon',
                     name = 'Morris', main = 'sampler.morris', plot = 'plot.morris')

# Integrate the new class into the mtk package
source('mtkMorrisDesigner.R')
```

Here, the mtk.designerAddons() tool generates a file named mtkMorrisDesigner.R which can be integrated directly into the **mtk** package via the R command source().

The other two tools mtk.evaluatorAddons() and mtk.analyserAddons() operate in the same way as mtk.designerAddons() does. They can be used respectively to integrate simulation models and to integrate methods for computing the sensitivity indices. An example of using the tool mtk.evaluatorAddons() can be found in the next section.

A case study

In this section, we present an example of a decision support model analyzed with the **mtk** package. The model used in Munier-Jolain et al. (2002) is a dynamic model simulating the effect of weeds (meadow foxtail) on the yield of a crop of wheat as a function of different agricultural practices, including soil preparation, weeding and crop varieties. The flow is simulated at a yearly time step. Five state variables are used, and their dynamics are modeled with a system of non-linear first order difference equations.

The model builds both on input variables describing the agricultural practices and on parameters describing the effect of the agricultural practices on the state variables of the model. The input variables are supposed to be fixed, but some parameters are uncertain. We will use the **mtk** package: i) to analyze the effect of the uncertainty of the parameters on the wheat yield (the state variable Y), and ii)

<i>State Variable</i>	Definition	Initial Value
S	Number of seeds of foxtail by m^2 found in the cultivated plot.	68 000
d	Number of foxtail plants by m^2 found at the beginning of the season.	400
$SSBa$	Number of seeds of foxtail by m^2 found on the surface of the soil after tillage.	3 350
$DCSa$	Number of seeds of foxtail by m^2 found under the surface of the soil after tillage.	280
Y	Yield of wheat on the plot (ton per m^2)	—

Table 2: The state variables used in the Weed model and the initial values of the state variables characterizing the population of foxtail at $t = 0$.

to determine the sensitivity of the state and output variable Y to the uncertainties contained in the different parameters.

The model takes into account three types of input: i) the initial values of the state variables characterizing the population of foxtail at $t = 0$ (see Table 2), ii) the year by year agricultural practices (tillage, weeding, cultivated crop varieties), and iii) 16 parameters of the model (they are supposed to be fixed, but contain some uncertainties).

Agricultural practices applied each year to the crop are described with the help of three binary variables: *Soil*, *Herb*, and *Crop*. If the soil is tilled, we set *Soil* = 1, otherwise *Soil* = 0. Similarly, we set *Herb* = 1 if herbicide is applied, *Herb* = 0 otherwise, and *Crop* = 1 if the cultivated plant is a variety of winter wheat, *Crop* = 0 otherwise. In this paper, we explore a simplified model where only winter wheat is supposed to be cultivated (*Crop* = 1) and the tillage is always realized every other year. Also, only two scenarios of the weeding treatment are explored: i) systematic treatment each year, and ii) systematic treatment except the third year.

Since the *factors* (parameters with uncertainty) are supposed to be fixed, their uncertainties can be represented with common probability distribution functions. Table 3 shows the domains of uncertainties associated with such parameters.

The computing code of the model is enclosed in the supplementary material provided with the package and in Faivre et al. (2013). Note that to integrate the model into the **mtk** package, we do not need to reprogram the model, but just wrap the main function `WEED.simule()` in the file `WeedModel_v2.R`, say, so that its inputs and outputs conform with the requirement of the function `mtk.evaluatorAddons()` presented in Section Extending the package with new or existing methods.

Once the model is wrapped, we append it to the **mtk** package so that it can be seamlessly used with the **mtk** package.

```
# Load the package mtk:
library(mtk)

# Transform the model into a mtk compliant class:
mtk.evaluatorAddons(where = 'WeedModel_v2.R', authors = 'D.Makowski(2012)',
                     name = 'Weed', main = 'WEED.simule')

# Load the mtk compliant class generated before into the mtk package:
source('mtkWeedEvaluator.R')
```

Uncertainty analysis

Recall that realizing a numerical experiment with the **mtk** package is composed of four steps: i) choose the *factors* and specify their uncertainties; ii) set up the processes involved in the numerical experiment; iii) form a workflow; and iv) run the workflow and report the results.

First, the uncertain domains associated with the *factors* are defined with function `make.mtkFactor()`. Sixteen *factors* are considered, and each is assumed to follow a uniform distribution whose range is fixed according to Table 3. The code below shows how we defined the uncertain domains of the *factors* within R.

```
# "table3.data" is a file referring to the Table 3 defined in the text.
```

Factors	Definition	Min Value	Max Value
<i>mu</i>	Annual decline rate	0.76	0.92
<i>v</i>	Proportion of non aborted seeds	0.54	0.66
<i>phi</i>	Loss of the fresh seeds	0.50	0.61
<i>beta.1</i>	Proportion of the foxtail seeds found under the surface after tillage	0.86	1.05
<i>beta.0</i>	Proportion of the foxtail seeds found under the surface without tillage	0.18	0.22
<i>chsi.1</i>	Proportion of the foxtail seeds found on the surface after tillage	0.27	0.33
<i>chsi.0</i>	Proportion of the foxtail seeds found on the surface without tillage	0.045	0.055
<i>delta.new</i>	Germination rate for fresh seeds	0.14	0.17
<i>delta.old</i>	Germination rate for old seeds	0.27	0.33
<i>mh</i>	Efficiency of the herbicide	0.88	1.08
<i>mc</i>	Mortality rate caused by cold weather	0	1
<i>Smax.1</i>	Number of the seeds harvested per plant (winter variety)	400	490
<i>Smax.0</i>	Number of the seeds harvested per plant (spring variety)	266	326
<i>Ymax</i>	Potential yield t/ha	7.2	8.8
<i>Rmax</i>	Parameter used to calculate the loss	0.0018	0.0022
<i>Gamma</i>	Another parameter used to calculate the loss	0.0045	0.0055

Table 3: The *factors* and their domains of uncertainties used in the Weed model. The domains of uncertainties are all modeled with a uniform probability distribution whose range is fixed from *Min Value* to *Max Value*.

```
table3 <- read.table("table3.data", header = TRUE)
facteurs <- list()
for(i in 1:16){
  facteur.i <- as.character(table3$Factors)[i]
  facteurs[facteur.i] <- make.mtkFactor(name = facteur.i,
    distribName = "unif",
    distribPara = list(min = table3$MinValue[i],
      max = table3$MaxValue[i]))
}
weedFactors <- mtkExpFactors(facteurs)
```

Here, the Basic Monte Carlo method is used for the experimental design, and 1000 samples are generated. The code below shows the underlying procedure:

```
plan <- mtkBasicMonteCarloDesigner(listParameters = list(size = 1000))
```

Two instantiations of the model are evaluated, and they correspond to the cases that weeding treatment is applied every year and that the weeding treatment is not applied only for the third year, respectively. Note that the agricultural practices are encapsulated into the argument *decision*: *decision* = 1 represents the first scenario and *decision* = 2 represents the second.

```
weed.case1 <- mtkWeedEvaluator(listParameters = list(decision = 1, outvar = 3))
weed.case2 <- mtkWeedEvaluator(listParameters = list(decision = 2, outvar = 3))
```

Consequently, two workflows are built, and they will be used to manage the analyses of the two models defined above.

```
exp1 <- mtkExpWorkflow(expFactors = weedFactors,
  processesVector = c(design = plan, evaluate = weed.case1))
exp2 <- mtkExpWorkflow(expFactors = weedFactors,
  processesVector = c(design = plan, evaluate = weed.case2))

set.seed(2) # to fix the seed of the random generator for exp1
run(exp1)
```

```
set.seed(2) # to fix the seed of the random generator for exp2
run(exp2)
```

Sometimes, users may choose not to use the reporting tools provided with the **mtk** package to present their results. The function `extractData()` allows them to fetch the data managed by the **mtk** package as independent data structures supported by the R computing platform and to manipulate them freely. Hereinafter, the simulated results (1000 simulations per year for 10 years) are stored in two variables: Y_1 and Y_2 . Notice that we are only interested in the yields of the third year (`outvar = 3`) since they are the only data which can reflect the effect of weeding treatment.

```
Y1 <- unlist(extractData(exp1, name = 'evaluate'))
Y2 <- unlist(extractData(exp2, name = 'evaluate'))

dev.new()
par(mfrow = c(2,2))
hist(Y1, main = '',
     xlab = 'Yield with herbicide systematically applied (t/ha)')
hist(Y2, main = '',
     xlab = 'Yield without applying herbicide for the 3rd year (t/ha)')
hist(Y1-Y2, main = ' ', xlab = 'Loss in yield (t/ha)')
hist(100*(Y1-Y2)/Y1, main = ' ', xlab = 'Relative loss in yield (%)')

summary(Y1-Y2)
```

The results are illustrated in Figure 5. It shows that the average loss is 0.33 t/ha , the median loss is 0.25 t/ha , and the 1st and 3rd quartiles of the distribution are equal to 0.12 and 0.48 t/ha respectively. The uncertainty analysis shows that the yield loss due to non-application of the weeding treatment has a one-in-two chance of exceeding 0.25 t/ha , and has a one-in-four chance of being less than 0.12 t/ha and a one-in-four chance of exceeding 0.48 t/ha . We can claim that the loss in yield is moderate, even taking into account the uncertainties of the *factors*.

Sensitivity analysis

The uncertainty analysis described above allows the estimation of the uncertainties about the yield losses, but it gives no information about where the uncertainties come from and which *factors* have the most important impact on them. In this section, we will discuss how to use the **mtk** package to calculate the sensitivity indices for the *factors* and how to identify the most influential *factors* according to their sensitivity. Two methods will be presented: Morris (Saltelli et al., 2005) and PLMM (Polynomial Linear metamodel; Faivre, 2013). These examples demonstrate how easy it is to use the **mtk** package to compare very different methods.

The R code below shows the sensitivity analysis with the Morris method. We can note the efficiency and effortlessness of the **mtk** package to fulfill such a procedure: We are neither concerned about where the methods are implemented (locally or remotely) nor worried about how data are organized within the processes.

```
# Specify the processes and form the workflows:
morris.sampler <- mtkMorrisDesigner(listParameters = list(r = 500, type = 'oat',
                                                       levels = 4 , grid.jump = 2, scale = TRUE))

weed.treated <- mtkWeedEvaluator(listParameters = list(decision = 1,
                                                       outvar = 3))
weed.no.treated <- mtkWeedEvaluator(listParameters = list(decision = 2,
                                                       outvar = 3))

morris.analyser <- mtkMorrisAnalyser()

exp.treated <- mtkExpWorkflow(expFactors = weedFactors,
                               processesVector = c(design = morris.sampler,
                                                   evaluate = weed.treated, analyze = morris.analyser))

exp.no.treated <- mtkExpWorkflow(expFactors = weedFactors,
                               processesVector = c(design = morris.sampler,
                                                   evaluate = weed.no.treated, analyze = morris.analyser))

# Run the workflows:
set.seed(2)
```

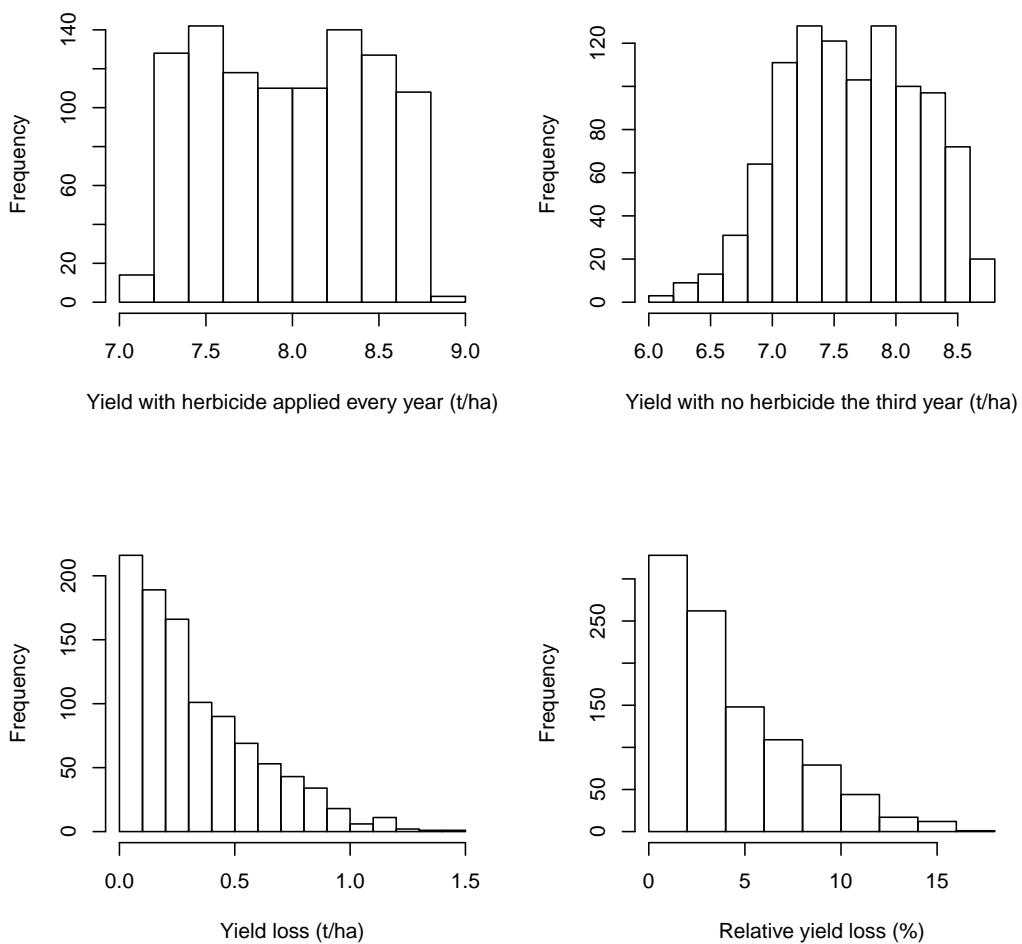


Figure 5: Histogram presenting the yields simulated with the Weed model (1000 simulations per year for 10 years). 1) Yields observed with applying herbicide every year (top-left) and without applying herbicide for the 3rd year (top-right); 2) Loss in yield due to not applying herbicide for the 3rd year expressed in t/ha (bottom-left) and in percentages (bottom-right).

```

run(exp.treated)
set.seed(2)
run(exp.no.treated)
# Report the results:
plot(getProcess(exp.treated, name = 'analyze'))
  title("With herbicide every year")
plot(getProcess(exp.no.treated, name = 'analyze'))
  title("With no herbicide the 3rd year")

```

Note that we make use one more time of the class ‘mtkWeedEvaluator’ to manage the model simulation, and that only the yields of the third year (*outvar* = 3) are explored.

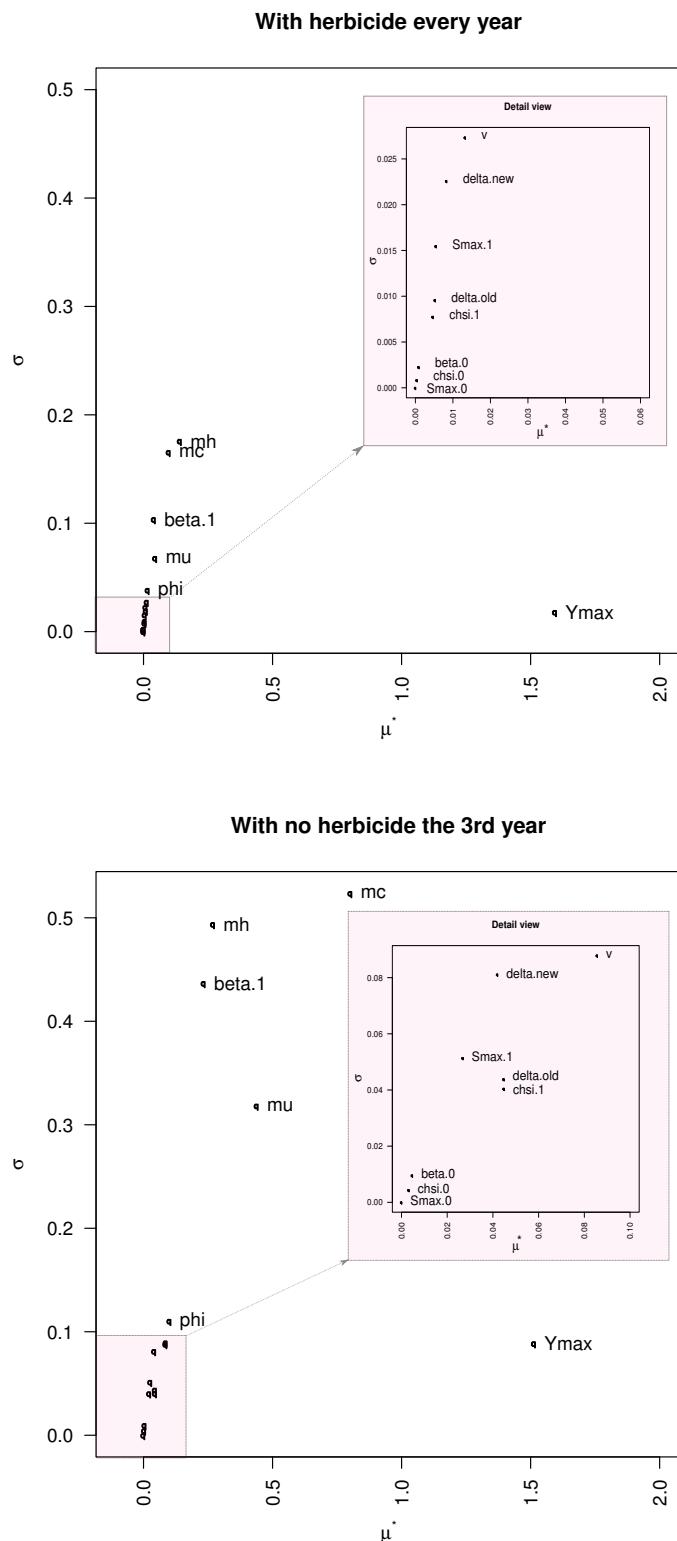


Figure 6: Sensitivity analysis results shown in graphical form for the outputs Y with respect to the 16 parameters with uncertainty, calculated with the Morris method for the Weed models: 1) with herbicide applied every year (top) and 2) without applying herbicide for the 3rd year (bottom). In the Morris method, the index μ^* (on the x -axis) is used to detect input factors with an important overall influence on the output, and the index σ (on the y -axis) is used to detect factors involved in interaction with other factors or whose effect is non-linear.

Figure 6 shows the results of the sensitivity analysis obtained with the Morris method for the two scenarios of agricultural practices. Remember that in the method Morris, the index μ^* is used to detect the factors with important overall influence on the output and the index σ is used to detect the factors involved in interaction with other factors or whose effect is non-linear.

We note that most factors have sensitivity indices μ^* and σ close to 0 either with or without weeding treatment. This shows that such factors have little effect on the yield performance.

On the contrary, when weeding is applied, the factor Y_{max} (the maximum yield potentially obtained with the underlying soil and the underlying cultivated wheat variety) has a sensitivity index μ^* larger than 1. This means that the factor Y_{max} has significant influence on the yield performance and that the main part of the variability of the yield performance might be explained by the uncertainties contained in the factor Y_{max} . Furthermore, some factors exhibit σ values slightly different from 0 meaning that their effect may be non-linear or interacting with others factors but with low consequences on the output.

Such a conclusion seems natural and easy to understand. In fact, when weeding treatment is applied, almost all foxtail is eliminated; their influences are wiped out and only the potential yield parameter Y_{max} becomes decisive for the yield performance.

On the other hand, when no weeding is applied the 3rd year, the index μ^* associated with the factor Y_{max} is not the only one to move away from 0, but also the factors mc , mu , mh , and $beta.1$. Meanwhile, the indices σ associated with the factors mc , mu , and $beta.1$ are all increased significantly. This means that when no weeding is applied, Y_{max} is no longer the only factor having significant impacts on the yield and the factors mc , mu and $beta.1$ also imply effects on the yield performance either in a non-linear way or in interaction with other factors.

To assess the relevance of the results, we have analyzed the same models with other methods. The methods RandLHS (Latin Hypercube Sampling; Carnell 2012) and PLMM are used respectively for the experimental design and the sensitivity analysis. The code below demonstrates the procedure. Note that we reuse the two simulators of the Weed models `weed.treated` and `weed.no.treated` previously defined.

```
# Specify the processes and form the workflows:
lhs.sampler <- mtkRandLHSDesigner(listParameters = list(size = 1000))
plmm.analyser <- mtkPLMMAalyser(listParameters = list(degree.pol = 2))

exp.plmm.treated <- mtkExpWorkflow(expFactors = weedFactors,
                                      processesVector = c(design = lhs.sampler,
                                                           evaluate = weed.treated, analyze = plmm.analyser))

exp.plmm.no.treated <- mtkExpWorkflow(expFactors = weedFactors,
                                         processesVector = c(design = lhs.sampler,
                                                               evaluate = weed.no.treated, analyze = plmm.analyser))

# Run the workflows:
set.seed(2)
run(exp.plmm.treated)
set.seed(2)
run(exp.plmm.no.treated)

# Report the results of the workflows:
plot(exp.plmm.treated, legend.loc = 'topleft')
plot(exp.plmm.no.treated, legend.loc = 'topleft')
```

Figure 7 shows the results of the sensitivity analysis obtained for the PLMM method with a polynomial metamodel of degree 2. The results uphold the conclusions obtained with the Morris method. The analyses were performed first with a regression modeling of the output on all the cross products of polynomials of factors with degree 2, and then extended by proceeding to a stepwise selection of explanatory variables. Figure 7 highlights that most of the main effects of the factors mc , mu , mh and Y_{max} are linear or polynomial and that interactions between factors are mainly between Y_{max} and $beta.1$. We can also notice that the R^2 , the percentage of variance explained by our metamodel, is close to 1 (zone marked with a dashed line).

Conclusions and perspectives

There is a rapidly growing trend to utilize uncertainty and sensitivity analyses for quantifying the uncertainties contained in a model and further assessing their impacts on the behaviors of the model. Numerous methods and theories emanating from different fields have been put forward, but the

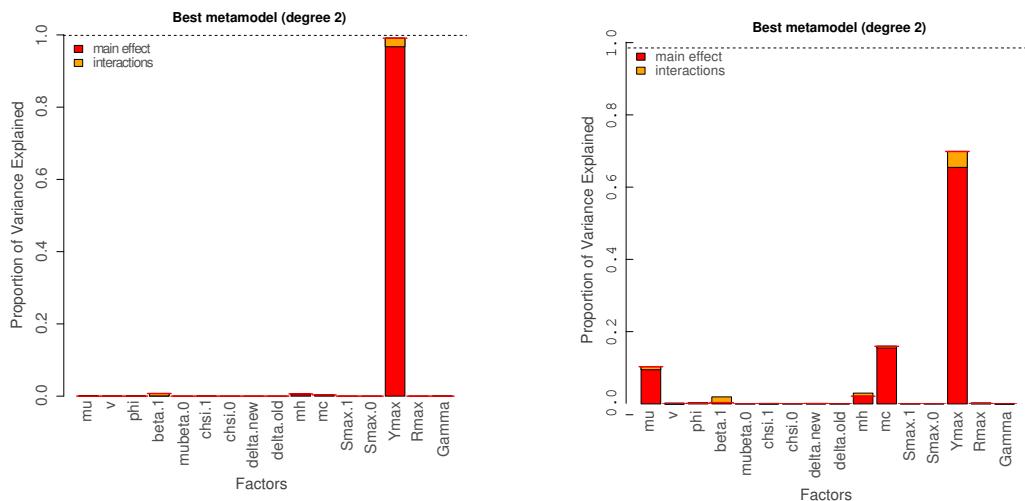


Figure 7: Sensitivity indices of the output Y with respect to the 16 parameters with uncertainty, calculated by the PLMM method with a polynomial metamodel of degree 2 for the Weed models: 1) with herbicide applied every year (left) and 2) without applying herbicide for the 3rd year (right). The y -axis represents the R^2 value corresponding to the fraction of the total variance of the output explained by the *factors*, and the dashed line shows the fraction of the total variance of the output explained by the metamodel.

issue related to software development is still lagging behind. The tools proposed are usually bespoke, context-specific and self-contained, and suffer from lack of generality and extensibility. Herein, we have developed a general-purpose framework to compile the available methods into a unique software platform which is able to provide the practitioners from different disciplines with a systematic and easy way to compare and find the best method for uncertainty discovery and sensitivity analysis.

The **mtk** package should be the first generic R platform available for uncertainty and sensitivity analyses, which allows us to collect all the methods actually available into a unique system, and present them according to the same semantics and with the same syntax. This makes the methods easy to use and their comparison effective since methods can be run with exactly the same data and in the same environment.

Building on an object-oriented framework and exploring the XML standards, the **mtk** package places its focus on the interoperability, and provides facilities for interplaying with other applications and sharing data and knowledge in a seamless way.

It is fully open-source and easy to extend. It allows users to add their own methods and models to the package easily. The power of a workflow-based approach allows researchers to organize their computing effectively and to extend the investigation in a quick manner. By decomposing the workflow into generic and elementary tasks, complex processing can be set up by combining the elementary tasks and be managed easily with the package. Moreover, the Web-based technologies and computing implemented in the package make its extension even more flexible since users have access to different ways to realize the extension: using the inheritance mechanism provided with the object-oriented framework, directly integrating native R functions, building the extension as an independent application, etc. Note also that the **mtk** package always presents the methods and models in the same way, wherever they are implemented (locally or remotely) and no matter how they are implemented (as an internal element of the package or an independent external application, etc.).

In spite of the advanced features, the **mtk** package is still work in progress. Future plans include implementing support for High Performance Computing to improve the efficiency for time-consuming processes (Leclaire and Reuillon, 2014). Further, a new version of the serialization function is also planned so that external platforms can easily integrate the **mtk** package as an internal component. Actually, state and data of the workflow managed by the package **mtk** can be exported into XML files, and used by external applications or platforms. Fine-tuning with real world examples is necessary so that an external application can use such information to wrap the **mtk** package as its internal component. When it comes to the issue of efficient large data management, we are studying the possibility to use the package **ff** for memory-efficient storage (Adler et al., 2014).

Acknowledgement

This research was partially supported by the French methodology research network for numerical experiments (MEXICO) (<http://reseau-mexico.fr>). We thank all the members of this partnership, who provided insight and expertise that greatly assisted the research.

Bibliography

- B. Adams, L. Bauman, W. Bohnhoff, K. Dalbey, J. Eddy, M. Ebeida, M. S. Eldred, P. D. Hough, K. Hu, J. Jakeman, L. P. Swiler, and D. M. Vigil. Dakota: A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Version 5.4 User's Manual. Technical Report, Sandia National Labs., Albuquerque, NM and Livermore, CA, 2013. [p206]
- D. Adler, C. Gläser, O. Nenadic, J. Oehlschlägel, and W. Zucchini. *ff: Memory-Efficient Storage of Large Data on Disk and Fast Access Functions*, 2014. URL <https://CRAN.R-project.org/package=ff>. R package version 2.2-13. [p223]
- K. Alden, M. Read, J. Timmis, P. S. Andrews, H. Veiga-Fernandes, and M. Coles. Spartan: A comprehensive tool for understanding uncertainty in simulations of biological systems. *PLoS Computational Biology*, 9(2), 2013. [p207]
- K. Alden, M. Read, P. Andrews, J. Timmis, H. Veiga-Fernandes, and M. Coles. *spartan: Simulation Parameter Analysis R Toolkit Application*, 2015. URL <https://CRAN.R-project.org/package=spartan>. R package version 2.2.1. [p207]
- M. Baudin, A. Dutfoy, B. Iooss, and A.-L. Popelin. Open TURNS: An industrial software for uncertainty quantification in simulation. *ArXiv e-prints*, 2015. [p207]
- J. Brock, C. Caupp, and H. Runke. *Evaluation of Water Quality Using DSSAM III Under Various Conditions of Nutrient Loadings from Municipal Wastewater and Agricultural Sources, Truckee River, Nevada: Comparison of Simulated Water Quality Conditions with Truckee River Water Quality Standards from McCarran to Pyramid Lake. Executive Summary*. Bureau of Water Quality Planning, Nevada Division of Environmental Protection, 1992. [p206]
- J. Cariboni, D. Gatelli, R. Liska, and A. Saltelli. The role of sensitivity analysis in ecological modelling. *Ecological Modelling*, 203(1–2):167–182, 2007. Special Issue on Ecological Informatics: Biologically-Inspired Machine Learning. [p206]
- R. Carnell. *Ihs: Latin Hypercube Samples*, 2012. URL <https://CRAN.R-project.org/package=lhs>. R package version 0.10. [p222]
- J. M. Chambers. *Software for Data Analysis: Programming with R*. Springer, 2008. [p208]
- J. M. Chambers. Object-oriented programming, functional programming and R. *Statistical Science*, 29(2):167–180, 2014. [p208, 211]
- B. Ciuffo, A. Miola, V. Punzo, and S. Sala. *Dealing with Uncertainty in Sustainability Assessment: Report on the Application of Different Sensitivity Analysis Techniques to Fieldspecific Simulation Models*. Publications Office, Joint Research Centre, Institute for Environment and Sustainability, Luxembourg, 2012. [p207]
- E. de Rocquigny, N. Devictor, and S. Tarantola. *Uncertainty in Industrial Practice: A Guide to Quantitative Uncertainty Management*. John Wiley & Sons, 2008. [p207]
- D. Dupuy, C. Helbert, and J. Franco. DiceDesign and DiceEval: Two R packages for design and analysis of computer experiments. *Journal of Statistical Software*, 65(11):1–38, 2015. URL <http://www.jstatsoft.org/v65/i11/>. [p207]
- R. Faivre. *Exploration par construction de métamodèles*. Savoir Faire. Quae, Versailles, France, 2013. Edited by Faivre R., Iooss B., Mahévas S., Makowski D., Monod H., Analyse de sensibilité et exploration de modèles. Applications aux modèles environnementaux. [p207, 219]
- R. Faivre, D. Makowski, J. Wang, H. Richard, and H. Monod. *Exploration numérique d'un modèle agronomique avec le package 'mtk'*. Savoir Faire. Quae, Versailles, France, 2013. Edited by Faivre R., Iooss B., Mahévas S., Makowski D., Monod H., Analyse de sensibilité et exploration de modèles. Applications aux modèles environnementaux. [p217]

- M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. [p210]
- J. Helton, J. Johnson, C. Sallaberry, and C. Storlie. Survey of sampling-based methods for uncertainty and sensitivity analysis. *Reliability Engineering and System Safety*, 91(10–11):1175–1209, 2006. [p206]
- G. M. Hornberger and R. C. Spear. An approach to the preliminary analysis of environmental systems. *Journal of Environmental Management*, 12:7–18, 1981. [p206]
- T. Ishigami and T. Homma. An importance quantification technique in uncertainty analysis for computer models. In *International Symposium on Uncertainty Modelling and Analysis (ISUMA'90)*, Dec. 3–6, University of Maryland, 1990. [p214]
- Joint Research Centre. SimLab. Software, 2006. URL <http://simlab.jrc.cec.eu.int>. [p207]
- M. Lamboni, H. Monod, and C. Bidot. *multisensi: Multivariate Sensitivity Analysis*, 2015. URL <https://CRAN.R-project.org/package=multisensi>. R package version 1.0-8. [p207]
- M. Leclaire and R. Reuillon. OpenMole: OPEN MOdeL Experiment. Open-Source Software, 2014. URL <http://www.openmole.org>. [p223]
- P. Lynch. The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, 227(7):3431–3444, Mar. 2008. [p206]
- S. Marino, I. B. Hogue, C. J. Ray, and D. E. Kirschner. A methodology for performing global uncertainty and sensitivity analysis in systems biology. *Journal of Theoretical Biology*, 254(1):178–196, 2008. [p206, 207]
- H. Monod, A. Bouvier, and A. Kobilinsky. *planor: Generation of Regular Factorial Designs*, 2015. URL <https://CRAN.R-project.org/package=planor>. R package version 0.2-4. [p207]
- N. Munier-Jolain, B. Chauvel, and J. Gasquez. Long-term modelling of weed control strategies: Analysis of threshold-based options for weed species with contrasted competitive abilities. *Weed Research*, 42:107–122, 2002. [p216]
- J. Papaix, S. Touzeau, H. Monod, and C. Lannou. Can epidemic control be achieved by altering landscape connectivity in agricultural systems? *Ecological Modelling*, 284:35–47, 2014. [p206]
- L. Pronzato and W. Müller. Design of computer experiments: Space filling and beyond. *Statistics and Computing*, 22(3):681–701, 2012. [p207]
- G. Pujol, B. Iooss, and A. Janon. *The sensitivity Package*, 2015. URL <https://CRAN.R-project.org/package=sensitivity>. R package version 1.11.1. [p206, 207, 216]
- M. Read, P. S. Andrews, J. Timmis, and V. Kumar. Techniques for grounding agent-based simulations in the real domain: A case study in experimental autoimmune encephalomyelitis. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):67–86, 2012. [p207]
- H. Richard, H. Monod, J. Wang, J. Couteau, N. Dumoulin, B. Poussin, J. Soulié, and E. Ramat. *La boîte à outils Mexico: un environnement générique pour piloter l'exploration numérique de modèles*. Savoir Faire. Quae, Versailles, France, 2013. Edited by Faivre R., Iooss B., Mahévas S., Makowski D., Monod H., Analyse de sensibilité et exploration de modèles. Applications aux modèles environnementaux. [p213]
- Y. Richet, D. Ginsbourger, O. Roustant, and Y. Deville. A grid computing environment for design and analysis of computer experiments. In *The R User Conference*, July 20–23 2010, Gaithersburg, Maryland, USA, 2010. [p207]
- Y. Richet et al. The Promethee project: A software environment for performing parametric computer simulations in dependability engineering, 2009. URL <http://promethee.irsn.org>. [p207]
- J. S. Risbey, S. Lewandowsky, C. Langlais, D. P. Monselesan, T. J. O’Kane, and N. Oreskes. Well-estimated global surface warming in climate projections selected for ENSO phase. *Nature Climate Change*, 4(9):835–840, Sept. 2014. [p206]
- A. Salinger, R. Pawlowski, J. Shadid, and B. van Bloemen Waanders. Computational analysis and optimization of a chemical vapor deposition reactor with large-scale computing. *Industrial and Engineering Chemistry Research*, 43(16):4612–4623, 2004. [p206]
- A. Saltelli, K. Chan, and E. M. Scott. *Sensitivity Analysis*. Wiley, 2000. [p207]

A. Saltelli, M. Ratto, S. Tarantola, and C. F. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley and Sons, 2005. [p206, 207, 211, 219]

V. G. Weirs, J. R. Kamm, L. P. Swiler, S. Tarantola, M. Ratto, B. M. Adams, W. J. Rider, and M. S. Eldred. Sensitivity analysis techniques applied to a system of hyperbolic conservation laws. *Reliability Engineering and System Safety*, 107:157–170, 2012. [p206]

Juhui Wang

MaIAGE, UR1404 – INRA, Domain de Vilvert, 78352 Jouy en Josas France

Juhui.Wang@jouy.inra.fr

Robert Faivre

MIAT, UR875 – INRA, Auzeville, 31326 Castanet-Tolosan France

Robert.Faivre@toulouse.inra.fr

Hervé Richard

BioSp – INRA, Domaine Saint-Paul, Site Agroparc, 84914 Avignon France

Herve.Richard@avignon.inra.fr

Hervé Monod

MaIAGE, UR1404 – INRA, Domain de Vilvert, 78352 Jouy en Josas France

Herve.Monod@jouy.inra.fr

treeClust: An R Package for Tree-Based Clustering Dissimilarities

by Samuel E. Buttrey and Lyn R. Whitaker

Abstract This paper describes **treeClust**, an R package that produces dissimilarities useful for clustering. These dissimilarities arise from a set of classification or regression trees, one with each variable in the data acting in turn as the response, and all others as predictors. This use of trees produces dissimilarities that are insensitive to scaling, benefit from automatic variable selection, and appear to perform well. The software allows a number of options to be set, affecting the set of objects returned in the call; the user can also specify a clustering algorithm and, optionally, return only the clustering vector. The package can also generate a numeric data set whose inter-point distances relate to the **treeClust** ones; such a numeric data set can be much smaller than the vector of inter-point dissimilarities, a useful feature in big data sets.

Introduction

Clustering is the act of dividing observations into groups. One critical component of clustering is the definition of distance or dissimilarity between two entities (two observations, or two clusters, or a cluster and an observation). We assume the usual case of a rectangular array of data in which p attributes are measured for each of n individuals. The common choice of the usual Euclidean distance needs to be extended when some of the attributes are categorical. A good dissimilarity should be able to incorporate categorical variables, impose appropriate attribute weights (in particular, be insensitive to linear scaling of the attributes), and handle missing values and outliers gracefully. [Buttrey and Whitaker \(2015\)](#) describe an approach which uses a set of classification and regression trees that collectively determine dissimilarities that seems to perform well in both numeric and mixed data, particularly in the presence of noise.

The next section covers the idea behind the tree-based clustering, while the following one (“[The treeClust package](#)”) describes the software we have developed for this purpose. The software is implemented as an R package, available under the name **treeClust** at the CRAN repository. The section on software also gives some of the attributes of the procedure, like its insensitivity to missing values, and of the software, like the ability to parallelize many of the computations. The “Dissimilarity-preserving data” section describes a feature of the software that has not yet been explored – the ability to generate a new numeric data set whose inter-point distances are similar to our tree-based dissimilarities. This allows a user to handle larger data sets, since in most cases the new data set will have many fewer entries than the vector of all pairwise dissimilarities. In the final section we provide an example using a data set from the UC Irvine repository ([Bache and Lichman, 2013](#)); this data set, and a script with the commands used in the example, are also available with the paper or from the authors.

Clustering with trees

The idea of tree-based clustering stems from this premise: objects that are similar tend to land in the same leaves of classification or regression trees. In a clustering problem there is no response variable, so we construct a tree for each variable in turn, using it as the response and all others are potential predictors. Trees are pruned to the “optimal” size (that is, the size for which the cross-validated deviance is smallest). Trees pruned all the way back to the root are discarded.

The surviving set of trees are used to define four different, related measures of dissimilarity. In the simplest, the dissimilarity between two observations i and j , say, $d_1(i, j)$, is given by the proportion of trees in which those two observations land in different leaves. (Our dissimilarities are not strictly distances, since we can have two distinct observations whose dissimilarity is zero.) That is, the dissimilarity between observations associated with a particular tree t is $d_1^t(i, j) = 0$ or 1, depending on whether i and j do, or do not, fall into the same leaf under tree t . Then over a set of T trees, $d_1(i, j) = \sum_t d_1^t(i, j) / T$.

A second dissimilarity, d_2 , generalizes the values of 0 and 1 used in d_1 . We start by defining a measure of tree “quality,” q , given by the proportion of deviance “explained” by the tree. The contribution to the deviance for each leaf is the usual sum of squares from the leaf mean for a numeric response, and the multinomial log-likelihood for a categorical one. Then, for tree t , we have $q_t = [(Deviance\ at\ root - Deviance\ in\ leaves) / Deviance\ in\ root]$. A “good” tree should have a larger q

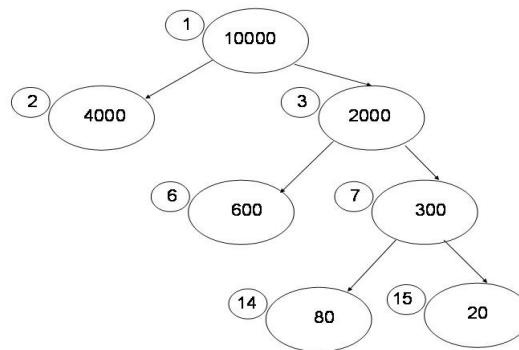


Figure 1: Example tree, showing node numbers (small circles) and deviances, from Buttrey and Whitaker (2015).

and a higher weight in the dissimilarity. So we define $d_2^t(i, j)$ to be 0 if the two observations fall in the same leaf, and otherwise $q_t / \max_k(q_k)$, where the maximum, of course, is taken over the whole set of trees. As before, $d_2(i, j) = \sum_t d_2^t(i, j) / T$.

In both of the d_1 and d_2 measures, each tree contributes only two values, zero and a constant that does not depend on which two different leaves a pair of observations fell into. The d_3 dissimilarity attempts to address that. Consider the tree in figure 1, taken from Buttrey and Whitaker (2015). In this picture, small numbers in circles label the leaves, while large numbers in ovals give the node-specific deviance values. A pair of observations falling into leaves 14 and 15 on this tree seem closer together than a pair falling into 2 and 15, for example. We measure the distance between two leaves by computing the increase in deviance we would observe, if we were to minimally prune the tree so that the two observations were in the same leaf, compared to the overall decrease in deviance for the full tree. For example, the picture in figure 1 shows a starting deviance (node 1) of 10000 and a final deviance (summing across leaves) of 4700. Now consider two observations landing in leaves 14 and 15. Those two observations would end up in the same leaf if we were to prune leaves 14 and 15 back to node 7. The resulting tree's deviance would be increased to 4900, so we define the dissimilarity between those two leaves – and between the two relevant observations – by $(4900 - 4700) / (10000 - 4700) = 0.038$. Observations in leaves 6 and 15 would contribute distance $(6000 - 4700) / (10000 - 4700) = 0.25$, while observations in leaves 2 and 14, say, would have distance 1 for this tree.

The final dissimilarity measure, d_4 , combines the d_2 -style tree-specific weights with the d_3 dissimilarity, simply by multiplying each d_3 dissimilarity by the tree's $q_t / \max_k q_k$. The best choice of dissimilarity measure, and algorithm with which to use it, varies from data set to data set, but our experience suggests that in general, distances d_2 and d_4 paired with the `pam()` clustering algorithm from the `cluster` (Maechler et al., 2015) package are good choices for data sets with many noise variables.

Other tree-based approaches

Other researchers have used trees to construct dissimilarity measures. One is based on random forests, a technique introduced in (Breiman, 2001) for regression and classification. The user manual (Breiman and Cutler, 2003) describes how this approach might be used to produce a proximity measure for unlabelled data. The original data is augmented with data generated at random with independent columns whose marginal distributions match those of the original. So the original data preserves the dependence structure among the columns, whereas the generated data's columns are independent. The original data is labelled with 1's, and the generated data with 0's; then the random forest is generated with those labels as the response. Proximity is measured by counting the number of times two observations fall in the same trees, dividing this number by twice the number of trees, and setting each observation's proximity with itself as 1. In our experiments we construct dissimilarity by subtracting proximity from 1. The random forest dissimilarity is examined in (Shi and Horvath, 2006), and we use it below to compare to our new approach.

Other clustering approaches based on trees includes the work of Fisher (1987), in which trees are used to categorize items in an approach called “conceptual clustering.” It appears to be intended for categorical predictors in a natural hierarchy. Ooi (2002) uses trees as a way to partition the feature space on the way to density estimation, and Smyth et al. (2006) use multivariate regression trees with principal component and factor scores. Both of these approaches are intended for numeric data.

We thank an anonymous referee for bringing to our attention the similarity between our approach

and the extensive literature on building phylogenetic trees. A thread in this literature is to start with a distance matrix and end with a phylogenetic or evolutionary tree, whereas we use the trees to build the distance. [Felsenstein \(2003\)](#) serves as a guide to developing phylogenetic trees, while [O'Meara \(2015\)](#) gives the CRAN Task View summary of R packages for phylogenetics. In the phylogenetic literature, methods for measuring distances and growing trees are tailored to the specific application. For example defining distances might involve informed choices of morphological or behavioral features or on aligning genetic material to capture dissimilarities at a molecular level. Our contribution, on the other hand, is to use trees to automate construction of dissimilarity matrices. Our context is more general in that we do not know *a priori* which variables are more important than others, which are noise, how to scale numeric variables, or how to collapse and combine categorical ones. Methods using application-specific knowledge might be expected to yield superior results, but it would be interesting to see if the ideas behind tree distances can be adapted for use in growing phylogenetic trees.

The **treeClust** package

Dependencies

This paper describes version 1.1.1 of the **treeClust** package. This package relies on the **rpart** package ([Therneau et al., 2015](#)) to build the necessary classification and regression trees. This package is “recommended” in R, meaning that users who download binary distributions of R will have this package automatically. **treeClust** also requires the **cluster** package for some distance computations; this, too, is “recommended.” Finally, the **treeClust** package allows some of the computations to be performed in parallel, using R’s **parallel** package, which is one of the so-called “base” packages that can be expected to be in every distribution of R. See “Parallel processing” for more notes on parallelization.

Functions

In this section we describe the functions that perform most of the work in **treeClust**, with particular focus on **treeClust()**, which is the function users would normally call, and on **treeClust.control()**, which provides a mechanism to supply options to **treeClust()**.

The **treeClust()** function

The **treeClust()** function takes several arguments. First among these, of course, is the data set, in the form of an R “`data.frame`” object. This rectangular object will have one row per observation and one column per attribute; those attributes can be categorical (including binary) or numeric. (For the moment there is no explicit support for ordered categoricals.) The user may also supply a choice of tree-based dissimilarity measure, indicated by an integer from 1 to 4. By default, the algorithm builds one tree for each column, but the `col.range` argument specifies that only a subset of trees is to be built. This might be useful when using parallel processing in a distributed computing architecture. Our current support for parallelization uses multiple nodes on a single machine; see “Parallel processing” below.

If the user plans to perform the clustering, rather than just compute the dissimilarities, he or she may specify the algorithm to be used in the clustering as argument `final.algorithm`. Choices include “`kmeans`” from base R, and “`pam`”, “`agnes`” and “`clara`” from the **cluster** package. (The first of these is an analog to “`kmeans`” using medoids, which are centrally located observations; the next two are implementations of hierarchical clustering, respectively agglomerative and divisive.) If any of these is specified, the number of clusters k must also be specified. By default, the entire output from the clustering algorithm is preserved, but the user can ask for only the actual vector of cluster membership. This allows users easy access to the most important part of the clustering and obviates the need to store all of the intermediate objects, like the trees, that go into the clustering. This election is made through the `control` argument to **treeClust()**, which is described next.

The **treeClust.control()** function

The `control` argument allows the user to modify some of the parameters to the algorithm, and particularly to determine which results should be returned. The value passed in this argument is normally the output of a function named **treeClust.control()**, which follows the convention of R

functions like `glm.control()` and `rpart.control()`. That is, it accepts a set of input names and values, and returns a list of control values in which the user's choices supersede default values.

The primary motivation for the control is to produce objects of manageable size. The user may elect to keep the entire set of trees, but there may be dozens or even hundreds of these, so by default the value of the `treeClust.control()`'s `return.trees` member is FALSE. Similarly, the dissimilarity object for a data set with n observations has $n(n - 1)/2$ entries, so by default `return.dists` is FALSE. The matrix of leaf membership is approximately the same size as the original data – it has one column for each tree, but not every column's tree is preserved. The `return.mat` entry defaults to TRUE, since we expect our users will often want this object. The `return.newdata` entry describes whether the user is asking for `newdata`, which is the numeric data set of observations in which the inter-point dissimilarities are preserved. This is described in the section on "Dissimilarity-preserving data" below. By default, this, too, is FALSE. It is also through `treeClust.control()` that the user may specify `cluster.only = TRUE`. In this case, the return value from the function is simply the numeric n -vector of cluster membership.

The `serule` argument describes how the pruning is to be done. By default, trees are pruned to the size for which the cross-validated deviance is minimized. When the `serule` argument is supplied, trees are pruned to the smallest size for which the cross-validated deviance is no bigger than the minimum deviance plus `serule` standard errors. So, for example, when `serule = 1`, the one-SE rule of Breiman et al. (1984) is implemented.

An argument named `DevRatThreshold` implements an experimental approach to detecting and removing redundant variables. By redundant, we mean variables that are very closely associated with other variables in the data set, perhaps because one was constructed from the other through transformation or binning, or perhaps because two variables describing the same measurement were acquired from two different sources. Let the sum of deviances across a tree's leaves be d_L , and the deviance at the root be d_R . Then for each tree we can measure the proportional reduction in deviance, $(d_R - d_L)/d_R$. A tree whose deviance reduction exceeds the value of `DevRatThreshold` is presumed to have arisen because of redundant variables. We exclude the predictor used at the root split from the set of permitted predictor variables for that tree and re-fit until the tree's deviance reduction does not exceed the threshold, or until there are no remaining predictors. (We thank our student Yoav Shaham for developing this idea.)

The final choice of the `tree.control()` is the `parallelnodes` argument, which allows trees to be grown in parallel. When this number is greater than 1, a set of nodes on the local machine is created for use in growing trees. (This set of nodes is usually referred to as a "cluster," but we want to reserve that word for the output of clustering.) Finally, additional arguments can be passed down to the underlying clustering routine, if one was specified, using R's ... mechanism.

Return value

If `cluster.only` has not been set to TRUE, the result of a call to `treeClust()` will be an object of class "treeClust" and contain at least five entries. Two of these are a copy of the call that created the object and the `d_num` argument supplied by the user. A "treeClust" object also contains a matrix named `tbl` with one row for each of the trees that was kept, and two columns. These are `DevRat`, which gives the tree's strength (q above), and `Size`, which gives the number of leaves in that tree. Finally, every "treeClust" object has a `final.algorithm` element, which may have the value "None", and a `final.clust` element, which may be NULL.

Dissimilarity functions

There are two functions that are needed to compute inter-point dissimilarities. The first of these, `tcdist()` (for "tree clust distance") is called by `treeClust()`. However, it can also be called from the command line if the user supplies the result of an earlier call. Unless the user has asked for `cluster.only = TRUE`, the return value of a call to `treeClust()` is an object of class "treeClust", and an object of this sort can be passed to `tcdist()` in order to compute the dissimilarity matrix. Note that certain elements will need to be present in the "treeClust" object, the specifics depending on the choice of dissimilarity measure. For example, most "treeClust" objects have the leaf membership matrix, which is required in order to compute dissimilarities d_1 and d_2 . For d_3 and d_4 the set of trees is required.

The `tcdist()` function makes use of another function, `d3.dist()`, when computing d_3 or d_4 distances. This function establishes all the pairwise distances among leaves in a particular tree, and then computes the pairwise dissimilarities among all observations by extracting the relevant leaf-wise distances. An unweighted sum of those dissimilarities across trees produces the final d_3 , and a weighted one produces d_4 . The `d3.dist()` function's argument `return.pd` defaults to FALSE; when set

to TRUE, the function returns the matrix of pairwise leaf distances. Those leaf distances are used by the `tcnnewdata()` function in order to produce a matrix called `newdata`, which has the property that the inter-point distances in this matrix mirror the `treeClust()` dissimilarities. Users might create this `newdata` matrix in large samples, where the vector of all pairwise distances is too large to be readily handled. This feature is described under “Dissimilarity-preserving data” below.

Support functions

A number of support functions are also necessary. We describe them here. Two of these arise from quirks in the way `rpart()` works. In a regression tree, an “`rpart`” object’s `frame` element contains a column named `dev` which holds the deviance associated with each node in the tree. However, in a classification tree, this column holds the number of misclassifications. So our function `rp.deviance()` computes the deviance at each node. We declined to name this function `deviance.rpart`, because we felt that users would expect it to operate analogously to the `deviance` method, `deviance.tree()`, for objects of class “`tree`” in the `tree` package (Ripley, 2015). However, that function produces a single number for the whole tree, whereas our function produces a vector of deviances, one for each node.

A second unexpected feature of `rpart()` is that there is no predict method which will name the leaf into which an observation falls. The `where` element gives this information for observations used to fit the tree, but we need it also for those that were omitted from the fitting because they were missing the value of the response (see section “Handling missing values” below). Our function `rpart.predict.leaves()` produces the leaf number (actually, the row of the “`rpart`” object’s `frame` element) into which each observation falls. For the idea of how to do this, we are grateful to Yuji (2015).

The support function `treeClust.rpart()` is not intended to be called directly by users. This function holds the code for the building of a single tree, together with the pruning and, if necessary, the re-building when the user has elected to exclude redundant variables (see `DevRatThreshold` above). It is useful to encapsulate this code in a stand-alone function because it allows that function to be called once for each column either serially or in parallel as desired.

The `make.leaf.paths()` function makes it easy to find the common ancestors of two leaves. Recall that under the usual leaf-naming convention, the children of leaf m have numbers $2m$ and $2m + 1$. The `make.leaf.paths()` function takes an integer argument `up.to` giving the largest row number; the result is a matrix with `up.to` rows, with row i giving the ancestors of i . For example, the ancestors of 61 are 1, 3, 7, 15, 30 and 61 itself. The number of columns in the output is j when $2^j \leq \text{up.to} < 2^{j+1}$, and unneeded entries are set to zero.

Finally we provide a function `cramer()` that computes the value of Cramér’s V for a two-way table. This helps users evaluate the quality of their clustering solutions.

Print, plot and summary methods

The objects returned from `treeClust()` can be large, so we supply a `print()` method by which they can be displayed. For the moment, the set of trees, if present, is ignored, and the `print()` method shows a couple of facts about the call, and then displays the table of deviances, with one row for each tree that was retained. Even this output can be too large to digest easily, and subsequent releases may try to produce a more parsimonious and informative representation. The `summary()` method produces the output from the `print()` method, only it excludes the table of deviances, so its output is always small.

We also supply a plot method for objects of class “`treeClust`”. This function produces a plot showing the values of q (that is, the `DevRat` column in the `tbl1` entry) on the vertical axis, scaled so that the maximum value is 1. The horizontal axis simply counts the trees starting at one. Each point on the graph is represented by a digit (or sometimes two) showing the size of the tree. For example, figure 2 shows the plot generated from a call to `plot()` passing a “`treeClust`” object generated from the splice data from the UCI Data Repository (Bache and Lichman, 2013) (see also the example below). The data has 60 columns (horizontal axis); this run of the function generated 59 trees (the “1” at $x = 32$ shows that the tree for the 32nd variable was dropped). The “best” tree is number 30, and it had three leaves (the digit “3” seen at coordinates (30, 1)).

Handling missing values

The `treeClust()` methodology uses the attributes of `rpart()` to handle missing values simply and efficiently. There are two ways a missing value can impact a tree (as part of a response or as a predictor)

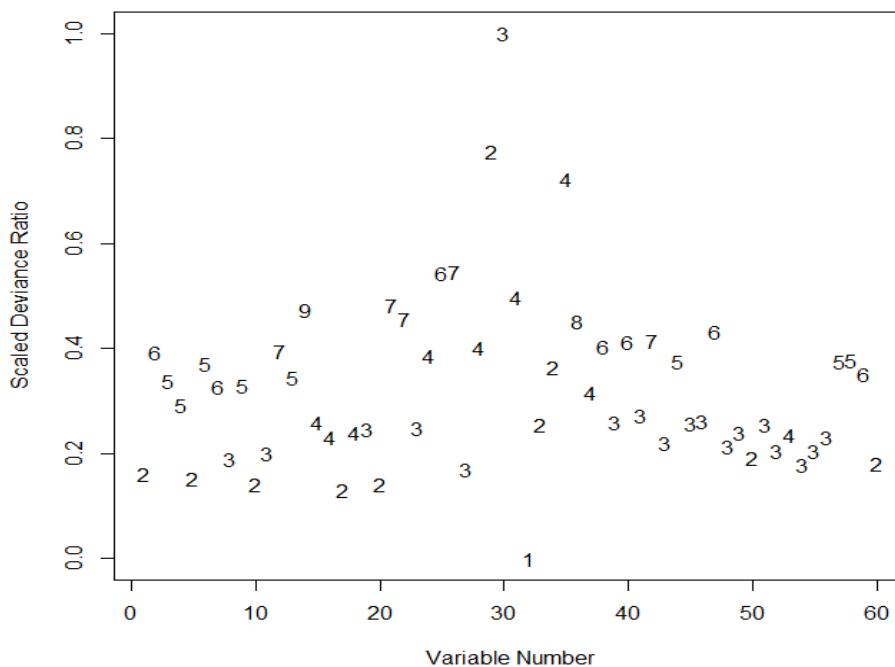


Figure 2: Example of plotting a “treeClust” object. This object contained 59 trees, the first of which had two leaves (digit “2” at (1, 0.16)).

and two times the impact can take place (when the tree is being built, and when observations are being assigned to leaves).

When a value is missing in a predictor, the tree-building routine simply ignores that entry, adjusting its splitting criterion to account for sample size. This is a reasonable and automatic approach when missing values are rare. At prediction time, an observation that is missing a value at a split is adjudicated using a surrogate split (which is an “extra” split, highly concordant with the best split, built for this purpose), or a default rule if no surrogate split is available. When there are large numbers of missing values, however, this approach may be inadequate and we are considering alternatives.

When an observation is missing a value of the response, that row is omitted in the building of the tree, but we then use the usual prediction routines to deduce the leaf into which that observation should fall, based on its predictors.

Parallel processing

The computations required by `treeClust()` can be substantial for large data sets with large numbers of columns, or with categorical variables with large numbers of levels. The current version of the package allows the user to require parallelization explicitly, by setting the `parallelNodes` argument to `treeClust.control()` to be greater than one. In that case, a set of nodes of that size is created using the `makePSOCKcluster()` function from R’s `parallel` package, and the `clusterApplyLB()` function from the same package invoked. The parallel nodes then execute the `treeClust.rpart()` tree-building function once for each variable. The time savings is not substantial for experiments on the splice data, experiments which we describe below, because even in serial mode the computations require only 30 seconds or so to build the trees. However, on a dataset with many more columns the savings can be significant. For example, on the Internet Ads data (Bache and Lichman, 2013), which contains 1,559 columns, building the trees required about 57 minutes in serial on our machine, and only about 8 minutes when a set of 16 nodes was used in parallel.

Big data and efficiency considerations

Not only can our approach be computationally expensive, but, if the user elects to preserve the set of trees and also the inter-point dissimilarities, the resulting objects can be quite large. For example, the object built using the splice data in the example in the example below is around 60 MB in size.

Of this, about 40 MB is devoted to the vector of dissimilarities and almost all of the rest to the set of trees. For larger data sets the resulting “treeClust” objects can be very much larger. The d_4 method, in particular, requires that all the trees be kept, at least long enough to compute the dissimilarities, since the necessary weightings are not known until the final tree is produced. In contrast the d_3 dissimilarities can be produced using only one tree at a time. Both of these two dissimilarities require more computation and storage than the d_1 and d_2 ones.

Some clustering techniques require the vector of dissimilarities be computed ahead of time when the data contains categorical variables, and this large vector is produced by `treeClust()` in the normal course of events. When the sample size is large, a more computationally efficient approach is to produce a new data set which has the attribute that its inter-point dissimilarities are the same as, or similar to, the `treeClust()` distances. That new data set can then serve as the input to a clustering routine like, say, k -means, which does not require that all the inter-point distances be computed and stored. The next section describes how the `treeClust()` function implements this idea.

Dissimilarity-preserving data

In this section we describe one more output from the `treeClust()` function that we believe will be useful to researchers. As we noted above, the act of computing and storing all $n(n - 1)$ pairwise distances is an expensive one. The well-known k -means algorithm stays away from this requirement, but is implemented in R only for numeric data sets. Researchers have taken various approaches to modifying the algorithm to accept categorical data, for example, [Ahmad and Dey \(2007\)](#). Our approach is to produce a data set in which the inter-point distances are similar to the ones computed by the `treeClust()` function, but arrayed as a numeric data set suitable for the R implementation of k -means or for `clara()`, the large-sample version of `pam()`. We call such a data set `newdata`.

As background, we remind the reader of the so-called Gower dissimilarity ([Gower, 1971](#)). Given two vectors of observations, say x_i and x_j , the Gower dissimilarity between those two vectors associated with a numeric column m is $g_m(i, j) = |x_{im} - x_{jm}| / \text{range } x_m$, where the denominator is the range of all the values in that column. That is, the absolute difference is scaled to fall in $[0, 1]$. Then, in a data set with all numeric columns, the overall Gower distance $g(i, j)$ is a weighted sum, $g(i, j) = \sum_m w_m g_m(i, j) / \sum_m w_m$, where the w_m are taken to be identically 1, and therefore the Gower dissimilarity between any two observations is also in $[0, 1]$. There is a straightforward extension to handle missing values we will ignore here.

The construction of `newdata` is straightforward for the d_1 and d_2 distances. When a tree has, say, l leaves, we construct a set of l 0-1 variables, where 1 indicates leaf membership. The resulting data set has Manhattan (sum of absolute values) distances that are exact multiples of the Gower dissimilarities we would have constructed from the matrix of leaf membership. (The Manhattan ones are larger by a factor of $(2 \times \text{number of trees})$). However, the `newdata` data set can be much smaller than the set of pairwise dissimilarities. In the splice data, the sample size $n = 3,190$, so the set of pairwise dissimilarities has a little more than 5,000,000 entries. The corresponding `newdata` object has 3,190 rows and 279 columns, so it requires under 900,000 entries. As sample sizes get larger the tractability of the `newdata` approach grows.

For d_2 , we replace the values of 1 that indicate leaf membership with the weights $q_t / \max_k(q_k)$ described earlier, so that a tree t with l leaves contributes l columns, and every entry in any of those l columns is either a 0 or the value $q_t / \max_k(q_k)$. In this case, as with d_1 , the Manhattan distance between any two rows is a multiple of the Gower dissimilarity that we would have computed with the leaf membership matrix, setting the weights $w_m = q_m / \max_k q_k$.

We note that, for d_1 and d_2 , the `newdata` items preserve Gower distances, not Euclidean ones. In the d_1 case, for example, the Euclidean distances will be the square roots of the Manhattan ones, and the square root of a constant multiple of the Gower ones. Applying the k -means algorithm to a `newdata` object will apply that Euclidean-based algorithm to these data points that have been constructed with the Gower metric. We speculate that this combination will produce good results, but research continues in this area.

The d_3 and d_4 dissimilarities require some more thought. With these dissimilarities, each tree of l leaves gives rise to a set of $l(l - 1)/2$ distances between pairs of leaves. We construct the $l \times l$ symmetric matrix whose i, j -th entry is the distance between leaves i and j . Then we use multidimensional scaling, as implemented in the built-in R function `cmdscale()`, to convert these distances into a $l \times (l - 1)$ matrix of coordinates, such that the Euclidean distance between rows i and j in the matrix is exactly the value in the (i, j) -th entry of the distance matrix. Each observation that fell into leaf i of the tree is assigned that vector of $l - 1$ coordinates. In the case of the d_4 dissimilarity, each element in the distance matrix for tree t is multiplied by the scaling value $q_t / \max_k q_k$.

The use of multidimensional scaling is convenient, but it produces a slight complication. When

d_3 or d_4 are used to construct a newdata object, each tree contributes a set of coordinates such that, separately, Euclidean distances among those points match the d_3 or d_4 dissimilarities for that tree. However, across all coordinates the newdata distances are not generally the same as the d_3 or d_4 ones. In fact the d_3 distances are Manhattan-type sums of Euclidean distances. The properties of this hybrid-type distance have yet to be determined.

An example

In this section we give an example of clustering with `treeClust()`. This code is also included in the ‘`treeClustExamples.R`’ script that is included with the paper. We use the `splice` data mentioned above (and this data set is also included with the paper.) This data starts with 3,190 rows and 63 columns, but after removing the class label and two useless columns, we are left with a data set named `splice` that is $3,190 \times 60$. The class label is stored in an item named `splice.class`. (A shorter version of the data set and class vector, from which 16 rows with ambiguous values have been removed, is also constructed in the script.) At the end of the section we compare the performance of the `treeClust()` approach with the random forest dissimilarity approach.

Our goal in this example is to cluster the data, without using the class label, to see whether the clusters correspond to classes. We select distance d_4 and algorithm “`pam`”, and elect to use six clusters for this example. Then (after we seed the random seed) the “`treeClust`” object is produced with a command like the one below. Here we are keeping the trees, as well as the dissimilarities, for later use:

```
set.seed(965)
splice.tc <- treeClust(splice, d.num = 4,
  control = treeClust.control(return.trees = TRUE, return.dists = TRUE),
  final.algorithm = "pam", k = 6)
```

This operation takes about 90 seconds on a powerful desktop running Windows; quite a lot of that time is used in the “`pam`” portion (the call with no final algorithm requires about 30 seconds). The resulting object can be plotted using `plot()`, and a short summary printed using `summary()`:

```
plot(splice.tc); summary(splice.tc)
```

The `splice.tc` object is quite large, since it contains both the set of trees used to construct the distance and the set of dissimilarities. Had the call to `treeClust.control()` included `cluster.only = TRUE`, the function would have computed the clustering internally and returned only the vector of cluster membership. As it is, since a `final.algorithm` was specified, `splice.tc` contains an object called `final.clust` which is the output from the clustering algorithm – in this case, `pam()`. Therefore the final cluster membership can be found as a vector named `splice.tc$final.clust$clustering`.

We might evaluate how well the clustering represents the underlying class structure by building the two-way table of cluster membership versus class membership, as shown here. The outer pair of parentheses ensures both that the assignment is performed and also that its result be displayed.

```
(tc.tbl <- table(splice.tc$final.clust$clustering, splice.class))
# with result as seen here
splice.class
  EI  IE  N,
1 275  67 53
2 246   0 246
3 235 170 72
4   6 529 198
5   3  1 597
6   2  1 489
```

Cluster six (bottom row) is almost entirely made up of items of class “N.”. Others are more mixed, but the `treeClust()` dissimilarity compares favorably in this example to some of its competitors ([Buttrey and Whitaker, 2015](#)). For example, we compute the random forest dissimilarities in this way:

```
require(randomForest)
set.seed(965)
splice.rfd <- as.dist(1 - randomForest(~ ., data = splice,
  proximity = TRUE)$proximity)
splice.pam.6 <- pam(splice.rfd, k = 6)
(rf.tbl <- table(splice.pam.6$cluster, splice.class))
```

That produces this two way-table:

```

splice.class
    EI   IE   N,
1   92   65  192
2   19     3    7
3  518  559 1035
4   59   57  216
5   75   69  195
6     4   15   10

```

Clearly, in this example, the `treeClust()` dissimilarity has out-performed the random forest one. The `cramer()` function computes the value of V for these two-way tables, as shown here:

```
round(c(cramer(tc.tbl), cramer(rf.tbl)), 3)
[1] 0.679 0.113
```

If we prefer the d_3 dissimilarity, it can be computed with another call to `treeClust()`, or, in this case, directly from the `splice.tc` object, since the latter has all the necessary trees stored in it. The d_3 dissimilarity would be produced by this command:

```
splice.tc.d3 <- tcdist(splice.tc, d.num = 3)
```

As a final example, we can compute the `newdata` object whose inter-point distances reflect the d_4 distances with a call to `tcnewdist()`. This object can then be used as the input to, for example, `kmeans()`.

```
splice.d4.newdata <- tcnewdata(splice.tc, d.num = 4)
```

In this example the `splice.d4.newdata` object is about 1/7 the size of the vector of d_4 dissimilarities.

We note that the results of a call to `treeClust()` are random, because random numbers are used in the cross-validation process to prime trees. Therefore users might find slightly different results than those shown above.

Conclusion

This paper introduces the R package `treeClust`, which produces inter-point dissimilarities based on classification and regression trees. These dissimilarities can then be used in clustering. Four dissimilarities are available; in the simplest, the dissimilarity between two points is proportional to the number of trees in which they fall in different leaves. Exploiting the properties of trees, the dissimilarities are resistant to missing values and outliers and unaffected by linear scaling. They also benefit from automatic variable selection and are resistant, in some sense, to non-linear monotonic transformations of the variables.

For large data sets, the trees can be built in parallel to save time. Furthermore the package can generate numeric data sets in which the inter-point distances are related to the ones computed from the trees. This last attribute, while still experimental, may hold promise for the clustering of large data sets.

Bibliography

- A. Ahmad and L. Dey. A k-mean clustering algorithm for mixed numeric and categorical data. *Data and Knowledge Engineering*, 63:503–527, 2007. [p233]
- K. Bache and M. Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>. [p227, 231, 232]
- L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. [p228]
- L. Breiman and A. Cutler. *Manual—Setting Up, Using, and Understanding Random Forests v4.0*, 2003. URL https://www.stat.berkeley.edu/~breiman/Using_random_forests_v4.0.pdf. [p228]
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. [p230]
- S. E. Buttrey and L. R. Whitaker. A scale-independent, noise-resistant dissimilarity for tree-based clustering of mixed data. In submission, Feb. 2015. [p227, 228, 234]

- J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, MA, 2003. [p229]
- D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987. [p228]
- J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971. [p233]
- M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, and K. Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2015. URL <https://CRAN.R-project.org/package=cluster>. R package version 2.0.3. [p134, 228]
- B. O'Meara. CRAN task view: Phylogenetics, especially comparative methods, 2015. URL <https://CRAN.R-project.org/view=Phylogenetics>. [p229]
- H. Ooi. Density visualization and mode hunting using trees. *Journal of Computational and Graphical Statistics*, 11(2):328–347, 2002. [p228]
- B. Ripley. *tree: Classification and regression trees*, 2015. URL <https://CRAN.R-project.org/package=tree>. R package version 1.0-36. [p231]
- T. Shi and S. Horvath. Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics*, 15(1):118–138, 2006. [p228]
- C. Smyth, D. Coomans, and Y. Everingham. Clustering noisy data in a reduced dimension space via multivariate regression trees. *Pattern Recognition*, 39:424–431, 2006. [p228]
- T. Therneau, B. Atkinson, and B. Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2015. URL <https://CRAN.R-project.org/package=rpart>. R package version 4.1-10. [p229]
- Yuji. Search for corresponding node in a regression tree using rpart, 2015. URL <http://stackoverflow.com/questions/5102754/search-for-corresponding-node-in-a-regression-tree-using-rpart>. [p231]

Samuel E. Buttrey
Department of Operations Research
Naval Postgraduate School
USA
buttrey@nps.edu

Lyn R. Whitaker
Department of Operations Research
Naval Postgraduate School
USA
whitaker@nps.edu

mmpp: A Package for Calculating Similarity and Distance Metrics for Simple and Marked Temporal Point Processes

by Hideitsu Hino, Ken Takano, and Noboru Murata

Abstract A simple temporal point process (SPP) is an important class of time series, where the sample realization of the process is solely composed of the times at which events occur. Particular examples of point process data are neuronal spike patterns or spike trains, and a large number of distance and similarity metrics for those data have been proposed. A marked point process (MPP) is an extension of a simple temporal point process, in which a certain vector valued mark is associated with each of the temporal points in the SPP. Analyses of MPPs are of practical importance because instances of MPPs include recordings of natural disasters such as earthquakes and tornadoes. In this paper, we introduce the R package **mmpp**, which implements a number of distance and similarity metrics for SPPs, and also extends those metrics for dealing with MPPs.

Introduction

A random point process is a mathematical model for describing a series of discrete events (Snyder and Miller, 1991). Let $\mathcal{X} = \{t; t_0 \leq t \leq t_0 + T\}$ be the base space, on which an *event* occurs. The base space can be quite abstract, but here we will take \mathcal{X} to be a semi-infinite real line representing time. A set of ordered points on \mathcal{X} is denoted as $x = \{x_1, x_2, \dots, x_n\}$, $x_i \leq x_{i+1}$, and called a *sample realization*, or simply *realization* of a point process.

Reflecting the importance of the analysis of point processes in a broad range of science and engineering problems, there are already some R packages for modeling and simulating point processes such as **splancs** (Rowlingson and Diggle, 1993), **spatstat** (Baddeley and Turner, 2005), **PtProcess** (Harte, 2010), and **stpp** (Gabriel et al., 2013). These packages support various approaches for the analysis of both simple and marked spatial or spatio-temporal point processes, namely, estimating an intensity function for sample points, visualizing the observed sample process, and running simulations based on the specified models.

To complement the above mentioned packages, in **mmpp** (Hino et al., 2015), we focus on the similarity or distance metrics between realizations of point processes. Similarity and distance metrics are fundamental notions for multivariate analysis, machine learning and pattern recognition. For example, with an appropriate distance metric, a simple k nearest neighbor classifier and regressor (Cover and Hart, 1967) works in a satisfactory way. Also, kernel methods (Shawe-Taylor and Cristianini, 2004) are a well known and widely used framework in machine learning, in which inferences are done solely based on the values of kernel function, which is considered as a similarity metric between two objects.

As for the distance and similarity metric for point processes, vast amount of methods are developed in the field of neuroscience (Kandel et al., 2000). In this field, neural activities are recorded as sequences of spikes (called *spike trains*), which is nothing but a realization of a simple point process (SPP). By its nature, the responses of neurons to the same stimulus can be different. To claim the repeatability and the reliability of experimental results, a number of different distance and similarity metrics between sequence of spikes are developed (Victor, 2005; Schreiber et al., 2003; Kreuz et al., 2007; Quiroga et al., 2002; van Rossum, 2001).

The package **mmpp** categorizes commonly used metrics for spike trains and offers implementations for them. Since a spike train is a realization of a simple point process, the original metrics developed in the field of neuroscience do not consider marked point process (MPP) realizations. **mmpp** extends conventional metrics for SPPs to MPPs. We have two main aims in the development of **mmpp**:

1. to have a systematic and unified platform for calculating the similarities and distances between SPPs, and
2. to support MPPs to offer a platform for performing metric-based analysis of earthquakes, tornados, epidemics, or stock exchange data.

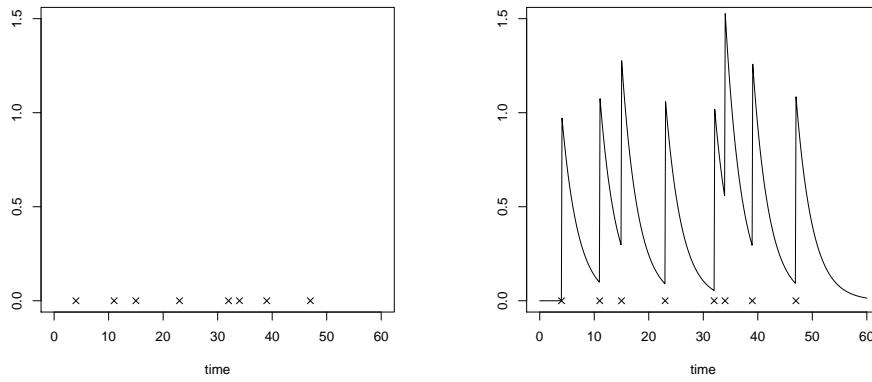


Figure 1: An example of continuation by smoothing. Left: event timings are marked with \times . Right: the corresponding continuous function $v_x(t)$.

Distances and similarities for point processes

Since realizations of temporal SPPs are ordered sets of the events, the commonly used Euclidean distance and inner product cannot be directly defined between them. Most of the metrics for SPP realizations first transform the realizations $x = \{x_1, \dots, x_n\}$, $x_i \leq x_{i+1}$ and $y = \{y_1, \dots, y_m\}$, $y_j \leq y_{j+1}$ to continuous functions $v_x(t)$ and $v_y(t)$, then define the distance or similarity metric between them. Based on the transformations, we categorize conventional methods for defining metrics on SPP realizations, and explain one by one in the following subsections.

We note that there are some attempts to directly define distances between SPP realizations. One of the most principled and widely used methods is based on the edit distance (Victor, 2005), and this method is extended to deal with MPP realizations by Suzuki et al. (2010). However, this approach is computationally expensive and prohibitive for computing the distance between spike trains with even a few dozen spikes. We exclude this class of metric from the current version of **mpp**.

In the following, we often use kernel smoothers and step functions for transforming SPP realizations. For notational convenience, we denote a kernel smoother with parameter τ by $h_\tau(t)$, and the Heaviside step function by

$$u(t) = \begin{cases} 1, & t \geq 0, \\ 0, & t < 0. \end{cases} \quad (1)$$

Examples of smoothers include the Gaussian kernel smoother $h_\tau^G(t) = \exp(-t^2/(2\tau^2))/\sqrt{2\pi\tau^2}$ and the Laplacian kernel smoother $h_\tau^L(t) = \exp(-|t|/\tau)/(2\tau)$.

Filtering to a continuous function

The most commonly used and intensively studied metrics for spike trains are based on the mapping of an event sequence to a real valued continuous function as

$$x = \{x_1, \dots, x_n\} \Rightarrow v_x(t) = \frac{1}{n} \sum_{i=1}^n h_\tau(t - x_i) \cdot u(t - x_i). \quad (2)$$

Figure 1 illustrates the transformation of an event sequence to a continuous function by the smoothing method.

Then, the inner product of x and y is defined by the usual ℓ_2 inner product in functional space by

$$k(x, y) = \int_0^\infty dt v_x(t) v_y(t) \in [-\infty, \infty], \quad (3)$$

and similarly the distance is defined by

$$d(x, y) = \sqrt{\int_0^\infty dt (v_x(t) - v_y(t))^2}. \quad (4)$$

When we use the Laplacian smoother $h_\tau^L(t) = \exp(-|t|/\tau)/(2\tau)$, the similarity and distance are

analytically given as

$$k(x, y) = \int_0^\infty dt v_x(t) v_y(t) = \frac{1}{4\tau nm} \sum_{i=1}^n \sum_{j=1}^m \exp\left(-\frac{1}{\tau}|x_i - y_j|\right), \quad (5)$$

and

$$\begin{aligned} d^2(x, y) &= k(x, x) + k(y, y) - 2k(x, y) \\ &= \frac{1}{4\tau n^2} \sum_{i=1}^n \sum_{j=1}^n \exp\left(-\frac{|x_i - x_j|}{\tau}\right) + \frac{1}{4\tau m^2} \sum_{i=1}^m \sum_{j=1}^m \exp\left(-\frac{|y_i - y_j|}{\tau}\right) \\ &\quad - \frac{1}{2\tau nm} \sum_{i=1}^n \sum_{j=1}^m \exp\left(-\frac{|x_i - y_j|}{\tau}\right), \end{aligned} \quad (6)$$

respectively. This distance eq. (6) is adopted by van Rossum (2001) for measuring the distance between spike trains. On the other hand, Schreiber et al. (2003) proposed to use the correlation defined by

$$\text{cor}(x, y) = \frac{\int_0^\infty dt v_x(t) v_y(t)}{\sqrt{\int_0^\infty dt v_x(t) v_x(t)} \sqrt{\int_0^\infty dt v_y(t) v_y(t)}} \quad (7)$$

to measure the similarity between spike trains. This class of measures is extended to take into account the effect of burst, i.e., the short interval in which events occur in high frequency, and refractory period, i.e., the short interval in which events tend to be suppressed immediately after the previous events (Houghton, 2009; Lytle and Fellous, 2011). These two effects, namely burst and refractory period, are commonly observed in neural activities. They are also observed in earthquake catalogues. After a large main shock, usually we observe high frequent aftershocks. On the other hand, suppression of events is sometimes happening, possibly because after a big event, the coda is so large that one cannot detect smaller events under the large ongoing signal from the big event (Kagan, 2004; Iwata, 2008).

The filtering-based metric is computed by using the function `fmetric` in `mmpp`. The first two arguments `S1` and `S2` are the (marked) point process realizations in the form of a ‘matrix’ object. The first column of `S1` and `S2` are the event timings and the rest are the marks. The argument `measure` can be either “sim” or “dist”, indicating similarity or distance, respectively. By default, the function assumes the Laplacian smoother. When the argument `h` of function `fmetric` is set to a function with scaling parameter τ as

```
> fmetric(S1, S2, measure = "sim", h = function(x, tau) exp(-x^2/tau), tau = 1)
```

the integrals in eq. (3) and eq. (4) are numerically done using the R function `integrate`. The function `h` should be square integrable and non-negative.

Intensity inner products

For analysis of point processes, the intensity function plays a central role. Paiva et al. (2009) proposed to use the intensity function for defining the inner product between SPP realizations. Let $N(t)$ be the number of events observed in the interval $(0, t]$. The intensity function of a counting process $N(t)$ is defined by

$$\lambda_x(t) = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \Pr[N(t + \epsilon) - N(t) = 1]. \quad (8)$$

We note that we can also consider the conditional intensity function reflecting the history up to the current time t , but we only explain the simplest case in this paper. Assuming that the SPP to be analysed is well approximated by a Poisson process, the intensity function is estimated by using a smoother $h_\tau(t)$ as

$$\hat{\lambda}_x(t) = \frac{1}{n} \sum_{i=1}^n h_\tau(t - x_i) \quad (9)$$

in non-parametric manner (Reiss, 1993). Using the estimates of intensity functions for processes behind x and y , Paiva et al. (2009) defined a similarity metric by

$$k(x, y) = \int_{-\infty}^\infty dt \hat{\lambda}_x(t) \hat{\lambda}_y(t) = \frac{1}{4\tau^2 nm} \sum_{i=1}^n \sum_{j=1}^m \int_{-\infty}^\infty dt h_\tau(t - x_i) h_\tau(t - y_j). \quad (10)$$

Particularly, when we use a Gaussian smoother $h_\tau^G(t) = \exp(-t^2/(2\tau^2)) / \sqrt{2\pi\tau}$, the integral is

analytically computed and we obtain an explicit formula

$$k(x, y) = \int_{-\infty}^{\infty} dt \hat{\lambda}_x(t) \hat{\lambda}_y(t) = \frac{1}{4\sqrt{\pi}\tau nm} \sum_{i=1}^n \sum_{j=1}^m \exp\left(-\frac{(x_i - y_j)^2}{4\tau^2}\right). \quad (11)$$

The distance metric is also defined as

$$d(x, y) = \int_{-\infty}^{\infty} dt (\hat{\lambda}_x(t) - \hat{\lambda}_y(t))^2, \quad (12)$$

and it is also simplified when we use the Gaussian smoother.

This class of measures is most in alignment with the statistical model of point processes. We estimated the intensity function in a versatile non-parametric approach, but it is reasonable to use other models such as the Hawkes model (Hawkes, 1971) when we should include the self-exciting nature of the process.

The intensity inner product metric is computed by using the function `iipmetric` in **mmp**. In the current version, the function assumes the Gaussian smoother, and its scaling parameter is specified by the argument `tau` as

```
> iipmetric(S1, S2, measure = "sim", tau = 1)
```

Co-occurrence metric

For comparing two SPP realizations, it is natural to *count* the number of events which can be considered to be co-occurring. There are two metrics for SPP realizations based on the notion of co-occurrence.

The first one proposed by Quian Quiroga et al. (2002) directly *counts near-by events*. The closeness of two events are defined by adaptively computed thresholds, making the method free from tuning parameters. Suppose we have two SPP realizations $x = \{x_1, \dots, x_n\}$ and $y = \{y_1, \dots, y_m\}$. For any events $x_i \in x$ and $y_j \in y$, a threshold under which the two events are considered to be synchronous with each other is defined as half of the minimum of the four inter event intervals around these two events:

$$\tau_{ij} = \frac{1}{2} \min\{x_{i+1} - x_i, x_i - x_{i-1}, y_{j+1} - y_j, y_j - y_{j-1}\}. \quad (13)$$

We note that τ_{ij} in the above definition depends on x and y , though, for the sake of notational simplicity, we simply denote it by τ_{ij} . Then, the function that counts the number of events in y which is coincided with those in x is defined by

$$c(x|y) = \sum_{i=1}^n \sum_{j=1}^m P_{ij}, \quad (14)$$

$$P_{ij} = \begin{cases} 1, & 0 < x_i - y_j < \tau_{ij}, \\ 1/2, & x_i = y_j, \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

Using this counting function, a similarity measure between x and y is defined as

$$k(x, y) = \frac{c(x|y) + c(y|x)}{\sqrt{nm}}, \quad (16)$$

and a distance measure is obtained by the transformation (Lytle and Fellous, 2011):

$$d(x, y) = 1 - k(x, y). \quad (17)$$

The second metric based on the counting co-occurrence is proposed by Hunter and Milton (2003), which transforms x to a continuous function $v_x(t)$, and sums up the near-by events in proportion to their degree of closeness. Denoting the closest event time in y from an event $x_i \in x$ by $y_{(x_i)}$, we define a function which measures degree of closeness by

$$dc_\tau(x_i) = \exp\left(-\frac{|x_i - y_{(x_i)}|}{\tau}\right). \quad (18)$$

Then, a similarity metric between x and y is defined by

$$k(x, y) = \frac{\frac{1}{n} \sum_{i=1}^n dc_\tau(x_i) + \frac{1}{m} \sum_{j=1}^m dc_\tau(y_j)}{2}, \quad (19)$$

and the distance is naturally defined by

$$d(x, y) = 1 - k(x, y). \quad (20)$$

The co-occurrence based metrics are computed by using the function `coocmetric`. By default, the function assumes the counting similarity in eq. (16). The smoothed counting similarity is computed by specifying the argument `type = "smooth"` as

```
> coocmetric(S1, S2, measure = "sim", type = "smooth", tau = 1)
```

Inter event interval

Assume an SPP realization $x = \{x_1, \dots, x_n\}$, $x_n < T$ such that for every event time x_i , $0 < x_i < T$, where T is the horizon of the time interval. In the inter event interval proposed by Kreuz et al. (2007), the SPP realization x is first modified to include artificial events corresponding to the beginning and end of the interval as

$$x = \{x_0 = 0, x_1, \dots, x_n, x_{n+1} = T\}. \quad (21)$$

Then each event is mapped to a function $v_x(t)$ as

$$v_x(t) = \sum_{i=0}^{n+1} f_i(t), \quad f_i(t) = \begin{cases} 0, & t \notin [x_i, x_{i+1}), \\ x_{i+1} - x_i, & t \in [x_i, x_{i+1}). \end{cases} \quad (22)$$

Two SPP realizations x and y are transformed to $v_x(t)$ and $v_y(t)$, then they are used to define an intermediate function

$$I_{xy}(t) = \frac{\min\{v_x(t), v_y(t)\}}{\max\{v_x(t), v_y(t)\}}. \quad (23)$$

This function takes value 1 when x is identical to y , and takes a smaller value when x and y are highly dissimilar. By using this intermediate function, the similarity measure is defined by

$$k(x, y) = \frac{1}{T} \int_0^T dt I_{xy}(t), \quad (24)$$

and the distance is defined by

$$d(x, y) = \frac{1}{T} \int_0^T dt (1 - I_{xy}(t)), \quad (25)$$

which is originally defined in Kreuz et al. (2007). A simple example of transformation $x \Rightarrow v_x(t)$, $y \Rightarrow v_y(t)$ and $x, y \Rightarrow 1 - I_{xy}(t)$ is illustrated in Figure 2.

The inter event interval metrics are computed by using the function `ieimetric` as

```
> ieimetric(S1, S2, measure = "sim")
```

Extension to marked point processes

Sometimes events considered in point processes entail certain vector valued *marks*. For example, seismic events are characterized by the time point the earthquake happens, and a set of attributes such as magnitude, depth, longitude, and latitude of the hypo-center. To deal with marked point processes, we extend the base space \mathcal{X} to $\mathcal{X} = \{t; t_0 \leq t \leq t_0 + T\} \times \mathbb{R}^p$, the Cartesian product of the time interval $[t_0, t_0 + T]$ and a region of the p dimensional Euclidean space corresponding to marks. Realizations of MPPs are denoted by, for example, $x = \{(x_1, r_1), \dots, (x_n, r_n)\}$ and $y = \{(y_1, s_1), \dots, (y_m, s_m)\}$.

There might be many possible ways of extension. The packages `mmp` takes the simplest way to deal with marks in a unified and computationally efficient manner, namely, the density or weight of marks are included in the metrics for SPPs by Gaussian windowing as shown in eq. (27).

Filtering to continuous function

In the same manner as eq. (2), the marked point process realization $x = \{(x_1, r_1), \dots, (x_n, r_n)\}$ is transformed to a continuous function as

$$x = \{(x_1, r_1), \dots, (x_n, r_n)\} \Rightarrow v_x(t, z) = \frac{1}{n} \sum_{i=1}^n h_M(z - r_i) h_\tau(t - x_i) \cdot u(t - x_i), \quad (26)$$

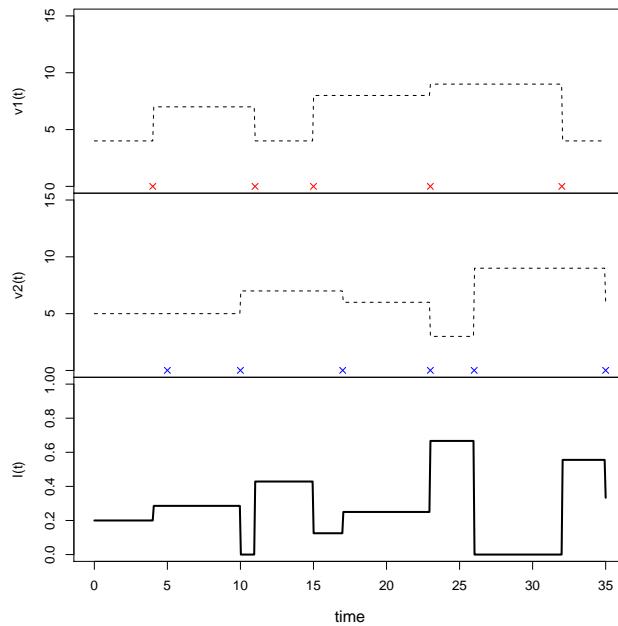


Figure 2: Example of the transformation of two point process realizations x and y into the intermediate function I_{xy} . The top panel shows x and corresponding continuous function $v_x(t)$. The middle panel shows y and $v_y(t)$. The bottom panel shows the intermediate function $1 - I_{xy}(t)$.

where

$$h_M(z) = (2\pi)^{-p/2} |M|^{1/2} \exp\left(-\frac{1}{2} z^\top M z\right) = (2\pi)^{-p/2} |M|^{1/2} \exp\left(-\frac{1}{2} \|z\|_M^2\right), M \in \mathbb{R}^{p \times p}, \quad (27)$$

where $|M|$ is the determinant of a matrix M , and $\|z\|_M^2 = z^\top M z$. Integrating with respect to both time t and mark z , we define the inner product by

$$k(x, y) = \int_{\mathbb{R}^p} dz \int_0^\infty dt v_x(t, z) v_y(t, z), \quad (28)$$

and the distance by

$$d^2(x, y) = \int_{\mathbb{R}^p} dz \int_0^\infty dt (v_x(t, z) - v_y(t, z))^2. \quad (29)$$

By virtue of Gaussian windowing, the integral with respect to the mark is explicitly written as

$$\int_{\mathbb{R}^p} dz \exp\left(-\frac{1}{2} \|z - r_i\|_M^2 - \frac{1}{2} \|z - s_j\|_M^2\right) = (2\pi)^{\frac{p}{2}} \sqrt{|2M|} \exp\left(-\frac{1}{4} \|r_i - s_j\|_M^2\right). \quad (30)$$

Furthermore, when we use Laplacian smoother for transforming temporal SPPs, we obtain

$$k(x, y) = \frac{|M|^{1/2}}{2^{p+2}\pi^{p/2}\tau nm} \sum_{i=1}^n \sum_{j=1}^m \exp\left(-\frac{\|r_i - s_j\|_M^2}{4}\right) \exp\left(-\frac{|x_i - y_j|}{\tau}\right). \quad (31)$$

The distance metric is also calculated in the same manner.

We note that the effect of marks depend on the units used for the various marks. It is reasonable to estimate the variance of each mark, and set the diagonal elements of M to be reciprocal of the variances, which is adopted as the default setting for M in **mmp**.

Intensity inner product

Extending kernel estimation eq. (9) to multivariate kernel estimation as

$$\hat{\lambda}_x(t, z) = \frac{1}{n} \sum_{i=1}^n h_M(z - r_i) h_\tau(t - x_i), \quad (32)$$

we obtain the estimate of the intensity function of the marked point process. We note that kernel density estimation for multidimensional variables is inaccurate in general, and we can instead estimate

the ground intensity function $\lambda(t)$ and the density function for mark $\lambda(z)$ separately. However, in many applications, the dimension of marks is not so high, and currently we adopt the kernel based estimator in eq. (32). The intensity inner product for MPP realizations is then defined by

$$k(x, y) = \int_{\mathbb{R}^p} dz \int_{-\infty}^{\infty} dt \hat{\lambda}_x(t, z) \hat{\lambda}_y(t, z). \quad (33)$$

When we use the Gaussian smoother $h_{\tau}^g(t) = \exp(-t^2/\tau) / \sqrt{2\pi\tau^2}$, the integral is explicitly computed and we obtain

$$k(x, y) = \frac{1}{\pi^{(p+1)/2} \tau^{1/2} |M|^{1/2} nm} \sum_{i=1}^n \sum_{j=1}^m \exp\left(-\frac{(x_i - y_j)^2}{2\tau}\right) \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{s}_j\|_M^2}{4}\right). \quad (34)$$

The distance metric is also defined by

$$d(x, y) = \int_{\mathbb{R}^p} dz \int_{-\infty}^{\infty} dt (\hat{\lambda}_x(t, z) - \hat{\lambda}_y(t, z))^2. \quad (35)$$

For estimating the intensity function, a simple Poisson process is assumed. This assumption is relaxed with more flexible models such as the ETAS model (Ogata, 1988, 1998), where the intensity function is estimated using the R packages **SAPP** (of Statistical Mathematics, 2014) and **etasFLP** (Chioldi and Adelfio, 2015), for example. The extension of the intensity-based metric to support other forms of intensity estimation such as the Hawkes and ETAS models remains our important future work.

Co-occurrence metric

To extend the co-occurrence metric based on counting the synchronous events, eq. (15) is replaced with a weighted counter

$$P_{ij} = \exp(-\|\mathbf{r}_i - \mathbf{s}_j\|_M^2) \times \begin{cases} 1, & 0 < x_i - y_j < \tau_{ij}, \\ 1/2, & x_i = y_j, \\ 0, & \text{otherwise.} \end{cases} \quad (36)$$

To extend the co-occurrence metric based on the smoothed count of the synchronous events, eq. (18) is replaced with a weighted smoothed counter

$$dc_{\tau, M}(x_i) = \exp\left(-\|\mathbf{r}_i - \mathbf{s}_{(x_i)}\|_M^2\right) \times \exp\left(-\frac{|x_i - y_{(x_i)}|}{\tau}\right). \quad (37)$$

Inter event interval

To weight the inter event interval by using marks associated with two MPP realizations x and y , we define index extraction operators as follows. We modify an MPP realization $x = \{(x_1, \mathbf{r}_1), \dots, (x_n, \mathbf{r}_n)\}$ to include artificial events and marks corresponding to the beginning and end of the interval as

$$x = \{(x_0 = 0, \mathbf{r}_0 = \mathbf{0}), (x_1, \mathbf{r}_1), \dots, (x_n, \mathbf{r}_n), (x_{n+1} = T, \mathbf{r}_{n+1} = \mathbf{0})\}. \quad (38)$$

Then we define operators

$$\underline{q} : [0, T] \times \mathcal{X} \rightarrow \mathbb{R} \\ (t, x) \mapsto i, \quad \text{s.t. } t \in [x_i, x_{i+1}], \quad (39)$$

$$\bar{q} : [0, T] \times \mathcal{X} \rightarrow \mathbb{R} \\ (t, x) \mapsto i + 1, \quad \text{s.t. } t \in [x_i, x_{i+1}]. \quad (40)$$

The intermediate function eq.(23) is modified to take into account the dissimilarity of marks:

$$I_{xy}(t) = \frac{\min(v_x(t), v_y(t))}{\max(v_x(t), v_y(t))} \frac{\exp(-\|\mathbf{r}_{\underline{q}(t,x)} - \mathbf{s}_{\underline{q}(t,y)}\|_M^2) + \exp(-\|\mathbf{r}_{\bar{q}(t,x)} - \mathbf{s}_{\bar{q}(t,y)}\|_M^2)}{2}. \quad (41)$$

The distance and similarity are then calculated using eq. (25) and eq. (24).

The usage of the functions **fmetric**, **iipmetric**, **coocmetric**, and **ieimetric** does not change for marked point process data, except for one additional argument **M**, which is the precision matrix M in

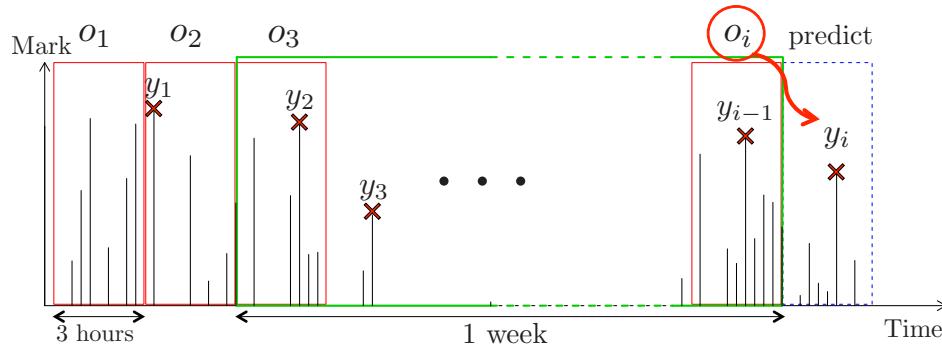


Figure 3: An illustrative diagram of the problem setting. The horizontal axis corresponds to time, and the vertical axis shows marks. Though the dimension of the mark is four, it is shown as one-dimensional axis. The process is split by a three hour window, and each window is assigned an output variable, which is the maximum magnitude in the next time window. Using the past one week data, the output variable of the next time window is predicted by nearest neighbor regression.

eq. (27). By default, it is automatically set to the diagonal matrix with the diagonal elements equal to the reciprocal of the variance of corresponding marks of S1 and S2. We can also specify the matrix M manually as

```
> fmetric(S1, S2, measure = "sim", M = diag(3))
```

where the number of marks is assumed to be three.

An example with the Miyagi20030626 data set

This section illustrates the use of the package with a simple experiment. We use the Miyagi20030626 dataset contained within the package.

```
> library(mmpp)
> data(Miyagi20030626)
```

The dataset is composed of 2305 aftershocks of the 26th July 2003 earthquake of M6.2 at the northern Miyagi-Ken Japan, which is a reparameterization of the main2006JUL26 dataset from the **SAPP** package. Each record has 5 dimensions, time, longitude, latitude, depth, and magnitude of its hypo-center. The time is recorded in seconds from the main shock.

To illustrate the use of the package, we consider a simple prediction task. We first split the original dataset by a time-window of length $60 \times 60 \times 3$, which means that the time interval of each partial point process split by this window is three hours.

```
> sMiyagi <- splitMPP(Miyagi20030626, h = 60*60*3, scaleMarks = TRUE)$S
```

Then, the maximum magnitude in each partial point process realization is computed.

```
> ## target of prediction is the maximum magnitude in the window
> m <- NULL
> for (i in 1:length(sMiyagi)) {
+   m <- c(m, max(sMiyagi[[i]]$magnitude))
+ }
```

The task we consider is the prediction of the maximum magnitude in the *next* three hours using the past one week of data. We formulate this problem as a regression problem. Let the partial point process realization in the i -th window be o_i , and let the maximum magnitude in the $i + 1$ -th window be m_i . Then the problem is predicting m_{i+1} given o_{i+1} and the past $24 \times 7/3 = 56$ hours of data $\{(o_{i-\ell}, m_{i-\ell})\}_{\ell=0}^{55}$. See Figure 3 for an illustrative diagram of the problem setting.

```
> m <- m[-1]
> sMiyagi[[length(sMiyagi)]] <- NULL
> ## number of whole partial MPPs split by a 3-hour time window
> N <- length(sMiyagi)
> ## training samples are past one week data
> Ntr <- 24*7/3
> ## number of different prediction methods
> Nd <- 10
```

For the purpose of illustrating the use of the package and to show the effect of different similarity metrics, we adopt the nearest neighbor regression. That is, given the current realization o_i , we find the most similar realization $o_j \in \{o_{i-l}\}_{l=0}^{55}$, and use the corresponding maximum magnitude m_j as the predictor for m_{i+1} . We use ten different similarity metrics supported in the package, and evaluate the mean absolute errors. The metrics used for these experiments are the filter based metric in eq. (3), the intensity inner product metric in eq. (10), the co-occurrence with counting in eq. (16), the co-occurrence with smoothed counting in eq. (19), and the inter event interval metric in eq. (24), and their MPP extensions.

```
> err <- matrix(0, N - Ntr, Nd)
> colnames(err) <- c("f SPP", "iip SPP", "cooc (s) SPP", "cooc (c) SPP", "iei SPP",
+                      "f MPP", "iip MPP", "cooc (s) MPP", "cooc (c) MPP", "iei MPP")
```

The following code performs the above explained experiment.

```
> for (t in 1:(N - Ntr)) {
+   qid <- Ntr + t
+   q <- sMiyagi[[qid]]
+   ## simple PP
+   ## fmetric with tau = 1
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, fmetric(q$time, sMiyagi[[qid - i]]$time))
+   }
+   err[t, 1] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## iipmetric with tau = 1
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, iipmetric(q$time, sMiyagi[[qid - i]]$time))
+   }
+   err[t, 2] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## coocmetric (smooth) with tau = 1
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, coocmetric(q$time, sMiyagi[[qid - i]]$time,
+                                         type = "smooth"))
+   }
+   err[t, 3] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## coocmetric (count)
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, coocmetric(q$time, sMiyagi[[qid - i]]$time,
+                                         type = "count"))
+   }
+   err[t, 4] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## iei metric
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, ieimetric(q$time, sMiyagi[[qid - i]]$time))
+   }
+   err[t, 5] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## marked PP with latitude, longitude, depth, and magnitude
+   ## fmetric with tau = 1
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, fmetric(q, sMiyagi[[qid - i]])))
+   }
+   err[t, 6] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## iipmetric with tau = 1
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, iipmetric(q, sMiyagi[[qid - i]])))
+   }
+   err[t, 7] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## coocmetric (smooth) with tau=1
+   sim2query <- NULL
```

```

+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, coocmetric(q, sMiyagi[[qid - i]], type = "smooth"))
+   }
+   err[t, 8] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## coocmetric (count)
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, coocmetric(q, sMiyagi[[qid - i]], type = "count"))
+   }
+   err[t, 9] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+   ## iei metric
+   sim2query <- NULL
+   for (i in 1:Ntr) {
+     sim2query <- c(sim2query, ieimetric(q, sMiyagi[[qid - i]]))
+   }
+   err[t, 10] <- abs(m[qid] - m[t:(Ntr + t - 1)][which.max(sim2query)])
+ }
> colMeans(err)
  f SPP    iip SPP    cooc (s) SPP    cooc (c) SPP    iei SPP
0.7002634 0.6839529 0.7263602 0.6632930 0.7905148
  f MPP    iip MPP    cooc (s) MPP    cooc (c) MPP    iei MPP
0.6839529 0.6317594 0.6643804 0.6622056 0.7698548

```

From this simple example, we can see that the prediction accuracy is improved by taking the marks into account.

Summary and future directions

mmpp is the first R package dedicated to the calculation of the similarity and distance metrics for marked point process realizations. It provides the implementation of several similarity metrics for simple point processes, originally proposed in the literature of neuroscience, and also provides extensions of these metrics to those for marked point processes.

A simple example of a real dataset presented in this paper illustrates the importance of taking the marks into account in addition to the event timings, and it also illustrates the possibilities of the package **mmpp** with a user guide for practitioners.

The development of the **mmpp** package has only just begun. Currently, we are considering supporting burst sensitive and refractory period sensitive versions of **fmetric**, since these properties are commonly observed in both neural activities and seismic event recordings. In the current version of **mmpp**, for treating MPPs, event timings and marks are assumed to be separable, and all the marks are simultaneously estimated by a kernel density estimator. This is a strong assumption and other possibilities for modeling MPPs should be considered. For example, it is popular to group spatio-temporal events, i.e., event timings and locations, and treat marks such as magnitude in seismic events as purely *marks*. Then, the separability between marks and spatio-temporal events can be tested by using test statistics proposed in [Schoenberg \(2004\)](#) and [Chang and Schoenberg \(2011\)](#). The separability assumption offers computational advantages, though, it would miss the intrinsic structure and relationship between event timings and marks. In principle, the separability hypothesis should be tested before calculating the metrics. Frameworks for flexible modeling of marked sample sequences with statistical validation such as nonparametric tests would be implemented in future version of **mmpp**. We are also considering to extend the intensity inner product metric to support other form of intensity estimation such as Hawkes and ETAS models.

Different similarity metrics capture different aspects of the point process realizations. Our final goal of the development of the package **mmpp** is to provide a systematic way to select or combine appropriate metrics for analysing given point process realizations and certain tasks such as prediction of magnitude or clustering similar seismic events.

Acknowledgement

The authors are grateful to T. Iwata for helpful discussions and suggestions. The authors would like to express their special thanks to the editor, the associate editor and three anonymous reviewers whose comments led to valuable improvements of the manuscript. Part of this work is supported by JSPS KAKENHI Grant Number 25870811, 26120504, and 25120009.

Bibliography

- A. Baddeley and R. Turner. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 1 2005. URL <http://www.jstatsoft.org/v12/i06>. [p237]
- C.-H. Chang and F. Schoenberg. Testing separability in marked multidimensional point processes with covariates. *The Annals of the Institute of Statistical Mathematics*, 63(6):1103–1122, 2011. doi: 10.1007/s10463-010-0284-7. [p246]
- M. Chiodi and G. Adelfio. *etasFLP: Mixed FLP and ML Estimation of ETAS Space-Time Point Processes*, 2015. URL <https://CRAN.R-project.org/package=etasFLP>. R package version 1.3.0. [p243]
- T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi: 10.1109/TIT.1967.1053964. [p237]
- E. Gabriel, B. S. Rowlingson, and P. J. Diggle. stpp: An R package for plotting, simulating and analyzing spatio-temporal point patterns. *Journal of Statistical Software*, 53(2):1–29, 4 2013. URL <http://www.jstatsoft.org/v53/i02>. [p237]
- D. Harte. PtProcess: An R package for modelling marked point processes indexed by time. *Journal of Statistical Software*, 35(8):1–32, 2010. URL <http://www.jstatsoft.org/v35/i08/>. [p237]
- A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1): 83–90, 1971. doi: 10.2307/2334319. [p240]
- H. Hino, K. Takano, Y. Yoshikawa, and N. Murata. mmpp: Various similarity and distance metrics for marked point processes, 2015. URL <https://CRAN.R-project.org/package=mmpp>. [p237]
- C. Houghton. Studying spike trains using a van Rossum metric with a synapse-like filter. *Journal of Computational Neuroscience*, 26(1):149–155, 2009. doi: 10.1007/s10827-008-0106-6. [p239]
- J. D. Hunter and J. G. Milton. Amplitude and frequency dependence of spike timing: Implications for dynamic regulation. *Journal of Neurophysiology*, 90(1):387–394, July 2003. doi: 10.1152/jn.00074.2003. [p240]
- T. Iwata. Low detection capability of global earthquakes after the occurrence of large earthquakes: Investigation of the Harvard CMT catalogue. *Geophysical Journal International*, 174(3):849–856, 2008. doi: 10.1111/j.1365-246X.2008.03864.x. [p239]
- Y. Y. Kagan. Short-term properties of earthquake catalogs and models of earthquake source. *Bulletin of the Seismological Society of America*, 94(4):1207–1228, 2004. doi: 10.1785/012003098. [p239]
- E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science*. McGraw-Hill Medical, 4th edition, July 2000. [p237]
- T. Kreuz, J. S. Haas, A. Morelli, H. D. Abarbanel, and A. Politi. Measuring spike train synchrony. *Journal of Neuroscience Methods*, 165(1):151–161, 2007. doi: 10.1016/j.jneumeth.2007.05.031. [p237, 241]
- D. Lyttle and J.-M. Fellous. A new similarity measure for spike trains: Sensitivity to bursts and periods of inhibition. *Journal of Neuroscience Methods*, 199(2):296–309, 2011. doi: 10.1016/j.jneumeth.2011.05.005. [p239, 240]
- T. I. of Statistical Mathematics. SAPP: *Statistical Analysis of Point Processes*, 2014. URL <https://CRAN.R-project.org/package=SAPP>. R package version 1.0.4. [p243]
- Y. Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 83(401):9–27, Mar. 1988. doi: 10.2307/2288914. [p243]
- Y. Ogata. Space-time point-process models for earthquake occurrences. *The Annals of the Institute of Statistical Mathematics*, 50(2):379–402, June 1998. doi: 10.1023/a:1003403601725. [p243]
- A. R. C. Paiva, I. Park, and J. C. Príncipe. A reproducing kernel Hilbert space framework for spike train signal processing. *Neural Comput.*, 21(2):424–449, Feb. 2009. doi: 10.1162/neco.2008.09-07-614. [p239]
- R. Quian Quiroga, T. Kreuz, and P. Grassberger. Event synchronization: A simple and fast method to measure synchronicity and time delay patterns. *Physical Review E*, 66:041904, Oct 2002. doi: 10.1103/PhysRevE.66.041904. [p237, 240]

- R.-D. Reiss. *A Course on Point Processes*. Springer Series in Statistics. Springer, 1993. [p239]
- B. Rowlingson and P. Diggle. Splancs: Spatial point pattern analysis code in S-PLUS. *Computers & Geosciences*, 19(5):627–655, 1993. doi: 10.1016/0098-3004(93)90099-Q. [p237]
- F. P. Schoenberg. Testing separability in spatial-temporal marked point processes. *Biometrics*, 60(2):471–481, 2004. doi: 10.1111/j.0006-341X.2004.00192.x. [p246]
- S. Schreiber, J. Fellous, D. Whitmer, P. Tiesinga, and T. Sejnowski. A new correlation-based measure of spike timing reliability. *Neurocomputing*, 52–54(0):925 – 931, 2003. doi: 10.1016/S0925-2312(02)00838-X. Computational Neuroscience: Trends in Research 2003. [p237, 239]
- J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. [p237]
- D. L. Snyder and M. I. Miller. *Random Point Processes in Time and Space*. Springer-Verlag, New York, NY, USA, 2nd edition, 1991. [p237]
- S. Suzuki, Y. Hirata, and K. Aihara. Definition of distance for marked point process data and its application to recurrence plot-based analysis of exchange tick data of foreign currencies. *International Journal of Bifurcation and Chaos*, 20(11):3699–3708, 2010. doi: 10.1142/S0218127410027970. [p238]
- M. C. W. van Rossum. A novel spike distance. *Neural Computation*, 13(4):751–763, 2001. doi: 10.1162/089976601300014321. [p237, 239]
- J. D. Victor. Spike train metrics. *Current Opinion in Neurobiology*, 15:585–592, October 2005. doi: 10.1016/j.conb.2005.08.002. [p237, 238]

Hideitsu Hino

*Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1 Tennoudai, Tsukuba, Ibaraki, 305-8573
Japan
hinohide@cs.tsukuba.ac.jp*

Ken Takano

*School of Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169-8555
Japan
ken.takano@toki.waseda.jp*

Noboru Murata

*School of Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555
Japan
noboru.murata@eb.waseda.ac.jp*

Open-Channel Computation with R

by Michael C. Koohofkan and Bassam A. Younis

Abstract The `rivr` package provides a computational toolset for simulating steady and unsteady one-dimensional flows in open channels. It is designed primarily for use by instructors of undergraduate- and graduate-level open-channel hydrodynamics courses in such diverse fields as river engineering, physical geography and geophysics. The governing equations used to describe open-channel flows are briefly presented, followed by example applications. These include the computation of gradually-varied flows and two examples of unsteady flows in channels—namely, the tracking of the evolution of a flood wave in a channel and the prediction of extreme variation in the water-surface profile that results when a sluice gate is abruptly closed. Model results for the unsteady flow examples are validated against standard benchmarks. The article concludes with a discussion of potential modifications and extensions to the package.

Introduction

Hydraulic engineers routinely rely on computational models to simulate the flow and conveyance of water for planning and designing channels and in-stream structures, managing reservoirs, and conducting flood damage and risk analyses. The simulation of fluid flow is a difficult and computationally-intensive endeavor, and simplifications are often made in order to reduce the task to one that is feasible with regard to time management and data availability. One of the most frequently-made assumptions is that of one-dimensional flow. This greatly simplifies the computation of a wide range of practically-relevant flows such as e.g. the prediction of the movement of a flood wave in a waterway, or the consequences of a sudden opening or closure of a control gate on the water level in a navigation channel. In making this assumption, the flow velocity—which in reality can vary in all three coordinate directions—is both depth- and laterally-averaged to produce a bulk or mean channel velocity that is a function of only time and distance along the channel:

$$\bar{u} = \frac{\int_{y=0}^{y=H} \int_{z=0}^{z=S} u(x, y, z, t) dy dz}{\iint dy dz} = f(x, t) \quad (1)$$

where u is velocity, y is the vertical dimension, z is the lateral (cross-channel) dimension, t is time and x is the along-channel direction. The integral bounds H and S are the flow depth and span, respectively. Treatment of bed friction is also frequently simplified in practical applications by relating the bed shear stresses to the “friction slope”, which is derived from the Energy Grade Line (EGL) equation (Leopold, 1953) and expressed using the semi-empirical Manning’s equation

$$S_f = \left(\frac{n u}{C_m R^{2/3}} \right)^2 \quad (2)$$

where S_f is the friction slope, n is the empirical Manning’s coefficient and R is the hydraulic radius. Manning’s equation and the EGL equation are widely used in the discipline to predict steady and gradually-varied flow conditions. The prediction of unsteady flows is somewhat more complex; in addition to the fact that the flow area is different from the area of the conveying channel, the governing equations are both non-linear and coupled. This complicates the task of obtaining solutions that are both numerically stable and accurate without the need for excessive computational resources.

A wide variety of software suites are available for computing steady and unsteady flows; some of the most well-known packages are listed in Table 1. These packages are typically accessed via a graphical user interface and often have specialized input and output data formatting requirements. These software suites are capable of modeling complex networks of irregular channels and are therefore of great utility to engineers and practitioners working on large-scale projects, but they also pose high barriers to entry in terms of cost, usability and computer hardware requirements as well as in terms of ancillary data needs. These software suites are so complex that many developers provide training courses for utilizing the software, with the assumption that users already possess an education or background in river mechanics and hydraulics.

The `rivr` package (Koohofkan, 2015) was designed to provide rapid solutions to steady and unsteady one-dimensional flows in open channels. The primary purpose of the package is to provide educators with an accessible open-source toolset that can be used to supplement lectures with interactive examples, and to facilitate the design of assignments that encourage students to explore open-channel flow concepts rather than focusing on the actual implementation of solutions. The R

Name	Developer	Free for use	Open source	Cross-platform
HEC-RAS	USACE	✓	✗	✗
SRH 1D	USBR	✓	✗	✗
ESTRY	BMT	✗	✗	✗
Mike 11	DHI	✗	✗	✗
SOBEK D-Flow 1D	Deltarès	✗	✗	✗

Table 1: Selection of popular software packages for modeling steady and unsteady flow.

Project for Statistical Computing provides an ideal platform for providing **rivr** (and educational tools in general) to students and instructors. R is both cross-platform and free and open-source software, meaning students can run the software on personal computers and educational institutions do not need enter expensive licensing agreements to provide it on institution-owned computers. Moreover, R provides a large and comprehensive library of functions for data visualization, analysis and export both through its core functionality and through a diversity of packages developed and supported by an active global community of over 2 million users, developers and scientists. R is often referred to as a “scripting language”, meaning that code blocks or snippets can be run either in an automated fashion or piecewise by users; scripting languages are ideal for teaching because they support interactive programming and exploration of data. Furthermore, support for dynamic document generation with R via packages such as **knitr** (Xie, 2014) provides an excellent mechanism for instructors to develop teaching materials that place concept and implementation side by side, as well as a promising format for students to submit assignments and reports. Finally, R provides the potential for instructors with virtually no web development proficiency to create interactive web-based tutorials and demonstrations using the **shiny** web application framework (Chang et al., 2015).

The flow algorithms provided by **rivr** are implemented in C++ and integrated into the package using **Rcpp** (Eddelbuettel and Francois, 2011) to leverage the speed advantage of compiled code. All algorithms were developed from the ground up and are based on well-established, peer-reviewed contributions to the field. Functions are provided for computing normal and critical depth, channel geometry and conveyance, and gradually-varied flow profiles. The package also provides an advanced function which implements two canonical formulations of unsteady flow—the Kinematic Wave Model and the Dynamic Wave Model—which are routinely taught in graduate-level courses in hydrology and open-channel hydraulics. The numerical schemes implemented in **rivr** are similarly well-established and reflect varying degrees of sophistication that are common topics in numerical methods courses, including implementations of iterative solutions to implicit equations; a solution scheme for nonlinear ordinary differential equations; and numerical methods for solving sets of partial differential equations ranging from one of the simplest formulations of an explicit finite-difference scheme to an advanced two-step predictor-corrector scheme that illustrates the trade-offs between performance and complexity of implementation. Detailed explanation of the theoretical foundations and numerical schemes is provided in the form of a technical vignette, and the source code is carefully organized and commented to facilitate modifications and extensions to the package such as new numerical schemes and alternative formulations for steady and unsteady flow modeling.

This article provides use examples for computing both gradually-varied and unsteady flows with **rivr**. In the first section, solutions to gradually-varied flow or “backwater curve” problems using the standard-step method are discussed. In the second section, solutions to unsteady flow problems using different theoretical formulations are described, followed by a comparison of two numerical solutions to an unsteady flow problem with discontinuous boundary conditions. The article concludes with a discussion of potential opportunities for extensions and contributions to the package.

Calculation of back-water curves

The water-surface profile resulting from non-uniform steady flow conditions is generally referred to as a gradually-varied flow (GVF) profile or “backwater curve”. Backwater curves are commonly used for floodplain delineation and assessing upstream effects of bridges, weirs and other channel structures. The GVF concept is based on a simplified form of the St. Venant equations (discussed in the following section) and is expressed as

$$\frac{dy}{dx} = \frac{S_0 - S_f}{1 - Fr^2} \quad (3)$$

where dy/dx is the water-surface slope, S_0 is the channel slope and Fr is the Froude Number. Eq. (3) describes steady-state conditions, i.e. the channel flow rate $Q = uA$ is constant. When $dy/dx = 0$, the flow depth is uniform and $S_0 = S_f$; the flow depth under these conditions is referred to as the

“normal depth” and can be calculated using Manning’s equation. The relative positioning of the normal depth and the “critical depth”—the depth at which the flow energy is minimized—determine the flow behavior. GVF profiles are classified based on the flow regime, the channel slope and the nature of the divergence from the steady-flow water-surface profile at a specific location or “control section” (Doubt, 1971). An M1 profile refers to sub-critical flows where the water depth at the control section is greater than the normal depth. An M2 profile refers to a water-surface profile where the water depth at the control section is below the normal depth, but above the critical depth. Other classes include “S” (steep channels where the normal depth is shallower than the critical depth), “C” (flow regimes where the normal depth and critical depth coincide), “H” (flat channels) and “A” (channels of “adverse” slope, i.e. sloped against the direction of flow).

The **rivr** package computes the GVF profile using the standard-step method (Chaudhry, 2007). A Newton-Raphson scheme is used where iterative methods are required to find the roots of an equation. The following example illustrates how to use **rivr** to calculate the variation of the water-surface profile with distance, and its return to the normal-depth level far upstream of a perturbation in flow depth. Both an M1 and an M2 profile are considered for a rectangular channel with a width of 100 feet, a slope of 0.001, a flow of 250 cubic feet per second, and a Manning’s roughness coefficient of 0.045. The water depth y at the control section is specified as 1 foot above the normal depth y_n for the M1 profile (2.71 feet) and 1.1 times the critical depth y_c for the M2 profile (0.64 feet). The elevation and x-coordinate of the control section are both specified as zero. Example code for computing the water-surface profile using the function `compute_profile` is shown below.

```

g <- 32.2          # gravitational acceleration (ft/s^2)
Cm <- 1.486        # conversion factor for Manning's equation in US customary units
slope <- 0.001      # channel slope (vertical ft / horizontal ft)
mannings <- 0.045    # Manning's roughness
flow <- 250          # channel flow rate (ft^3/s)
width <- 100          # channel bottom width (ft)
sideslope <- 0        # channel sideslope (horizontal ft / vertical ft)

# calculate normal depth for channel, initial guess of 2 feet
yn <- normal_depth(slope, mannings, flow, 2, Cm, width, sideslope)

# calculate critical depth, initial guess of 1 ft
yc <- critical_depth(flow, 1, g, width, sideslope)

# compute the M1 profile 3000 ft upstream with a step size of 50 ft
m1 <- compute_profile(slope, mannings, flow, yn + 1, Cm, g,
                       width, sideslope, z0 = 0, x0 = 0, stepdist = 50, totaldist = 3000)

# compute the M2 profile under the same assumptions
m2 <- compute_profile(slope, mannings, flow, 1.1 * yc, Cm, g,
                       width, sideslope, z0 = 0, x0 = 0, stepdist = 50, totaldist = 3000)

```

The effect of model resolution on results is shown in Figure 1. For the M1 profile, the solution is quite stable even for very coarse (> 100 feet) resolutions. In contrast, solutions to the M2 profile are quite sensitive to model resolution owing to the steep gradient in dy/dx where $y \rightarrow y_c$. However, the solution to the M2 profile is not sensitive to resolutions finer than 10 feet. A step size of 10 feet is used in subsequent analyses to ensure smooth calculations of the water-surface slope for both profiles.

Manning’s coefficient is an empirical quantity that cannot be directly measured, and engineers often rely on tabulated values to inform their selection of a roughness value based on channel bank material and conditions such as vegetation and compaction. It is generally necessary to investigate the sensitivity of the solution to the choice of Manning’s roughness coefficient, or to calibrate a GVF model based on channel depth observations. The next example considers the GVF profile of the control section specified above for a range of Manning’s roughness values spanning 0.0225 to 0.0675. Figure 2 shows that the normal depth (apparent as the section where the water-surface slope is linear) changes depending on the value of the roughness coefficient; this comes as no surprise, as the concept of normal depth is itself derived from Manning’s equation. The specification of a fixed control section depth therefore confounds analysis of the relationship between Manning’s coefficient and the nature of the GVF profile.

In order to control for the dependence of the normal depth on Manning’s coefficient, the analysis is repeated using a control section depth of $1.25y_n$ for the M1 profile and $0.75y_n$ for the M2 profile, where y_n is the normal depth associated with each roughness value. Note that the critical depth is by definition independent of Manning’s coefficient. Figure 3 shows the water-surface elevation profile as a percent difference from the normal depth for each roughness value; it is clear from the figure that bed

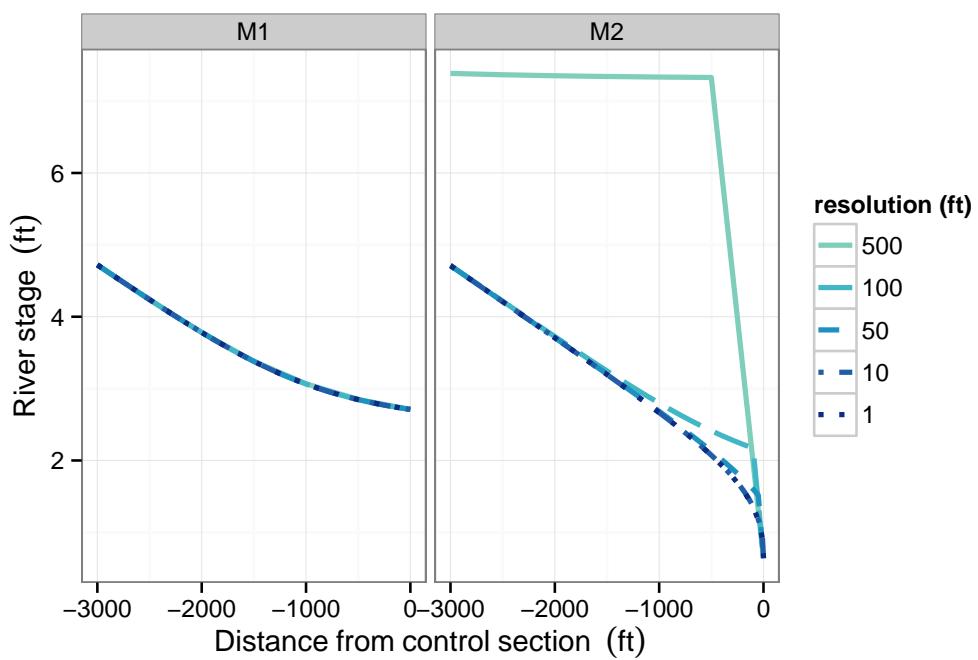


Figure 1: M1 and M2 water-surface elevation profiles computed under different resolutions. When steep gradients in the water-surface profile are present, finer spatial resolutions are needed to provide accurate solutions.

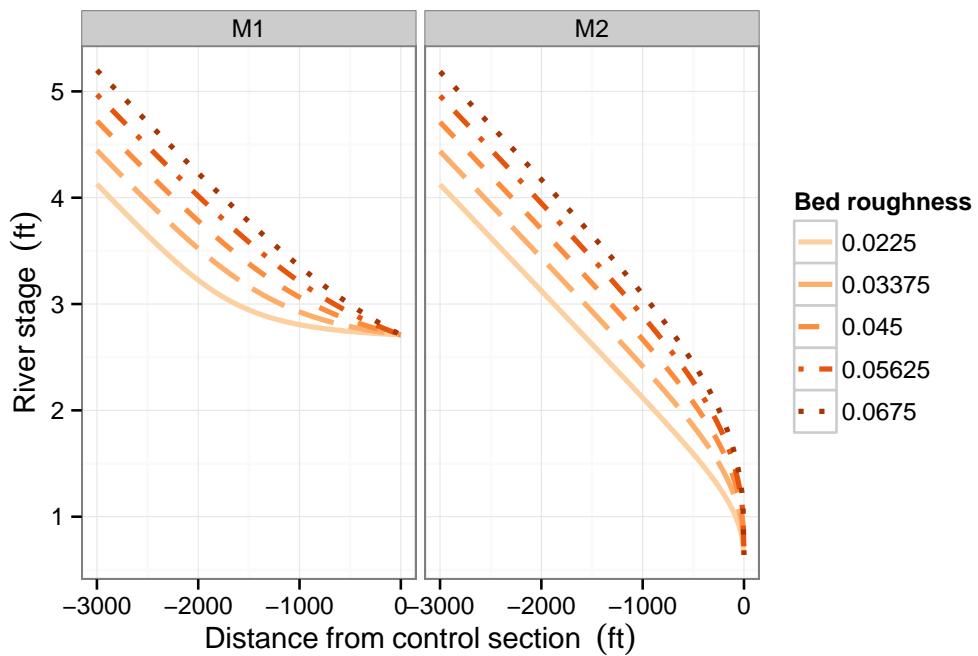


Figure 2: M1 and M2 water-surface elevation profiles for a range in values for Manning's roughness coefficient.

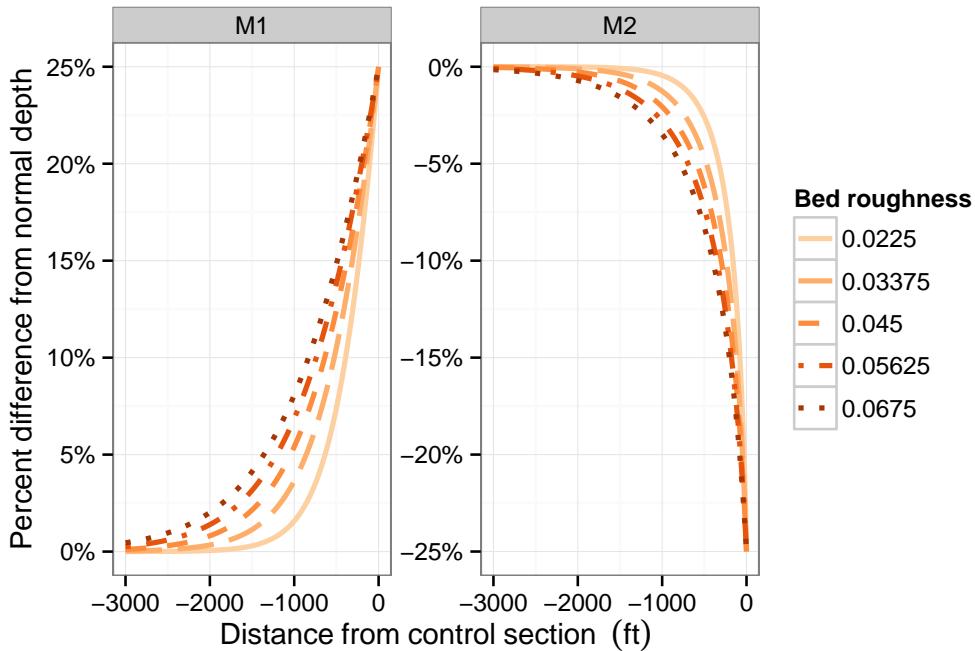


Figure 3: M1 and M2 water-surface elevation profiles as percent difference from the normal depth under different roughness conditions.

roughness affects the distance that a depth disturbance propagates upstream under sub-critical flow conditions. From this we can interpret Manning's coefficient to not only determine the steady-state water-surface profile, but also indicate the degree to which a disturbed channel resists adjustment towards said profile. Note that the M1 and M2 profiles are not symmetric; that is, the point at which the water depth returns to the normal depth from a 25% increase in water depth at the control section is further upstream compared to that for a 25% decrease in depth. This is because the velocity term in the energy equation is non-linear (u^2) and therefore the energy gradient is steeper for the M2 profile.

Computation of unsteady flows

Open-channel unsteady flows are typically described using the St. Venant equations, also known as the shallow water equations. The St. Venant equations are derived from the Navier-Stokes equations based on a number of simplifying assumptions, including uniform density and hydrostatic pressure conditions. The one-dimensional St. Venant equations are written as

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \quad (4)$$

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} (Qu + g\bar{y}A) - gA \left(S_0 - S_f \right) = 0 \quad (5)$$

where Eq. (4) is the continuity, or mass conservation, equation (assuming no infiltration or precipitation) and Eq. (5) is the momentum equation. The terms Q , u , A and S_0 and S_f are as defined previously and g is gravitational acceleration. The term \bar{y} results from averaging the pressure component and refers to the distance from the free surface to the centroid of the cross section. The `rivr` package provides multiple methods for solving these equations to compute unsteady flows through a prismatic channel. These methods are discussed in the following sections.

Simulations with the Kinematic Wave Model

The Kinematic Wave Model (KWM) provides a simplified solution to the St. Venant equations based on a truncated momentum equation that ignores inertial terms (Lighthill and Whitham, 1955). Eq. (5)

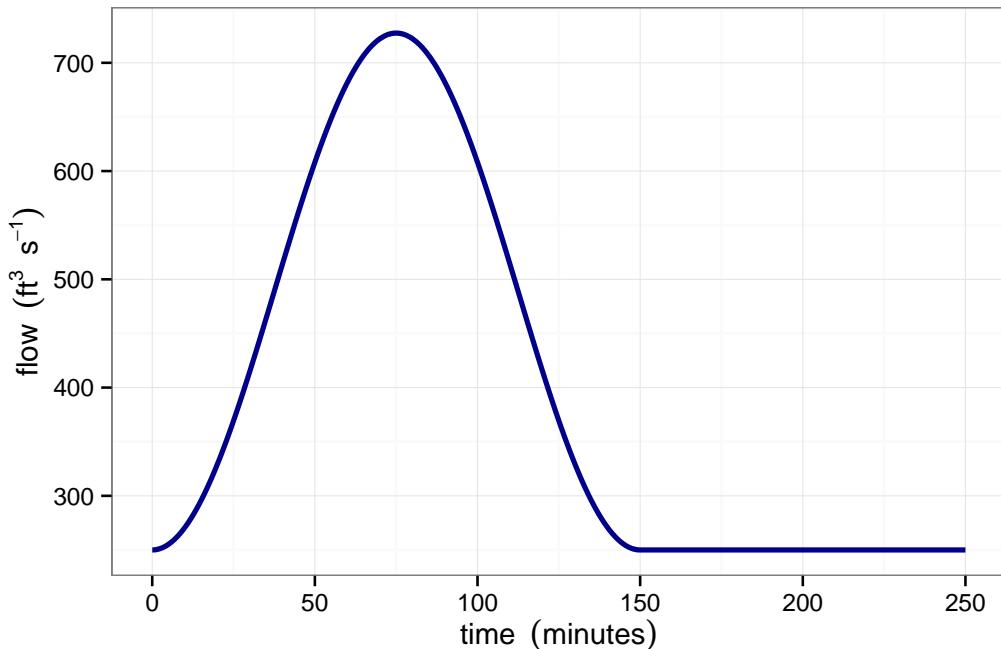


Figure 4: Flood wave hydrograph specified as the initial condition of the unsteady flow examples.

is instead expressed through the relation

$$A = \left(\frac{n Q P^{2/3}}{C_m \sqrt{S_0}} \right)^{3/5} \quad (6)$$

where P is the wetted perimeter and S_0 is substituted for S_f in Manning's equation.

The KWM engine (and other engines, as we show later) provided by **rivr** can be called via the function `route_wave`. The behavior of the KWM is demonstrated by using it to calculate the evolution of a flood wave down a 100-foot wide rectangular channel that is 150,000 feet long, with a slope of 0.001 and a Manning's roughness of 0.045. The boundary.condition argument is a vector specifying the flow at the upstream boundary for an arbitrary period of time in intervals equal to the timestep argument, in seconds. The monitor.nodes and monitor.times arguments are used to specify the function outputs. The hydrograph (change in flow over time) at the upstream boundary, shown in Figure 4, is modeled as a symmetrical flood wave and is conveniently expressed as

$$Q(t) = \begin{cases} 250 + \frac{750}{\pi} \left(1 - \cos \frac{\pi t}{4500} \right) & \text{if } t \leq 9000 \\ 250 & \text{if } t \geq 9000 \end{cases} \quad (7)$$

where t is in seconds.

```
slope <- 0.001      # channel slope (vertical ft / horizontal ft)
extent <- 150000    # channel length (ft)
mannings <- 0.045   # Manning's roughness
width <- 100        # channel bottom width
sideslope <- 0      # channel side slope (horizontal ft / vertical ft)
Cm <- 1.486         # conversion factor for Manning's equation
g <- 32.2           # gravitational acceleration (ft/s^2)

numnodes <- 601      # number of finite-difference nodes
dx <- extent/(numnodes - 1)  # distance between nodes (ft)
dt <- 100            # time interval (s)

monpoints <- c(1, 201, 401, 601)      # Nodes to monitor
montimes <- seq(1, length(boundary), by = 10)  # time steps to monitor
```

Figure 5: Progression of the flood wave calculated using the KWM engine with a spatial resolution of 25 feet and a temporal resolution of 12 seconds.

```

initflow <- 250                                # initial flow condition (ft^3/s)
times <- seq(0, 76000, by = dt)
boundary <- ifelse(times < 9000,                # upstream hydrograph (ft^3/s)
  250 + (750/pi) * (1 - cos(pi * times/(4500))), 250)

route_wave(slope, mannings, Cm, g, width, sideslope, initflow, boundary,
timestep = dt, spacestep = dx, numnodes = numnodes,
monitor.nodes = monpoints, monitor.times = montimes,
engine = "Kinematic")

```

While the KWM is easy to use and implement, it ignores wave damping and attenuation resulting in unrealistic predictions of the shape of the flood wave as it progresses downstream. The truncation of Eq. (5) results in what is known as “kinematic shock”, i.e. the shape of the flood wave becomes increasingly asymmetrical as it progresses downstream. This effect is apparent in Figure 5, which shows the wave front approaching a vertical line as the flood wave travels along the channel.

The **rivr** package implements the KWM with an explicit, first-order accurate backwards-difference scheme. While such schemes are simple to conceptualize and implement, they are sensitive to numerical error and can be unstable. A necessary condition for stability is that the Courant Number C is less than unity, i.e.

$$C \equiv \frac{u\Delta t}{\Delta x} \leq 1 \quad (8)$$

where u is defined here as the initial velocity at the upstream boundary and Δx and Δt are the spatial and temporal resolution of the model. Numerical error is propagated through successive calculations and results in a systematic decrease in the accuracy of the solution as the flood wave progresses downstream, apparent in Figure 5 as a decreasing peak flow magnitude. The sensitivity of the solution generated by the KWM engine to model resolution is assessed by varying Δx and Δt while maintaining a Courant number of 0.7 to ensure numerical stability. The effect of numerical error on model accuracy is inferred by evaluating the shape of the hydrograph at a cross-section 50,000 feet downstream. Figure 6 shows the shape of the flood wave at said cross-section for selected spatial resolutions. As resolution decreases, the peak flow downstream decreases due to the effects of numerical error. In addition, the severity of the kinematic shock decreases with decreasing resolution due to greater numerical dispersion. Numerical error can be reduced by applying finer model resolutions at the cost of increased computation time. Table 2 summarizes the effect of model resolution on the peak flow value 50,000 feet downstream; for the example shown here, increasing the model resolution incurs

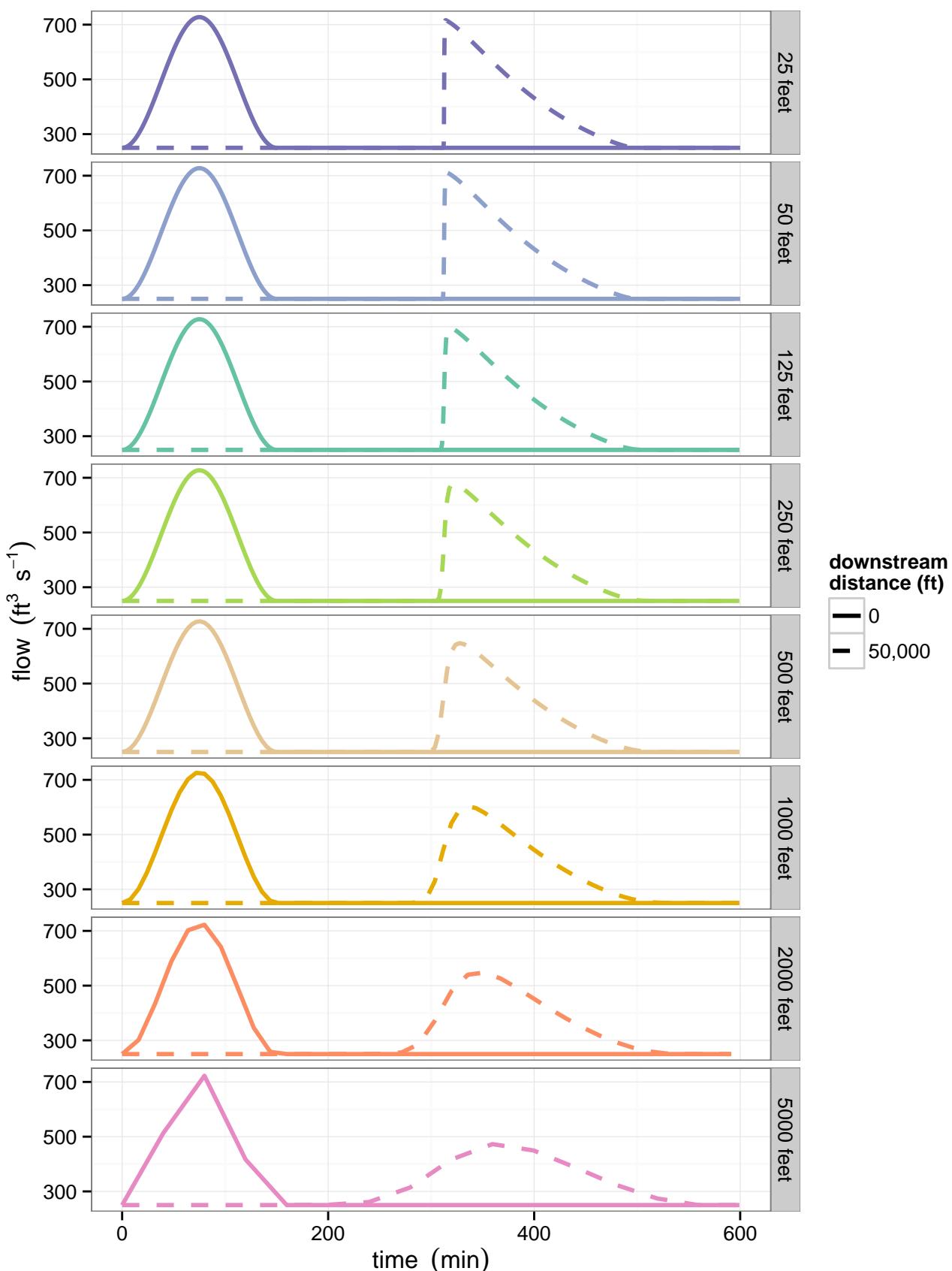


Figure 6: Evolution of the flood wave hydrograph 50,000 feet downstream using the KWM engine, for a variety of model resolutions. Note that at coarser resolutions the wave appears to attenuate, but this is solely the result of numerical error.

Δx (ft)	Δt (s)	% error (peak flow)	cost (s)
25	11.98	-1.53	221.02
50	23.97	-2.19	53.25
125	59.92	-4.04	8.48
250	119.84	-6.68	2.48
500	239.68	-10.96	0.50
1,000	479.36	-17.02	0.14
2,000	958.72	-24.69	0.05
5,000	2,396.80	-35.02	0.02

Table 2: KWM engine resolution and accuracy 50,000 feet downstream. Temporal resolution is matched to the selected spatial resolution to maintain a Courant number of 0.7. Finer resolutions reduce the impact of numerical error but show diminishing returns for resolutions finer than 50 feet. Computation times are for R v3.2.2 (x64) running on an Intel i7-4500U CPU with 8GB of RAM.

substantial costs in computation time with diminishing returns on accuracy for spatial resolutions finer than 50 feet.

Simulations with the Dynamic Wave Model

The previous example showed that the KWM gives unrealistic predictions of a flood wave routed through a prismatic channel with mild slope. This is largely due to the exclusion of inertial forces by truncating the momentum equation. Unsteady flow models that solve complete forms of Eq. (5), known as Dynamic Wave Models (DWMs), can provide more realistic simulations of unsteady flow. The **rivr** package provides a DWM engine accessed in a similar manner to the KWM engine. The DWM engine uses the MacCormack predictor-corrector scheme (MacCormack, 2003) to provide solutions that are second-order accurate in space. As shown in Figure 7, flood waves modeled with the DWM attenuate as they travel through the channel.

The DWM engine differs from the KWM engine in that the downstream boundary condition must be known. The **rivr** package employs the Method of Characteristics (MOC) to resolve fluid flow across the upstream and downstream boundaries. By using MOC, the DWM engine can accommodate a variety of boundary specifications; the upstream and downstream boundaries can be individually specified in terms of either flows or water depths. The downstream boundary can alternatively be specified as a zero-gradient condition, which allows flow to be routed out of the channel by assuming that the flow or water depth is constant across the final (downstream) two nodes in the model domain. This results in “smearing” of the flood wave at the downstream boundary, but in some practical cases this may be preferable to direct specification of depth or flow. When depths are specified at the boundaries, the characteristic equations are solved directly; when flows are specified, a Newton-Raphson scheme is used to iteratively solve for depths and velocities simultaneously.

To illustrate use of the DWM, the example from the previous section is repeated. The channel geometry, extent and upstream boundary condition are unchanged. A zero-gradient condition is specified at the downstream boundary. Example code using the DWM engine is shown below.

```
slope <- 0.001      # channel slope (vertical ft / horizontal ft)
extent <- 150000    # channel length (ft)
mannings <- 0.045   # Manning's roughness
width <- 100        # channel bottom width
sideslope <- 0       # channel side slope (horizontal ft / vertical ft)
Cm <- 1.486         # conversion factor for Manning's equation
g <- 32.2           # gravitational acceleration (ft/s^2)

numnodes <- 601      # number of finite-difference nodes
dx <- extent/(numnodes - 1)  # distance between nodes (ft)
dt <- 100            # time interval (s)

monpoints <- c(1, 201, 401, 601)      # Nodes to monitor
montimes <- seq(1, length(boundary), by = 10) # time steps to monitor

initflow <- 250      # initial flow condition (ft^3/s)
times <- seq(0, 76000, by = dt)
```

Figure 7: Progression of the flood wave simulated using the DWM engine with a spatial resolution of 50 feet and a temporal resolution of 2.1 seconds. Flow is routed out of the channel by assuming the flow gradient is zero at the downstream boundary.

Δx (ft)	Δt (s)	% error (peak flow)	% error (time to peak)	cost (s)
50	2.05	0.79	0.98	1,690.05
125	5.13	0.77	1.03	268.67
250	10.27	0.74	1.11	67.32
500	20.54	0.68	1.26	16.87
1,000	41.07	0.44	1.96	4.28

Table 3: DWM engine resolution and accuracy 50,000 feet downstream. Error values for peak flow magnitude and timing are based on a standard benchmark ([Sobey, 2001](#)).

```

boundary <- ifelse(times < 9000,                                # upstream hyrograph (ft^3/s)
  250 + (750/pi) * (1 - cos(pi * times/(4500))), 250)
downstream <- rep(-1, length(boundary))                         # values < 0 specify zero-gradient

route_wave(slope, mannings, Cm, g, width, sideslope, initflow, boundary,
           downstream, timestep = dt, spacestep = dx, numnodes = numnodes,
           monitor.nodes = mpoints, monitor.times = mtimes, engine = "Dynamic",
           boundary.type = "QQ")

```

The DWM engine, like the KWM engine, uses an explicit scheme and is subject to stability constraints. In fact, the DWM engine is more prone to numerical instability compared to the KWM engine and typically requires Courant numbers on the order of 0.06 (i.e. an order-of-magnitude increase in the required temporal resolution compared to the KWM engine for a given spatial resolution). This disadvantage is offset by a significant reduction in numerical error provided by the advanced finite-differencing scheme. As shown in Figure 8, spatial resolutions of 1,000 feet provide accurate predictions at 50,000 feet downstream, at the cost of severe instability near the downstream boundary. Resolutions of 500 feet provide stable solutions throughout the full channel extent for a modest computation cost, as shown in Table 3.

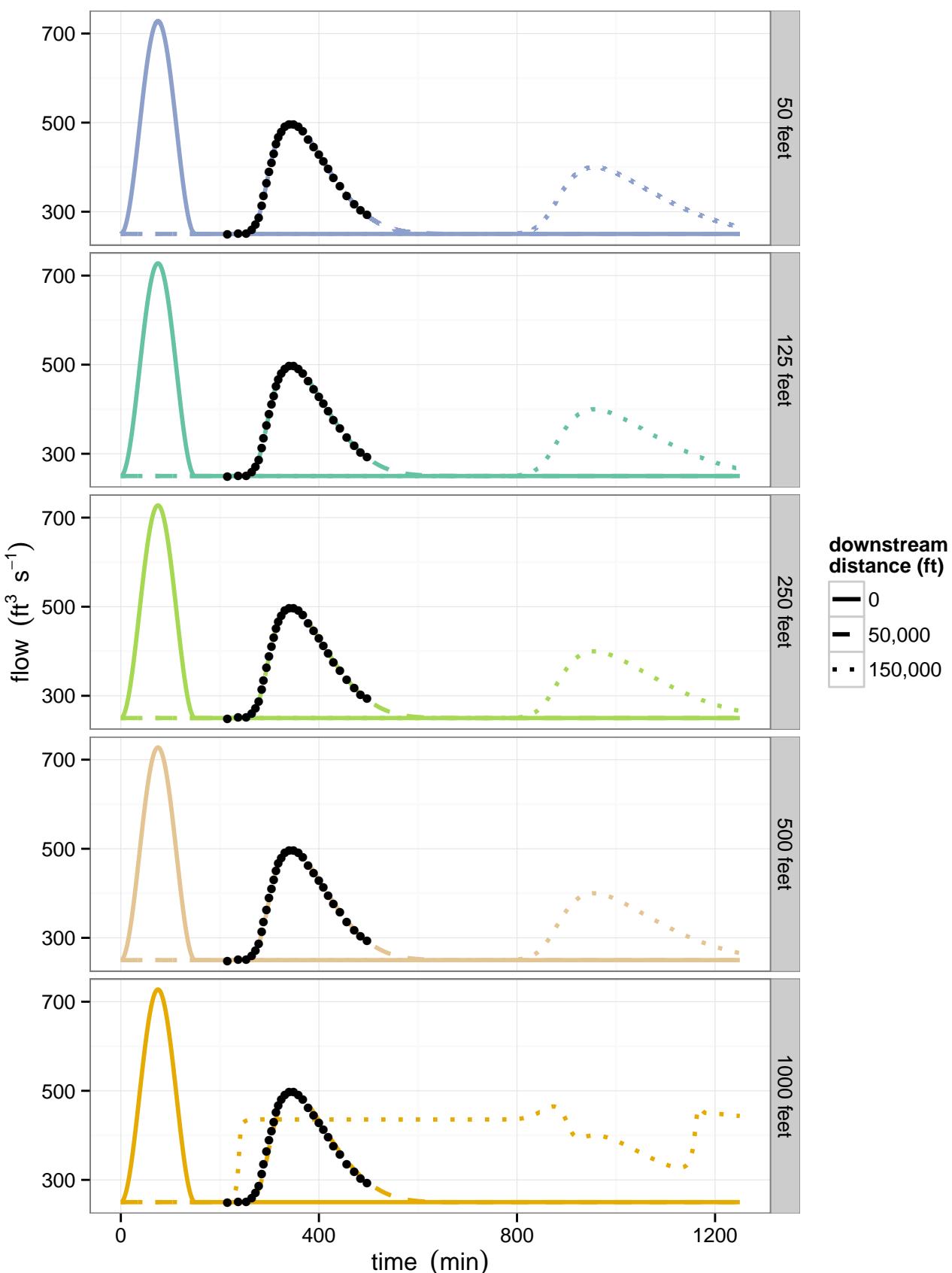


Figure 8: Evolution of the flood wave across both boundaries and at 50,000 feet downstream, for a variety of model resolutions. Validation data from Sobey (2001) are shown as points. The solution becomes unstable at very coarse resolutions but otherwise maintains a high degree of accuracy.

Simulations with discontinuous boundary conditions

The DWM engine can accommodate boundary conditions that specify zero flows (velocities) but non-zero depths, such as can occur by the sudden closure of sluice gates. The last unsteady-flow example illustrates use of the DWM engine to simulate unsteady flow conditions following sudden closure of a sluice gate at the downstream end of a trapezoidal spillway. This example mimics example 14-1 in Chaudhry (2007). The upstream boundary condition is specified as a constant depth to represent water level in a large reservoir. Example code for specifying discontinuous boundary conditions is shown below.

```
slope <- 0.00008      # channel slope (vertical m / horizontal m)
extent <- 5000        # channel length (m)
mannings <- 0.013     # Manning's roughness
width <- 6.1          # channel width (m)
sideslope <- 1.5      # channel sideslope (horizontal m / vertical m)
g <- 9.81              # gravitational acceleration (m/s^2)
Cm <- 1.0              # conversion factor for Manning's equation (SI units)
extent <- 5000        # channel length (m)

numnodes <- 51         # number of nodes
dx <- extent/(numnodes - 1)  # spatial resolution (m)
dt <- 10                # temporal resolution (s)

mp <- c(1, 16, 26, 31, 51)  # monitor nodes
mt <- c(1, 61, 101, 161, 201) # monitor time steps

ic <- 126                # initial condition (m^3/s)
bctime <- 2000             # constant depth upstream (m)
bc <- rep(5.79, round(bctime/dt) + 1) # flow dropped to 0 on first timestep
dc <- rep(0, length(bc))   # flow dropped to 0 on first timestep

results <- route_wave(slope, mannings, Cm, g, width, sideslope, ic, bc, dc,
timestep = dt, spacetime = dx, numnodes = numnodes, engine = "Dynamic",
boundary.type = "yQ", monitor.nodes = mp, monitor.times = mt)
```

Figure 9 shows that the MacCormack scheme can produce small instabilities in the solution where gradients are high. The **rivr** package provides access to an alternative DWM engine—the Lax diffusive scheme (Lax, 1954)—which is also second-order accurate in space. Whereas the Lax scheme is less accurate than the MacCormack scheme at coarser resolutions, it can provide smoother solutions to discontinuous boundary problems. The Lax scheme can be accessed with `route_wave` via the argument `scheme = "Lax"`.

Opportunities for extension

The **rivr** package was designed such that additional analysis capabilities could be implemented in future versions with minimal modification to the existing code base. A wide variety of extensions could be supported by the existing interface, both in terms of numerical methods and problem complexity. In particular, there are many avenues for contribution to the unsteady flow analysis capabilities of **rivr**.

Additional numerical schemes for unsteady flow analysis in prismatic channels—such as implicit finite-difference schemes and finite-element methods—could be implemented with only superficial changes to the current function interface provided by `route_wave`, i.e. new algorithms could be made accessible through additional keyword options in existing arguments provided the underlying engines followed the input template used by the KWM and DWM engines. Variable-step methods could also be implemented with minimal changes, most simply by allowing the user to define an initial or “default” time step and using an interpolation function to compute the upstream boundary condition at intermediate time steps when required. The authors consider the degree of effort required to implement such additional numerical schemes in **rivr** to be on par with typical expectations for an individual project assignment in a graduate-level computational hydraulics or numerical methods course.

Support for prismatic channels with arbitrary or compound geometry could be accomplished with only minor restructuring of the channel geometry function, but would introduce additional complexity to the specification of channel roughness (Yen, 2002) and the computation of normal and

Figure 9: Evolution of the water-surface profile through time following sudden closure of a control gate. The Lax scheme produces smoother profiles, while the MacCormack scheme predicts steeper gradients and produces small numerical instabilities.

critical depth ([Chaudhry and Bhallamudi, 1988](#); [Younis et al., 2009](#)). Support for compound channels is currently being considered for the next major version release of **rivr**.

Specification of variable channel geometry and slope, additional friction terms (e.g. bedform drag, hydraulic structures) and additional source or sink terms (e.g. hillslope runoff, infiltration) would require some modification to the current implementation but would largely consist of allowing users to specify channel properties as vectors rather than as single values, and modifying the underlying C++ code to access the appropriate vector elements when computing solutions at individual nodes. These modifications would best be developed prior to, or in conjunction with, the implementation of an implicit scheme. Such changes would, however, add considerable complexity to the formulation of unsteady flow problems and could potentially distract from the primary purpose of the package as an accessible tool for teaching.

Support for channel networks would likely require the development of a new function interface that would essentially couple three or more instances of `route_wave` (e.g. two channels merging into one) and provide a keyword argument for selecting different methods for resolving flow depth and velocity at the junctions. Implementing such functionality would likely be a much larger endeavor compared to the other extensions discussed, and could pose additional challenges in terms of obtaining reasonably-fast solutions for moderately-large channel networks. It is important to note, however, that discussion of channel network modeling is often neglected in open-channel hydraulics courses due to the high complexity of problem formulation and solution algorithms; making channel network solution algorithms demonstrable through **rivr** would therefore be of considerable educational value.

Concluding remarks

This article describes **rivr**, a first-of-its-kind R/C++ package for one-dimensional open-channel flow computation with educational applications to hydraulic engineering degree programs and related disciplines. The package provides a reliable and flexible toolset for modeling open-channel flows via a simple function interface. Model outputs are formatted to facilitate analysis, tabulation, visualization and export. Example computations of backwater curves and flood wave routing are provided to demonstrate functionality and validate model results, and potential extensions to the package are discussed. This article, along with the package documentation, provides information on the governing equations and numerical implementations as well as a suite of use examples.

Bibliography

- W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2015.
URL <https://CRAN.R-project.org/package=shiny>. R package version 0.12.2. [p150, 250, 279]
- M. H. Chaudhry. *Open-Channel Flow*. Springer, 2007. [p251, 260]
- M. H. Chaudhry and S. M. Bhallamudi. Computation of critical depth in symmetrical compound channels. *Journal of Hydraulic Research*, 26(4):377–396, 1988. [p261]
- P. D. Doubt. Classification system for varied flow in prismatic channels. Technical Release 47, United States Department of Agriculture, Soil Conservation Service, Engineering Division, Littleton, Colorado, USA, Feb. 1971. [p251]
- D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL <http://www.jstatsoft.org/v40/i08/>. [p250]
- M. C. Koohofkan. *rivr: Steady and Unsteady Open-Channel Flow Computation*, 2015. URL <https://CRAN.R-project.org/package=rivr>. R package version 1.1. [p249]
- P. D. Lax. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications on Pure and Applied Mathematics*, 7(1):159–193, 1954. [p260]
- L. B. Leopold. Downstream change of velocity in rivers. *American Journal of Science*, 251(8):606–624, 1953. [p249]
- M. J. Lighthill and G. B. Whitham. On kinematic waves. I. flood movement in long rivers. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 229(1178):281–316, 1955. [p253]
- R. MacCormack. The effect of viscosity in hypervelocity impact cratering. *Journal of Spacecraft and Rockets*, 40(5):757–763, 2003. [p257]
- R. J. Sobey. H11: Hydrograph routing. In *Review of One-Dimensional Hydrodynamic and Transport Models*, chapter 14. Bay-Delta Modeling Forum, June 2001. URL <http://www.cwemf.org/1-DReview/>. [p258, 259]
- Y. Xie. knitr: A comprehensive tool for reproducible research in R. In V. Stodden, F. Leisch, and R. D. Peng, editors, *Implementing Reproducible Computational Research*. Chapman and Hall/CRC, 2014. URL <http://www.crcpress.com/product/isbn/9781466561595>. [p250]
- B. C. Yen. Open channel flow resistance. *Journal of Hydraulic Engineering*, 128(1):20–39, 2002. [p260]
- B. A. Younis, V. Sousa, and I. Meireles. Prediction of the asymptotic water depth in rough compound channels. *Journal of Irrigation and Drainage Engineering*, 135(2):231–234, 2009. [p261]

Michael C. Koohofkan
Hydrologic Sciences Graduate Group
University of California, Davis, CA 95616, USA
koohofkan@ucdavis.edu

Bassam A. Younis
Department of Civil and Environmental Engineering
University of California, Davis, CA 95616, USA
bayounis@ucdavis.edu

Generalized Hermite Distribution Modelling with the R Package `hermite`

by David Moriña, Manuel Higueras, Pedro Puig and María Oliveira

Abstract The Generalized Hermite distribution (and the Hermite distribution as a particular case) is often used for fitting count data in the presence of overdispersion or multimodality. Despite this, to our knowledge, no standard software packages have implemented specific functions to compute basic probabilities and make simple statistical inference based on these distributions. We present here a set of computational tools that allows the user to face these difficulties by modelling with the Generalized Hermite distribution using the R package `hermite`. The package can also be used to generate random deviates from a Generalized Hermite distribution and to use basic functions to compute probabilities (density, cumulative density and quantile functions are available), to estimate parameters using the maximum likelihood method and to perform the likelihood ratio test for Poisson assumption against a Generalized Hermite alternative. In order to improve the density and quantile functions performance when the parameters are large, Edgeworth and Cornish-Fisher expansions have been used. Hermite regression is also a useful tool for modeling inflated count data, so its inclusion to a commonly used software like R will make this tool available to a wide range of potential users. Some examples of usage in several fields of application are also given.

Introduction

The Poisson distribution is without a doubt the most common when dealing with count data. There are several reasons for that, including the fact that the maximum likelihood estimate of the population mean is the sample mean and the property that this distribution is closed under convolutions (see Johnson et al. 2005). However, it is very common in practice that data presents overdispersion or zero inflation, cases where the Poisson assumption does not hold. In these situations it is reasonable to consider discrete distributions with more than one parameter. The class of all two-parameter discrete distributions closed under convolutions and satisfying that the sample mean is the maximum likelihood estimator of the population mean are characterized in Puig (2003). One of these families is just the Generalized Hermite distribution. Several generalizations of the Poisson distribution have been considered in the literature (see, for instance, Gurland 1957; Lukacs 1970; Kemp and Kemp 1965, 1966), that are *compound-Poisson* or *contagious* distributions. They are families with probability generating function (PGF) defined by

$$P(s) = \exp(\lambda(f(s) - 1)) = \exp(a_1(s - 1) + a_2(s^2 - 1) + \cdots + a_m(s^m - 1) + \cdots), \quad (1)$$

where $f(s)$ is also a PGF and $\sum_{i=1}^m a_i = \lambda$. Many well known discrete distributions are included in these families, like the Negative Binomial, Polya-Aeppli or the Neyman A distributions. The Generalized Hermite distribution was first introduced in Gupta and Jain (1974) as the situation where a_m is significant compared to a_1 in (1), while all the other terms a_i are negligible, resulting in the PGF

$$P(s) = \exp(a_1(s - 1) + a_m(s^m - 1)). \quad (2)$$

After fixing the value of the positive integer $m \geq 2$, the *order* or *degree* of the distribution, the domain of the parameters is $a_1 > 0$ and $a_m > 0$. Note that when a_m tends to zero, the distribution tends to a Poisson. Otherwise, when a_1 tends to zero it tends to m times a Poisson distribution. It is immediate to see that the PGF in (2) is the same than the PGF of $X_1 + mX_2$, where X_i are independent Poisson distributed random variables with population mean a_1 and a_m respectively. From here, it is straightforward to calculate the population mean, variance, skewness and excess kurtosis of the Generalized Hermite distribution:

$$\begin{aligned} \mu &= a_1 + ma_m, & \sigma^2 &= a_1 + m^2a_m, \\ \gamma_1 &= \frac{a_1 + m^3a_m}{(a_1 + m^2a_m)^{3/2}}, & \gamma_2 &= \frac{a_1 + m^4a_m}{(a_1 + m^2a_m)^2}. \end{aligned} \quad (3)$$

A useful expression for the probability mass function of the Generalized Hermite distribution in terms of the population mean μ and the population index of dispersion $d = \sigma^2/\mu$ is provided in Puig (2003).

$$P(Y = k) = P(Y = 0) \frac{\mu^k(m-d)^k}{(m-1)^k} \sum_{j=0}^{[k/m]} \frac{(d-1)^j(m-1)^{(m-1)j}}{m^j \mu^{(m-1)j} (m-d)^{mj} (k-mj)! j!}, \quad k = 0, 1, \dots \quad (4)$$

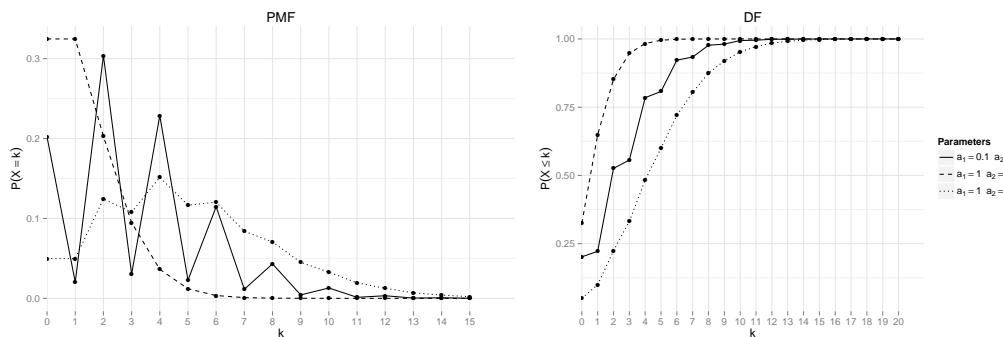


Figure 1: Hermite probability mass and distribution functions for the indicated parameter values.

where $P(Y = 0) = \exp(\mu(-1 + \frac{d-1}{m}))$ and $[k/m]$ is the integer part of $\frac{k}{m}$. Note that m can be expressed as $m = \frac{d-1}{1+\log(p_0)/\mu}$. Because the denominator is a measure of zero inflation, m can be understood as an index of the relationship between the overdispersion and the zero inflation.

The probabilities can be also written in terms of the parameters a_1, a_m using the identities given in (3).

The case $m = 2$ in (2) is covered in detail in Kemp and Kemp (1965) and Kemp and Kemp (1966) and the resulting distribution is simply called Hermite distribution. In that case, the probability mass function, in terms of the parameters a_1 and a_2 , has the expression

$$P(Y = k) = e^{-a_1 - a_2} \sum_{j=0}^{[k/2]} \frac{a_1^{k-2j} a_2^j}{(k-2j)! j!}, k = 0, 1, \dots \quad (5)$$

The probability mass function and the distribution function for some values of a_1 and a_2 are shown in Figure 1.

Gupta and Jain (1974) also develop a recurrence relation that can be used to calculate the probabilities in a numerically efficient way:

$$p_k = \frac{\mu}{k(m-1)} (p_{k-m}(d-1) + p_{k-1}(m-d)), k \geq m, \quad (6)$$

where $p_k = P(Y = k)$ and the first values can be computed as $p_k = p_0 \frac{\mu^k}{k!} \left(\frac{m-d}{m-1}\right)^k, k = 1, \dots, m-1$. Although overdispersion or multimodality are common situations when dealing with count data and the Generalized Hermite distribution provides an appropriate framework to face these situations, the use of techniques based on this distribution was not easy in practice as they were not available in any standard statistical software. A description of the **hermite** package's main functionalities will be given in Section [Package hermite](#). Several examples of application in different fields will be discussed in Section [Examples](#), and finally some conclusions will be commented in Section [Conclusions](#).

Package hermite

Like the common distributions in R, the package **hermite** implements the probability mass function (**dhermite**), the distribution function (**phermite**), the quantile function (**qhermite**) and a function for random generation (**rhermite**) for the Generalized Hermite distribution. It also includes the function **glm.hermite**, which allows to calculate, for a univariate sample of independent draws, the maximum likelihood estimates for the parameters and to perform the likelihood ratio test for a Poisson null hypothesis against a Generalized Hermite alternative. This function can also carry out Hermite regression including covariates for the population mean, in a very similar way to that of the well known R function **glm**.

Probability mass function

The probability mass function of the Generalized Hermite distribution is implemented in **hermite** through the function **dhermite**. A call to this function might be

```
dhermite(x, a, b, m = 2)
```

The description of these arguments can be summarized as follows:

- x : Vector of non-negative integer values.
- a : First parameter for the Hermite distribution.
- b : Second parameter for the Hermite distribution.
- m : Degree of the Generalized Hermite distribution. Its default value is 2, corresponding to the classical Hermite distribution introduced in [Kemp and Kemp \(1965\)](#).

The recurrence relation (6) is used by `dhermite` for the computation of probabilities. For large values of any parameter a or b (above 20), the probability of Y taking x counts is approximated using an Edgeworth expansion of the distribution function (7), i.e. $P(Y = x) = F_H(x) - F_H(x - 1)$. The Edgeworth expansion does not guarantee positive values for the probabilities in the tails, so in case this approximation returns a negative probability, the probability is calculated by using the normal approximation $P(Y = x) = \Phi(x^+) - \Phi(x^-)$ where Φ is the standard normal distribution function and

$$x^\pm = \frac{x \pm 0.5 - a - mb}{\sqrt{a + m^2 b}}$$

are the typified continuous corrections.

The normal approximation is justified taking into account the representation of any Generalized Hermite random variable Y , as $Y = X_1 + mX_2$ where X_i are independent Poisson distributed with population means a, b . Therefore, for large values of a or b , the Poissons are well approximated by normal distributions.

Distribution function

The distribution function of the Generalized Hermite distribution is implemented in `hermite` through the function `phermite`. A call to this function might be

```
phermite(q, a, b, m = 2, lower.tail = TRUE)
```

The description of these arguments can be summarized as follows:

- q : Vector of non-negative integer quantiles.
- `lower.tail`: Logical; if `TRUE` (the default value), the computed probabilities are $P(Y \leq x)$, otherwise, $P(Y > x)$.

All remaining arguments are defined as specified for `dhermite`.

If a and b are large enough (a or $b > 20$), X_1 and X_2 are approximated by $N(a, \sqrt{a})$ and $N(b, \sqrt{b})$ respectively, so Y can be approximated by a normal distribution with mean $a + mb$ and variance $a + m^2 b$. This normal approximation is improved by means of an Edgeworth expansion ([Barndorff-Nielsen and Cox, 1989](#)), using the following expression

$$F_H(x) \approx \Phi(x^*) - \phi(x^*) \cdot \left(\frac{1}{6} \gamma_1 H_{e2}(x^*) + \frac{1}{24} \gamma_2 H_{e3}(x^*) + \frac{1}{72} \gamma_1^2 H_{e5}(x^*) \right), \quad (7)$$

where Φ and ϕ are the typified normal distribution and density functions respectively, $H_{en}(x)$ are the n th-degree probabilists' Hermite polynomials ([Barndorff-Nielsen and Cox, 1989](#))

$$\begin{aligned} H_{e2}(x) &= x^2 - 1 \\ H_{e3}(x) &= x^3 - 3x \\ H_{e5}(y) &= x^5 - 10x^3 + 15x, \end{aligned}$$

x^* is the typified continuous correction of x considered in [Pace and Salvan \(1997\)](#)

$$x^* = 1 + \frac{1}{24(a + m^2 b)} \cdot \frac{x + 0.5 - a - mb}{\sqrt{a + m^2 b}},$$

and γ_1 and γ_2 are respectively the skewness and the excess kurtosis of Y expressed in (3).

Quantile function

The quantile function of the Generalized Hermite distribution is implemented in `hermite` through the function `qhermite`. A call to this function might be

```
qhermite(p, a, b, m = 2, lower.tail = TRUE)
```

The description of these arguments can be summarized as follows:

- p : Vector of probabilities.

All remaining arguments are defined as specified for `hermite`. The quantile is right continuous: `qhermite(p,a,b,m)` is the smallest integer x such that $P(Y \leq x) \geq p$, where Y follows an m -th order Hermite distribution with parameters a and b .

When the parameters a or b are over 20, a Cornish-Fisher expansion is used (Barndorff-Nielsen and Cox, 1989) to approximate the quantile function. The Cornish-Fisher expansion uses the following expression

$$y_p \approx \left(u_p + \frac{1}{6} \gamma_1 H e_2(u_p) + \frac{1}{24} \gamma_2 H e_3(u_p) - \frac{1}{36} \gamma_1^2 (2u_p^3 - 5u_p) \right) \sqrt{a + m^2 b} + a + mb,$$

where u_p is the p quantile of the typified normal distribution.

Random generation

The random generation function `rhermite` uses the relationship between Poisson and Hermite distributions detailed in Sections [Introduction](#) and [Probability mass function](#). A call to this function might be

```
rhermite(n, a, b, m = 2)
```

The description of these arguments can be summarized as follows:

- n : Number of random values to return.

All remaining arguments are defined as specified for `dhermite`.

Maximum likelihood estimation and Hermite regression

Given a sample $X = x_1, \dots, x_n$ of a population coming from a generalized Hermite distribution with mean μ , index of dispersion d and order m , the log-likelihood function is

$$l(X; \mu, d) = n \cdot \mu \cdot \left(-1 + \frac{d-1}{m} \right) + \log \left(\frac{\mu(m-d)}{m-1} \right) \sum_{i=1}^n x_i + \sum_{i=1}^n \log(q_i(\theta)), \quad (8)$$

where $q_i(\theta) = \sum_{j=0}^{[x_i/m]} \frac{\theta^j}{(x_i-mj)!j!}$ and $\theta = \frac{(d-1)(m-1)^{(m-1)}}{m\mu^{(m-1)}(m-d)^m}$.

The maximum likelihood equations do not always have a solution. This is due to the fact that this is not a regular family of distributions because its domain of parameters is not an open set. The following result gives a sufficient and necessary condition for the existence of such a solution (Puig, 2003):

Proposition 1 Let x_1, \dots, x_n be a random sample from a generalized Hermite population with fixed m . Then, the maximum likelihood equations have a solution if and only if $\frac{\mu^{(m)}}{\bar{x}^m} > 1$, where \bar{x} is the sample mean and $\mu^{(m)}$ is the m -th order sample factorial moment, $\mu^{(m)} = \frac{1}{n} \cdot \sum_{i=1}^n x_i(x_i - 1) \cdots (x_i - m + 1)$.

If the likelihood equations do not have a solution, the maximum of the likelihood function (8) is attained at the border of the domain of parameters, that is, $\hat{\mu} = \bar{x}$, $\hat{d} = 1$ (Poisson distribution), or $\hat{\mu} = \bar{x}$, $\hat{d} = m$ (m times a Poisson distribution). The case $\hat{\mu} = \bar{x}$, $\hat{d} = m$ corresponds to the very improbable situation where all the observed values were multiples of m . Then, in general, when the condition of Proposition 1 is not satisfied, the maximum likelihood estimators are $\hat{\mu} = \bar{x}$, $\hat{d} = 1$. This means that the data is fitted assuming a Poisson distribution.

The package `hermite` allows to estimate the parameters μ and d given an univariate sample by means of the function `glm.hermite`:

```
glm.hermite(formula, data, link = "log", start = NULL, m = NULL)
```

The description of the arguments can be summarized as follows:

- `formula`: Symbolic description of the model. A typical predictor has the form `response ~ terms` where `response` is the (numeric) response vector and `terms` is a series of terms which specifies a linear predictor for `response`.

- **data:** An optional data frame containing the variables in the model.
- **link:** Character specification of the population mean link function: "log" or "identity". By default link = "log".
- **start:** A vector containing the starting values for the parameters of the specified model. Its default value is NULL.
- **m:** Value for parameter m . Its default value is NULL, and in that case it will be estimated as \hat{m} , more details below.

The returned value is an object of class `glm.hermite`, which is a list including the following components:

- **coefs:** The vector of coefficients.
- **loglik:** Log-likelihood of the fitted model.
- **vcov:** Covariance matrix of all coefficients in the model (derived from the Hessian returned by the `maxLik()` output).
- **hess:** Hessian matrix, returned by the `maxLik()` output.
- **fitted.values:** The fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
- **w:** Likelihood ratio test statistic.
- **pval:** Likelihood ratio test p-value.

If the condition given in Proposition 1 is not met for a sample x , the `glm.hermite` function provides the maximum likelihood estimates $\hat{\mu} = \bar{x}$ and $\hat{d} = 1$ and a warning message advising the user that the MLE equations have no solutions.

The function `glm.hermite` can also be used for Hermite regression as described below and as will be shown through practical examples in Sections [Giles \(2007\)](#) and [DiGiorgio et al. \(2004\)](#).

Covariates can be incorporated into the model in various ways (see, e.g., [Giles 2007](#)). In function `glm.hermite`, the distribution is specified in terms of the dispersion index and its mean, which is then related to explanatory variables as in linear regression or other generalized linear models. That is, for Hermite regression, we assume Y_i follows a generalized Hermite distribution of order m , where we retain the dispersion index $d (> 1)$ as a parameter to be estimated and let the mean μ_i for the i -th observation vary as a function of the covariates for that observation, i.e., $\mu_i = h(\mathbf{x}_i^t \boldsymbol{\beta})$, where \mathbf{x}_i is a vector of covariates, t denotes the transpose vector, $\boldsymbol{\beta}$ is the corresponding vector of coefficients to be estimated and h is a link function. Note that because the dispersion index d is taken to be constant, this is a linear mean-variance (NB1) regression model.

The link function provides the relationship between the linear predictor and the mean of the distribution function. Although the log is the canonical link for count data as it ensures that all the fitted values are positive, the choice of the link function can be somewhat influenced by the context and data to be treated. For example, the identity link is the accepted standard in biodosimetry as there is no evidence that the increase of chromosomal aberration counts with dose is of exponential shape ([IAEA, 2011](#)). Therefore, function `glm.hermite` allows both link functions.

A consequence of using the identity-link is that the maximum likelihood estimate of the parameters obtained by maximizing the log-likelihood function of the corresponding model may lead to negative values for the mean. Therefore, in order to avoid negative values for the mean, constraints in the domain of the parameters must be included when the model is fitted. In function `glm.hermite`, this is carried out by using the function `maxLik` from package `maxLik` ([Henningsen and Toomet, 2011](#)), which is used internally for maximizing the corresponding log-likelihood function. This function allows constraints which are needed when the identity-link is used.

It should also be noted that results may depend of the starting values provided to the optimization routines. If no starting values are supplied, the starting values for the coefficients are computed/fixed internally. Specifically, when the log-link is specified, the starting values are obtained by fitting a standard Poisson regression model through a call to the internal function `glm.fit` from package `stats`. If the link function is the identity and no initial values are provided by the user, the function takes 1 as initial value for the coefficients. In both cases, the initial value for the dispersion index \hat{d} is taken to be 1.1.

Regarding the order of the Hermite distribution, it can be fixed by the user. If it is not provided (default option), when the model includes covariates, the order \hat{m} is selected by discretized maximum likelihood method, fitting the coefficients for each value of \hat{m} between 1 (Poisson) and 10, and selecting the case that maximizes the likelihood. In addition, if no covariates are included in the model and no initial values are supplied by the user, the naïve estimate $\hat{m} = \frac{s^2/\bar{x}-1}{1+\log(p_0)/\bar{x}}$, where p_0 is the proportion

of zeros in the sample is also considered. In the unlikely case the function returns $\hat{m} = 10$, we recommend to check the likelihood of the next orders ($m = 11, 12, \dots$) fixing this parameter in the function until a local minimum is found.

When dealing with the Generalized Hermite distribution it seems natural to wonder if data could be fitted by using a Poisson distribution. Because the Poisson distribution is included in the Generalized Hermite family, this is equivalent to test the null hypothesis $H_0 : d = 1$ against the alternative $H_1 : d > 1$. To do this, an immediate solution is to use the likelihood ratio test, which test-statistic is given by $W = 2 \left(l(X; \hat{\mu}, \hat{d}) - l(X; \hat{\mu}, 1) \right)$, where l is the log-likelihood function.

Under the null hypothesis W is not asymptotically χ^2_1 distributed as usual, because $d = 1$ is on the border of the domain of the parameters. Using the results of [Self and Liang \(1987\)](#) and [Geyer \(1994\)](#) it can be shown that in this case the asymptotic distribution of W is a 50:50 mixture of a zero constant and a χ^2_1 distribution. The α percentile for this mixture is the same as the 2α upper tail percentile for a χ^2_1 ([Puig, 2003](#)). The likelihood ratio test is also performed through `glm.hermite` function, using the maximum likelihood estimates $\hat{\mu}$ and \hat{d} .

A summary method for objects of class `glm.hermite` is included in the **hermite** package, giving a summary of relevant information, including the residuals minimum, maximum, median and first and third quartiles, the table of coefficients including the corresponding standard errors and significance tests based on the Normal reference distribution for regression coefficients and the likelihood ratio test against the Poisson distribution for the dispersion index. The AIC value for the proposed model is also reported.

Examples

Several examples of application of the package **hermite** in a wide range of contexts are discussed in this section, including classical and recent real datasets and simulated data.

Hartenstein (1961)

This example by [Hartenstein \(1961\)](#) describes the counts of Collenbola microarthropods in 200 samples of forest soil. The frequency distribution is shown in Table 1.

Microarthropods per sample	0	1	2	3	4	5
Frequency	122	40	14	16	6	2

Table 1: Frequency distribution of Collenbola microarthropods.

This dataset was analyzed in [Puig \(2003\)](#) with a Generalized Hermite distribution of order $m = 3$. The maximum likelihood estimation gave a mean of $\hat{\mu} = \bar{x} = 0.75$, and an index of dispersion of $\hat{d} = 1.8906$.

Using `glm.hermite` we calculate the parameter estimates:

```
> library("hermite")
> data <- c(rep(0, 122), rep(1, 40), rep(2, 14), rep(3, 16), rep(4, 6), rep(5, 2))
> mle1 <- glm.hermite(data ~ 1, link = "log", start =NULL, m = 3)$coefs
  (Intercept) dispersion.index          order
  -0.2875851      1.8905920      3.0000000
```

We can see that these parameter estimates are equivalent to those reported in [Puig \(2003\)](#).

The estimated expected frequencies are shown in Table 2.

Microarthropods per sample	0	1	2	3	4	5
Frequency	118.03	49.11	10.22	14.56	5.61	1.15

Table 2: Expected frequency distribution of Collenbola microarthropods.

The frequencies in Table 2 have been obtained running the following code and using the transformation

$$\begin{aligned} b &= \frac{\mu(d-1)}{m(m-1)}, \\ a &= \mu - mb. \end{aligned} \tag{9}$$

```
> a <- -exp(mle1$coefs[1])*(mle1$coefs[2] - mle1$coefs[3])/(mle1$coefs[3] - 1)
> b <- exp(mle1$coefs[1])*(mle1$coefs[2] - 1)/(mle1$coefs[3]*(mle1$coefs[3] - 1))
> exp <- round(dhermite(seq(0,5,1), a, b, m = 3)*200, 2)
```

Note that the null hypothesis of Poisson distributed data is strongly rejected, with a likelihood ratio test statistic $W = 48.66494$ and its corresponding p-value= $1.518232e - 12$, as we can see with

```
> mle1$w
[1] 48.66494
> mle1$pval
[1] 1.518232e-12
```

Giles (2010)

In [Giles \(2010\)](#), the author explores an interesting application of the classical Hermite distribution ($m = 2$) in an economic field. In particular, he proposes a model for the number of currency and banking crises. The reported maximum likelihood estimates for the parameters were $\hat{a} = 0.936$ and $\hat{b} = 0.5355$, slightly different from those obtained using `glm.hermite`, which are $\hat{a} = 0.910$ and $\hat{b} = 0.557$. The actual and estimated expected counts under Hermite and Poisson distribution assumptions are shown in Table 3.

Currency and banking crises	0	1	2	3	4	5	6	7
Observed	45	44	19	17	19	13	6	4
Expected (Hermite)	38.51	35.05	37.40	24.36	15.96	8.33	4.23	1.88
Expected (Poisson)	22.07	44.66	45.20	30.49	15.43	6.25	2.11	0.61

Table 3: Observed and expected frequency distributions of currency and banking crises.

In this example, the likelihood ratio test clearly rejects the Poisson assumption in favor of the Hermite distribution ($W = 40.08$, p-value= $1.22e - 10$).

The expected frequencies of the Hermite distribution shown in Table 3 have been calculated running the code,

```
> exp2 <- round(dhermite(seq(0, 7, 1), 0.910, 0.557, m = 2)*167, 2)
> exp2
```

Giles (2007)

In [Giles \(2007\)](#), the author proposes an application of Hermite regression to the 965 number 1 hits on the Hot 100 chart over the period January 1955 to December 2003. The data were compiled and treated with different approaches by Giles (see [Giles 2006](#) for instance), and is available for download at the author website <http://web.uvic.ca/~dgiles/>. For all recordings that reach the number one spot, the number of weeks that it stays at number one was recorded. The data also allow for reentry into the number one spot after having being relegated to a lower position in the chart. The actual and predicted counts under Poisson and Hermite distributions are shown in Table 4.

Several dummy covariates were also recorded, including indicators of whether the recording was by Elvis Presley or not, the artist was a solo female, the recording was purely instrumental and whether the recording topped the charts in nonconsecutive weeks.

The estimates and corresponding standard errors are obtained through the instructions

```
> data(hot100)
> fit.hot100 <- glm.hermite(Weeks ~ Elvis+Female+Inst+NonCon, data = hot100,
+                               start = NULL, m = 2)
> fit.hot100$coefs
  (Intercept)          Elvis          Female          Inst          NonCon
  0.4578140      0.9126080      0.1913968      0.3658427      0.6558621
  dispersion.index      order
  1.5947901      2.0000000
> sqrt(diag(fit.hot100$vcov))
[1] 0.03662962 0.16208682 0.07706250 0.15787552 0.12049736 0.02533045
```

For instance, we can obtain the predicted value for the average number of weeks that an Elvis record hits the number one spot. According to the model, we obtain a predicted value of 3.9370, whereas the

Weeks	Actual	Poisson	Hermite
0	337	166.95	317.85
1	249	292.91	203.58
2	139	256.94	214.60
3	93	150.26	109.61
4	47	65.90	67.99
5	35	23.12	29.32
6	21	6.76	13.78
7	13	1.69	5.20
8	9	0.37	2.04
9	8	0.07	0.69
10	5	0.01	0.24
11	2	0.00	0.07
12	2	0.00	0.02
13	4	0.00	0.01
14	0	0.00	0.00
15	1	0.00	0.00

Table 4: Observed and expected frequency distribution of Hot 100 data.

observed corresponding value is 3.9375. The likelihood ratio test result justifies the fitting through a Hermite regression model instead of a Poisson model:

```
> fit.hot100$w; fit.hot100$pval
[1] 385.7188
[1] 3.53909e-86
```

DiGiorgio et al. (2004)

In [diGiorgio et al. \(2004\)](#) the authors perform an experimental simulation of *in vitro* whole body irradiation for high-LET radiation exposure, where peripheral blood samples were exposed to 10 different doses of 1480MeV oxygen ions. For each dose, the number of dicentrics chromosomes per blood cell were scored. The corresponding data is included in the package **hermite**, and can be loaded into the R session by

```
> data(hi_let)
```

In [Puig and Barquinero \(2011\)](#) the authors apply Hermite regression (to contrast the Poisson assumption) for fitting the dose-response curve, i.e. the yield of dicentrics per cell as a quadratic function of the absorbed dose linked by the identity function (which is commonly used in biodosimetry). This model can be fitted using the `glm.hermite` function in the following way:

```
> fit.hlet.id <- glm.hermite(Dic ~ Dose+Dose2-1, data = hi_let, link = "identity")
```

Note that the model defined in `fit.hlet.id` has no intercept.

A summary of the most relevant information can be obtained using the `summary()` method as in

```
> summary(fit.hlet.id)
Call:
glm.hermite(formula = Dic ~ Dose + Dose2 - 1, data = hi_let,
link = "identity")
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-0.06261842	-0.03536264	0.00000000	0.10373982	1.42251927

Coefficients:

	Estimate	Std. Error	z value	p-value
Dose	0.4620671	0.03104362	14.884450	4.158952e-50
Dose2	0.1555365	0.04079798	3.812357	1.376478e-04
dispersion.index	1.2342896	0.02597687	107.824859	1.468052e-25
order	2.0000000	NA	NA	NA

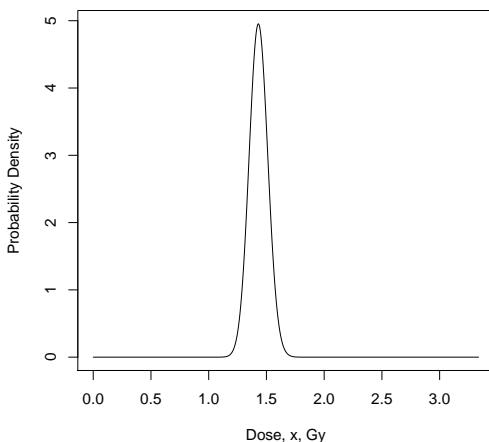


Figure 2: Absorbed dose density plot.

(Likelihood ratio test against Poisson is reported by `*z value*` for `*dispersion.index*`)

AIC: 5592.422

We can see the maximum likelihood estimates and corresponding standard errors in the output from the `summary` output. Note also that the likelihood ratio test rejects the Poisson assumption ($W = 107.82$, $p\text{-value}=1.47e - 25$).

Higuera et al. (2015)

In the first example in Higuera et al. (2015) the Bayesian estimation of the absorbed dose by Cobalt-60 gamma rays after the *in vitro* irradiation of a sample of blood cells is given by a density proportional to the probability mass function of a Hermite distribution taking 102 counts whose mean and variance are functions of the dose x , respectively $\mu(x) = 45.939x^2 + 5.661x$ and $v(x) = 8.913x^4 - 22.553x^3 + 69.571x^2 + 5.661x$.

The reparametrization in terms of a and b as a function of the dose is given by the transformation

$$a(x) = 2\mu(x) - v(x), \quad b(x) = \frac{v(x) - \mu(x)}{2}.$$

This density only makes sense when $a(x)$ and $b(x)$ are positive. The dose $x > 0$ and consequently $b(x)$ is always positive and $a(x)$ is positive for $x < 3.337$. Therefore, the probability density outside $(0, 3.337)$ is 0.

The following code generates the plot of the resulting density,

```
> u <- function(x) 45.939*x^2 + 5.661*x
> v <- function(x) 8.913*x^4 - 22.553*x^3 + 69.571*x^2 + 5.661*x
> a <- function(x) 2*u(x) - v(x); b <- function(x) (v(x) - u(x))/2
> dm <- uniroot(function(x) a(x), c(1, 4))$root; dm
> nc <- integrate(Vectorize(function(x) dhermite(102, a(x), b(x))), 0, dm)$value
> cd <- function(x){ vapply(x, function(d) dhermite(102, a(d), b(d)), 1)/nc }
> x <- seq(0, dm, .001)
> plot(x, cd(x), type = "l", ylab = "Probability Density", xlab = "Dose, x, Gy")
```

Figure 2 shows this resulting density.

Random number generation

A vector of random numbers following a Generalized Hermite distribution can be obtained by means of the function `rhermite`. For instance, the next code generates 1000 observations according to an Hermite regression model, including Bernoulli and normal covariates `x1` and `x2`:

```
> n <- 1000
> ##### Regression coefficients
> b0 <- -2
> b1 <- 1
```

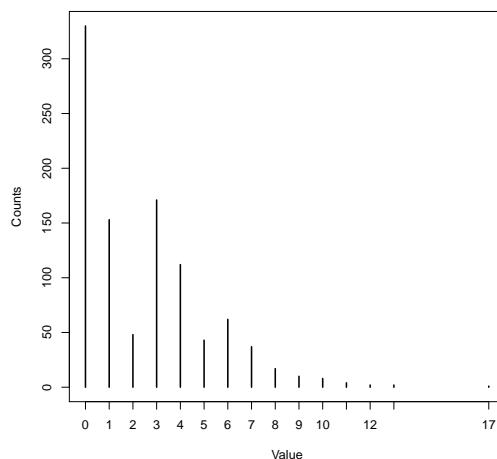


Figure 3: Random generated Hermite values.

```
> b2 <- 2
> ##### Covariate values
> set.seed(111111)
> x1 <- rbinom(n, 1, .75)
> x2 <- rnorm(n, 1, .1)
> u <- exp(b0 + b1*x1 + b2*x2)
> d <- 2.5
> m <- 3
> b <- u*(d - 1)/(m*(m - 1))
> a <- u - m*b
> x <- rhermite(n, a, b, m)
```

This generates a multimodal distribution, as can be seen in Figure 3. The probability that this distribution take a value under 5 can be computed using the function `phermite`:

```
> phermite(5, mean(a), mean(b), m = 3)
[1] 0.8734357
```

Conversely, the value that has an area on the left of 0.8734357 can be computed using the function `qhermite`:

```
> qhermite(0.8734357, mean(a), mean(b), m = 3)
[1] 5

> mle3 <- glm.hermite(x ~ factor(x1) + x2, m = 3)
> mle3$coefs
  (Intercept)   factor(x1)1           x2 dispersion.index      order
  -1.809163     1.017805      1.839606      2.491119    3.000000

> mle3$w;mle3$pval
[1] 771.7146
[1] 3.809989e-170
```

In order to check the performance of the likelihood ratio test, we can simulate a Poisson sample and run the function `glm.hermite` again:

```
> y <- rpois(n, u)
> mle4 <- glm.hermite(y ~ factor(x1) + x2, m = 3)
> mle4$coefs[4]
dispersion.index
  1
> mle4$w; mle4$pval
[1] -5.475874e-06
[1] 0.5
```

We can see that in this case the maximum likelihood estimate of the dispersion index d is almost 1 and that the Poisson assumption is not rejected ($p\text{-value} = 0.5$). If the MLE equations have no solution, the function `glm.hermite` will return a warning:

```
> z <- rpois(n, 20)
> mle5 <- glm.hermite(z ~ 1, m = 4)
Warning message:
In glm.hermite(z ~ 1, m = 4) : MLE equations have no solution
```

In this case, we have $\frac{\mu^{(4)}}{z^4} = 0.987$ and therefore the condition of Proposition 1 is not met.

Conclusions

Hermite distributions can be useful for modeling count data that presents multi-modality or overdispersion, situations that appear commonly in practice in many fields. In this article we present the computational tools that allow to overcome these difficulties by means of the Generalized Hermite distribution (and the classical Hermite distribution as a particular case) compiled as an R package. The **hermite** package also allows the user to perform the likelihood ratio test for Poisson assumption and to estimate parameters using the maximum likelihood method. Hermite regression is also a useful tool for modeling inflated count data, and it can be carried out by the **hermite** package in a flexible framework and including covariates. Currently, the **hermite** package is also used by the **radir** package (Moriña et al., 2015) that implements an innovative Bayesian method for radiation biodosimetry introduced in Higueras et al. (2015).

Acknowledgments

This work was partially funded by the grant MTM2012-31118, by the grant UNAB10-4E-378 co-funded by FEDER “A way to build Europe” and by the grant MTM2013-41383P from the Spanish Ministry of Economy and Competitiveness co-funded by the European Regional Development Fund (EDRF). We would like to thank professor David Giles for kindly providing some of the data sets used as examples in this paper.

Bibliography

- O. E. Barndorff-Nielsen and D. R. Cox. *Asymptotic Techniques for Use in Statistics*. Chapman & Hall, London, 1989. [p265, 266]
- M. diGiorgio, A. A. Edwards, J. E. Moquet, P. Finnion, P. A. Hone, D. C. Lloyd, A. J. Kreiner, J. A. Schuff, M. R. Taja, M. B. Vallerga, F. O. López, A. Burlón, M. E. Debray, and A. Valda. Chromosome aberrations induced in human lymphocytes by heavycharged particles in track segment mode. *Radiation Protection Dosimetry*, 108:47–53, 2004. doi: 10.1093/rpd/nch012. [p270]
- C. J. Geyer. On the asymptotics of constrained M-estimation. *The Annals of Statistics*, 22(4):1993–2010, Dec. 1994. doi: 10.1214/aos/1176325768. [p268]
- D. E. Giles. Superstardom in the US popular music industry revisited. *Economics Letters*, 92(1):68–74, July 2006. doi: 10.1016/j.econlet.2006.01.022. [p269]
- D. E. Giles. Modeling inflated count data. In L. Oxley and D. Kulasiri, editors, *Proceedings of the MODSIM 2007 International Congress on Modelling and Simulation*, pages 919–925. Modelling and Simulation Society of Australia and New Zealand, 2007. [p267, 269]
- D. E. Giles. Hermite regression analysis of multi-modal count data. *Economics Bulletin*, 30(4):2936–2945, 2010. [p269]
- R. Gupta and G. Jain. A generalized Hermite distribution and its properties. *SIAM Journal on Applied Mathematics*, 27(2):359–363, Sept. 1974. doi: 10.1137/0127027. [p263, 264]
- J. Gurland. Some interrelations among compound and generalized distributions. *Biometrika*, 44(1/2): 265–268, June 1957. doi: 10.2307/2333264. [p263]
- R. Hartenstein. On the distribution of forest soil microarthropods and their fit to ‘contagious’ distribution functions. *Ecology*, 42(1):190, Jan. 1961. doi: 10.2307/1933288. [p268]
- A. Henningsen and O. Toomet. maxLik: A package for maximum likelihood estimation in R. *Computational Statistics*, 26(3):443–458, 2011. doi: 10.1007/s00180-010-0217-1. [p267]

- M. Higueras, P. Puig, E. A. Ainsbury, and K. Rothkamm. A new inverse regression model applied to radiation biodosimetry. *Proceedings of the Royal Society A*, 471(2176):1–18, 2015. doi: 10.1098/rspa.2014.0588. [p271, 273]
- IAEA. Cytogenetic dosimetry: Applications in preparedness for and response to radiation emergencies. 2011. URL <http://www.iaea.org/newscenter/focus/actionplan/reports/enhancetransparency2012.pdf>. [p267]
- N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate Discrete Distributions*. John Wiley & Sons, New Jersey, 3rd edition, 2005. [p263]
- A. W. Kemp and C. D. Kemp. An alternative derivation of the Hermite distribution. *Biometrika*, 53(3–4):627–628, Dec. 1966. doi: 10.1093/biomet/53.3-4.627. [p263, 264]
- C. D. Kemp and A. W. Kemp. Some properties of the ‘hermite’ distribution. *Biometrika*, 52(3):381–394, Dec. 1965. [p263, 264, 265]
- E. Lukacs. *Characteristic Functions*. Hafner Publishing Company, 1970. [p263]
- D. Moriña, M. Higueras, P. Puig, E. A. Ainsbury, and K. Rothkamm. radir package: An R implementation for cytogenetic biodosimetry dose estimation. *Journal of Radiological Protection*, 35(3):557–569, 2015. [p273]
- L. Pace and A. Salvan. *Principles of Statistical Inference from a Neo-Fisherian Perspective*. World Scientific Publishing, Singapore, 1997. [p265]
- P. Puig. Characterizing additively closed discrete models by a property of their maximum likelihood estimators, with an application to generalized Hermite distributions. *Journal of the American Statistical Association*, 98(463):687–692, Sept. 2003. [p263, 266, 268]
- P. Puig and J. F. Barquinero. An application of compound Poisson modelling to biological dosimetry. *Proceedings of the Royal Society A*, 467:897–910, 2011. doi: 10.1098/rspa.2010.0384. [p270]
- S. G. Self and K.-Y. Liang. Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. *Journal of the American Statistical Association*, 82(398):605, June 1987. doi: 10.2307/2289471. [p268]

David Moriña

Centre for Research in Environmental Epidemiology (CREAL), Barcelona, Spain

Universitat Pompeu Fabra (UPF), Barcelona, Spain

CIBER Epidemiología y Salud Pública (CIBERESP), Barcelona, Spain

Grups de Recerca d’Àfrica i Amèrica Llatines (GRAAL), Unitat de Bioestadística, Facultat de Medicina, Universitat Autònoma de Barcelona

Barcelona 08003

Spain

dmorina@creal.cat

Manuel Higueras

Centre for Radiation, Chemical and Environmental Hazards, Public Health England (PHE)

Departament de Matemàtiques, Universitat Autònoma de Barcelona

Chilton, Oxfordshire OX11 0RQ

United Kingdom

mhigueras@mat.uab.cat

Pedro Puig

Departament de Matemàtiques, Universitat Autònoma de Barcelona

Bellaterra, Barcelona 08193

Spain

ppuig@mat.uab.cat

Maria Oliveira

Departamento de Estadística e Investigación Operativa, Universidad de Santiago de Compostela

Santiago de Compostela 15782

Spain

maria.oliveira@usc.es

Code Profiling in R: A Review of Existing Methods and an Introduction to Package **GUIProfiler**

by Angel Rubio and Fernando de Villar

Abstract Code analysis tools are crucial to understand program behavior. Profile tools use the results of time measurements in the execution of a program to gain this understanding and thus help in the optimization of the code. In this paper, we review the different available packages to profile R code and show the advantages and disadvantages of each of them. In addition, we present **GUIProfiler**, a package that fulfills some unmet needs.

Package **GUIProfiler** generates an HTML report with the timing for each code line and the relationships between different functions. This package mimics the behavior of the MATLAB profiler. The HTML report includes information on the time spent on each of the lines of the profiled code (the slowest code is highlighted). If the package is used within the RStudio environment, the user can navigate across the bottlenecks in the code and open the editor to modify the lines of code where more time is spent. It is also possible to edit the code using Notepad++ (a free editor for Windows) by simply clicking on the corresponding line. The graphical user interface makes it easy to identify the specific lines which slow down the code.

The integration in RStudio and the generation of an HTML report makes **GUIProfiler** a very convenient tool to perform code optimization.

Introduction

Software profiling is the analysis of a computer program performed by measuring the time spent on each line of code, code coverage or memory usage during its execution. Profiling is the first step towards efficient programming. The development of efficient software depends on identification of key bottlenecks. Focusing the optimization only on the bottlenecks is known to maximize efficiency in both development time and program runtime (Wilson et al., 2014). In interpreted languages (including R) a few lines can form major bottlenecks. See for example Visser et al. (2015).

With the advent of data mining, data analytics and big data analysis, code profiling is gaining prominence. In these fields, one potential limitation to scientific advance is inefficient code. Since the factors that affect the execution time are difficult to foresee beforehand, and the bottlenecks (if the code is large) are especially difficult to identify, the need of a profiling tool is apparent. In addition to that, inefficient code is also prone to have bugs. Our experience is that profiling is also an indirect way to fix errors in software.

The base distribution of R includes a profiling tool that consists of the functions `Rprof` to start and stop the profiling and `summaryRprof` as a parser of the output. The description of `Rprof` given in the help file is: “Profiling works by writing out the call stack every interval seconds (...).” This means that R uses the operating system interrupts to sample and write the call-stack (by default every 20 msecs). Therefore, lines that take more than 20 msecs appear at least once in the file written by the internal R profiler. On the other hand, a “fast” line of code where the execution time is strictly less than 20 msecs may or may not appear in the output file (with the probability of being included proportional to its execution time). Therefore, the output of profiling the same code is not identical for different runs. Overall it can be noted that statistical profiling (i.e., the one implemented in R) intrudes very little on the executed code (i.e., it almost does not affect its execution time) and is considered to be an efficient way to achieve proper profiling.

In previous versions of R (prior to 3.0), `Rprof` only worked at the function level (i.e., the profiler only provided information on the functions in the stack). Since version 3.0, it is possible to perform line profiling. If the `line.profiling` option is selected, the file generated by `Rprof` also includes information on the specific *line* of code in the stack (not only the *function*). This extension made it possible to identify the specific lines that slow down the code.

The output of `Rprof` is quite simple: a file that shows the name and some other characteristics of a function every time it is found to be in the stack. Despite this simplicity, it is necessary to parse this file to understand the content and R provides the “`summaryRprof` function (...) that can be used to process the output file to produce a summary of the usage” (from its help file).

The functionality of `summaryRprof` falls short in some aspects. If a function is called several times, `summaryRprof` fails to identify the specific function call that is slowing down the computation. As

functions can be nested in other functions, finding the true bottleneck is not obvious. In `summaryRprof`, it is not possible to track the hierarchy among the functions. `summaryRprof` was developed prior to the `line.profiling` option and does not take full advantage of this information. This fact will be shown in detail in the analysis of the different profile tools below.

These limitations have fostered the development of other command line functions and packages such as `proftable`, `aprof`, `proftools`, `profr`, and `lineprof`. `proftable` (Klevtsov, 2014) is a convenient command line function that parses the output of `Rprof` and shows the output in a reasonable and intuitive way. `aprof` (Visser et al., 2015) displays graphically the time spent on each line of code and provides an estimate on how optimizing a single line of code will affect the overall performance. It also shows the output of memory profiling. `proftools` (Tierney and Jarjour, 2013) includes useful command line tools to perform the profiling. Among other functionalities, it shows a graph of the different hierarchical relationships between the called functions. `lineprof` (Wickham, 2014a) can be considered an evolution of `profr` (Wickham, 2014b) and is developed by the same author. It is a profiling package that provides output integrated in the RStudio environment. It shows a nice graphical output and also includes memory profiling capabilities. In addition to these tools, the `pbdPAPI` package (Schmidt et al., 2015) offers access to low-level hardware counter information and is mainly used for advanced profiling. On the other hand, the most convenient package to test the performance of a single line of code is `microbenchmark` (Mersmann, 2014). It cannot be applied, though, to profile complex code. To our knowledge, these are all the available tools to aid in R profiling.

All these packages are, as most R packages, command line tools. Although they represent an important advance if compared with the `summaryRprof` function, none of them are especially user-friendly. In comparison, MATLABTM (The MathWorks Inc., 2015) provides a profiler that includes a convenient user interface by means of an HTML report. Profiling MATLAB code is easy and straightforward with the aid of its profile tool. An R profiler with this convenient front-end that includes GUI capabilities would be highly desirable.

Fortunately, the development of this tool is not such a challenging task. Several packages such as `Nozzle.R1` (Gehlenborg, 2013) or `knitr` (Xie, 2014) are available on CRAN that automatically generate HTML reports based on an easy syntax. `GUIProfiler` (de Villar and Rubio, 2015) is an R package that automatically generates an HTML report that summarizes the profiling results. These reports are generated with the aid of `Nozzle.R1`. Its integration in the RStudio environment makes it especially user friendly.

The following sections show how to use package `GUIProfiler` as well as provide a review on the profiling capabilities of the aforementioned tools using the same sample code for all of them.

Case study using `GUIProfiler`

The first lines of code are required for the installation of the package. As `GUIProfiler` is hosted on CRAN, the installation is straightforward:

```
install.packages("GUIProfiler")
library("GUIProfiler")
```

If the package is properly installed, no errors should appear after calling the `library` command. `GUIProfiler` has a practical limitation that must be taken into account: The profiled code must be stored on an accessible file. It is therefore better to run a source command instead of writing the lines directly in the command line. The reason of this limitation is that the report is based on the output from `Rprof` and that it only includes information on the functions stored at files, not from the script that calls them. In addition to that, it only accepts one function per file. R does allow to include several functions in a single file. We are currently working to circumvent this limitation.

Here we present some sample code (included also in the `GUIProfiler` documentation) that will be used with all the profiling tools. The HTML report generated by package `GUIProfiler` is shown in Figure 1 and the code is,

```
temp <- tempdir()
# Definition of two functions
normal.solve <- function(A, b) {
  Output <- solve(crossprod(A), t(A) %*% b)
}
chol.solve <- function(A, b) {
  L <- chol(crossprod(A))
  Output1 <- backsolve(L, t(A) %*% b, transpose = TRUE)
  Output2 <- backsolve(L, Output1)
}
```

```

compareMethods <- function() {
  library(MASS)
  # Call the functions
  source(paste(temp, "/normal.solve.R", sep = ""))
  source(paste(temp, "/chol.solve.R", sep = ""))
  # Solving a big system of equations
  nrows <- 1000
  ncols <- 500
  A <- matrix(rnorm(nrows * ncols), nrows, ncols)
  b <- rnorm(nrows)
  # Testing different possibilities
  Sol1 <- qr.solve(A, b)
  # Using QR factorization
  Sol2 <- coefficients(lm.fit(A, b))
  # lm.fit, based on QR but with some overhead
  Sol3 <- ginv(A) %*% b
  # Using the pseudoinverse based on SVD
  Sol4 <- normal.solve(A, b)
  # Using a function based on the normal equations.
  Sol5 <- chol.solve(A, b)
  # Using Choleski factorization.
}
# Dump these functions to three different files
dump("normal.solve", file = paste(temp, "/normal.solve.R", sep = ""))
dump("chol.solve", file = paste(temp, "/chol.solve.R", sep = ""))
dump("compareMethods", file = paste(temp, "/compareMethods.R", sep = ""))
source(paste(temp, "/compareMethods.R", sep = ""))

```

This code implements the minimum squares solution of a linear system of equations using different methods: QR factorization, the `lm.fit` function (that internally also uses the QR factorization), computation of a generalized pseudoinverse (that internally uses the SVD factorization), using the normal equations and, finally, using the Choleski factorization. The solutions using either of them are identical, i.e., the vectors `Sol1`, `Sol2`, `Sol3`, `Sol4` and `Sol5` are identical up to computer precision.

The lines to profile the code are only the last ones in the example code of the `RRprofReport` function. Specifically,

```

# Profile the code
RRprofStart()
compareMethods()
RRprofStop()
RRprofReport()

```

Each of these lines are self-explanatory. In the first line we activate and start the **GUIProfiler**. The following line executes the function that is being profiled. Once this line finishes, the profiling is stopped and the last line generates the report based on the output of the R profiler. In the RStudio environment, this report is shown in the viewer pane. In addition, the markers pane indicates the lines of code where more time was spent. It is possible to navigate through the source code by simply clicking on the corresponding markers.

If the program is not executed in the RStudio environment, `RRprofReport()` opens a new browser window. Figure 1 shows a snapshot of the generated report. The report consists of two groups of tables: a summary of the called functions with the time spent on each of them and a group of tables with the time spent on each line of code for each function. A convenient feature, if the browser is the Internet Explorer and Notepad++ is installed, is that the line numbers of the functions are clickable: Once a line number is clicked, the corresponding file is opened with the cursor on the selected line (as shown in Figure 2). On the right panel of Figure 2, the layout of **GUIProfiler** is shown in the RStudio environment. Note that RStudio version ≥ 0.99 is required. We tested **GUIProfiler** on RStudio 0.99.467. The navigation across the different functions can be done using the markers tab.

Comparison between different profiling tools

Using the same example, we compare the functionalities of the different profiling tools in their ability to provide insight on the profiled code. All the profiling tools (including **GUIProfiler**) manipulate the file generated by `Rprof` to provide a more readable and useful output. Therefore, most of the code to profile a function is shared by the different packages: First of all there is a call to `Rprof` to start

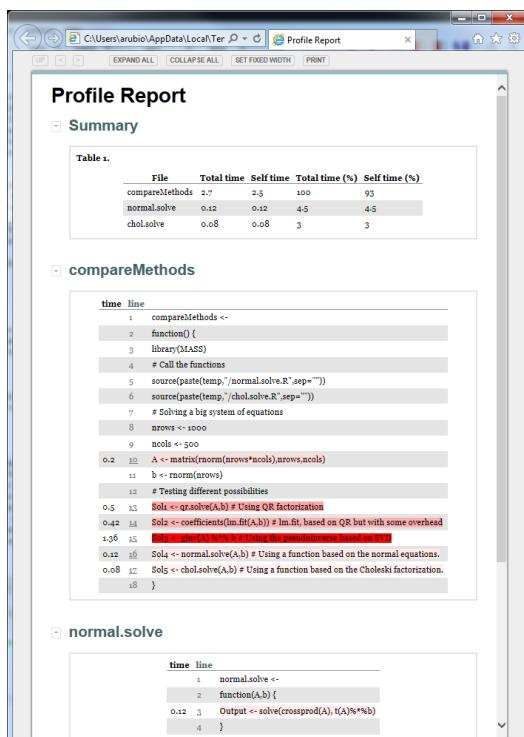


Figure 1: HTML report generated by GUIProfiler.

	HTML report	Graphical output	Function nesting	Line profiling	Memory profiling	Connection with editors
GUIProfiler	✓	✓	✓	✓	✗	✓(*)
summaryRprof	✗	✗	✗	✓	minimal	✗
proftable	✗	✗	✓	✓	✗	✗
aprof	✗	✓	✓	✓	✓	✗
protoools	✗	✓	✓	✗	✗	✗
profr	✗	✓	✓	✗	✗	✗
lineprof	✓	✓	✓	✓	✓	✗
MATLAB	✓	✓	✓	✓	✓(**)	✓

Table 1: Comparison of different profile tools for R. (*) **GUIProfiler** connects the results with RStudio and with the Notepad++ editor. (**) Non-documented characteristic.

profiling, the code itself to be profiled and a second call to Rprof to stop profiling. The differences between them are the way the output of Rprof is summarized and displayed.

Table 1 shows a comparison between the functionalities of Rprof parsers used to profile R code. The first column indicates if the package generates an HTML report. The second column indicates if the package generates some visual graphical output to show the results. The “Function nesting” column shows whether the package is able to display the hierarchy across the function calls. The “Line profiling” column states whether the package provides information related with each of the lines in the code, not only the functions (i.e., whether it takes advantage of the line.profiling option). The “Memory profiling” column states whether the package shows results of profiling memory usage and finally, the “Connection with editors” column states whether the package has a direct link with an editor to fix the potential bottlenecks.

We also included the features of the MATLAB profiler. As can be seen, it offers all these functionalities. One anecdotal note of the MATLAB profiler is that, even though it implements memory profiling (in a very effective and user friendly manner), this feature is non-documented. **summaryRprof**, **lineprof** and **aprof** implement memory profiling in R. **lineprof** and **GUIProfiler** provide an HTML report. **GUIProfiler** is the only one that provides a connection with editors.

We include Table 2 to show the dependencies for each of the packages. As a general rule, packages with few dependencies are easier to install and to run in different conditions (i.e. on a server

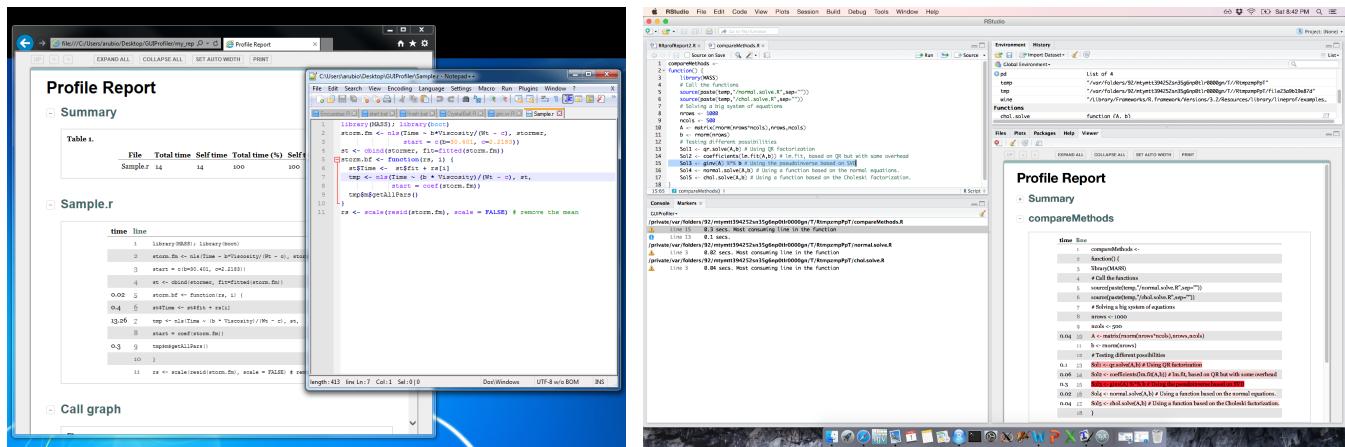


Figure 2: Layout of **GUIProfiler**. Left: Internet Explorer and Notepad++. Right: RStudio environment. The report is shown in the viewer tab on the right, and the markers tab on the left can be used to navigate across the code.

Tool	Dependencies
GUIProfiler	Nozzle.R1 , Rgraphviz (Hansen et al., 2015), graph (Gentleman et al., 2015), protoools
summaryRprof	None
protoable	None
aprof	grDevices
protoools	Rgraphviz , graph
profr	stringr (Wickham, 2015), plyr (Wickham, 2011)
lineprof	devtools (installation; Wickham and Chang, 2015), environment (C compiler), shiny (Chang et al., 2015, and its dependencies)

Table 2: Dependencies of the packages analyzed. Only packages not included in the standard distribution are mentioned.

enviroment). The following sections describe how to profile the example using the tools shown in Table 1.

summaryRprof

The code to perform the profiling of the example using **summaryRprof** is:

```
Rprof(tmp <- tempfile(), line.profiling = TRUE)
compareMethods()
Rprof(append = FALSE)
summaryRprof(tmp)
unlink(tmp)
```

and the output,

```
$by.self
self.time self.pct total.time total.pct
"La.svd"          0.80    40.82      0.80    40.82
".Call"           0.40    20.41      0.40    20.41
".Fortran"        0.38    19.39      0.38    19.39
"crossprod"       0.12     6.12      0.12     6.12
".External"        0.10     5.10      0.10     5.10
"%*%"            0.08     4.08      0.08     4.08
...More lines not included...
```

```
$by.total
```

```

total.time total.pct self.time self.pct
"compareMethods"      1.96   100.00    0.00    0.00
"ginv"                0.90    45.92    0.00    0.00
"svd"                 0.82    41.84    0.02    1.02
"La.svd"              0.80    40.82    0.80    40.82
".Call"                0.40    20.41    0.40    20.41
"coefficients"        0.40    20.41    0.00    0.00
"lm.fit"               0.40    20.41    0.00    0.00
".Fortran"             0.38    19.39    0.38    19.39
"qr"                  0.38    19.39    0.00    0.00
...More lines not included...

```

```

$sample.interval
[1] 0.02

```

```

$sampling.time
[1] 1.96

```

If we focus on the "by.self" part, the information is not too useful: the .Call or .Fortran functions can be used anywhere and depending on their argument, their behavior is completely different. On the other hand, the crossprod function is used many times in the code and the table only shows the overall time spent on it. Although the line.profiling option was set, line information does not appear anywhere in the output. The latest version of R (by setting `summaryRprof(tmp,lines = "show")`) provides basic information on line profiling.

The "by.total" part states that the most costly functions are ginv, svd and La.svd. However, these functions are in fact all the same: ginv calls svd that, in turn, calls La.svd. The output does not show this hierarchy in the calls to the function. In addition, the `summaryRprof` output does not show locations of the calls to the corresponding functions within the code. The output presents therefore serious limitations for its practical use that can be solved with other tools described below.

proftable

`proftable` is a convenient function that solves some of the aforementioned problems of `summaryRprof`: Each line of code is clearly identified and can be easily tracked. The function can be accessed from GitHub. The code to run the example is:

```

Rprof(tmp <- tempfile(), line.profiling = TRUE)
compareMethods()
Rprof(append = FALSE)
source("https://raw.githubusercontent.com/noamross/noamtools/master/R/proftable.R")
proftable(tmp)

```

And the output,

```

PctTime Call
36.364 compareMethods > 1#15 > ginv > svd > La.svd
20.000 compareMethods > 1#14 > coefficients > lm.fit > .Call
19.091 compareMethods > 1#13 > qr.solve > qr > qr.default > .Fortran
4.545 compareMethods > 1#15 > ginv > %*%
3.636 compareMethods > 1#10 > matrix > rnorm > .External
2.727 compareMethods > 1#15 > ginv > svd > La.svd > matrix
2.727 compareMethods > 1#16 > normal.solve > 2#3 > solve > crossprod
2.727 compareMethods > 1#16 > normal.solve > 2#3 > solve > solve.default
1.818 compareMethods > 1#17 > chol.solve > 3#3 > chol > crossprod
0.909 C:\\Some directories...\\chol.solve.R > 3: > #File

#File 1: C:\\Users\\arubio\\AppData\\Local\\Temp\\RtmpEH6ki0\\compareMethods.R
#File 2: C:\\Users\\arubio\\AppData\\Local\\Temp\\RtmpEH6ki0\\normal.solve.R
#File 3: C:\\Users\\arubio\\AppData\\Local\\Temp\\RtmpEH6ki0\\chol.solve.R

```

Parent Call: None

Total Time: 2.2 seconds
Percent of run time represented: 94.5 %

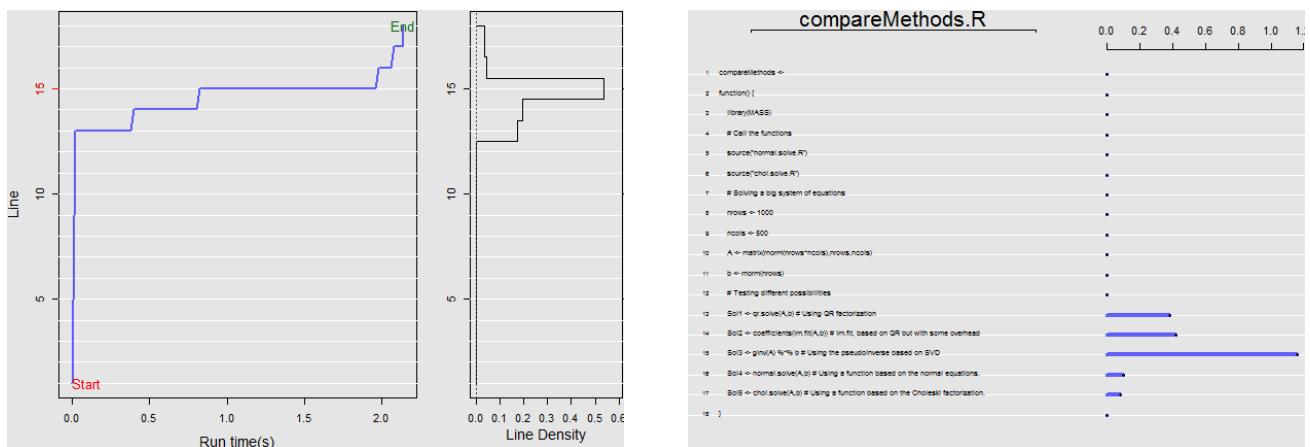


Figure 3: Output generated by `profileplot` of an ‘`aprof`’ object. Left panel: accumulated time and time spent on each line of code. Right: content for each line of code and the spent on each of them.

In this case the hierarchical relationships between `ginv`, `svd` and `La.svd` (as well as other ones) are clearly stated. Each line of code may appear several times. This “excess” of information can be useful to identify within each line of code which is the most costly part. However, it is also confusing. For example, line 15 of the first archive appears several times in the list since `svd` and matrix multiplication (both inside the `ginv` function) are costly operations.

`iprofable` includes the location of each line in each file, making their analysis within a context easier. `iprofable` is a simple, yet very useful tool.

aprof

`aprof` also works with the output of `Rprof` and provides visual aids based on line profiling. It helps to identify the most promising sections of code to optimize. One interesting unique functionality is that `aprof` also projects the potential gains. The last version has also memory profiling included. The code to run `aprof` is:

```

library(aprof)
Rprof(tmp <- tempfile(), line.profiling = TRUE)
compareMethods()
Rprof(append = FALSE)
fooaprof <- aprof(paste(temp, "/compareMethods.R", sep = ""), tmp)
plot(fooaprof)
profileplot(fooaprof)
summary(fooaprof)

```

Running the provided code in `aprof` generates nice plots that describe the time spent on each line of code (and the code itself). Figure 3 shows the `aprof` output. The summary function estimates the speed-up by optimizing a single line of code (or all the lines of code). The output to the console is:

Largest attainable speed-up factor for the entire program

when 1 line is sped-up with factor (S):

	Speed up factor (S) of a line	1	2	4	8	16	S -> Inf**
Line*: 15 :	1.00	1.40	1.75	2.00	2.15	2.33	
Line*: 13 :	1.00	1.06	1.10	1.12	1.13	1.14	
Line*: 14 :	1.00	1.06	1.10	1.12	1.13	1.14	
Line*: 16 :	1.00	1.03	1.05	1.06	1.06	1.07	
Line*: 17 :	1.00	1.03	1.05	1.06	1.06	1.07	
Line*: 10 :	1.00	1.01	1.02	1.02	1.02	1.02	

Lowest attainable execution time for the entire program when

lines are sped-up with factor (S):

```

Speed up factor (S) of a line
      1     2     4     8    16
All lines 2.840 1.420 0.710 0.355 0.178
Line*: 15 : 2.840 2.030 1.625 1.423 1.321
Line*: 13 : 2.840 2.670 2.585 2.543 2.521
Line*: 14 : 2.840 2.670 2.585 2.543 2.521
Line*: 16 : 2.840 2.750 2.705 2.683 2.671
Line*: 17 : 2.840 2.750 2.705 2.683 2.671
Line*: 10 : 2.840 2.810 2.795 2.788 2.784

```

```

Total sampling time: 2.84 seconds
* Expected improvement at current scaling
** Asymtotic max. improvement at current scaling

```

The line of code ‘`fooaprof <- aprof("myfile.R...")`’ can be run for each of the files to get the profiling of all the executed functions. In this case it can be done by,

```

compareaprof <- aprof(paste(temp, "/compareMethods.R", sep = ""), tmp)
plot(compareaprof)
profileplot(compareaprof)
compareaprof <- aprof(paste(temp, "/normal.solve.R", sep = ""), tmp)
plot(compareaprof)
profileplot(compareaprof)
compareaprof <- aprof(paste(temp, "/chol.solve.R", sep = ""), tmp)
plot(compareaprof)
profileplot(compareaprof)

```

Alternatively, if the user wants to go into more detail for each of the functions, `targetedSummary` can be used to disentangle nested functions.

The information provided by `aprof` and **GUIProfiler** is very similar: `aprof` provides the output as an image and **GUIProfiler** as an HTML report.

proftools

proftools provides tools for examining Rprof profile output. It shows graphically the dependencies among the different functions and the time spent on each of them. The code to profile the example is:

```

library(proftools)
Rprof(tmp <- tempfile(), line.profiling = TRUE)
compareMethods()
Rprof(append = FALSE)
pd <- readProfileData(tmp)
plotProfileCallGraph(pd, style = google.style, score = "total", nodeSizeScore = "none")

```

The user can get this information in the console by using the function `printProfileCallGraph`. The corresponding output is for this example:

```

Call graph
index % time   % self   % children      name
[1] 100.00      0.00   100.00  compareMethods [1]
1.06 0.00      %*% [15]
0.00 2.13      chol.solve [19]
0.00 21.28     coefficients [8]
0.00 43.62     ginv [2]
0.00 6.38      matrix [13]
0.00 3.19      normal.solve [16]
0.00 22.34     qr.solve [6]
-----
0.00 43.62     compareMethods [1]
[2] 43.62      0.00   43.62  ginv [2]
4.26 0.00      %*% [15]
0.00 39.36     svd [4]
-----
39.36 0.00      svd [4]
[3] 39.36      39.36   0.00  La.svd [3]

```

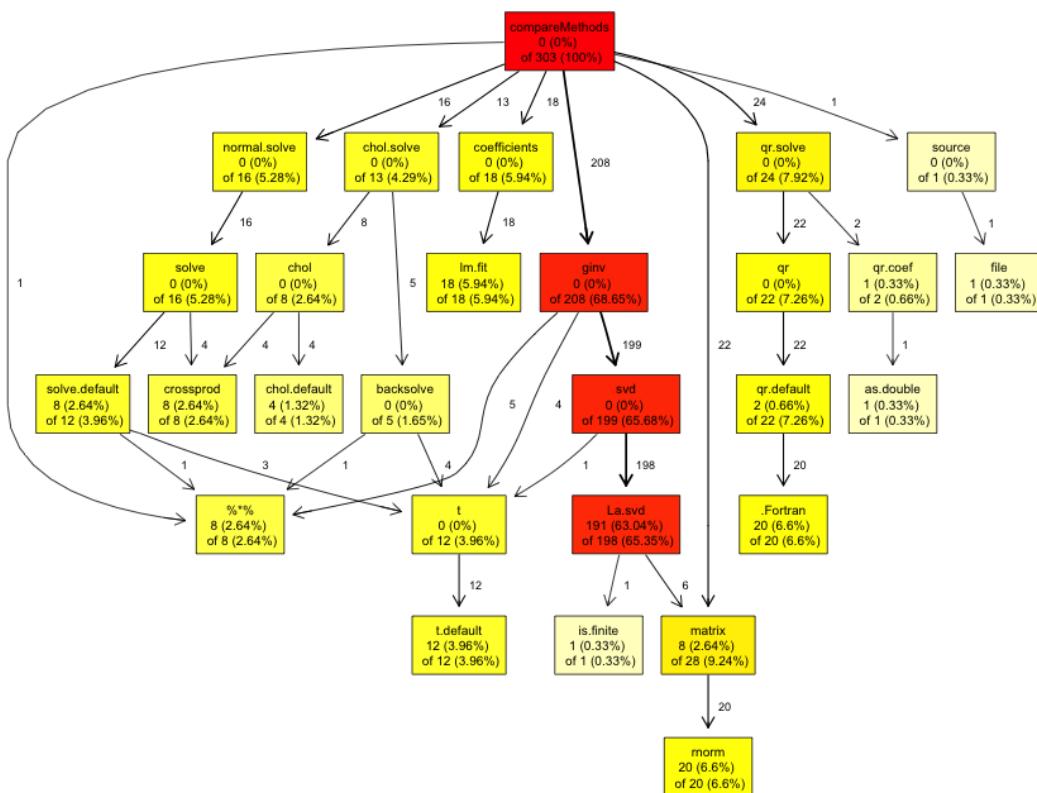


Figure 4: Output of proftools.

```

0.00      39.36      ginv [2]
[4]      39.36      0.00      39.36      svd [4]
39.36      0.00      La.svd [3]

-----
21.28      0.00      qr.default [11]
1.06      0.00      qr.coef [23]
[5]      22.34      22.34      0.00      .Fortran [5]

-----
0.00      22.34      compareMethods [1]
[6]      22.34      0.00      22.34      qr.solve [6]
0.00      21.28      qr [10]
0.00      1.06      qr.coef [23]

-----
21.28      0.00      lm.fit [9]
[7]      21.28      21.28      0.00      .Call [7]

... Additional lines not shown here...

```

Figure 4 shows the dependencies between the different functions. Probably, this tool provides the most intuitive representation of the different relationships among the functions in the code. We have taken advantage of this in **GUIProfiler** and this graph is also included in the generated report.

proftools is, however, “function based” and the `line.profiling` option is not used at all. In fact, if `Rprof` is used without the `line.profiling` option, the result is identical.

profr

profr is one of the oldest packages to parse the `Rprof` output. The first version appeared in May, 2008. Its usage, as for the other tools described here, is straightforward:

```
library(profr)
profcompareMethods <- profr(compareMethods())
```

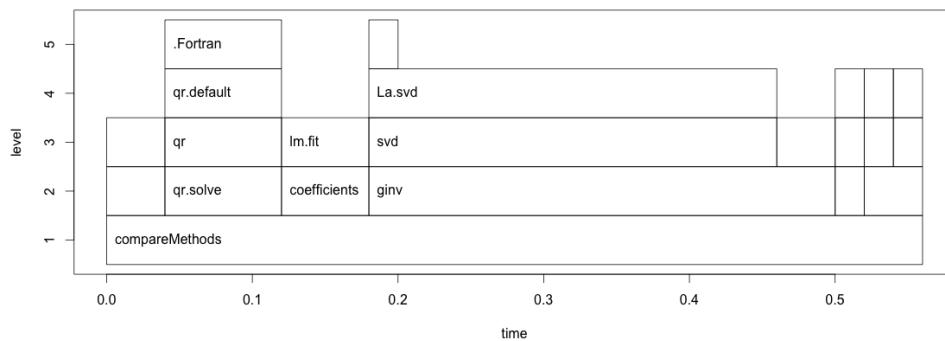


Figure 5: Output of **profr**.

```
head(profcompareMethods)
plot(profcompareMethods)
```

The output provided by **profr** is the following:

level	g_id	t_id	f	start	end	n	leaf	time	source		
8	1	1	1	compareMethods	0.00	0.76	1	FALSE	0.76	.GlobalEnv	
9	2	1	1		matrix	0.00	0.08	1	FALSE	0.08	base
10	2	2	1		qr.solve	0.08	0.20	1	FALSE	0.12	base
11	2	3	1		coefficients	0.20	0.26	1	FALSE	0.06	stats
12	2	4	1		ginv	0.26	0.66	1	FALSE	0.40	MASS
13	2	5	1		normal.solve	0.66	0.74	1	FALSE	0.08	.GlobalEnv

There is a useful plot command that shows the time spent on each function. In turn, the functions are grouped by levels. The output is shown in Figure 5.

profr was (with `summaryRprof`) the first attempt to display the result of profiling R code. It was developed before `line.profiling` was available and does not make use of it. The graphical representation is much more informative than the text output. However, although the hierarchical tree of `proftools` and the graph shown in `profr` show basically the same information, in our opinion, the tree is more visually apparent and attracts attention directly to the bottlenecks of the code.

lineprof

`lineprof` can be considered as an evolution of `profr` (both have the same developer and maintainer). Although it is not yet on CRAN, the installation from GitHub is straightforward.

```
install.packages("devtools")
library(devtools)
devtools::install_github("hadley/lineprof")
```

`lineprof` presents some characteristics that make it unique: It is integrated in the RStudio environment using the `shiny` package, it provides memory profiling out of the box and finally, using the `shiny` environment it is possible to navigate across the different functions to find out the bottlenecks in the code. The application to the example is also straightforward:

```
library(lineprof)
x <- lineprof(compareMethods())
shine(x)
```

The resulting figure shows the output in the viewer pane in RStudio. The code for each function is shown there. The blue lines of code are hyperlinks to the corresponding functions. The last column represents the memory spent on each line of code. It can be seen that the creation of the 1000×500 matrix is the most expensive line of code in terms of memory use.

We experienced some minor issues when using `lineprof` that are worth mentioning. The representation in RStudio is a little bit buggy: The columns for each of the results (time spent, memory used, etc) are not properly shown. The navigation across the functions, although very intuitive, does not link directly to the RStudio editor to work on the code. Finally, when working on the `shiny` environment, the console appears to be busy and the user has to break it using Ctrl-C or Esc. We expect that most of these minor problems will be fixed in the stable release.

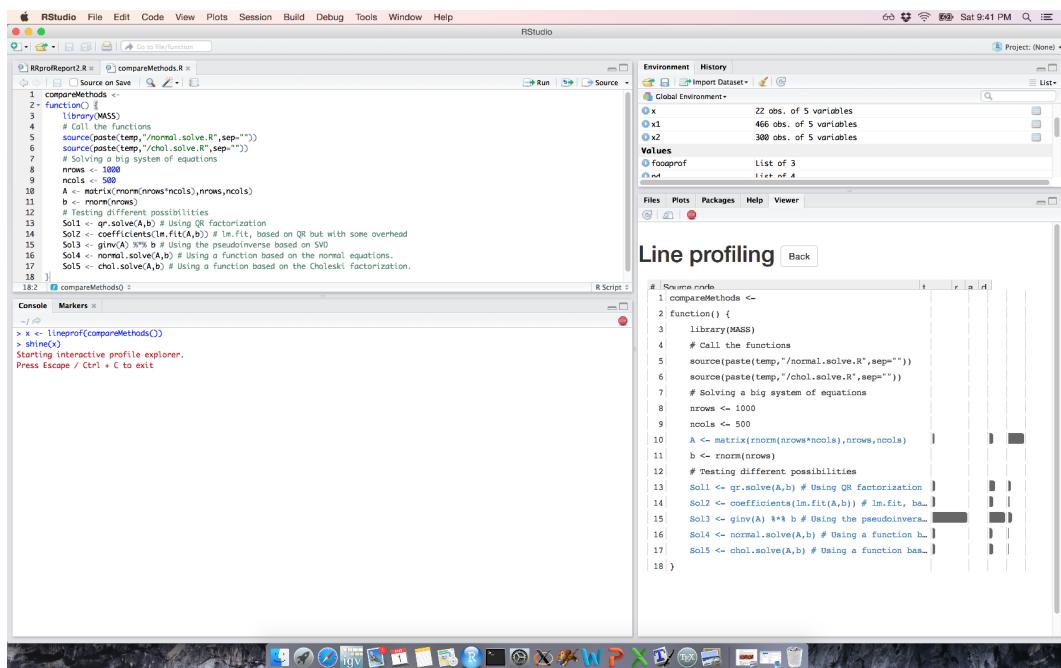


Figure 6: Output of **lineprof**.

Discussion and conclusion

This paper describes **GUIProfiler**, a graphical user interface to the line profiler provided by R and reviews, in relation to **GUIProfiler**, all other available profiling packages. Based on an example, we presented how to profile this code using the different packages and compared their advantages and disadvantages.

This review shows that, in spite of being far from the intuitiveness and user friendliness of the MATLAB profiler, the different developed tools are getting closer to it. For example, the graphical display of the relationship between functions in **proftools** is useful and intuitive and MATLAB does not provide anything like it. **lineprof** allows the navigation across the different functions very much like MATLAB's profiler does.

Regarding the described tools, **proftable**, **profR** and **proftools** are comparable. **proftable** provides an abstract on the time spent on each line of the code that is clearer and more useful than the one from **summaryRprof**. **profR** and **proftools** show graphically the relationships between the different profiled functions and the time spent on each of them. Unfortunately, neither of them provide profiling at the line level.

The other group is formed by **aprof**, **lineprof** and **GUIProfiler**. All of them provide similar information: time spent on each line of code stating the actual code of the line. There are also some differences among them. **aprof** shows these results using plots and provides estimates of the expected improvements when speeding up the most costly lines of code. **lineprof** displays these results using the **shiny** environment. Both **lineprof** and **aprof** provide memory profiling. **GUIProfiler**, on the other hand, builds an HTML report which is interactively linked with the program code. The preference between both tools may be a question of taste, but in our opinion, the HTML report is more advantageous.

We would like to note that there might be other characteristics which could be taken into account when evaluating a profiling tool such as those considered in this paper. For example, we assumed that graphical output is something desirable. However, this is not the case if the profiled code is run on a server with no graphical capabilities. In the case of **lineprof**, for example, a runtime environment is required to compile it as well as package **shiny** and several other dependencies. Although these dependencies are not of concern when R is run on a personal computer, they can be problematic if the software is running on a server.

GUIProfiler has a convenient characteristic that is missing in the other packages: It connects the profiling tool with an editor to fix the bottlenecks. In the RStudio environment the navigation across the markers pane directly opens the editor on the clicked line of code. If R is used outside of the RStudio environment, **GUIProfiler** opens Notepad++ when clicking on the number of the line. This functionality is browser dependent and, at present, it is only implemented for the Internet Explorer

(and thus the Windows OS) by using ActiveX controls. However, we assume that most of the users will use the RStudio environment where the connection with an external editor is not necessary. Finally we would like to emphasize that **GUIProfiler** is a useful tool that interactively and graphically helps the users in the difficult task of profiling.

Acknowledgments

This work was partially supported by the Spanish Minister of Science and Innovation with project "Sanscript" IPT-2012-0093-010000. This funding is gratefully acknowledged.

Bibliography

- W. Chang, J. Cheng, J. J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2015. URL <https://CRAN.R-project.org/package=shiny>. R package version 0.12.2. [p150, 250, 279]
- F. de Villar and A. Rubio. *GUIProfiler: Graphical User Interface for Rprof()*, 2015. URL <https://CRAN.R-project.org/package=GUIProfiler>. R package version 2.0.1. [p276]
- N. Gehlenborg. *Nozzle.R1: Nozzle Reports*, 2013. URL <https://CRAN.R-project.org/package=Nozzle.R1>. R package version 1.1-1. [p276]
- R. Gentleman, E. Whalen, W. Huber, and S. Falcon. *graph: A Package to Handle Graph Data Structures*, 2015. R package version 1.48.0. [p279]
- K. D. Hansen, J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, and D. Sarkar. *Rgraphviz: Provides Plotting Capabilities for R Graph Objects*, 2015. R package version 2.14.0. [p279]
- A. A. Klevtsov. *proftable: An Alternative to summaryRprof()*, 2014. URL <https://github.com/noamross/noamtools/blob/master/R/proftable.R>. R function. [p276]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2014. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4-2. [p276]
- D. Schmidt, C. Heckendorf, and W. C. Chen. *pbdPAPI: Programming with Big Data – Performance Analysis Tools for R*, 2015. URL <https://github.com/wrathematics/pbdPAPI>. R package. [p276]
- The MathWorks Inc. *MATLAB – The Language of Technical Computing, Version R2015a*. Natick, Massachusetts, 2015. URL <http://www.mathworks.com/products/matlab/>. [p276]
- L. Tierney and R. Jarjour. *proftools: Profile Output Processing Tools for R*, 2013. URL <https://CRAN.R-project.org/package=proftools>. R package version 0.1-0. [p276]
- M. D. Visser, S. M. McMahon, C. Merow, P. M. Dixon, S. Record, and E. Jongejans. Speeding up ecological and evolutionary computations in R; Essentials of high performance computing for biologists. *PLoS Computational Biology*, 11(3):e1004140, 2015. [p275, 276]
- H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 2011. URL <http://www.jstatsoft.org/v40/i01/>. [p279]
- H. Wickham. *lineprof: An Alternative Display for Line Profiling Information*, 2014a. R package version 0.1. [p276]
- H. Wickham. *profr: An Alternative Display for Profiling Information*, 2014b. URL <https://CRAN.R-project.org/package=profr>. R package version 0.3.1. [p276]
- H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2015. URL <https://CRAN.R-project.org/package=stringr>. R package version 1.0.0. [p279]
- H. Wickham and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2015. URL <https://CRAN.R-project.org/package=devtools>. R package version 1.9.1. [p152, 279]
- G. Wilson, D. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. Haddock, K. Huff, I. M. Mitchell, M. D. Plumley, et al. Best practices for scientific computing. *PLoS Biology*, 12(1):e1001745, 2014. [p275]
- Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2014. URL <https://CRAN.R-project.org/package=knitr>. R package version 1.6. [p276]

Angel Rubio
CEIT and TECNUN
Universidad de Navarra
San Sebastian
Spain
arubio@ceit.es

Fernando de Villar
TECNUN
Universidad de Navarra
San Sebastian
Spain
fdevillar@gmail.com

The R Consortium and the R Foundation

by *Martyn Plummer*

Abstract The R Consortium was announced at the useR! 2015 conference in Aalborg, Denmark on 30 June. It is a non-profit organization set up to provide infrastructure for the R community. The purpose of this article is to explain some of the background to the setting up of the Consortium and how it interacts with the R Foundation.

One of the most striking developments in the recent history of R is the extent to which it is now used in a commercial environment. This success has generated new demands for the R project. In particular, there has been an ongoing conversation about how commercial organizations can make financial contributions to the R project and support infrastructure for the R community. This is of particular concern to companies based in the United States, who may find it difficult to make donations to the R Foundation for Statistical Computing, as we are a non-profit organization based in Europe. It was also clear that the interested companies wanted to play an active role in supporting infrastructure for R. Representatives from companies based in the United States discussed these issues with the R Foundation at the [DSC 2014](#) meeting in July 2014, where it was decided that the best way forward would be to create a trade association under the name “The R Consortium”. Members of a trade association (formally, a non-profit organization under US Internal Revenue code 501(c)(6)) are businesses that agree to work together to advance a common interest.

Once the idea of a trade association was approved by the R Foundation, representatives of the founding members, including John Chambers representing the R Foundation, worked with the Linux Foundation to follow the legal steps to create the Consortium. The Linux Foundation has wide experience in setting up what they call “collaborative projects”. The R Consortium is one of many such projects now listed at <http://collabprojects.linuxfoundation.org/>. Other examples include the Core Infrastructure Initiative and the Open Virtualization Alliance.

All current members of the R Consortium, apart from the R Foundation, are commercial organizations, and their status within the R Consortium (platinum, gold, or silver) depends on their level of financial contributions. The R Foundation has a special status in the bylaws of the R Consortium, and has permanent membership without making any financial contributions. This membership includes a seat on the R Consortium Board of Directors, where our representative is currently John Chambers. In addition, the R Foundation has a representative on the [Infrastructure Steering Committee](#) (ISC), the body that decides what projects should receive funding from the R Consortium. Our current representative on the ISC is Luke Tierney.

In November 2015, the R Consortium announced its first funding award to Gábor Csárdi to develop [R-hub](#). At the time of writing the ISC also has a [call for proposals](#) set to end on 10 January 2016. Grants awarded by the R Consortium could have values up to \$20–30k. Proposals are open to everyone, not just members of the R Consortium.

The stated goals of the R Consortium emphasize that it will not influence the development of R itself (noted in the R Consortium [FAQ](#)); this remains entirely under the control of the R Core Team and the R Foundation remains the sole representative of the R project at the organization level.

It should also be noted that the R Consortium was set up to answer the particular needs of companies in the United States. The R Foundation is also open to collaboration with other non-profit organizations working to support the development of R world-wide.

*Martyn Plummer
Co-president, The R Foundation
martyn.plummer@r-project.org*

Conference Report: useR! 2015

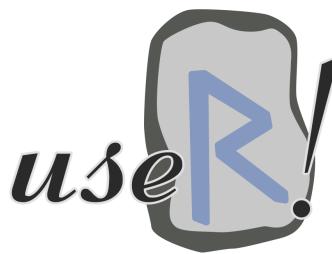
by Torben Tvedebrink

The 11th international R user conference, useR! 2015, took place in Aalborg, Denmark, 1–3 July 2015. The Department of Mathematical Sciences, Aalborg University, hosted the conference, which took place in Aalborg Congress and Culture Centre.

We had originally hoped for 300–400 participants and some support from sponsors. The meeting attracted a total of 660 participants from 42 countries with an almost uniform split on academia and industry. Furthermore, the industry's generous support made it possible to provide free meals, drinks and a well-suited venue for the conference.

Our social events included welcome reception at the waterfront in the House of Music, a poster session with free bar and food, and a trip to Denmark's second largest forest (Rold Forest) where we held the conference dinner. During the conference dinner competitive games took place such as long sawing, axe hurling and archery.

We received more than 250 abstracts of which some 220 were accepted either as posters, lightning talks or oral presentations. The final programme consisted of six invited talks, 126 oral presentations, 14 lightning talks and 77 posters were presented at the conference.



```
> sessionInfo()
[1] "June 30 - July 3, 2015"
[2] "Aalborg, Denmark"
```

Pre-conference tutorials

Inspired by the initiative of useR! 2014 in Los Angeles we decided to provide the tutorials free of charge to useR! participants. This reduced the book-keeping load and allowed people to attend tutorials without considering the additional cost per tutorial. Based on the submitted tutorial proposal, the programme committee elected tutorials below:

- Applied Spatial Data Analysis with R (*Virgilio Gómez Rubio*)
- Bayesian Networks and Graphical Models with R (*Søren Højsgaard and Therese Graversen*)
- Data Manipulation with **dplyr** (*Hadley Wickham*)
- Efficient Statistical Consulting using R Workflow for Data Analysis Projects (*Peter Baker*)
- Handling Missing Values with a Special Focus on the Use of Principal Components Methods (*François Husson*)
- RHadoop (*Andrie de Vries and Simon Field*)
- Rocker: Using R on Docker (*Dirk Eddelbuettel*)
- Statistical Analysis of Network Data (*Gabor Csardi*)
- Analysis and Visualization of Large Complex Data with Tessera (*Ryan Hafen and Stephen Elston*)
- Applied Machine Learning and Efficient Model Selection with **mlr** (*Bernd Bischl and Michel Lang*)
- Bioconductor for High-Throughput Sequence Analysis (*Martin Morgan*)
- Getting to Know **grid** Graphics (*Paul Murrell*)
- Introduction to Bayesian Data Analysis with R (*Rasmus Bååth*)
- **spatstat**: An R Package for Analysing Spatial Point Patterns (*Adrian Baddeley and Ege Rubak*)
- Testing R Code (*Richard J. Cotton*)
- Using Pandoc's Markdown with R (*Gergely Daróczi*)

More than 80% of the conference participants registered at Tutorial Tuesday and most of these participated in one or two tutorials making the “open source” offer of free participation a success.

Invited talks

With the aim of getting the “use” of useR! in focus we invited speakers with varying backgrounds to give the six plenary talks of useR! 2015. Most of the presented topics were also discussed in the submitted sessions.

- *Thomas Lumley*: How Flexible Computing Expands What an Individual Can Do
- *Adrian Baddeley*: How R Has Changed Spatial Statistics
- *Steffen Lauritzen*: Linear Estimating Equations for Gaussian Graphical Models with Symmetry
- *Di Cook*: A Survey of Two Decades of Efforts to Build Interactive Graphics Capacity in R
- *Romain François*: My R Adventures
- *Susan Holmes*: Multitype Data Integration: Challenges from the Human Microbiome

Contributed sessions

After the selection of submitted abstracts we attempted to group the contributed talks in sessions of similar talks. The overall headings of the five parallel sessions were:

- | | | |
|-------------------|-----------------------------|---------------------------|
| • Ecology | • Computational performance | • Commercial offerings |
| • Networks | • Business | • Interactive graphics |
| • Reproducibility | • Spatial | • Teaching |
| • Interfacing | • Databases | • Statistical methodology |
| • Case study | • Medicine | • Machine learning |
| • Clustering | • Regression | • Visualisation |
| • Data management | | |

These themes were also represented in the poster session and in the six kaleidoscope sessions. In addition to posters and presentations, there were 14 Lightning Talks, a 5-minute presentation on any R-related topic aimed particularly at those new to R. Participants seemed to appreciate this fast-paced introduction to a wide range of topics.

Organisers

The selection of abstracts for presentations would not have been possible without the thorough review process of the programme committee. We are grateful to the programme committee of useR! 2015: Peter Dalgaard, Dirk Eddelbuettel, Poul Svante Eriksen, Julie Josse, Martin Maechler, Katharine Mullen, Helle Sørensen, Heather Turner, Hadley Wickham, Achim Zeileis, and Søren Højsgaard (chair).

The local “green shirt” heroes making the useR! 2015 in Aalborg possible consisted of several students and local statisticians: Mikkel Meyer Andersen, Anders Ellern Bilgrau, Claus Dethlefsen, Mateusz ‘Matt’ Dziubinski, Poul Svante Eriksen, Søren Højsgaard, Rikke Nørmark Mortensen, Maria Rodrigo-Domingo, Ege Rubak, and Torben Tvedebrink (chair).

Further information

The useR! 2015 website, www.R-project.org/useR-2015 provides a record of the conference. Where authors have made them available, slides are accessible via the online conference schedule (Oral Sessions).

A blog post summarising the planning and execution of useR! 2015 can be found at the [Revolution Analytics’ blog](#).

Torben Tvedebrink

Department of Mathematical Sciences, Aalborg University

Fredrik Bajers Vej 7G, DK-9220 Aalborg

Denmark

tvede@math.aau.dk

News from the Bioconductor Project

Bioconductor Team

Biostatistics and Bioinformatics

Roswell Park Cancer Institute, Buffalo, NY, USA

The [Bioconductor](#) project provides tools for the analysis and comprehension of high-throughput genomic data. The 1104 software packages available in Bioconductor can be viewed at <http://bioconductor.org/packages/>. Navigate packages using 'biocViews' terms and title search. Each package has an html page with a description, links to vignettes, reference manuals, and usage statistics. Start using Bioconductor version 3.2 by installing R 3.2.3 and evaluating the commands

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

Install additional packages and dependencies, e.g., [AnnotationHub](#), with

```
source("http://bioconductor.org/biocLite.R")
biocLite("AnnotationHub")
```

Bioconductor 3.2 Release Highlights

Bioconductor 3.2 was released on 14 October 2015. It is compatible with R 3.2 and consists of 1104 software packages, 257 experiment data packages, and 917 up-to-date annotation packages. There are 80 new software packages and many updates and improvements to existing packages. The [release announcement](#) includes descriptions of new packages and updated NEWS files provided by package maintainers.

Our collection of microarray, transcriptome and organism-specific *annotation packages* use the 'select' interface (keys, columns, keytypes) to access static information on gene annotations (**org.*** packages) and gene models (**TxDb.*** packages); these augment packages such as [biomaRt](#) for interactive querying of web-based resources. The [AnnotationHub](#) continues to complement our traditional offerings with diverse whole genome annotations from Ensembl, ENCODE, dbSNP, UCSC, and elsewhere; example uses are described in the [AnnotationHub How-To vignette](#).

Other activities

The Bioconductor project has moved from the Fred Hutchinson Cancer Research Center to Roswell Park Cancer Institute, in Buffalo, NY. The transition has brought with it many challenges and opportunities, including the departure of some long-term project personnel and the addition of new team members. In particular, Marc Carlson, Sonali Arora and Paul Shannon were instrumental in the design and implementation of [AnnotationHub](#) (and annotations in general), tools for biological network analysis, educational material and many other areas. The Bioconductor community is grateful to them for their many valuable contributions.

The Bioconductor [F1000 research channel](#) is a recently-launched forum for publication of task-oriented work flows. The channel is peer-reviewed, providing authors with a compelling venue for dissemination of their analysis methods. Users gain fully reproducible descriptions of tasks that cover current, genome-scale analysis problem from start to finish.

The Bioconductor [support forum](#) plays an increasing important role in providing with timely, knowledgeable, and accurate answers to user questions. A particularly valuable feature is the opportunity for community members to announce tutorial and other availability, such as [Bioconductor for Genomic Data Science](#) offered by long-term contributor Kasper D. Hansen. A highlight of the Bioconductor European Developer Meeting, held in

Cambridge, UK on 7 and 8 December, was recognition of the contributions Aaron Lun and Michael Love make to the success of the Bioconductor support forum through their patient and knowledgeable responses to diverse user questions.

Continued availability of Bioconductor [Docker](#) and [Amazon](#) images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Docker images are available for release and development versions of Bioconductor, with analysis-specific images pre-loaded with packages relevant to common analysis scenarios, e.g., of sequencing, microarray, flow cell, or proteomic data. Both Amazon and Docker images include Rstudio Server for easy web-browser based access.

New Bioconductor package contributors are encouraged to consult the package [guidelines](#) and [submission](#) sections of the Bioconductor web site, and use the [BiocCheck](#) package, in addition to R `CMD check`, for guidance on conforming to Bioconductor package standards. New package submissions are automatically built across Linux, Mac, and Windows platforms, providing an opportunity to address cross-platform issues; many new package contributors take advantage of this facility to refine their package before it is subject to technical preview. Keep abreast of packages added to the ‘devel’ branch and other activities by following @Bioconductor on Twitter.

The Bioconductor web site advertises [training and community events](#), including the BioC 2016, the Bioconductor annual conference, to be held in Stanford, CA, immediately before the annual *useR!* conference, Friday through Sunday, 24–25 June.

Changes in R

From version 3.2.2 to version 3.2.3

by the R Core Team

CHANGES IN R 3.2.3

NEW FEATURES

- Some recently-added Windows time zone names have been added to the conversion table used to convert these to Olson names. (Including those relating to changes for Russia in Oct 2014, as in PR#16503.)
- (Windows) Compatibility information has been added to the manifests for ‘Rgui.exe’, ‘Rterm.exe’ and ‘Rscript.exe’. This should allow `win.version()` and `Sys.info()` to report the actual Windows version up to Windows 10.
- Windows “wininet” FTP first tries EPSV / PASV mode rather than only using active mode (reported by Dan Tenenbaum).
- `which.min(x)` and `which.max(x)` may be much faster for logical and integer `x` and now also work for long vectors.
- The ‘emulation’ part of `tools::texi2dvi()` has been somewhat enhanced, including supporting `quiet = TRUE`. It can be selected by `texi2dvi = "emulation"`.
(Windows) MiKTeX removed its `texi2dvi.exe` command in Sept 2015: `tools::texi2dvi()` tries `texify.exe` if it is not found.
- (Windows only) Shortcuts for printing and saving have been added to menus in `Rgui.exe`. (Request of PR#16572.)
- `loess(..., iterTrace=TRUE)` now provides diagnostics for robustness iterations, and the `print()` method for `summary(<loess>)` shows slightly more.
- The included version of PCRE has been updated to 8.38, a bug-fix release.
- `View()` now displays nested data frames in a more friendly way. (Request with patch in PR#15915.)

INSTALLATION and INCLUDED SOFTWARE

- The included configuration code for `libintl` has been updated to that from `gettext` version 0.19.5.1 — this should only affect how an external library is detected (and the only known instance is under OpenBSD). (Wish of PR#16464.)
- `configure` has a new argument ‘`--disable-java`’ to disable the checks for Java.
- The `configure` default for `MAIN_LDFLAGS` has been changed for the FreeBSD, NetBSD and Hurd OSes to one more likely to work with compilers other than `gcc` (FreeBSD 10 defaults to `clang`).
- `configure` now supports the OpenMP flags ‘`-fopenmp=libomp`’ (`clang`) and ‘`-qopenmp`’ (`Intel C`).
- Various macros can be set to override the default behaviour of `configure` when detecting OpenMP: see file ‘`config.site`’.
- Source installation on Windows has been modified to allow for MiKTeX installations without `texi2dvi.exe`. See file ‘`MkRules.dist`’.

BUG FIXES

- `regexp(pat, x, perl = TRUE)` with Python-style named capture did not work correctly when `x` contained NA strings. (PR#16484)
- The description of dataset `ToothGrowth` has been improved/corrected. (PR#15953)
- `model.tables(type = "means")` and hence `TukeyHSD()` now support "aov" fits without an intercept term. (PR#16437)
- `close()` now reports the status of a `pipe()` connection opened with an explicit open argument. (PR#16481)
- Coercing a list without names to a data frame is faster if the elements are very long. (PR#16467)
- (Unix-only) Under some rare circumstances piping the output from `Rscript` or `R -f` could result in attempting to close the input file twice, possibly crashing the process. (PR#16500)
- (Windows) `Sys.info()` was out of step with `win.version()` and did not report Windows 8.
- `topenv(baseenv())` returns `baseenv()` again as in R 3.1.0 and earlier. This also fixes `compilerJIT(3)` when used in '`.Rprofile`'.
- `detach()`ing the **methods** package keeps `.isMethodsDispatchOn()` true, as long as the `methods` namespace is not unloaded.
- Removed some spurious warnings from `configure` about the preprocessor not finding header files. (PR#15989)
- `rchisq(*, df=0, ncp=0)` now returns 0 instead of NaN, and `dchisq(*, df=0, ncp=*)` also no longer returns NaN in limit cases (where the limit is unique). (PR#16521)
- `pchisq(*, df=0, ncp >0, log.p=TRUE)` no longer underflows (for `ncp > ~60`).
- `nchar(x, "w")` returned -1 for characters it did not know about (e.g. zero-width spaces): it now assumes 1. It now knows about most zero-width characters and a few more double-width characters.
- Help for `which.min()` is now more precise about behavior with logical arguments. (PR#16532)
- The print width of character strings marked as "latin1" or "bytes" was in some cases computed incorrectly.
- `abbreviate()` did not give names to the return value if `minlength` was zero, unlike when it was positive.
- (Windows only) `dir.create()` did not always warn when it failed to create a directory. (PR#16537)
- When operating in a non-UTF-8 multibyte locale (e.g. an East Asian locale on Windows), `grep()` and related functions did not handle UTF-8 strings properly. (PR#16264)
- `read.dcf()` sometimes misread lines longer than 8191 characters. (Reported by Hervé Pagès with a patch.)
- `within(df, ...)` no longer drops columns whose name start with a ". ".
- The built-in HTTP server converted entire Content-Type to lowercase including parameters which can cause issues for multi-part form boundaries (PR#16541).

- Modifying slots of S4 objects could fail when the **methods** package was not attached. (PR#16545)
- `splineDesign(*,outer.ok=TRUE)` (**splines**) is better now (PR#16549), and `interpSpline()` now allows `sparse=TRUE` for speedup with non-small sizes.
- If the expression in the traceback was too long, `traceback()` did not report the source line number. (Patch by Kirill Müller.)
- The browser did not truncate the display of the function when exiting with `options("deparse.max.lines")` set. (PR#16581)
- When `bs(*,Boundary.knots=)` had boundary knots inside the data range, extrapolation was somewhat off. (Patch by Trevor Hastie.)
- `var()` and hence `sd()` warn about factor arguments which are deprecated now. (PR#16564)
- `loess(*,weights = *)` stored wrong weights and hence gave slightly wrong predictions for newdata. (PR#16587)
- `aperm(a,*)` now preserves `names(dim(a))`.
- `poly(x,...)` now works when either `raw=TRUE` or `coef` is specified. (PR#16597)
- `data(package=*)` is more careful in determining the path.
- `prettyNum(*,decimal.mark,big.mark)`: fixed bug introduced when fixing PR#16411.

CHANGES IN R 3.2.2

SIGNIFICANT USER-VISIBLE CHANGES

- It is now easier to use secure downloads from ‘`https://`’ URLs on builds which support them: no longer do non-default options need to be selected to do so. In particular, packages can be installed from repositories which offer ‘`https://`’ URLs, and those listed by `setRepositories()` now do so (for some of their mirrors).
Support for ‘`https://`’ URLs is available on Windows, and on other platforms if support for `libcurl` was compiled in and if that supports the `https` protocol (system installations can be expected to do). So ‘`https://`’ support can be expected except on rather old OSes (an example being OS X ‘Snow Leopard’, where a non-system version of `libcurl` can be used).
(Windows only) The default method for accessing URLs *via* `download.file()` and `url()` has been changed to be “`winet`” using Windows API calls. This changes the way proxies need to be set and security settings made: there have been some reports of ‘`ftp:`’ sites being inaccessible under the new default method (but the previous methods remain available).

NEW FEATURES

- `cmdscale()` gets new option `list`. for increased flexibility when a list should be returned.
- `configure` now supports `texinfo` version 6.0, which (unlike the change from 4.x to 5.0) is a minor update. (Wish of PR#16456.)
- (Non-Windows only) `download.file()` with `default.method = "auto"` now chooses “`libcurl`” if that is available and a ‘`https://`’ or ‘`ftps://`’ URL is used.

- (Windows only) `setInternet2(TRUE)` is now the default. The command-line option `--internet2` and environment variable `R_WIN_INTERNET2` are now ignored.

Thus by default the "internal" method for `download.file()` and `url()` uses the "wininet" method: to revert to the previous default use `setInternet2(FALSE)`.

This means that '`https://`' URLs can be read by default by `download.file()` (they have been readable by `file()` and `url()` since R 3.2.0).

There are implications for how proxies need to be set (see `?download.file`).

- `chooseCRANmirror()` and `chooseBioCmirror()` now offer HTTPS mirrors in preference to HTTP mirrors. This changes the interpretation of their `ind` arguments: see their help pages.
- `capture.output()` gets optional arguments `type` and `split` to pass to `sink()`, and hence can be used to capture messages.

C-LEVEL FACILITIES

- Header '`Rconfig.h`' now defines `HAVE_ALLOCA_H` if the platform has the '`alloca.h`' header (it is needed to define `alloca` on Solaris and AIX, at least: see 'Writing R Extensions' for how to use it).

INSTALLATION and INCLUDED SOFTWARE

- The `libtool` script generated by `configure` has been modified to support FreeBSD >= 10 (PR#16410).

BUG FIXES

- The HTML help page links to demo code failed due to a change in R 3.2.0. (PR#16432)
- If the `na.action` argument was used in `model.frame()`, the original data could be modified. (PR#16436)
- `getGraphicsEvent()` could cause a crash if a graphics window was closed while it was in use. (PR#16438)
- `matrix(x, nr, nc, byrow = TRUE)` failed if `x` was an object of type "expression".
- `strptime()` could overflow the allocated storage on the C stack when the timezone had a non-standard format much longer than the standard formats. (Part of PR#16328.)
- `options(OutDec = s)` now signals a warning (which will become an error in the future) when `s` is not a string with exactly one character, as that has been a documented requirement.
- `prettyNum()` gains a new option `input.d.mark` which together with other changes, e.g., the default for `decimal.mark`, fixes some `format()`ting variants with non-default `getOption("OutDec")` such as in PR#16411.
- `download.packages()` failed for type equal to either "both" or "binary". (Reported by Dan Tenerbaum.)
- The `dendrogram` method of `labels()` is much more efficient for large dendograms, now using `rapply()`. (Comment #15 of PR#15215)
- The "port" algorithm of `nls()` could give spurious errors. (Reported by Radford Neal.)

- Reference classes that inherited from reference classes in another package could invalidate methods of the inherited class. Fixing this requires adding the ability for methods to be “external”, with the object supplied explicitly as the first argument, named `.self`. See “Inter-Package Superclasses” in the documentation.
- `readBin()` could fail on the SPARC architecture due to alignment issues. (Reported by Radford Neal.)
- `qt(*,df=Inf,ncp=.)` now uses the natural `qnorm()` limit instead of returning `NaN`. (PR#16475)
- Auto-printing of S3 and S4 values now searches for `print()` in the base namespace and `show()` in the **methods** namespace instead of searching the global environment.
- `polym()` gains a `coefs = NULL` argument and returns class “poly” just like `poly()` which gets a new `simple=FALSE` option. They now lead to correct `predict()`ions, e.g., on subsets of the original data.
- `rhyper(nn,<large>)` now works correctly. (PR#16489)
- `ttkimage()` did not (and could not) work so was removed. Ditto for `tkimage.cget()` and `tkimage.configure()`. Added two Ttk widgets and missing subcommands for Tk’s `image` command: `ttskscale()`, `ttspsinbox()`, `tkimage.delete()`, `tkimage.height()`, `tkimage.inuse()`, `tkimage.type()`, `tkimage.types()`, `tkimage.width()`. (PR#15372, PR#16450)
- `getClass("foo")` now also returns a class definition when it is found in the cache more than once.

Changes on CRAN

2015-06-01 to 2015-11-30

by Kurt Hornik and Achim Zeileis

New packages in CRAN task views

Bayesian `eco`, `rstan`.

ChemPhys `compositions`.

ClinicalTrials `samplesize`.

Cluster `BayesLCA`, `NbClust`, `dbscan`, `longclust`.

Distributions `BivarP`, `CompGLM`, `Compounding`, `DiscreteLaplace`, `DiscreteWeibull`, `EMMIXskew`, `ExtDist`, `FMStable`, `GIGRvg`, `GenBinomApps`, `GenOrd`, `HAC`, `JohnsonDistribution`, `LIHNPSD`, `MM`, `MitISEM`, `MixedTS`, `NormalLaplace`, `ORDER2PARENT`, `OrdNor`, `PerMallows`, `SCI`, `TTmoment`, `csn`, `degreenet`, `dirmult`, `dislap`, `emg`, `fpow`, `frmqa`, `gambin`, `gb`, `gldist`, `kolmim`, `logitnorm`, `minimax`, `mnormpow`, `mvprpb`, `nCDunnett`, `polyAeppli`, `poweRlaw`, `qmap`, `rtdists`, `sfsmisc`, `skellam`, `symmoments`, `vines`.

Econometrics `gets`, `midasr`, `sfa`, `spfrontier`, `ssfa`.

Environmetrics `eco`, `mra`.

ExperimentalDesign `ALTopt`, `Crossover`, `EngrExpt`, `LDOD`, `MAMS`, `MaxPro`, `OPDOE`, `OptGS`, `OptInterim`, `PopED`, `VdgRsm`, `agridat`, `bcrm`, `blockTools`, `blocksdesign`, `daewr`, `designGG`, `geospt`, `oapackage`, `pipe.design`, `rodd`, `simrel`, `sp23design`, `toxtestD`.

Finance `FatTailsR`, `PortRisk`, `Rblpapi`, `covmat`, `credule`.

HighPerformanceComputing `flowr`, `future`, `partDSA`, `toaster`.

MachineLearning `FCNN4R`, `Rborist`, `ranger`.

MetaAnalysis `MetaPath`, `RcmdrPlugin.RMTCJags`, `etma`, `joint.Cox`, `meta4diag`, `metaSEM`, `metagear`, `xmeta`.

Multivariate `cwhmisc`, `delt`, `knnncat`.

NaturalLanguageProcessing `stringi`, `textreuse`.

NumericalMathematics `Pade`, `lamW`, `mvQuad`.

OfficialStatistics `hot.deck`, `mipfp`.

Optimization `ECOSolveR`, `NlcOptim`, `Rdsdp`, `bvls`, `copulaedas`, `kofnGA`, `lbfgsb3`, `matchingR`, `nls2`, `nnls`, `onls`, `qap`, `tabuSearch`.

Phylogenetics `SigTree`, `markophylo`, `pmc`.

Psychometrics `OpenMx`, `pwrRasch`.

ReproducibleResearch `papeR`.

SocialSciences `Amelia`, `MCMCglmm`^{*}, `PAFit`, `PSAGraphics`, `RSiena`, `VGAM`^{*}, `VIM`, `arm`, `betareg`, `biglm`, `catspec`, `demography`, `dispmod`, `elrm`, `ergm`, `influence.ME`, `logistf`, `logmult`, `lsmeans`^{*}, `mi`^{*}, `mlogit`, `multgee`, `multiplex`, `nlstools`, `norm`, `np`, `simpleboot`, `statnet`, `visreg`.

Spatial `OasisR`, `rgrass7`, `spBayesSurv`, `spatsurv`, `tmap`.

SpatioTemporal `STMedianPolish`, `ctmcmove`, `ctmm`, `moveHMM`, `rsatscan`.

Survival `dynpred`, `glrt`, `parfm`, `survJamda`.

TimeSeries `EMD`, `LSTS`, `PCA4TS`, `Rlibeemd`, `TSMining`, `Wats`, `ZIM`, `autovarCore`, `bentcableAR`, `changepoint`, `dtwclust`, `ecp`, `fanplot`, `hht`, `imputeTS`, `jmotif`, `mlVAR`, `spectral.methods`, `tseriesEntropy`, `wbsts`.

WebTechnologies `RSocrata`, `bigml`, `googlePublicData`, `htmltab`, `pxweb`, `rsnps`.

gR `BDgraph`, `FBFsearch`, `huge`, `ndtv`, `networkDynamic`.

(* = core package)

New contributed packages

ACDm Tools for Autoregressive Conditional Duration Models. Author: Markus Belfrage.

ACSNMineR Gene Enrichment Analysis from ACSN Maps or Gmt Files. Authors: Paul Deveau [aut, cre], Eric Bonnet [aut].

ACSWR A Companion Package for the Book “A Course in Statistics with R”. Author: Prabhanjan Tattar.

AF Model-Based Estimation of Confounder-Adjusted Attributable Fractions. Authors: Elisabeth Dahlqwist and Arvid Sjolander.

AFM Atomic Force Microscope Image Analysis. Authors: Mathieu Beauvais [aut, cre], Irma Liascukiene [aut], Jessem Landoulsi [aut].

AHR Estimation and Testing of Average Hazard Ratios. Author: Matthias Brueckner.

ANOM Analysis of Means. Author: Philip Pallmann.

APSIM General Utility Functions for the ‘Agricultural Production Systems Simulator’. Author: Justin Fainges.

ART Aligned Rank Transform for Nonparametric Factorial Analysis. Author: Pablo J. Villacorta.

AdaptGauss Gaussian Mixture Models (GMM). Authors: Michael Thrun, Onno Hansen-Goos, Rabea Griese, Catharina Lippmann, Jorn Lotsch, Alfred Ultsch.

AggregateR Aggregate Numeric, Date and Categorical Variables by an ID. Authors: Matthias Bogaert, Michel Ballings, Dirk Van den Poel.

AlgebraicHaploPackage Haplotype Two Snips Out of a Paired Group of Patients. Author: Jan Wolfertz.

ArgumentCheck Improved Communication to Users with Respect to Problems in Function Arguments. Author: Benjamin Nutter.

AssocTests Genetic Association Studies. Authors: Lin Wang [aut], Wei Zhang [aut], Qizhai Li [aut], Weicheng Zhu [ctb].

AutoModel Automated Hierarchical Multiple Regression with Assumptions Checking. Author: Alex Lishinski.

AutoregressionMDE Minimum Distance Estimation in Autoregressive Model. Author: Jiwoong Kim.

AzureML Discover, Publish and Consume Web Services on Microsoft Azure Machine Learning. Authors: Raymond Laghaeian [aut, cre], Brianna Gerads [aut], Ritika Ravichandra [aut], Alex Wang [aut].

BCEE The Bayesian Causal Effect Estimation Algorithm. Authors: Denis Talbot, Geneviève Lefebvre, Juli Atherton.

BEDMatrix Matrices Backed by Binary PED Files (PLINK). Authors: Alexander Grueneberg [aut, cre], Lian Lian [ctb], Gustavo de los Campos [ctb].

BIGDAWG Case-Control Analysis of Multi-Allelic Loci. Authors: Derek Pappas, Steve Mack, Jill Hollenbach.

BTLLasso Modelling Heterogeneity in Paired Comparison Data. Author: Gunther Schauberger.

Bagidis BAses GIving DIstances. Author: Catherine Timmermans.

BayesBD Bayesian Boundary Detection in Images. Author: Meng Li.

BayesMAMS Designing Bayesian Multi-Arm Multi-Stage Studies. Authors: Philip Pallmann, Amanda Turner.

BiTrinA Binarization and Trinarization of One-Dimensional Data. Authors: Stefan Mundus, Christoph Müssel, Florian Schmid, Ludwig Lausser, Tamara J. Blätte, Martin Hopfensitz, Hans A. Kestler.

Biocomb Feature Selection and Classification with the Embedded Validation Procedures for Biomedical Data Analysis. Authors: Natalia Novoselova, Junxi Wang, Frank Pessler, Frank Klawonn.

Blossom Statistical Comparisons with Distance-Function Based Permutation Tests. Authors: Marian Talbert, Jon Richards, Paul Mielke, and Brian Cade.

CALF Coarse Approximation Linear Function. Authors: Stephanie Lane [aut, cre], Clark Jeffries [aut], Diana Perkins [aut].

CANSIM2R Directly Extracts Complete CANSIM Data Tables. Author: Marco Lugo.

COMBIA Synergy/Antagonism Analyses of Drug Combinations. Author: Muhammad Kashif.

CTTShiny Classical Test Theory via Shiny. Authors: William Kyle Hamilton [aut, cre], Atsushi Mizumoto [aut].

CUB A Class of Mixture Models for Ordinal Data. Authors: Maria Iannario, Domenico Piccolo.

CUSUMdesign Compute Decision Interval and Average Run Length for CUSUM Charts. Authors: Douglas M. Hawkins, David H. Olwell, Boxiang Wang.

Canopy Accessing Intra-Tumor Heterogeneity and Tracking Longitudinal and Spatial Clonal Evolutionary History by Next-Generation Sequencing. Authors: Yuchao Jiang, Nancy R. Zhang.

CensMixReg Censored Linear Mixture Regression Models. Authors: Luis Benites Sanchez, Victor Hugo Lachos.

ChannelAttribution Markov Model for the Online Multi-Channel Attribution Problem. Author: Davide Altomare.

CircOutlier Detecting of Outliers in Circular Regression. Authors: Azade Ghazanfarihesari, Majid Sarmad.

- ClamR** Time Series Modeling for Climate Change Proxies. Author: Jonathan M. Lees.
- ClustGeo** Clustering of Observations with Geographical Constraints. Authors: Amaury Labenne, Marie Chavent, Vanessa Kuentz-Simonet and Jerome Saracco.
- ClusterStability** Assessment of Stability of Individual Objects or Clusters in Partitioning Solutions. Authors: Etienne Lord, Francois-Joseph Lapointe, and Vladimir Makarenkov.
- Combine** Game-Theoretic Probability Combination. Authors: Alaa Ali, Marta Padilla and David R. Bickel.
- CommT** Comparative Phylogeographic Analysis using the Community Tree Framework. Author: Michael Gruenstaeudl.
- CompR** Paired Comparison Data Analysis. Author: Michel Semenou.
- CompareCausalNetworks** Interface to Diverse Estimation Methods of Causal Networks. Authors: Christina Heinze, Nicolai Meinshausen.
- ComplexAnalysis** Numerically Evaluate Integrals and Derivatives (also Higher Order) of Vector- And Complex-Valued Functions. Author: Char Leung.
- ConsRank** Compute the Median Ranking(s) According to the Kemeny's Axiomatic Approach. Authors: Antonio D'Ambrosio, Sonia Amodio.
- CopyNumber450kCancer** Baseline Correction for Copy Number Data from Cancer Samples. Author: Nour-al-dain Marzouka [aut, cre].
- CoxPlus** Cox Regression (Proportional Hazards Model) with Multiple Causes and Mixed Effects. Author: Jing Peng.
- CryptRndTest** Statistical Tests for Cryptographic Randomness. Author: Haydar Demirhan.
- D3M** Two Sample Test with Wasserstein Metric. Author: Yusuke Matsui & Teppei Shimamura.
- DCchoice** Analyzing Dichotomous Choice Contingent Valuation Data. Authors: Tomoaki Nakatani [aut, cph] (original developer), Hideo Aizaki [aut, cre] (code patches), Kazuo Sato [ctb] (theoretical part of the manual).
- DIFboost** Detection of Differential Item Functioning (DIF) in Rasch Models by Boosting Techniques. Author: Gunther Schauberger.
- DJL** Distance Measure Based Judgment and Learning. Author: Dong-Joon Lim.
- DRIP** Discontinuous Regression and Image Processing. Author: Yicheng Kang.
- DT** A Wrapper of the JavaScript Library 'DataTables'. Authors: Yihui Xie [aut, cre], Joe Cheng [ctb], jQuery contributors [ctb, cph] (jQuery in htmlwidgets/lib), SpryMedia Limited [ctb, cph] (DataTables in htmlwidgets/lib), Brian Reavis [ctb, cph] (selectize.js in htmlwidgets/lib), Leon Gersen [ctb, cph] (noUiSlider in htmlwidgets/lib), Bartek Szopka [ctb, cph] (jquery.highlight.js in htmlwidgets/lib), RStudio Inc [cph]. In view: *ReproducibleResearch*.
- DTRlearn** Learning Algorithms for Dynamic Treatment Regimes. Authors: Ying Liu, Yuanjia Wang, Donglin Zeng.
- DYM** Did You Mean? Author: Kosei Abe.
- DataLoader** Import Multiple File Types. Authors: Srivenkatesh Gandhi, Kreshnaa Raam S Bethusamy.
- DiffusionRgqdd** Inference and Analysis for Generalized Quadratic Diffusions. Authors: Etienne A.D. Pienaar [aut, cre], Melvin M. Varughese [ctb].

Directional Directional Statistics. Authors: Michail Tsagris, Giorgos Athineou.

DynTxRegime Methods for Estimating Dynamic Treatment Regimes. Authors: S. T. Holloway, E. B. Laber, K. A. Linn, B. Zhang, M. Davidian, and A. A. Tsiatis.

ECOSolveR Embedded Conic Solver in R. Authors: Anqi Fu [aut], Balasubramanian Narasimhan [aut, cre]. In view: *Optimization*.

EDFIR Estimating Discrimination Factors. Authors: Alex Bond and Robert Robere.

EEM Read and Preprocess Fluorescence Excitation-Emission Matrix (EEM) Data. Author: Vipavee Trivittayasil.

EGRETci Exploration and Graphics for RivEr Trends (EGRET) Confidence Intervals. Authors: Robert Hirsch [aut], Laura DeCicco [aut, cre].

ELMR Extreme Machine Learning (ELM). Author: Alessio Petrozziello [aut, cre].

EMbC Expectation-Maximization Binary Clustering. Authors: Joan Garriga, John R.B. Palmer, Aitana Oltra, Frederic Bartumeus.

EPGLM Gaussian Approximation of Bayesian Binary Regression Models. Author: James Ridgway.

ESKNN Ensemble of Subset of K-Nearest Neighbours Classifiers for Classification and Class Membership Probability Estimation. Authors: Asma Gul, Aris Perperoglou, Zardad Khan, Osama Mahmoud, Werner Adler, Miftahuddin Miftahuddin, and Berthold Lausen.

EditImputeCont Simultaneous Edit-Imputation for Continuous Microdata. Authors: Quanli Wang, Hang J. Kim, Jerome P. Reiter, Lawrence H. Cox and Alan F. Karr.

EloChoice Preference Rating for Visual Stimuli Based on Elo Ratings. Author: Christof Neumann.

EpiBayes Implements Hierarchical Bayesian Models for Epidemiological Applications. Authors: Matthew Branan, Marta Remmenga, Lori Gustafson, Jennifer Hoeting.

EstHer Estimation of Heritability in High Dimensional Sparse Linear Mixed Models using Variable Selection. Authors: Anna Bonnet and Celine Levy-Leduc.

EurosarcBayes Bayesian Single Arm Sample Size Calculation Software. Author: Peter Dutton.

ExplainPrediction Explanation of Predictions for Classification and Regression Models. Author: Marko Robnik-Sikonja.

FACTMLE Maximum Likelihood Factor Analysis. Authors: Koulik Khamaru, Rahul Mazumder.

FCGR Fatigue Crack Growth in Reliability. Authors: Antonio Meneses, Salvador Naya, Javier Tarrio-Saavedra, Ignacio Lopez-Ullibarri.

FCNN4R Fast Compressed Neural Networks for R. Author: Grzegorz Klima. In view: *MachineLearning*.

FENmlm Fixed Effects Nonlinear Maximum Likelihood Models. Author: Laurent Berge.

FIACH Retrospective Noise Control for fMRI. Author: Tim Tierney.

FSA Functions for Simple Fisheries Stock Assessment Methods. Author: Derek Ogle [aut, cre].

FSAdata Data to Support Fish Stock Assessment (FSA) Package. Author: Derek Ogle [aut, cre].

FastBandChol Fast Estimation of a Covariance Matrix by Banding the Cholesky Factor.

Author: Aaron Molstad.

FastGP Efficiently Using Gaussian Processes with Rcpp and RcppEigen. Authors: Giri Gopalan, Luke Bornn.

FastKM A Fast Multiple-Kernel Method Based on a Low-Rank Approximation. Authors: Rachel Marceau, Wenbin Lu, Michele M. Sale, Bradford B. Worrall, Stephen R. Williams, Fang-Chi Hsu, Jung-Ying Tzeng, and Shannon T. Holloway.

Fgmutils Forest Growth Model Utilities. Authors: Clayton Vieira Fraga, Ana Paula Simiqueli, Wagner Amorim da Silva Altoe.

ForecastCombinations Forecast Combinations. Author: Eran Raviv. In view: *TimeSeries*.

FractalParameterEstimation Estimation of Parameters p and q for Randomized Sierpinski Carpet for [p-p-p-q]-Model. Author: Philipp Hermann.

Fragman Fragment Analysis in R. Authors: Giovanny Covarrubias-Pazaran, Luis Diaz-Garcia, Brandon Schlautman, Walter Salazar, Juan Zalapa.

FuzzyLP Fuzzy Linear Programming. Authors: Carlos A. Rabelo [aut, cre], Pablo J. Villacorta [ctb].

GERGM Estimation and Fit Diagnostics for Generalized Exponential Random Graph Models. Authors: Matthew J. Denny, James D. Wilson, Skyler Cranmer, Bruce A. Desmarais, Shankar Bhamidi.

GFD Tests for General Factorial Designs. Authors: Sarah Friedrich, Frank Konietschke, Markus Pauly.

GSSE Genotype-Specific Survival Estimation. Authors: Baosheng Liang, Yuanjia Wang and Donglin Zeng.

GenCAT Genetic Class Association Testing (GenCAT). Authors: Eric Reed, Sara Nuñez, Jing Qian, Andrea Foulkes.

GeneralOaxaca Blinder-Oaxaca Decomposition for Generalized Linear Model. Authors: Aurelien Nicosia and Simon Baillargeon-Ladouceur.

GeoBoxplot Geographic Box Plot. Author: Ao Li.

GiANT Gene Set Uncertainty in Enrichment Analysis. Authors: Florian Schmid, Christoph Müssel, Johann M. Kraus, Hans A. Kestler.

Goslate Goslate Interface. Authors: Author: Florian Schwendinger [aut, cre], Kurt Hornik [cbt], Zhou Qiang [cph].

Grace Graph-Constrained Estimation and Hypothesis Testing. Author: Sen Zhao.

GroupTest Multiple Testing Procedure for Grouped Hypotheses. Author: Zhigen Zhao.

GsymPoint Estimation of the Generalized Symmetry Point, an Optimal Cutpoint in Continuous Diagnostic Tests. Authors: Mónica López-Ratón, Carmen Cadarso-Suárez, Elisa M. Molanes-López, Emilio Letón.

HBglm Hierarchical Bayesian Regression for GLMs. Authors: Asad Hasan, Alireza S. Mahani.

HDGLM Tests for High Dimensional Generalized Linear Models. Author: Bin Guo.

HMDHFDplus Read HMD and HFD Data from the Web. Authors: Tim Riffe, Carl Boe, Josh Goldstein.

HRM High-Dimensional Repeated Measures. Authors: Martin Happ [aut, cre], Harrar W. Solomon [aut], Arne C. Bathke [aut].

Harvest.Tree Harvest the Classification Tree. Author: Bingyuan Liu/Yan Yuan/Qian Shi.

HiCfeat Multiple Logistic Regression for 3D Chromatin Domain Border Analysis. Author: Raphael Mourad.

HydeNet Hybrid Bayesian Networks Using R and JAGS. Authors: Jarrod E. Dalton and Benjamin Nutter.

ICBayes Bayesian Semiparametric Models for Interval-Censored Data. Authors: Chun Pan, Bo Cai, Lianming Wang, and Xiaoyan Lin.

ICC.Sample.Size Calculation of Sample Size and Power for ICC. Authors: Alasdair Rathbone [aut, cre], Saurabh Shaw [aut], Dinesh Kumbhare [aut].

IDTurtle Identify Turtles by their Plastral Biometries. Author: Aitor Valdeon.

IRISMustangMetrics Statistics and Metrics for Seismic Data. Authors: Jonathan Callahan [aut, cre], Rob Casey [aut], Mary Templeton [aut].

IRISSeismic Classes and Methods for Seismic Data Analysis. Authors: Jonathan Callahan [aut, cre], Rob Casey [aut], Mary Templeton [aut].

ITEMAN Classical Item Analysis. Author: Cengiz Zopluoglu.

IalsaSynthesis Synthesizing Information Across Collaborating Research. Authors: Will Beasley [aut, cre], Andrey Koval [aut], Integrative Analysis of Longitudinal Studies of Aging (IALSA) [cph].

ImportExport Import and Export Data. Authors: Roger Pros, Isaac Subirana, Joan Vila.

Information Data Exploration with Information Theory (Weight-of-Evidence and Information Value). Author: Larsen Kim [aut, cre].

InformationValue Performance Analysis and Companion Functions for Binary Classification Models. Author: Selva Prabhakaran.

IntegratedJM Joint Modelling of the Gene-Expression and Bioassay Data, Taking Care of the Effect Due to a Fingerprint Feature. Authors: Rudradev Sengupta, Nolen Joy Perualila.

InterSIM Simulation of Inter-Related Genomic Datasets. Authors: Prabhakar Chalise, Rama Raghavan, Brooke Fridley.

JGEE Joint Generalized Estimating Equation Solver. Author: Gul Inan.

JPEN Covariance and Inverse Covariance Matrix Estimation Using Joint Penalty. Author: Ashwini Maurya.

JRF Joint Random Forest (JRF) for the Simultaneous Estimation of Multiple Related Networks. Authors: Francesca Petralia [aut, cre], Pei Wang [aut], Zhidong Tu [aut], Won-min Song [aut], Adele Cutler [ctb], Leo Breiman [ctb], Andy Liaw [ctb], Matthew Wiener [ctb].

JacobiEigen Classical Jacobi Eigensolution Algorithm. Author: Bill Venables.

KERE Expectile Regression in Reproducing Kernel Hilbert Space. Authors: Yi Yang, Teng Zhang, Hui Zou.

KoulMde Koul's Minimum Distance Estimation in Linear Regression and Autoregression Model. Author: Jiwoong Kim [aut, cre].

LANDD Liquid Association for Network Dynamics Detection. Authors: Shangzhao Qiu, Yan Yan, Tianwei Yu.

LFDR.MLE Estimation of the Local False Discovery Rates by Type II Maximum Likelihood Estimation. Authors: Ye Yang, Marta Padilla, Alaa Ali, Kyle Leckett, Zhenyu Yang, Zuojing Li, Corey M. Yanofsky and David R. Bickel.

LGEWIS Tests for Genetic Association/Gene-Environment Interaction in Longitudinal Gene-Environment-Wide Interaction Studies. Authors: Zihuai He, Seunggeun Lee, Bhramar Mukherjee, Min Zhang.

LGRF Set-Based Tests for Genetic Association in Longitudinal Studies. Author: Zihuai He.

LOGIT Functions, Data and Code for Binary and Binomial Data. Authors: Joseph M Hilbe, Rafael S. de Souza.

LPM Linear Parametric Models Applied to Hydrological Series. Authors: Corrado Tallerini [aut, cre], Salvatore Grimaldi [aut].

LRcontrast Dose Response Signal Detection under Model Uncertainty. Author: Kevin Kokot.

LSDinterface Reading LSD Results (.res) Files. Author: Marcelo C. Pereira.

LSTS Locally Stationary Time Series. Authors: Ricardo Olea, Wilfredo Palma, Pilar Rubio. In view: *TimeSeries*.

Langevin Langevin Analysis in One and Two Dimensions. Authors: Philip Rinn [aut, cre], Pedro G. Lind [aut], David Bastine [ctb].

LaplaceDeconv Laplace Deconvolution with Noisy Discrete Non-Equally Spaced Observations on a Finite Time Interval. Authors: Yves Rozenholc and Marianna Pensky.

Libra Linearized Bregman Algorithms for Generalized Linear Models. Authors: Feng Ruan, Jiechao Xiong and Yuan Yao.

LifeHist Life History Models of Individuals. Author: Ruben H. Roa-Ureta.

LightningR Tools for Communication with Lightning-Viz Server. Author: Ermlab.

LinearRegressionMDE Minimum Distance Estimation in Linear Regression Model. Author: Jiwoong Kim [aut, cre].

LinkedMatrix Column-Linked and Row-Linked Matrices. Authors: Gustavo de los Campos [aut], Alexander Grueneberg [aut, cre].

LncMod Predicting Modulator and Functional/Survival Analysis. Authors: Yongsheng Li, Zishan Wang, Juan Xu*, Xia Li*.

LotkasLaw Runs Lotka's Law which is One of the Special Applications of Zipf's Law. Authors: Kenneth Bunker [aut, cre], Dr. Alon Friedman [ctb].

MANCIE Matrix Analysis and Normalization by Concordant Information Enhancement. Authors: Tao Wang, Chongzhi Zang.

MBTAr Access Data from the Massachusetts Bay Transit Authority (MBTA) Web API. Author: Justin de Benedictis-Kessner [aut, cre].

MCDM Multi-Criteria Decision Making Methods. Author: Blanca A. Ceballos Martin.

MDimNormn Multi-Dimensional MA Normalization for Plate Effect. Author: Mun-Gwan Hong.

MFAg Multiple Factor Analysis (MFA). Author: Paulo Cesar Ossani Marcelo Angelo Cirillo.

MIXFIM Evaluation of the FIM in NLMMs using MCMC. Authors: Marie-Karelle Riviere-Jourdan and France Mentre.

MLCIRTwithin Latent Class Item Response Theory Models Under Within-Item Multi-Dimensionality. Authors: Francesco Bartolucci, Silvia Bacci.

MM2S Single-Sample Classifier of Medulloblastoma Subtypes for Medulloblastoma Patient Samples, Mouse Models, and Cell Lines. Authors: Deena M.A. Gendoo, Benjamin Haibe-Kains.

MM2Sdata Gene Expression Datasets for the 'MM2S' Package. Authors: Deena M.A. Gendoo, Benjamin Haibe-Kains.

MMWRweek Convert Dates to MMWR Day, Week, and Year. Author: Jarad Niemi.

MRQoL Minimal Clinically Important Difference and Response Shift Effect for Health-Related Quality of Life. Author: Ahmad Ousmen.

MScombine Combine Data from Positive and Negative Ionization Mode Finding Common Entities. Author: Monica Calderon-Santiago.

MVT Estimation and Testing for the Multivariate t-Distribution. Author: Felipe Osorio.

ManlyMix Manly Mixture Modeling and Model-Based Clustering. Authors: Xuwen Zhu [aut, cre], Volodymyr Melnykov [aut], Michael Hutt [ctb, cph] (NM optimization in c), Stephen Moshier [ctb, cph] (eigen calculations in c), Rouben Rostamian [ctb, cph] (memory allocation in c).

MatchLinReg Combining Matching and Linear Regression for Causal Inference. Authors: Alireza S. Mahani, Mansour T.A. Sharabiani.

MaxentVariableSelection Selecting the Best Set of Relevant Environmental Variables along with the Optimal Regularization Multiplier for Maxent Niche Modeling. Author: Alexander Jueterbock.

Mediana Clinical Trial Simulations. Authors: Gautier Paux, Alex Dmitrienko.

MethylCapSig Detection of Differentially Methylated Regions using MethylCap-Seq Data. Authors: Deepak N. Ayyala, David E. Frankhouser, Javkhlan-Ochir Ganbat, Guido Marcucci, Ralf Bundschuh, Pearly Yan and Shili Lin.

MixedPoisson Mixed Poisson Models. Authors: Alicja Wolny-Dominiak and Michal Trzeciok.

MoTBFs Learning Hybrid Bayesian Networks using Mixtures of Truncated Basis Functions. Authors: Inmaculada Pérez-Bernabé, Antonio Salmerón.

MonoPhy Allows to Explore Monophyly (or Lack of it) of Taxonomic Groups in a Phylogeny. Author: Orlando Schwery.

MotilityLab Quantitative Analysis of Motion. Authors: Katharina Dannenberg, Jeffrey Berry, Johannes Textor.

MultiGHQuad Multidimensional Gauss-Hermite Quadrature. Author: Karel Kroeze.

MvBinary Modelling Multivariate Binary Data with Blocks of Specific One-Factor Distribution. Authors: Matthieu Marbac and Mohammed Sedki.

NCA Necessary Condition Analysis. Author: Jan Dul.

NEpiC Network Assisted Algorithm for Epigenetic Studies Using Mean and Variance Combined Signals. Author: Peifeng Ruan.

NIPTeR Fast and Accurate Trisomy Prediction in Non-Invasive Prenatal Testing. Author: Dirk de Weerd.

NetSwan Network Strengths and Weaknesses Analysis. Author: Serge Lhomme.

NlcOptim Solve Nonlinear Optimization with Nonlinear Constraints. Authors: Xianyan Chen, Xiangrong Yin. In view: *Optimization*.

NostalgiR Advanced Text-Based Plots. Author: Hien D. Nguyen.

OBsMD Objective Bayesian Model Discrimination in Follow-Up Designs. Author: Marta Nai Ruscone based on Laura Deldossi's code.

OECD Search and Extract Data from the OECD. Author: Eric Persson [aut, cre].

OTE Optimal Trees Ensembles for Regression, Classification and Class Membership Probability Estimation. Authors: Zardad Khan, Asma Gul, Aris Perperoglou, Osama Mahmoud, Werner Adler, Miftahuddin and Berthold Lausen.

OasisR Outright Tool for the Analysis of Spatial Inequalities and Segregation. Author: Mihai Tivadar. In view: *Spatial*.

OptGS Near-Optimal and Balanced Group-Sequential Designs for Clinical Trials with Continuous Outcomes. Authors: James Wason [aut, cre], John Burkardt [ctb]. In view: *ExperimentalDesign*.

OptiQuantR Simplifies and Automates Analyzing and Reporting OptiQuant's log Data. Author: Joao Pinelo Silva [aut, cre].

OriGen Fast Spatial Ancestry via Flexible Allele Frequency Surfaces. Authors: John Michael O Ranola, John Novembre, and Kenneth Lange.

OrthoPanels Dynamic Panel Models with Orthogonal Reparameterization of Fixed Effects. Authors: Davor Cubranic [aut, cre], Mark Pickup [aut], Paul Gustafson [aut], Geoffrey Evans [aut].

PCA4TS Segmenting Multiple Time Series by Contemporaneous Linear Transformation. Authors: Jinyuan Chang, Bin Guo and Qiwei Yao. In view: *TimeSeries*.

PKNCA Perform Pharmacokinetic Non-Compartmental Analysis. Authors: Bill Denney, Clare Buckeridge.

PRIMsrc PRIM Survival Regression Classification. Authors: Jean-Eudes Dazard [aut, cre], Michael Choe [ctb], Michael LeBlanc [ctb], Alberto Santana [ctb].

PabonLasso Pabon Lasso Graphs and Comparing Situations of a Unit in Two Different Times. Authors: H Nezami and H Tabesh and AA Azarian.

Pade Padé Approximant Coefficients. Author: Avraham Adler [aut, cph, cre]. In view: *NumericalMathematics*.

PanelCount Random Effects and/or Sample Selection Models for Panel Count Data. Author: Jing Peng.

ParallelPC Parallelised Versions of Constraint Based Causal Discovery Algorithms. Authors: Thuc Duy Le, Tao Hoang, Shu Hu, and Liang Zhang.

Pasha Preprocessing of Aligned Sequences from HTS Analyses. Authors: Romain Fenouil, Nicolas Descotes, Lionel Spinelli, Jean-Christophe Andrau.

Perc Using Percolation and Conductance to Find Information Flow Certainty in a Direct Network. Authors: Kevin Fujii [aut], Jian Jin [aut, cre], Aaron Shev [aut], Brianne Beisner [aut], Brenda McCowan [aut, cph], Hsieh Fushing [aut, cph].

PharmacoGx Analysis of Large-Scale Pharmacogenomic Data. Authors: Petr Smirnov, Zhaleh Safikhani, Benjamin Haibe-Kains.

Phxnlme Run Phoenix NLME and Perform Post-Processing. Authors: Chay Ngee Lim [aut,cre], Shuang Liang [aut], Kevin Feng [aut,cre], Grygoriy Vasilin [aut], Angela Birnbaum [aut,ths], Jason Chittenden [aut], Bob Leary [ctb], Ana Henry [ctb], Mike Dunlavey [ctb], Samer Mouksassi [com].

PlotPrjNetworks Useful Networking Tools for Project Management. Author: Javier Celigueta Muñoz.

PortfolioEffectEstim High Frequency Price Estimators by PortfolioEffect. Authors: Andrey Kostin [aut, cre], Aleksey Zemnitskiy [aut], Oleg Nechaev [aut].

PortfolioEffectHFT High Frequency Portfolio Analytics by PortfolioEffect. Authors: Aleksey Zemnitskiy [aut, cre], Andrey Kostin [aut], Oleg Nechaev [aut], Craig Otis and others [ctb, cph] (OpenFAST library), Daniel Lemire, Muraoka Taro and others [ctb, cph] (JavaFastPFOR library), Joe Walnes, Jorg Schaible and others [ctb, cph] (XStream library), Dain Sundstrom [ctb, cph] (Snappy library), Extreme! Lab, Indiana University [ctb, cph] (XPP3 library), The Apache Software Foundation [ctb, cph] (Apache Log4j and Commons Lang libraries), Google, Inc. [ctb, cph] (GSON library), Free Software Foundation [ctb, cph] (GNU Trove and GNU Crypto libraries). In view: [Finance](#).

ProNet Biological Network Construction, Visualization and Analyses. Authors: Xiang-Yun Wu and Xia-Yu Xia.

ProTrackR Manipulate and Play ‘ProTracker’ Modules. Author: Pepijn de Vries [aut, cre, dtc].

PsiHat Several Local False Discovery Rate Estimators. Authors: Alaa Ali, Kyle Leckett, Marta Padilla, David R. Bickel (contributions from Bradley Efron, Brit B. Turnbull, Balasubramanian Narasimhan from package locfdr).

PythonInR Use Python from Within R. Author: Florian Schwendinger [aut, cre].

QPot Quasi-Potential Analysis for Stochastic Differential Equations. Authors: Christopher Moore [aut], Christopher Stieha [aut, cre], Ben Nolting [aut], Maria Cameron [aut], Karen Abbott [aut], James Gregson [cph] (author of expression_parser library: https://github.com/jamesgregson/expression_parser).

QuantumClone Clustering Mutations using High Throughput Sequencing (HTS) Data. Author: Paul Deveau [aut, cre].

RClone Partially Clonal Populations Analysis. Authors: Sophie Arnaud-Haond [aut], Diane Bailleul [aut, cre], CLONIX [ctb].

RCriteo Loading Criteo Data into R. Author: Johannes Burkhardt.

RFormatter R Source Code Formatter. Author: Benjamin Fischer [aut, cre].

RGoogleAnalyticsPremium Unsampled Data in R for Google Analytics Premium Accounts. Author: Jalpa Joshi Dave.

RKlout Fetch Klout Scores for Twitter Users. Author: Binayak Goswami.

RMixpanel R API for Mixpanel. Author: Meinhard Ploner.

RNaviCell Visualization of High-Throughput Data on Large-Scale Biological Networks. Authors: Eric Bonnet [aut, cre], Paul Deveau [aut].

RNeo4j Neo4j Driver for R. Author: Nicole White.

ROMIplot Plots Surfaces of Rates of Mortality Improvement. Authors: Roland Rau, Tim Riffe.

ROptimizely R Optimizely API. Authors: Keerthi Chandra [aut, cre], Chris Johannessen [aut], Siddharth Somayajula [ctb].

- RRNA** Secondary Structure Plotting for RNA. Author: JP Bida.
- RStoolbox** Tools for Remote Sensing Data Analysis. Authors: Benjamin Leutner [cre, aut], Ned Horning [aut].
- RViennaCL** ViennaCL C++ Header Files. Author: Charles Determan Jr.
- RateDistortion** Routines for Solving Rate-Distortion Problems. Author: Chris R. Sims.
- Rblpapi** R Interface to Bloomberg. Authors: Whit Armstrong, Dirk Eddelbuettel and John Laing. In view: *Finance*.
- Rcereal** C++ Header Files of ‘cereal’. Authors: Wush Wu, Randolph Voorhies, and Shane Grant.
- RcppFaddeeva** ‘Rcpp’ Bindings for the ‘Faddeeva’ Package. Authors: Baptiste Auguie [aut, cre], Dirk Eddelbuettel [aut], Steven G. Johnson [aut] (Author of Faddeeva).
- RcppShark** R Interface to the Shark Machine Learning Library. Author: Aydin Demircioglu.
- RedditExtractoR** Reddit Data Extraction Toolkit. Author: Ivan Rivera.
- Relatedness** An Algorithm to Infer Relatedness. Author: Fabien Laporte.
- RepeatABEL** GWAS for Multiple Observations on Related Individuals. Author: Lars Ronnegard.
- RevEcoR** Reverse Ecology Analysis on Microbiome. Authors: Yang Cao, Fei Li.
- Rip46** Utils for IP4 and IP6 Addresses. Author: Neal Fultz.
- RobustEM** Robust Mixture Modeling Fitted via Spatial-EM Algorithm for Model-Based Clustering and Outlier Detection. Authors: Aishat Aloba, Kai Yu, Xin Dang, Yixin Chen, and Henry Bart Jr.
- Rphylopars** Phylogenetic Comparative Tools for Missing Data and Within-Species Variation. Authors: Eric W. Goolsby, Jorn Bruggeman, Cecile Ane.
- Rsurrogate** Robust Estimation of the Proportion of Treatment Effect Explained by Surrogate Marker Information. Author: Layla Parast.
- Rtwalk** The R Implementation of the ‘t-walk’ MCMC Algorithm. Author: J Andres Christen.
- RxODE** Facilities for Simulating from ODE-Based Models. Authors: Melissa Hallow [aut, cre], Wenping Wang [aut], David A. James [aut].
- SACCR** SA Counterparty Credit Risk under Basel III. Author: Tasos Grivas.
- SACOBRA** Self-Adjusting COBRA. Authors: Wolfgang Konen [aut], Samineh Bagheri [cre, aut], Patrick Koch [aut].
- SAGA** Software for the Analysis of Genetic Architecture. Authors: Heath Blackmon and Jeffery P. Demuth.
- SALT_sampler** Efficient Sampling on the Simplex. Authors: Hannah Director, Scott Vander Wiel, James Gattiker.
- SDR** Subgroup Discovery Algorithms for R. Authors: Angel M. Garcia [aut, cre], Pedro Gonzalez [aut, cph], Cristobal J. Carmona [aut, cph], Francisco Charte [ctb].
- SEHmodel** Spatial Exposure-Hazard Model for Exposure and Impact Assessment on Exposed Individuals. Authors: Marc Bourotte [ctb], Melen Leclerc [aut], Jean-Francois Rey [aut, cre], Samuel Soubeyrand [ctb], Emily Walker [aut].

SHELF Tools to Support the Sheffield Elicitation Framework (SHELF). Author: Jeremy Oakley.

SIBER Stable Isotope Bayesian Ellipses in R. Authors: Andrew Jackson and Andrew Parnell.

SOPIE Non-Parametric Estimation of the Off-Pulse Interval of a Pulsar. Author: Willem Daniel Schutte.

SPCALDA A New Reduced-Rank Linear Discriminant Analysis Method. Authors: Yue S. Niu, Ning Hao, and Bin Dong.

SSRMST Sample Size Calculation using Restricted Mean Survival Time. Author: Miki Horiguchi.

STAND Statistical Analysis of Non-Detects. Authors: E. L. Frome and D. P. Frome.

STI Calculation of the Standardized Temperature Index. Author: Marc Fasel [aut, cre].

STMedianPolish Spatio-Temporal Median Polish. Authors: William Martinez, Carlos Melo. In view: *SpatioTemporal*.

SchemaOnRead Automated Schema on Read. Author: Michael North [aut, cre].

Sejong KoNLP static dictionaries and Sejong project resources. Author: Heewon Jeon.

SensMixed Analysis of Sensory and Consumer Data in a Mixed Model Framework. Authors: Alexandra Kuznetsova [aut, cre], Per Bruun Brockhoff [aut, ths], Rune Haubo Bojesen Christensen [aut].

ShapeChange Change-Point Estimation using Shape-Restricted Splines. Authors: Xiyue Liao and Mary C Meyer.

Shrinkage Several Shrinkage Effect-Size Estimators. Authors: Corey M. Yanofsky, Zahra Montazeri, Marta Padilla, Alaa Ali and David R. Bickel.

SimDesign Structure for Organizing Monte Carlo Simulation Designs. Author: Phil Chalmers [aut, cre].

SimHaz Simulated Survival and Hazard Analysis for Time-Dependent Exposure. Authors: Danyi Xiong, Teeranan Pokaprakarn, Hiroto Udagawa, Nusrat Rabbee.

SimReg Similarity Regression Functions. Author: Daniel Greene.

SocialMediaLab Tools for Collecting Social Media Data and Generating Networks for Analysis. Author: Timothy Graham & Robert Ackland.

Sofi Interfaz interactiva con fines didacticos. Author: Jose D. Loera.

SoyNAM Soybean Nested Association Mapping Dataset. Authors: Alencar Xavier, William Beavis, James Specht, Brian Diers, Reka Howard, William Muir, Katy Rainey.

SpaDES Develop and Run Spatially Explicit Discrete Event Simulation Models. Authors: Alex M Chubaty [aut, cre], Eliot J B McIntire [aut], Steve Cumming [ctb], Her Majesty the Queen in Right of Canada, as represented by the Minister of Natural Resources Canada [cph].

SparseFactorAnalysis Scaling Count and Binary Data with Sparse Factor Analysis. Authors: Marc Ratkovic, In Song Kim, John Londregan, and Yuki Shiraito.

SparseLearner Sparse Learning Algorithms Using a LASSO-Type Penalty for Coefficient Estimation and Model Prediction. Authors: Pi Guo, Yuantao Hao.

StMoMo Stochastic Mortality Modelling. Authors: Andres Villegas, Pietro Millossovich, Vladimir Kaishev.

StockChina Real-Time Stock Price & Volume in China Market. Author: Xiaodong Deng.

SuperExactTest Exact Test and Visualization of Multi-Set Intersections. Authors: Minghui Wang, Yongzhong Zhao and Bin Zhang.

SurvRank Rank Based Survival Modelling. Author: Michael Laimighofer [aut, cre, ctb].

SwarmSVM Ensemble Learning Algorithms Based on Support Vector Machines. Authors: Tong He, Aydin Demircioglu.

SyncMove Subsample Temporal Data to Synchronal Events and Compute the MCI. Authors: Martin Rimmmer [aut, cre], Thomas Mueller [aut].

TCGA2STAT Simple TCGA Data Access for Integrated Statistical Analysis in R. Authors: Ying-Wooi Wan, Genevera I. Allen, Matthew L. Anderson, Zhandong Liu.

TLBC Two-Level Behavior Classification. Author: Katherine Ellis.

TMB Template Model Builder: A General Random Effect Tool Inspired by ADMB. Authors: Kasper Kristensen [aut, cre, cph], Brad Bell [cph], Hans Skaug [ctb], Arni Magnusson [ctb], Casper Berg [ctb], Anders Nielsen [ctb], Martin Maechler [ctb], Theo Michelot [ctb], Mollie Brooks [ctb], Cole Monnahan [ctb].

TMDB Access to TMDb API - Apiary. Author: Andrea Capozio.

TOC Total Operating Characteristic Curve and ROC Curve. Authors: Robert G. Pontius, Alí Santacruz, Amin Tayyebi, Benoit Parmentier, Kangping Si.

TP.idm Estimation of Transition Probabilities for the Illness-Death Model. Authors: Vanesa Balboa-Barreiro, Jacobo de Una-Alvarez and Luis Meira-Machado.

TRADEr Tree Ring Analysis of Disturbance Events in R. Authors: Pavel Fibich, Jan Altman, Tuomas Aakala, Jiri Dolezal.

TSMining Mining Univariate and Multivariate Motifs in Time-Series Data. Author: Cheng Fan. In view: *TimeSeries*.

TSTr Ternary Search Tree for Auto-Completion and Spell Checking. Authors: Ricardo Merino [aut, cre], Samantha Fernandez [ctb].

TTS Master Curve Estimates Corresponding to Time-Temperature Superposition. Authors: Antonio Meneses, Salvador Naya, Javier Tarrio-Saavedra.

TauStar Efficient Computation of the t^* Statistic of Bergsma and Dassios (2014). Authors: Luca Weihs [aut, cre], Emin Martinian [ctb] (Created the red-black tree library included in package.).

TestingSimilarity Bootstrap Test for Similarity of Dose Response Curves Concerning the Maximum Absolute Deviation. Author: Kathrin Moellenhoff.

TipDatingBeast Using Tip Dates with Phylogenetic Trees in BEAST. Authors: Adrien Rieux, Camilo Khatchikian.

Tmisc Turner Miscellaneous. Author: Stephen Turner.

Traitspace A Predictive Model for Trait Based Community Assembly of Plant Species. Authors: Chaitanya Joshi, Xin Li, Daniel Laughlin.

TransModel Fit Linear Transformation Models for Censored Data. Authors: Jie Zhou, Jiajia Zhang, Wenbin Lu.

TruncatedNormal Truncated Multivariate Normal. Author: Zdravko I. Botev.

UncerIn2 Implements Models of Uncertainty into the Interpolation Functions. Author: Tomas Burian.

UpSetR A More Scalable Alternative to Venn and Euler Diagrams for Visualizing Intersecting Sets. Authors: Jake Conway [cre], Nils Gehlenborg [aut].

VTrack A Collection of Tools for the Analysis of Remote Acoustic Telemetry Data. Authors: Ross G. Dwyer, Mathew E. Watts, Hamish A. Campbell & Craig E. Franklin.

WiSEBoot Wild Scale-Enhanced Bootstrap. Authors: Megan Heyman, Snigdhansu Chatterjee.

WikidataR API Client Library for ‘Wikidata’. Authors: Oliver Keyes [aut, cre], Christian Graul [ctb].

WufooR R Wrapper for the ‘Wufoo.com’ - The Form Building Service. Author: John Malc.

XHWE X Chromosome Hardy-Weinberg Equilibrium. Authors: Xiao-Ping You, Qi-Lei Zou, Jian-Long Li, Ji-Yuan Zhou.

XMRF Markov Random Fields for High-Throughput Genetics Data. Authors: Ying-Wooi Wan, Genevera I. Allen, Yulia Baker, Eunho Yang, Pradeep Ravikumar, Zhandong Liu.

ZRA Dynamic Plots for Time Series Forecasting. Author: David Beiner.

aTSA Alternative Time Series Analysis. Author: Debin Qiu.

abbyyR Access to Abbyy Optical Character Recognition (OCR) API. Author: Gaurav Sood [aut, cre].

abcrf Approximate Bayesian Computation via Random Forests. Authors: Jean-Michel Marin [aut], Pierre Pudlo [aut, cre], Christian P. Robert [ctb].

abodOutlier Angle-Based Outlier Detection. Author: Jose Jimenez.

acmeR Implements ACME Estimator of Bird and Bat Mortality by Wind Turbines. Authors: Robert Wolpert [aut, cre], Jacob Coleman [aut].

addhazard Fit Additive Hazards Models for Survival Analysis. Authors: Jie (Kate) Hu [aut, cre], Norman Breslow [aut], Gary Chan [aut].

alineR Alignment of Phonetic Sequences Using the ‘ALINE’ Algorithm. Authors: Sean Downey [aut, cre], Guowei Sun [aut].

ameco European Commission Annual Macro-Economic (AMECO) Database. Author: Eric Persson [aut, cre].

analogsea Interface to ‘Digital Ocean’. Authors: Scott Chamberlain [aut, cre], Hadley Wickham [aut], Winston Chang [aut], RStudio [cph].

anonymizer Anonymize Data Containing Personally Identifiable Information. Author: Paul Hendricks [aut, cre].

aop Adverse Outcome Pathway Analysis. Author: Lyle D. Burgoon.

apaStyle Generate APA Tables for MS Word. Author: Jort de Vreeze [aut, cre]. In view: *ReproducibleResearch*.

apaTables Create American Psychological Association (APA) Style Tables. Author: David Stanley [aut, cre].

appnn Amyloid Propensity Prediction Neural Network. Authors: Carlos Família, Sarah R. Dennison, Alexandre Quintas, David A. Phoenix.

apricom Tools for the a Priori Comparison of Regression Modelling Strategies. Authors: Romin Pajouheshnia [aut, cre], Wiebe Pestman [aut], Rolf Groenwold [aut].

arqas Application in R for Queueing Analysis and Simulation. Author: Borja Varela.

- artfima** Fit ARTFIMA Model. Authors: A. I. McLeod, Farzad Sabzikar, Mark M. Meerschaert.
- asdreader** Reading ASD Binary Files in R. Author: Pierre Roudier [aut, cre].
- asht** Applied Statistical Hypothesis Tests. Author: Michael P. Fay.
- asremlPlus** Augments the Use of ‘Asreml’ in Fitting Mixed Models. Author: Chris Brien.
- assertive.base** A Lightweight Core of the ‘assertive’ Package. Authors: Richard Cotton [aut, cre], Sunkyu Choi [trl], Ivanka Skakun [trl], Gergely Daroczi [trl], Anton Antonov [trl], Hisham Ben Hamidane [trl], Anja Billing [trl], Aditya Bhagwat [trl], Rasmus Baath [trl], Mine Cetinkaya-Rundel [trl], Aspasia Chatziefthymiou [trl].
- assertive.code** Assertions to Check Properties of Code. Author: Richard Cotton [aut, cre].
- assertive.data** Assertions to Check Properties of Data. Author: Richard Cotton [aut, cre].
- assertive.data.uk** Assertions to Check Properties of Strings. Author: Richard Cotton [aut, cre].
- assertive.data.us** Assertions to Check Properties of Strings. Author: Richard Cotton [aut, cre].
- assertive.datetimes** Assertions to Check Properties of Dates and Times. Author: Richard Cotton [aut, cre].
- assertive.files** Assertions to Check Properties of Files. Author: Richard Cotton [aut, cre].
- assertive.matrices** Assertions to Check Properties of Matrices. Author: Richard Cotton [aut, cre].
- assertive.models** Assertions to Check Properties of Models. Author: Richard Cotton [aut, cre].
- assertive.numbers** Assertions to Check Properties of Numbers. Author: Richard Cotton [aut, cre].
- assertive.properties** Assertions to Check Properties of Variables. Author: Richard Cotton [aut, cre].
- assertive.reflection** Assertions for Checking the State of R. Author: Richard Cotton [aut, cre].
- assertive.sets** Assertions to Check Properties of Sets. Author: Richard Cotton [aut, cre].
- assertive.strings** Assertions to Check Properties of Strings. Author: Richard Cotton [aut, cre].
- assertive.types** Assertions to Check Types of Variables. Author: Richard Cotton [aut, cre].
- autovarCore** Automated Vector Autoregression Models and Networks. Author: Ando Emerencia [aut, cre]. In view: *TimeSeries*.
- averisk** Calculation of Average Population Attributable Fractions and Confidence Intervals. Author: John Ferguson [aut, cre].
- bWGR** Bagging Whole-Genome Regression. Authors: Alencar Xavier, William Muir, Katy Rainey.
- backShift** Learning Causal Cyclic Graphs from Unknown Shift Interventions. Author: Christina Heinze.
- backpipe** Backward Pipe Operator. Authors: Christopher Brown [cre, aut], Decision Patterns [cph].

- backtestGraphics** Interactive Graphics for Portfolio Data. Authors: David Kane, Ziqi Lu, Fan Zhang, Miller Zijie Zhu.
- bacr** Bayesian Adjustment for Confounding. Author: Chi Wang.
- bapred** Batch Effect Removal (in Phenotype Prediction using Gene Data). Authors: Roman Hornung, David Causeur.
- batman** Convert Categorical Representations of Logicals to Actual Logicals. Authors: Oliver Keyes [aut, cre], Ruben C. Arslan [ctb], Christopher Akiki [ctb], Mine Cetinkaya-Rundel [ctb], Peter Meissner [ctb], Ilaria Prosdocimi [ctb], Thomas Leeper [ctb], Amy Lee [ctb], Adolfo Álvarez [ctb].
- batteryreduction** An R Package for Data Reduction by Battery Reduction. Authors: Chunqiao Luo [aut, cre], Ralph D'Agostino [aut].
- bcRep** Advanced Analysis of B Cell Receptor Repertoire Data. Author: Julia Bischof.
- bcrypt** ‘Blowfish’ Password Hashing Algorithm. Authors: Jeroen Ooms [cre, aut], Damien Miller [cph], Niels Provos [cph].
- bdots** Bootstrapped Differences of Time Series. Authors: Michael Seedorff, Jacob Oleson, Grant Brown, Joseph Cavanaugh, and Bob McMurray.
- bedr** Genomic Region Processing using Tools Such as Bedtools, Bedops and Tabix. Authors: Daryl Waggott, Syed Haider, Emilie Lalonde, Clement Fung, Paul C. Boutros.
- betalink** Beta-Diversity Within a Metaweb of Species Interactions. Author: Timothee Poisot.
- bimixt** Estimates Mixture Models for Case-Control Data. Authors: Michelle Winerip, Garrick Wallstrom, Joshua LaBaer.
- biomartr** Functional Annotation and Biological Data Retrieval with R. Author: Hajk-Georg Drost.
- blavaan** Bayesian Latent Variable Analysis. Authors: Edgar Merkle [aut, cre], Yves Rosseel [aut].
- bmeta** A Package for Bayesian Meta-Analysis and Meta-Regression. Authors: Tao Ding, Gianluca Baio.
- bnclassify** Learning Discrete Bayesian Network Classifiers from Data. Authors: Mihaljevic Bojan [aut, cre], Bielza Concha [aut], Larranaga Pedro [aut], Wickham Hadley [ctb] (some code extracted from memoise package). In view: [MachineLearning](#).
- bnstruct** Bayesian Network Structure Learning from Data with Missing Values. Authors: Francesco Sambo [aut, cre], Alberto Franzin [aut].
- brotli** A Compression Format Optimized for the Web. Authors: Jeroen Ooms [aut, cre], Google, Inc [aut, cph] (Brotli C++ library).
- brr** Bayesian Inference on the Ratio of Two Poisson Rates. Author: Stéphane Laurent.
- brranching** Fetch Phylogenies from Many Sources. Author: Scott Chamberlain [aut, cre].
- bssn** Birnbaum-Saunders Model Based on Skew-Normal Distribution. Authors: Rocio Paola Maehara and Luis Benites Sanchez.
- btergm** Temporal Exponential Random Graph Models by Bootstrapped Pseudolikelihood. Authors: Philip Leifeld [aut, cre], Skyler J. Cranmer [ctb], Bruce A. Desmarais [ctb].
- caRpools** CRISPR AnalyzeR for Pooled CRISPR Screens. Authors: Jan Winter, Florian Heigwer.
- calACS** Count All Common Subsequences. Author: Alan Gu.

camtrapR Camera Trap Data Management and Preparation of Occupancy and Spatial Capture-Recapture Analyses. Authors: Juergen Niedballa [aut, cre], Alexandre Courtiol [aut], Rahel Sollmann [aut], John Mathai [ctb], Seth Timothy Wong [ctb], An The Truong Nguyen [ctb], Azlan bin Mohamed [ctb], Andrew Tilker [ctb], Andreas Wilting [ctb, ths].

captioner Numbers Figures and Creates Simple Captions. Author: Letaw Alathea [aut, cre].

captr Client for the Captricity API. Author: Gaurav Sood [aut, cre].

cardioModel Cardiovascular Safety Exposure-Response Modeling in Early-Phase Clinical Studies. Authors: Daniela J Conrado [aut, cre], William S Denney [aut], Gregory J Hather [aut], Danny Chen [ctb].

cartography Thematic Cartography. Authors: Timothée Giraud [cre, aut], Nicolas Lambert [aut].

cate High Dimensional Factor Analysis and Confounder Adjusted Testing and Estimation. Authors: Jingshu Wang [aut], Qingyuan Zhao [aut, cre].

causaldrf Tools for Estimating Causal Dose Response Functions. Authors: Douglas Galagate [cre], Joseph Schafer [aut].

cdcfluvview Retrieve U.S. Flu Season Data from the CDC FluView Portal. Author: Bob Rudis.

chunked Chunkwise Text-File Processing for ‘dplyr’. Author: Edwin de Jonge.

ckanr Client for the Comprehensive Knowledge Archive Network (‘CKAN’) API. Authors: Scott Chamberlain [aut, cre], Imanuel Costigan [ctb], Wush Wu [ctb], Florian Mayer [ctb].

clarifai Access to Clarifai API. Author: Gaurav Sood [aut, cre].

cleango Cleaning Geometries from Spatial Objects. Author: Emmanuel Blondel.

climtrends Statistical Methods for Climate Sciences. Author: Jose Gama [aut, cre].

clipr Read and Write from the System Clipboard. Authors: Matthew Lincoln [aut, cre], Louis Maddox [ctb].

clisymbols Unicode Symbols at the R Prompt. Authors: Gabor Csardi [aut, cre], Sindre Sorhus [aut].

clogitboost Boosting Conditional Logit Model. Authors: Haolun Shi [aut, cre], Guosheng Yin [aut].

clttools Central Limit Theorem Experiments (Theoretical and Simulation). Authors: Simiao Ye, Jingning Mei.

cmsaf Tools for CM SAF Netcdf Data. Author: Steffen Kothe.

cnmlcd Maximum Likelihood Estimation of a Log-Concave Density Function. Authors: Yu Liu, Yong Wang.

coala A Framework for Coalescent Simulation. Authors: Paul Staab [aut, cre, cph], Dirk Metzler [ths].

codyn Community Dynamics Metrics. Authors: Lauren Hallett [aut], Sydney K. Jones [aut], Andrew A. MacDonald [aut], Dan F. B. Flynn [aut], Peter Slaughter [aut], Julie Ripplinger [aut], Scott L. Collins [aut], Corinna Gries [aut], Matthew B. Jones [aut, cre].

colorhclplot Colorful Hierarchical Clustering Dendrograms. Author: Damiano Fantini.

- cometExactTest** Exact Test from the Combinations of Mutually Exclusive Alterations (CoMEt) Algorithm. Authors: Max Leiserson [aut, cre], Hsin-Ta Wu [aut], Fabio Vandin [ctb], Vivian Hsiao [ctb], Benjamin Raphael [ctb].
- conover.test** Conover-Iman Test of Multiple Comparisons Using Rank Sums. Author: Alexis Dinno.
- contoureR** Contouring of Non-Regular Three-Dimensional Data. Author: Nicholas Hamilton.
- controlTest** Median Comparison for Two-Sample Right-Censored Survival Data. Author: Eric Kawaguchi.
- convOSPAT** Convolution-Based Nonstationary Spatial Modeling. Author: Mark D. Risser [aut, cre].
- cord** Community Estimation in G-Models via CORD. Authors: Xi (Rossi) LUO, Florentina Bunea, Christophe Giraud.
- coreNLP** Wrappers Around Stanford CoreNLP Tools. Authors: Taylor Arnold, Lauren Tilton.
- corkscrew** Preprocessor for Data Modeling. Authors: Navin Loganathan [aut], Mohan Manivannan [aut], Santhosh Sasanapuri [aut, cre], LatentView Analytics [ctb].
- corregp** Functions and Methods for Correspondence Regression. Author: Koen Plevoets [aut, cre].
- covBM** Brownian Motion Processes for ‘nlme’ Models. Author: Oliver Stirrup [aut, cre].
- covmat** Covariance Matrix Estimation. Author: Rohit Arora. In view: *Finance*.
- cowplot** Streamlined Plot Theme and Plot Annotations for ‘ggplot2’. Authors: Claus O. Wilke [aut, cre], Hadley Wickham [cph].
- cowsay** Messages, Warnings, Strings with Ascii Animals. Authors: Scott Chamberlain [aut, cre], Tyler Rinker [aut], Thomas Leeper [aut], Noam Ross [aut], Rich FitzJohn [aut], Carson Sievert [aut], Kiyoko Gotanda [aut], Andy Teucher [aut], Karl Broman [aut], Franz-Sebastian Krah [aut].
- cpgen** Parallelized Genomic Prediction and GWAS. Author: Claas Heuer.
- creditr** Credit Default Swaps in R. Authors: Heidi Chen, Yuanchu Dang, David Kane, Yang Lu, Skylar Smith, Kanishka Malik, and Miller Zijie Zhu.
- credule** Credit Default Swap Functions. Authors: Bertrand Le Nezet [cre, aut, cph], Richard Brent [ctb, cph], John Burkardt [ctb, cph]. In view: *Finance*.
- crmPack** Object-Oriented Implementation of CRM Designs. Authors: Daniel Sabanes Bove, Wai Yin Yeung, Giuseppe Palermo, Thomas Jaki.
- crop** Graphics Cropping Tool. Author: Marius Hofert [aut, cre].
- crrp** Penalized Variable Selection in Competing Risks Regression. Author: Zhixuan Fu.
- crskdiag** Diagnostics for Fine and Gray Model. Author: Jianing Li.
- cruts** Interface to Climatic Research Unit Time-Series Version 3.21 Data. Author: Benjamin M. Taylor.
- ctmcmove** Modeling Animal Movement with Continuous-Time Discrete-Space Markov Chains. Author: Ephraim Hanks. In view: *SpatioTemporal*.
- ctmm** Continuous-Time Movement Modeling. Authors: Chris H. Fleming and J. M. Calabrese. In view: *SpatioTemporal*.

ctsem Continuous Time Structural Equation Modelling. Authors: Manuel Voelkle [aut, cph] (Original development of continuous time model specification within OpenMx, advisor for further development), Han Oud [aut, cph] (Original development of continuous time model specification within OpenMx), Charles Driver [aut, cre, cph] (Further development of continuous time model specification within OpenMx, package development, documentation and maintenance).

cvxbiclustr Convex Biclustering Algorithm. Authors: Eric C. Chi, Genevera I. Allen, Richard G. Baraniuk.

cycleRtools Tools for Cycling Data Analysis. Author: Jordan Mackie [aut, cre].

cymruservices Query Team Cymru IP Address, Autonomous System Number (ASN), Border Gateway Protocol (BGP), Bogon and Malware Hash Data Services. Author: Bob Rudis [aut, cre].

d3heatmap Interactive Heat Maps Using ‘htmlwidgets’ and ‘D3.js’. Authors: Joe Cheng [aut, cre], Tal Galili [aut], RStudio, Inc. [cph], Michael Bostock [ctb, cph] (D3.js library), Justin Palmer [ctb, cph] (d3.tip library).

dMod Dynamic Modeling and Parameter Estimation in ODE Models. Author: Daniel Kaschek.

dad Three-Way Data Analysis Through Densities. Authors: Rachid Boumaza, Pierre Santagostini, Smail Yousfi, Sabine Demotes-Mainard.

datafsm Estimating Finite State Machine Models from Data. Authors: Nay John J. [cre, aut], Gilligan Jonathan M. [aut].

dataonderivatives Easily Source Publicly Available Data on Derivatives. Author: Imanuel Costigan [aut, cre].

datastepr An Implementation of a SAS-Style Data Step. Author: Brandon Taylor.

dbSCAN Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms. Authors: Michael Hahsler [aut, cre, cph], Sunil Arya [ctb, cph], David Mount [ctb, cph]. In view: *Cluster*.

dc3net Inferring Disease Networks via Differential Network Inference. Author: Gokmen Altay.

ddR Distributed Data Structures in R. Authors: Edward Ma, Indrajit Roy, Michael Lawrence.

deformula Integration of One-Dimensional Functions with Double Exponential Formulas. Author: Hiroyuki Okamura.

dequer An R ‘Deque’ Container. Author: Drew Schmidt [aut, cre].

describer Describe Data in R Using Common Descriptive Statistics. Author: Paul Hendricks [aut, cre].

desiR Desirability Functions for Ranking, Selecting, and Integrating Data. Author: Stanley E. Lazic.

detector Detect Data Containing Personally Identifiable Information. Author: Paul Hendricks [aut, cre].

diezeit R Interface to the ZEIT ONLINE Content API. Author: Christian Graul.

diffR Display Differences Between Two Files using Codediff Library. Author: John Muschelli [aut, cre].

dina Bayesian Estimation of DINA Model. Author: Steven Andrew Culpepper [aut, cph, cre].

- distcomp** Computations over Distributed Data without Aggregation. Authors: Balasubramanian Narasimhan [aut, cre], Marina Bendersky [aut], Sam Gross [aut], Terry M. Therneau [ctb], Thomas Lumley [ctb].
- diverse** Diversity Measures for Complex Systems. Authors: Miguel R. Guevara, Dominik Hartmann, Marcelo Mendoza.
- dml** Distance Metric Learning in R. Authors: Yuan Tang, Gao Tao, Xiao Nan.
- docopulae** Optimal Designs for Copula Models. Author: Andreas Rappold [aut, cre].
- docxtractr** Extract Data Tables from Microsoft Word Documents. Author: Bob Rudis [aut, cre].
- dotwhisker** Dot-and-Whisker Plots of Regression Results. Authors: Frederick Solt, Yue Hu.
- downscale** Downscaling Species Occupancy. Author: Charles Marsh [aut, cre].
- drLumi** Multiplex Immunoassays Data Analysis. Authors: Hector Sanz [aut, cre], John Aponte [aut], Jaroslaw Harezlak [aut], Yan Dong [aut], Magdalena Murawska [aut], Clarissa Valim [aut], Aintzane Ayestaran [ctb], Ruth Aguilar [ctb], Gemma Moncunill [ctb].
- dtwSat** Time-Weighted Dynamic Time Warping for Remote Sensing Time Series Analysis. Author: Victor Maus [aut, cre].
- dtwclust** Time Series Clustering with Dynamic Time Warping. Author: Alexis Sardá-Espinosa. In view: *TimeSeries*.
- easyVerification** Ensemble Forecast Verification for Large Datasets. Authors: Jonas Bhend [aut, cre], Jacopo Ripoldi [ctb], Claudia Mignani [ctb], Irina Mahlstein [ctb], Rebecca Hiller [ctb], Christoph Spirig [ctb], Mark Liniger [ctb], Andreas Weigel [ctb], Joaquín Bedia Jimenez [ctb], Matteo De Felice [ctb].
- easypower** Sample Size Estimation for Experimental Designs. Author: Aaron McGarvey.
- ecospace** Simulating Community Assembly and Ecological Diversification Using Ecospace Frameworks. Author: Phil Novack-Gottshall [aut, cre].
- ecotoxicology** Methods for Ecotoxicology. Author: Jose Gama [aut, cre, trl].
- edgar** Platform for EDGAR Filing Management. Authors: Gunratan Lonare, Bharat Patil.
- edgebundleR** Circle Plot with Bundled Edges. Authors: Garth Tarr [aut, cre], Ellis Patrick [aut], Kent Russell [ctb].
- eel** Extended Empirical Likelihood. Authors: Fan Wu and Yu Zhang.
- eemR** Tools for Pre-Processing Emission-Excitation-Matrix (EEM). Author: Philippe Massicotte [aut, cre].
- efflog** The Causal Effects for a Causal Loglinear Model. Author: Gloria Gheno [aut, cre].
- elasso** Enhanced Least Absolute Shrinkage and Selection Operator Regression Model. Author: Pi Guo.
- epandist** Statistical Functions for the Censored and Uncensored Epanechnikov Distribution. Author: Mathias Borritz Milfeldt [aut, cre].
- epanetReader** Read Epanet Files into R. Author: Bradley J. Eck.
- epiDisplay** Epidemiological Data Display Package. Author: Virasakdi Chongsuvivatwong.
- epifit** Flexible Modelling Functions for Epidemiological Data Analysis. Authors: Kazutaka Doi [aut, cre], Kei Sakabe [ctb], Masataka Taruri [ctb].

- erp.easy** Event-Related Potential (ERP) Data Exploration Made Easy. Author: Travis Moore [aut, cre].
- etma** Epistasis Test in Meta-Analysis (ETMA). Author: Chin Lin. In view: *MetaAnalysis*.
- exreport** Fast, Reliable and Elegant Reproducible Research. Authors: Jacinto Arias [aut, cre], Javier Cozar [aut].
- extremogram** Estimation of Extreme Value Dependence for Time Series Data. Authors: Nadezda Frolova, Ivor Cribben.
- eyetrackingR** Eye-Tracking Data Analysis. Authors: Jacob Dink [aut, cre], Brock Ferguson [aut].
- ezec** Easy Interface to Effective Concentration Calculations. Authors: Zhian N. Kamvar [cre, aut], Niklaus J. Grunwald [ths, ctb].
- ezsummary** Summarise Data in the Quick and Easy Way. Author: Hao Zhu [aut, cre].
- fancycut** A Fancy Version of ‘base::cut’. Author: Adam Rich [aut, cre].
- fastdigest** Fast, Low Memory-Footprint Digests of R Objects. Authors: Gabriel Becker, Bob Jenkins (SpookyHash algorithm and C++ implementation).
- fasttime** Fast Utility Function for Time Parsing and Conversion. Author: Simon Urbanek.
- favnums** A Dataset of Favourite Numbers. Authors: Oliver Keyes [aut, cre], Alex Bellos [cph].
- filematrix** File-Backed Matrix Class with Convenient Read and Write Access. Author: Andrey Shabalin.
- flacco** Feature-Based Landscape Analysis of Continuous and Constraint Optimization Problems. Authors: Pascal Kerschke [aut, cre], Jan Dagefoerde [aut].
- flexPM** Flexible Parametric Models for Censored and Truncated Data. Author: Paolo Frumento.
- flowDiv** Cytometric Diversity Indices from ‘FlowJo’ Workspaces. Authors: Bruno M.S. Wanderley, María Victoria Quiroga, André M. Amado, Fernando Unrein.
- flows** Flow Selection and Analysis. Authors: Timothée Giraud [cre, aut], Laurent Beauguitte [aut], Marianne Guérois [ctb].
- forestmodel** Forest Plots from Regression Models. Author: Nick Kennedy.
- fourPNO** Bayesian 4 Parameter Item Response Model. Author: Steven Andrew Culpepper [aut, cre].
- frailtySurv** General Semiparametric Shared Frailty Model. Authors: John V. Monaco [aut, cre], Malka Gorfine [aut], Li Hsu [aut].
- franc** Detect the Language of Text. Authors: Gabor Csardi, Titus Wormer, Maciej Ceglowski, Jacob R. Rideout, and Kent S. Johnson.
- freqdom** Frequency Domain Analysis for Multivariate Time Series. Authors: Hormann S., Kidzinski L.
- ftsspec** Spectral Density Estimation and Comparison for Functional Time Series. Author: Shahin Tavakoli [aut, cre].
- fulltext** Full Text of Scholarly Articles Across Many Data Sources. Author: Scott Chamberlain [aut, cre].
- functools** Functional Programming in R. Author: Paul Hendricks [aut, cre].

- funcy** Functional Clustering Algorithms. Authors: Christina Yassouridis [aut, cre], Dominik Ernst [ctb], Madison Giacofci [ctb], Sophie Lambert-Lacroix [ctb], Guillemette Marot [ctb], Franck Picard [ctb], Nicoleta Serban [ctb], Huijing Jiang [ctb], Gareth James [ctb], Catherine Sugar [ctb], Hans-Georg Mueller [ctb], Jie Peng [ctb], Chiou Jeng-Min [ctb], Pai-Ling Li [ctb].
- fungible** Fungible Coefficients and Monte Carlo Functions. Authors: Niels G. Waller and Jeff Jones.
- funr** Simple Utility Providing Terminal Access to all R Functions. Author: Sahil Seth [aut, cre].
- future** A Future API for R. Author: Henrik Bengtsson [aut, cre, cph]. In view: *HighPerformanceComputing*.
- gamsel** Fit Regularization Path for Generalized Additive Models. Authors: Alexandra Chouldechova and Trevor Hastie.
- gaston** Genetic Data Manipulation (Quality Control, GRM and LD Computations, PCA), Linear Mixed Models (AIREML Algorithm), Association Testing. Author: Hervé Perdry & Claire Dandine-Roulland.
- gdtools** Utilities for Graphical Rendering. Authors: David Gohel [aut, cre], Hadley Wickham [aut], Jeroen Ooms [ctb], Yixuan Qiu [ctb], RStudio [cph].
- gendist** Generated Probability Distribution Models. Author: Shaiful Anuar Abu Bakar.
- generator** Generate Data Containing Fake Personally Identifiable Information. Author: Paul Hendricks [aut, cre].
- geo** Draw and Annotate Maps, Especially Charts of the North Atlantic. Authors: Hoskuldur Bjornsson, Sigurdur Thor Jonsson, Arni Magnusson, and Bjarki Thor Elvarsson.
- geoknife** Web-Processing of Large Gridded Datasets. Authors: Jordan Read [aut, cre], Jordan Walker [aut], Alison Appling [aut], David Blodgett [aut], Emily Read [aut], Luke Winslow [aut].
- geosptdb** Spatio-Temporal; Inverse Distance Weighting and Radial Basis Functions with Distance-Based Regression. Authors: Carlos Melo, Oscar Melo.
- gesis** R Client for GESIS Data Catalogue (DBK). Author: Eric Persson [aut, cre].
- getMet** Get Meteorological Data for Hydrological Modeling. Authors: Andrew Sommerlot [aut, cre], Daniel Fuka [aut], Zachary Easton [aut].
- ggfortify** Data Visualization Tools for Statistical Analysis Results. Authors: Masaaki Horikoshi and Yuan Tang.
- ggplot2movies** Movies Data. Authors: Hadley Wickham [aut, cre], RStudio [cph].
- ggsn** North Symbols and Scale Bars for Maps Created with ‘ggplot2’ or ‘ggmap’. Author: Oswaldo Santos Baquero [aut, cre].
- gimms** Download and Process GIMMS NDVI3g Data. Author: Florian Detsch.
- gitlabr** Access to the Gitlab API. Author: Jirka Lewandowski [aut, cre].
- gkmSVM** Gapped-Kmer Support Vector Machine. Author: Mahmoud Ghandi.
- glamlasso** Lasso Penalization in Large Scale Generalized Linear Array Models. Author: Adam Lund.
- glm.ddR** Distributed ‘glm’ for Big Data using ‘ddR’ API. Authors: Vishrut Gupta, Arash Fard.

globals Identify Global Objects in R Expressions. Author: Henrik Bengtsson [aut, cre, cph].

glycanr Tools for Analysing N-Glycan Data. Author: Ivo Ugrina [aut, cre].

gmapsdistance Distance and Travel Time Between Two Points from Google Maps. Author: Rodrigo Azuero Melo.

gmum.r GMUM Machine Learning Group Package. Authors: Wojciech Czarnecki, Stanisław Jastrzebski, Marcin Data, Igor Sieradzki, Mateusz Bruno-Kaminski, Karol Jurek, Piotr Kowenzowski, Michał Pletty, Konrad Talik, Maciej Zgliczynski.

gmwm Generalized Method of Wavelet Moments. Authors: James Balamuta [aut, cph], Stephane Guerrier [ctb, cre, cph], Roberto Molinari [ctb, cph], Wenchao Yang [ctb].

gofCopula Goodness-of-Fit Tests for Copulæ. Authors: Ostap Okhrin, Shulin Zhang, Qian M. Zhou, Simon Trimborn.

googleAuthR Easy Authentication with Google OAuth2 APIs. Author: Mark Edmondson [aut, cre].

googlesheets Manage Google Spreadsheets from R. Authors: Jennifer Bryan [aut, cre], Joanna Zhao [aut].

gpuR GPU Functions for R Objects. Author: Charles Determan Jr. In view: *HighPerformanceComputing*.

gquad G Quadruplex Motif Prediction Tool. Author: Hannah O. Ajoge.

graticule Meridional and Parallel Lines for Maps. Author: Michael D. Sumner [aut, cre].

growthcurver Simple Metrics to Summarize Growth Curves. Author: Kathleen sprouffske [aut, cre].

grpregOverlap Penalized Regression Models with Overlapping Grouped Covariates. Authors: Yaohui Zeng, Patrick Breheny.

gtrendsR R Functions to Perform and Display Google Trends Queries. Authors: Philippe Massicotte [aut, cre], Dirk Eddelbuettel [aut].

gyriq Kinship-Adjusted Survival SNP-Set Analysis. Authors: Martin Leclerc and Lajmi Lakhal Chaieb.

hashids Generate Short Unique YouTube-Like IDs (Hashes) from Integers. Authors: Alex Shum [aut, cre], Ivan Akimov [aut] (original author of hashids – implemented in javascript), David Aurelio [ctb] (implemented hashids in python 2 and 3).

hashr Hash R Objects to Integers Fast. Authors: Mark van der Loo [aut, cre], Paul Hsieh [ctb].

hdnom Nomograms for High-Dimensional Cox Models. Authors: Miao Zhu Li, Nan Xiao.

hierband Convex Banding of the Covariance Matrix. Authors: Jacob Bien, Florentina Bunea, and Luo Xiao.

highmean Two-Sample Tests for High-Dimensional Mean Vectors. Authors: Lifeng Lin and Wei Pan.

hindexcalculator H-Index Calculator using Data from a Web of Science (WoS) Citation Report. Author: Sepand Alavifard [aut, cre].

hit Hierarchical Inference Testing. Author: Jonas Klasen [aut, cre].

homomorpheR Homomorphic Computations in R. Author: Balasubramanian Narasimhan [aut, cre].

- hotspot** Software Hotspot Analysis. Author: Csaba Farago.
- hqreg** Regularization Paths for Huber Loss Regression and Quantile Regression Penalized by Lasso or Elastic-Net. Author: Congrui Yi.
- httpcode** HTTP Status Code Helper. Author: Scott Chamberlain [aut, cre].
- humaniformat** A Parser for Human Names. Author: Oliver Keyes.
- iLaplace** Improved Laplace Approximation for Integrals of Unimodal Functions. Authors: Erlis Ruli [aut, cre], Nicola Sartori [aut], Laura Ventura [aut].
- iNEXT** Interpolation and Extrapolation for Species Diversity. Authors: T. C. Hsieh, K. H. Ma and Anne Chao.
- iaQCA** The Irvine Robustness Assessment for Qualitative Comparative Analysis. Authors: C. Ben Gibson [aut, cre], Burrel Vann Jr [aut].
- icsw** Inverse Compliance Score Weighting. Authors: Peter M. Aronow, Dean Eckles and Kyle Peyton.
- idendr0** Interactive Dendograms. Author: Tomas Sieger.
- idm** Incremental Decomposition Methods. Authors: Alfonso Iodice D' Enza [aut], Angelos Markos [aut, cre], Davide Buttarazzi [ctb].
- ifaTools** Toolkit for Item Factor Analysis with OpenMx. Author: Joshua N. Pritikin [cre, aut].
- imPois** Imprecise Inferential Framework for Poisson Sampling Model. Authors: Chel Hee Lee [aut, cre, cph], Mikelis Bickis [aut, ths, cph].
- imager** Image Processing Library Based on CImg. Authors: Simon Barthelme [aut, cre], Antoine Cecchi [ctb].
- immer** Item Response Models for Multiple Ratings. Authors: Alexander Robitzsch [aut, cre], Jan Steinfeld [aut].
- imputeMissing** Impute Missing Values in a Predictive Context. Authors: Matthijs Meire, Michel Ballings, Dirk Van den Poel.
- imputeTS** Time Series Missing Value Imputation. Author: Steffen Moritz. In view: [Time-Series](#).
- inbreedR** Analysing Inbreeding Based on Genetic Markers. Authors: Martin A. Stoffel [aut, cre], Mareike Esser [aut].
- inegiR** Integrate INEGI's (Mexican Stats Office) API with R. Author: Eduardo Flores.
- influenceR** Software Tools to Quantify Structural Importance of Nodes in a Network. Authors: Jacobs Simon [aut], Khanna Aditya [aut, cre], Madduri Kamesh [ctb], Bader David [ctb].
- injectoR** R Dependency Injection. Author: Lev Kuznetsov.
- instaR** Access to Instagram API via R. Authors: Pablo Barbera [aut, cre], Tiago Dantas [ctb], Jonne Guyt [ctb].
- interactionTest** Calculates Critical Test Statistics to Control False Discovery and Familywise Error Rates in Marginal Effects Plots. Authors: Justin Esarey and Jane Lawrence Sumner.
- interplot** Plot the Effects of Variables in Interaction Terms. Authors: Frederick Solt, Yue Hu.
- invLT** Inversion of Laplace-Transformed Functions. Author: Christopher Barry.

- ioncopy** Calling Copy Number Alterations in Amplicon Sequencing Data. Author: Jan Budczies.
- iotools** I/O Tools for Streaming. Authors: Simon Urbanek, Taylor Arnold.
- ipflasso** Integrative Lasso with Penalty Factors. Authors: Anne-Laure Boulesteix, Mathias Fuchs.
- iptools** Manipulate, Validate and Resolve IP Addresses. Authors: Bob Rudis [aut, cre], Oliver Keyes [aut].
- isoph** Isotonic Proportional Hazards Model. Authors: Yunro Chung [cre], Anastasia Ivanova, Michael G. Hudgens and Jason P. Fine.
- jetset** One-to-One Gene-Probeset Mapping for Affymetrix Human Microarrays. Authors: Qiyuan Li, Aron Eklund.
- jmotif** Tools for Time Series Analysis Based on Symbolic Aggregate Dicretization. Author: Pavel Senin [aut, cre]. In view: *TimeSeries*.
- jrvFinance** Basic Finance; NPV/IRR/Annuities/Bond-Pricing; Black Scholes. Author: Jayanth Varma [aut, cre].
- jug** Create a Simple Web API for your R Functions. Author: Bart Smeets.
- jvnVaR** Value at Risk. Author: Hung Vu.
- kdecopula** Kernel Smoothing for Bivariate Copula Densities. Author: Thomas Nagler [aut, cre].
- kergp** Gaussian Process Laboratory. Authors: Yves Deville, David Ginsbourger, Olivier Roustant. Contributors: Nicolas Durrande.
- kernDeepStackNet** Kernel Deep Stacking Networks. Authors: Thomas Welchowski and Matthias Schmid.
- kerndwd** Distance Weighted Discrimination (DWD) and Kernel Methods. Authors: Boxiang Wang, Hui Zou.
- keyplayer** Locating Key Players in Social Networks. Author: Weihua An; Yu-Hsin Liu.
- keypress** Wait for a Key Press in a Terminal. Author: Gabor Csardi [aut, cre].
- kineticF** Framework for the Analysis of Kinetic Visual Field Data. Author: Dipesh E Patel & Mario Cortina-Borja.
- kmeans.ddR** Distributed k-Means for Big Data using 'ddR' API. Authors: Vishrut Gupta, Arash Fard.
- knitLatex** Knitr Helpers — Mostly Tables. Author: John Shea [aut, cre]. In view: *ReproducibleResearch*.
- ksrlive** Identify Kinase Substrate Relationships Using Dynamic Data. Author: Westa Domanova.
- kwb.hantush** Calculation of Groundwater Mounding Beneath an Infiltration Basin. Author: Michael Rustler.
- labelrank** Predicting Rankings of Labels. Authors: Artur Aiguzhinov [cre], Carlos Soares [aut].
- landest** Landmark Estimation of Survival and Treatment Effect. Author: Layla Parast.
- landsat8** Landsat 8 Imagery Rescaled to Reflectance, Radiance and/or Temperature. Author: Alexandre dos Santos.

- lasvmR** A Simple Wrapper for the LASVM Solver. Author: Aydin Demircioglu.
- latex2exp** Use L^AT_EX Expressions in Plots. Author: Stefano Meschiari [aut, cre]. In view: *ReproducibleResearch*.
- lcopula** Liouville Copulas. Authors: Leo Belzile [aut, cre], Christian Genest [aut, ctb], Alexander J. McNeil [ctb], Johanna G. Neslehova [ctb].
- ldamatch** Multivariate Condition Matching by Backwards Elimination Using Linear Discriminant Analysis. Authors: Kyle Gorman [aut, cre], Geza Kiss [ctb].
- ldatuning** Tuning of the LDA Models Parameters. Author: Murzintcev Nikita [aut, cre].
- leaflet** Create Interactive Web Maps with the JavaScript ‘Leaflet’ Library. Authors: Joe Cheng [aut, cre], Yihui Xie [aut], Hadley Wickham [ctb], jQuery Foundation and contributors [ctb, cph] (jQuery library), Vladimir Agafonkin [ctb, cph] (Leaflet library), CloudMade [cph] (Leaflet library), Leaflet contributors [ctb] (Leaflet library), Leaflet Providers contributors [ctb, cph] (Leaflet Providers plugin), RStudio [cph].
- learNN** Examples of Neural Networks. Author: Bastiaan Quast [aut, cre].
- lfda** Local Fisher Discriminant Analysis. Author: Yuan Tang with great contributions from Zachary Deane-Mayer.
- lift** Compute the Top Decile Lift and Plot the Lift Curve. Authors: Steven Hoornaert, Michel Ballings, Dirk Van den Poel.
- liftr** Dockerize R Markdown Documents. Authors: Miao Zhu Li [ctb], Tengfei Yin [ctb], Nan Xiao [aut, cre].
- likelihoodAsy** Functions for Likelihood Asymptotics. Authors: Ruggero Bellio and Donald Pierce.
- lira** LInear Regression in Astronomy. Author: Mauro Sereno.
- listWithDefaults** List with Defaults. Author: Russell S. Pierce.
- littler** R at the Command-Line via ‘r’. Author: Dirk Eddelbuettel.
- logisticPCA** Binary Dimensionality Reduction. Author: Andrew J. Landgraf.
- longurl** Expand Short URLs Using the ‘LongURL’ API. Author: Bob Rudis [aut, cre].
- loo** Efficient Leave-One-Out Cross-Validation and WAIC for Bayesian Models. Authors: Aki Vehtari [aut], Andrew Gelman [aut], Jonah Gabry [cre, aut], Juho Piironen [ctb], Ben Goodrich [ctb].
- lookupTable** Look-Up Tables using S4. Authors: Enzo Jia [aut, cre], Marc Maier [aut].
- lpbrim** LP-BRIM Bipartite Modularity. Authors: Timothee Poisot, Daniel B Stouffer.
- lqr** Robust Linear Quantile Regression. Authors: Christian E. Galarza, Luis Benites, Victor H. Lachos.
- lrsg** Linear Regression by Gibbs Sampling. Author: Adam Mantz.
- lse1** Solving Least Squares Problems under Equality/Inequality Constraints. Authors: Yong Wang [aut, cre], Charles L. Lawson [aut], Richard J. Hanson [aut].
- lsl** Latent Structure Learning. Author: Po-Hsien Huang [aut, cre].
- lucr** Currency Formatting and Conversion. Author: Oliver Keyes.
- lulcc** Land Use Change Modelling in R. Author: Simon Moulds.
- luzlogr** Lightweight Logging for R Scripts. Author: Ben Bond-Lamberty [aut, cre].

- mHG** Minimum-Hypergeometric Test. Author: Kobi Perl.
- maGUI** A Graphical User Interface for Microarray Data Analysis. Authors: Dhammapal Bharne, Praveen Kant, Vaibhav Vindal.
- magclass** Data Class and Tools for Handling Spatial-Temporal Data. Authors: Jan Philipp Dietrich, Benjamin Bodirsky, Misko Stevanovic, Lavinia Baumstark, Christoph Bertram, Markus Bonsch, Anastasis Giannousakis, Florian Humpenoeder, David Klein, Ina Neher, Michaja Pehl, Anselm Schultes.
- marketeR** Enhanced Analytics for Marketers Navigating the Ocean of Web Data. Author: Felix Mikaelian.
- markophylo** Markov Chain Models for Phylogenetic Trees. Authors: Utkarsh J. Dang and G. Brian Golding. In view: *Phylogenetics*.
- matlabr** An Interface for MATLAB using System Calls. Author: John Muschelli.
- matlib** Matrix Functions for Teaching and Learning Linear Algebra and Multivariate Statistics. Authors: Michael Friendly [aut, cre], John Fox [ctb], Georges Monette [ctb].
- mcemGLM** Maximum Likelihood Estimation for Generalized Linear Mixed Models. Author: Felipe Acosta Archila.
- merTools** Tools for Analyzing Mixed Effect Regression Models. Authors: Jared E. Knowles [aut, cre], Carl Frederick [aut].
- meta4diag** Meta-Analysis for Diagnostic Test Studies. Authors: Jingyi Guo and Andrea Riebler. In view: *MetaAnalysis*.
- metaSEM** Meta-Analysis using Structural Equation Modeling. Author: Mike W.-L. Cheung. In view: *MetaAnalysis*.
- metafuse** Fused Lasso Approach in Regression Coefficient Clustering. Authors: Lu Tang, Peter X.K. Song.
- metansue** Meta-Analysis of Studies with Non Statistically-Significant Unreported Effects. Author: Joaquim Radua.
- meteR** Fitting and Plotting Tools for the Maximum Entropy Theory of Ecology (METE). Authors: Andy Rominger, Cory Merow.
- meteo** Spatio-Temporal Analysis and Mapping of Meteorological Observations. Authors: Milan Kilibarda, Aleksandar Sekulic, Tomislav Hengl, Edzer Pebesma, Benedikt Graeler.
- metricsgraphics** Create Interactive Charts with the JavaScript ‘MetricsGraphics’ Library. Authors: Bob Rudis [aut, cre], Ali Almossawi [ctb, cph] (MetricsGraphics library), Hamilton Ulmer [ctb, cph] (MetricsGraphics library), Mozilla [cph] (MetricsGraphics library), jQuery Foundation and contributors [ctb, cph] (jQuery library).
- mev** Multivariate Extreme Value Distributions. Author: Leo Belzile [aut, cre].
- mgm** Estimating Mixed Graphical Models. Author: Jonas Haslbeck.
- mhde** Minimum Hellinger Distance Test for Normality. Authors: Paul W. Eslinger [aut, cre], Heather Orr [aut, ctb].
- midrangeMCP** Multiples Comparisons Procedures Based on Studentized Midrange and Range Distributions. Authors: Ben Deivide [cre], Daniel Furtado [aut].
- missDeaths** Correctly Analyse Disease Recurrence with Missing at Risk Information using Population Mortality. Authors: Tomaz Stupnik [aut, cre], Maja Pohar Perme [aut].

- mixtox** Curve Fitting and Mixture Toxicity Assessment. Author: Xiangwei Zhu.
- mlVAR** Multi-Level Vector Autoregression. Authors: Sacha Epskamp, Marie K. Deserno and Laura F. Bringmann. In view: *TimeSeries*.
- mldr.datasets** R Ultimate Multilabel Dataset Repository. Authors: David Charte [cre], Francisco Charte [aut].
- mlma** Multilevel Mediation Analysis. Authors: Qingzhao Yu, Bin Li.
- mlsjunkgen** Use the MLS Junk Generator Algorithm to Generate a Stream of Pseudo-Random Numbers. Author: Steve Myles [aut, cre].
- mmc** Multivariate Measurement Error Correction. Author: Jaejoon Song.
- mmtfa** Model-Based Clustering and Classification with Mixtures of Modified t Factor Analyzers. Authors: Jeffrey L. Andrews, Paul D. McNicholas, and Mathieu Chalifour.
- modQR** Multiple-Output Directional Quantile Regression. Authors: Miroslav Šiman [aut], Pavel Boček [aut, cre].
- modelObj** A Model Object Framework for Regression Analysis. Author: Shannon T. Holloway.
- mogavs** Multiobjective Genetic Algorithm for Variable Selection in Regression. Authors: Tommi Pajala [aut, cre], Pekka Malo [aut], Ankur Sinha [aut], Timo Kuosmanen [ctb].
- molaR** Dental Surface Complexity Measurement Tools. Authors: James D. Pampush [aut, cre, cph], Julia M. Winchester [aut, cph], Paul E. Morse [aut, cph], Alexander Q. Vining [aut, cph].
- momr** Mining Metaomics Data (MetaOMineR). Authors: Edi Prifti, Emmanuelle Le Chatelier.
- monogeneaGM** Geometric Morphometric Analysis of Monogenean Anchors. Author: Tsung Fei Khang.
- monographaR** Taxonomic Monographs Tools. Author: Marcelo Reginato.
- moveHMM** Animal Movement Modelling using Hidden Markov Models. Authors: Theo Michelot, Roland Langrock, Toby Patterson, Eric Rexstad. In view: *SpatioTemporal*.
- mplot** Graphical Model Stability and Variable Selection Procedures. Authors: Garth Tarr [aut, cre], Samuel Mueller [aut], Alan Welsh [aut].
- mri** Modified Rand Index (1 and 2.1 and 2.2) and Modified Adjusted Rand Index (1 and 2.1 and 2.2). Author: Marjan Cugmas.
- ms.sev** Package for Calculation of ARMSS, Local MSSS and Global MSSS. Authors: Helga Westerlind, Ali Manouchehrinia.
- multiwave** Estimation of Multivariate Long-Memory Models Parameters. Authors: Sophie Achard [aut, cre], Irene Gannaz [aut].
- mvQuad** Methods for Multivariate Quadrature. Author: Constantin Weiser (HHU of Duesseldorf / Germany). In view: *NumericalMathematics*.
- mvtboost** Tree Boosting for Multivariate Outcomes. Author: Patrick Miller [aut, cre].
- nbconvertR** Vignette Engine Wrapping IPython Notebooks. Author: Philipp Angerer.
- nhanesA** NHANES Data Retrieval. Author: Christopher Endres.
- nivm** Noninferiority Tests with Variable Margins. Author: Michael P. Fay.

- nlnet** Nonlinear Network Reconstruction and Clustering Based on DCOL (Distance Based on Conditional Ordered List). Authors: Haodong Liu, Tianwei Yu.
- nlrr** Non-Linear Relative Risk Estimation and Plotting. Author: Yiqiang Zhan [aut, cre].
- nomclust** Hierarchical Nominal Clustering Package. Authors: Sulc Zdenek, Rezankova Hana.
- nparACT** Non-Parametric Measures of Actigraphy Data. Author: Christine Blume Nayan-tara Santhi Manuel Schabus.
- npregfast** Nonparametric Estimation of Regression Models with Factor-by-Curve Interactions. Authors: Marta Sestelo [aut, cre], Nora M. Villanueva [aut], Javier Roca-Pardinas [aut].
- npsf** Nonparametric and Stochastic Efficiency and Productivity Analysis. Authors: Oleg Badunenko [aut, cre], Yaryna Kolomiytseva [aut], Pavlo Mozharovskyi [aut].
- npsurv** Non-Parametric Survival Analysis. Author: Yong Wang.
- oaColors** OpenAnalytics Colors Package. Author: Jason Waddell.
- oaPlots** OpenAnalytics Plots Package. Authors: Jason Waddell, Willem Ligtenberg.
- oai** General Purpose ‘Oai-PMH’ Services Client. Author: Scott Chamberlain [aut, cre].
- oapackage** Orthogonal Array Package. Author: Pieter Thijs Eendebak. In view: *ExperimentalDesign*.
- odds.converter** Betting Odds Conversion. Author: Marco Blume.
- oglmx** Estimation of Ordered Generalized Linear Models. Author: Nathan Carroll.
- olctools** Open Location Code Handling in R. Author: Oliver Keyes.
- onewaytests** One-Way Independent Groups Design. Authors: Osman Dag, Anil Dolgun, N. Meric Konar.
- optCluster** Determine Optimal Clustering Algorithm and Number of Clusters. Authors: Michael Sekula, Somnath Datta, and Susmita Datta.
- optigrab** Command-Line Parsing for an R World. Author: Christopher Brown.
- optimixture** Optimal Mixture Weights in Multiple Importance Sampling. Authors: Hera Y. He, Art B. Owen.
- ordinalNet** Penalized Ordinal Regression. Author: Mike Wurm [aut, cre].
- osrm** Interface Between R and the OpenStreetMap-Based Routing Service OSRM. Author: Timothée Giraud [cre, aut].
- pAnalysis** Benchmarking and Rescaling R2 using Noise Percentile Analysis. Authors: Joseph G Kreke, Sangeet Khemlani, Greg Trafton.
- packagetrackr** Track R Package Downloads from RStudio’s CRAN Mirror. Author: Jirka Lewandowski.
- packcircles** Circle Packing. Authors: Michael Bedward [aut, cre], David Eppstein [aut] (Original author of Python code for graph-based circle packing ported to C++ for this package).
- paco** Procrustes Application to Cophylogenetic Analysis. Authors: Juan Antonio Balbuena, Timothee Poisot, Matthew Hutchinson, Fernando Cagua.
- pageviews** An API Client for Wikimedia Traffic Data. Author: Oliver Keyes.

- palettetown** Use Pokemon Inspired Colour Palettes. Author: Tim Lucas.
- palr** Colour Palettes for Data. Author: Michael D. Sumner [aut, cre].
- pangaeair** Client for the ‘Pangaea’ Database. Authors: Scott Chamberlain [aut, cre], Kara Woo [aut], Andrew MacDonald [aut], Naupaka Zimmerman [aut], Gavin Simpson [aut].
- parallelML** A Parallel-Voting Algorithm for many Classifiers. Author: Wannes Rosiers (InfoFarm).
- params** Simplify Parameters. Authors: Sahil Seth [aut, cre], Yihui Xie [ctb] (kable from knitr R/table.R).
- parsec** Partial Orders in Socio-Economics. Authors: Alberto Arcagni [aut, cre], Marco Fattore [ctb].
- patPRO** Visualizing Temporal Microbiome Data. Authors: Hannigan GD, Loesche MA, Hodkinson BP, Mehta S, Grice EA.
- pbdZMQ** Programming with Big Data – Interface to ZeroMQ. Authors: Wei-Chen Chen [aut, cre], Drew Schmidt [aut], Whit Armstrong [ctb] (some functions are modified from the rzmq for backward compatibility), Brian Ripley [ctb] (C code of shellexec), R Core team [ctb] (some functions are modified from the R source code).
- pcadapt** Principal Component Analysis for Outlier Detection. Authors: Keurcien Luu, Michael G.B. Blum.
- pch** Piecewise Constant Hazards Models for Censored and Truncated Data. Author: Paolo Frumento.
- pco** Panel Cointegration Tests. Author: Georgi Marinov.
- personograph** Pictographic Representation of Treatment Effects. Authors: Joel Kuiper [aut, cre], Iain Marshall [aut].
- phonenumer** Convert Letters to Numbers and Back as on a Telephone Keypad. Author: Steve Myles [aut, cre].
- phyext2** An Extension (for Package ‘SigTree’) of Some of the Classes in Package ‘phylobase’. Author: J. Conrad Stack.
- phylometrics** Estimating Statistical Errors of Phylogenetic Metrics. Authors: Xia Hua, Lindell Bromham.
- phylosignal** Exploring the Phylogenetic Signal in Continuous Traits. Author: Francois Keck.
- phyndr** Matches Tip and Trait Data. Authors: Rich FitzJohn, Matt Pennell and Will Cornwell.
- pid** Process Improvement using Data. Author: Kevin Dunn [aut, cre].
- piecewiseSEM** Piecewise Structural Equation Modeling. Author: Jon Lefcheck.
- pinnacle.API** A Wrapper for the Pinnacle Sports API. Authors: Marco Blume, Nicholas Jhirad, Amine Gassem.
- pixiedust** Tables so Beautifully Fine-Tuned You Will Believe It’s Magic. Author: Benjamin Nutter [aut, cre].
- pkgconfig** Private Configuration for R Packages. Author: Gabor Csardi.
- pla** Parallel Line Assays. Author: Jens Henrik Badsberg.

plfMA A GUI to View, Design and Export Various Graphs of Data. Authors: Dhammapal Bharne, Vaibhav Vindal.

plotly Create Interactive Web Graphics via Plotly's JavaScript Graphing Library. Authors: Carson Sievert [aut, cre], Chris Parmer [aut, cph], Toby Hocking [aut], Scott Chamberlain [aut], Karthik Ram [aut], Marianne Corvellec [aut], Pedro Despouy [aut].

pnea Parametric Network Enrichment Analysis. Authors: Mirko Signorelli, Veronica Vinciotti and Ernst C. Wit.

pnf Prime Numbers and Integer Factorization. Author: Abby Rothway.

poisson Simulating Homogenous & Non-Homogenous Poisson Processes. Authors: Christian Brock [aut], Daniel Slade [ctb].

polyfreqs Bayesian Population Genomics in Autopolyploids. Author: Paul Blischak [aut, cre].

popEpi Functions for Epidemiological Analysis using Population Data. Authors: J Miettinen, M Rantanen, K Seppa, J Pitkaniemi.

poprxl Read GenAlEx Files Directly from Excel. Author: Zhian N. Kamvar [cre, aut].

praise Praise Users. Authors: Gabor Csardi, Sindre Sorhus.

prclust Penalized Regression-Based Clustering Method. Authors: Chong Wu, Wei Pan.

prepdat Preparing Experimental Data for Statistical Analysis. Authors: Ayala S. Allon [aut, cre], Roy Luria [aut], Jim Grange [ctb], Nachshon Meiran [ctb].

preprocmb Tools for Preprocessing Combinations. Author: Markus Vattulainen.

prettymapr Scale Bar, North Arrow, and Pretty Margins in R. Author: Dewey Dunnington.

primes Generate and Test for Prime Numbers. Author: Oliver Keyes.

prism Access Data from the Oregon State Prism Climate Project. Authors: Hart Edmund [aut, cre], Kendon Bell [aut].

pro Point-Process Response Model for Optogenetics. Authors: Xi (Rossi) LUO with contributions from Dylan Small and Vikaas Sohal.

profilr Quickly Profile Data in R. Author: Paul Hendricks [aut, cre].

progenyClust Finding the Optimal Cluster Number Using Progeny Clustering. Author: C.W. Hu.

prop.comb.RR Analyzing Combination of Proportions and Relative Risk. Authors: Maria Alvarez and Javier Roca-Pardinas.

proton The Proton Game. Authors: Przemysław Biecek [aut, cre], Witold Chodor [trl], Foundation SmarterPoland.pl [cph].

prozor Minimal Protein Set Explaining Peptide Spectrum Matches. Author: Witold Wolski.

pscore Standardizing Physiological Composite Risk Endpoints. Author: Joshua F. Wiley.

ptycho Bayesian Variable Selection with Hierarchical Priors. Authors: Laurel Stell and Chiara Sabatti.

purge Purge Training Data from Models. Authors: Marc Maier [cre], Chaoqun Jia [ctb], MassMutual Advanced Analytics [aut].

purrr Functional Programming Tools. Authors: Hadley Wickham [aut, cre], Lionel Henry [ctb], RStudio [cph].

- pweight** P-Value Weighting. Authors: Edgar Dobriban [aut, cre], Kristen Fortney [aut].
- pwrRasch** Statistical Power Simulation for Testing the Rasch Model. Authors: Takuya Yanagida [cre, aut], Jan Steinfeld [aut], Thomas Kiefer [ctb]. In view: *Psychometrics*.
- pystr** Python String Methods in R. Author: Nicole White.
- qap** Heuristics for the Quadratic Assignment Problem (QAP). Authors: Michael Hahsler [aut, cre, cph], Franz Rendl [ctb, cph]. In view: *Optimization*.
- qlcData** Processing Data for Quantitative Language Comparison (QLC). Author: Michael Cysouw.
- qlcVisualize** Visualization for Quantitative Language Comparison (QLC). Author: Michael Cysouw.
- qrage** Tools that Create D3 JavaScript Force Directed Graph from R. Authors: Shingo Yamamoto [aut, cre], RStudio, Inc. [cph], Michael Bostock [ctb, cph] (D3.js library), jQuery Foundation [cph] (jQuery library and jQuery UI library), jQuery contributors [ctb, cph] (jQuery library), jQuery UI contributors [ctb, cph] (jQuery UI library).
- qrcode** QRcode Generator for R. Author: Victor Teh.
- qrjoint** Joint Estimation in Linear Quantile Regression. Author: Surya Tokdar.
- qtlcharts** Interactive Graphics for QTL Experiments. Authors: Karl W Broman [aut, cre], Michael Bostock [ctb, cph] (d3.js library in htmlwidgets/lib), Justin Palmer [ctb, cph] (d3.tip library in htmlwidgets/lib), Cynthia Brewer [cph] (ColorBrewer library in htmlwidgets/lib), Mark Harrower [cph] (ColorBrewer library in htmlwidgets/lib), The Pennsylvania State University [cph] (ColorBrewer library in htmlwidgets/lib), jQuery Foundation [cph] (jQuery library in htmlwidgets/lib), jQuery contributors [ctb] (jQuery library in htmlwidgets/lib), jQuery UI contributors [ctb] (jQuery UI library in htmlwidgets/lib).
- qualvar** Implements Indices of Qualitative Variation Proposed by Wilcox (1973). Author: Joel Gombin.
- quantable** Streamline Descriptive Analysis of Quantitative Data Matrices. Author: Witold Wolski.
- quantileDA** Quantile Classifier. Authors: Christian Hennig, Cinzia Viroli.
- quickmapr** Quickly Map and Explore Spatial Data. Author: Jeffrey W. Hollister [aut, cre].
- qut** Quantile Universal Threshold. Authors: Jairo Diaz, Sylvain Sardy, Caroline Giacobino, Nick Hengartner.
- rAmCharts** JavaScript Charts API Tool. Authors: Jeffery Petit [aut, cre], Antanas Marcellionis [aut, cph] ('AmCharts' library in th directory htmlwidgets/lib/amcharts), Benoit Thieurmel [aut, ctb], DataKnowledge [ctb] (See official web site at <http://www.datak.fr>).
- rGroovy** Groovy Language Integration. Author: Thomas P. Fuller.
- rPowerSampleSize** Sample Size Computations Controlling the Type-II Generalized Family-Wise Error Rate. Authors: Pierre Lafaye de Micheaux, Benoit Liquet and Jeremie Riou.
- radiomics** Radiomic Image Processing Toolbox. Author: Joel Carlson.
- rafalib** Convenience Functions for Routine Data Exploration. Authors: Rafael A. Irizarry and Michael I. Love.

- randomForest.ddR** Distributed ‘randomForest’ for Big Data using ‘ddR’ API. Authors: Vishrut Gupta, Arash Fard, Winston Li, Matthew Saltz.
- randomizeR** Randomization for Clinical Trials. Authors: Thi Mui Pham [ctb], David Schindler [aut], Diane Uschner [aut, cre].
- ranger** A Fast Implementation of Random Forests. Author: Marvin N. Wright. In view: *MachineLearning*.
- rase** Range Ancestral State Estimation for Phylogeography and Comparative Analyses. Authors: Ignacio Quintero [aut, cre], Forrest W. Crawford [aut], Petr Keil [aut].
- rcanvec** Access and Plot CanVec and CanVec+ Data for Rapid Basemap Creation in Canada. Author: Dewey Dunnington.
- rchess** Chess Move, Generation/Validation, Piece Placement/Movement, and Check/Checkmate/Stalemate Detection. Author: Joshua Kunst.
- rcrypt** Symmetric File Encryption Using GPG. Author: Brett Klamer [aut, cre].
- rddtools** Toolbox for Regression Discontinuity Design (‘RDD’). Authors: Matthieu Stigler [aut], Bastiaan Quast [aut, cre].
- reReg** Recurrent Event Regression. Author: Sy Han (Steven) Chiou.
- recoder** A Simple and Flexible Recoder. Author: Ali Sanaei.
- reda** Recurrent Event Data Analysis. Authors: Wenjie Wang [aut, cre], Haoda Fu [aut], Jun Yan [ctb].
- refund.shiny** Interactive Plotting for Functional Data Analyses. Authors: Julia Wrobel [aut, cre], Jeff Goldsmith [aut].
- relen** Compute Relative Entropy. Author: Soeren Braehmer.
- repjson** Tools for Handling EpiJSON (Epidemiology Data) Files. Authors: Andy South [aut, cre], Thomas Finnie [aut], Ellie Sherrard-Smith [aut], Ana Bento [aut], Thibaut Jombart [aut].
- repo** A Resource Manager for R Objects. Author: Francesco Napolitano.
- reservoir** Tools for Analysis, Design, and Operation of Water Supply Storages. Authors: Sean Turner [aut, cre], Stefano Galelli [aut].
- resumer** Build Resumes with R. Author: Jared Lander [aut, cre]. In view: *ReproducibleResearch*.
- rgeolocate** IP Address Geolocation. Authors: Oliver Keyes [aut, cre], Drew Schmidt [aut], David Robinson [ctb], Maxmind, Inc. [cph], Pascal Gloor [cph].
- rglwidget** ‘rgl’ in ‘htmlwidgets’ Framework. Author: Duncan Murdoch.
- rhandsontable** Interface to the ‘Handsontable.js’ Library. Authors: Jonathan Owen [aut, cre, cph], Jeff Allen [ctb], Yihui Xie [ctb], Enzo Martoglio [ctb], Warpechowski Marcin [ctb, cph] (Handsontable.js library), Handsontable.sp. z o.o. [ctb, cph] (Handsontable.js library), Aisch Gregor [ctb, cph] (Chroma.js library), Wood Tim [ctb, cph] (Moment.js library), Chernev Iskren [ctb, cph] (Moment.js library), Moment.js contributors [ctb, cph] (Moment.js library), Bushell David [ctb, cph] (Pikaday.js library), jQuery Foundation [ctb, cph] (jQuery.js library), Splunk Inc [ctb, cph] (Sparkline.js library).
- ridigbio** Interface to the iDigBio Data API. Authors: Francois Michonneau [aut, cph], Matthew Collins [aut, cre], Scott Chamberlain [ctb].
- riskR** Risk Management. Author: Marcelo Brutti Righi.

- rleafmap** Interactive Maps with R and Leaflet. Author: Francois Keck.
- rnetcarto** Fast Network Modularity and Roles Computation by Simulated Annealing (Rgraph C Library Wrapper for R). Authors: Guilhem Doulcier [aut, cre] (R bindings, current implementation of the simulated annealing algorithm), Roger Guimera [ctb] (Author of the original rgraph library), Daniel B. Stouffer [aut, ths].
- rnn** Recurrent Neural Network. Author: Bastiaan Quast [aut, cre].
- rollply** Moving-Window Add-on for ‘plyr’. Author: Alexandre Genin.
- rosm** Plot Raster Map Tiles from Open Street Map and Other Sources. Authors: Dewey Dunnington [aut, cre], Timothée Giraud [ctb].
- rotl** Interface to the ‘Open Tree of Life’ API. Authors: Francois Michonneau [aut, cre], Joseph Brown [aut], David Winter [aut].
- rpca** RobustPCA: Decompose a Matrix into Low-Rank and Sparse Components. Author: Maciek Sykulski [aut, cre].
- rpdo** Pacific Decadal Oscillation Index. Authors: Joe Thorley [aut, cre], Nathan Mantua [aut, dtc], Steven R. Hare [aut, dtc].
- rpivotTable** Build Powerful Pivot Tables and Dynamically Slice & Dice your Data. Authors: Enzo Martoglio [aut, cre] (R interface), Nicolas kruchten [ctb, cph] (pivotable library).
- rpnf** Point and Figure Package. Author: Sascha Herrmann.
- rrepast** Invoke ‘Repast Symphony’ Simulation Models. Authors: Antonio Prestes Garcia [aut, cre], Alfonso Rodriguez-Paton [aut, ths].
- rscopus** Scopus Database API Interface. Author: John Muschelli.
- rsggm** Robust Sparse Gaussian Graphical Modeling via the Gamma-Divergence. Author: Kei Hirose.
- rstan** R Interface to Stan. Authors: Jiqiang Guo [aut], Daniel Lee [ctb], Krzysztof Sakrejda [ctb], Jonah Gabry [aut], Ben Goodrich [cre, aut], Joel de Guzman [cph] (Boost), Eric Niebler [cph] (Boost), Thomas Heller [cph] (Boost), John Fletcher [cph] (Boost). In view: *Bayesian*.
- rstatscn** R Interface for China National Data. Author: Xuehui YANG.
- rstpm2** Flexible Link-Based Survival Models. Authors: Mark Clements [aut, cre], Xing-Rong Liu [aut], Paul Lambert [ctb].
- rsvd** Randomized Singular Value Decomposition. Author: N. Benjamin Erichson [aut, cre].
- rtimes** Client for New York Times APIs. Author: Scott Chamberlain [aut, cre].
- rtkore** STK++ Core Library Integration to R using Rcpp. Authors: Serge Iovleff [aut, cre], Parmeet Bhatia [ctb].
- rtson** Typed JSON. Author: Alexandre Maurel.
- runittottestthat** Convert ‘RUnit’ Test Functions into ‘testthat’ Tests. Author: Richard Cotton [aut, cre].
- rusda** Interface to USDA Databases. Author: Franz-Sebastian Krah.
- rwirelesscom** Basic Wireless Communications Simulation. Author: Alberto Gutierrez [aut, cre].
- rwunderground** R Interface to Weather Underground API. Author: Alex Shum.

sSDR Tools Developed for Structured Sufficient Dimension Reduction (sSDR). Authors:

Yang Liu, Francesca Chiaromonte, Bing Li.

sValues Measures of the Sturdiness of Regression Coefficients. Author: Carlos Cinelli.

satellite Various Functions for Handling and Manipulating Remote Sensing Data. Authors:

Thomas Nauss, Hanna Meyer, Florian Detsch, Tim Appelhans.

sbmSDP Semidefinite Programming for Fitting Block Models of Equal Block Sizes. Author: Arash A. Amini.

scatterD3 D3 JavaScript Scatterplot from R. Author: Julien Barnier [aut, cre].

schumaker Schumaker Shape-Preserving Spline. Authors: Stuart Baumann [aut, cre], Margaryta Klymak[ctb].

scmamp Statistical Comparison of Multiple Algorithms in Multiple Problems. Authors: Borja Calvo [aut, cre], Guzman Santafe [aut].

score A Package to Score Behavioral Questionnaires. Author: Jaejoon Song.

scorer Quickly Score Models. Author: Paul Hendricks [aut, cre].

sdPrior Scale-Dependent Hyperpriors in Structured Additive Distributional Regression. Author: Nadja Klein.

searchConsoleR Google Search Console APIv3 R Client. Authors: Mark Edmondson [aut, cre], Jennifer Bryan [ctb].

seeclikfixr Access Data from the SeeClickFix Web API. Authors: Justin de Benedictis-Kessner [aut, cre], Christian Lemp [ctb].

seismic Predict Information Cascade by Self-Exciting Point Process. Authors: Hera He, Murat Erdogdu, Qingyuan Zhao.

sejmRP An Information About Deputies and Votings in Polish Diet. Authors: Piotr Smuda [aut, cre], Przemyslaw Biecek [aut], Tomasz Mikolajczyk [ctb].

selectiveInference Tools for Selective Inference. Authors: Ryan Tibshirani, Rob Tibshirani, Jonathan Taylor, Joshua Loftus, Stephen Reid.

sgRSEA Enrichment Analysis of CRISPR/Cas9 Knockout Screen Data. Authors: Jungsik Noh, Beibei Chen.

shinystan Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models. Authors: Jonah Gabry [aut, cre], Stan Development Team [ctb], Michael Andrae [ctb], Michael Betancourt [ctb], Bob Carpenter [ctb], Yuanjun Gao [ctb], Andrew Gelman [ctb], Ben Goodrich [ctb], Daniel Lee [ctb], Dongying Song [ctb], Rob Trangucci [ctb].

signalHsmm Predict Presence of Signal Peptides. Authors: Michal Burdukiewicz [cre, aut], Piotr Sobczyk [aut].

simest Constrained Single Index Model Estimation. Authors: Arun Kumar Kuchibhotla, Rohit Kumar Patra.

simmer Just Let it Simmer. Authors: Bart Smeets [aut], Iñaki Ucar [aut, cre].

simmr A Stable Isotope Mixing Model. Author: Andrew Parnell.

simplr Basic Symbolic Expression Simplification. Authors: Oliver Flasch [aut, cre], Felix Gorschlüter [aut], Leo Liberti [ctb].

simr Power Analysis for Generalised Linear Mixed Models by Simulation. Authors: Peter Green, Catriona MacLeod.

- sinaplot** An Enhanced Chart for Simple and Truthful Representation of Single Observations over Multiple Classes. Authors: Nikos Sidiropoulos [aut, cre], Sina Hadi Sohi [aut], Nicolas Rapin [aut], Frederik Otzen Bagger [aut].
- sisal** Sequential Input Selection Algorithm. Author: Mikko Korpela [aut, cre].
- smallarea** Fits a Fay Herriot Model. Author: Abhishek Nandy.
- smerc** Statistical Methods for Regional Counts. Author: Joshua French.
- smint** Smooth Multivariate Interpolation for Gridded and Scattered Data. Authors: Yves Deville Yann Richet. Fortran codes by William Thacker, Manjula Iyer Jingwei Zhang, Laynet Watson, Jeffrey Birch, Manjula Iyer, Michael Berry and Robert Renka. Fortran codes by Alain Hebert.
- snn** Stabilized Nearest Neighbor Classifier. Authors: Wei Sun, Xingye Qiao, and Guang Cheng.
- sodavis** SODA: Main and Interaction Effects Selection for Discriminant Analysis and Logistic Regression. Authors: Yang Li, Jun S. Liu.
- sodium** A Modern and Easy-to-Use Crypto Library. Author: Jeroen Ooms.
- solidearthtide** Solid Earth Tide Computation. Authors: Jose Gama [aut, cre], Dennis Milbert [aut, cph].
- spTDyn** Spatially Varying and Spatio-Temporal Dynamic Linear Models. Authors: K. Shuvo Bakar, Philip Kokic, Huidong Jin.
- spanel** Spatial Panel Data Models. Author: Taha Zaghoudi.
- sparsereg** Sparse Bayesian Models for Regression, Subgroup Analysis, and Panel Data. Authors: Marc Ratkovic and Dustin Tingley.
- spatialfil** Application of 2D Convolution Kernel Filters to Matrices or 3D Arrays. Authors: Nicola Dinapoli, Roberto Gatta.
- spduration** Split-Population Duration (Cure) Regression. Authors: Andreas Beger [aut, cre], Daina Chiba [aut], Daniel Hill [aut], Nils Metternich [aut], Shahryar Minhas [aut], Michael Ward [cph].
- speciesgeocodeR** Prepare Species Distributions for the Use in Phylogenetic Analyses. Author: Alexander Zizka [aut, cre].
- specmine** Metabolomics and Spectral Data Analysis and Mining. Authors: Christopher Costa, Marcelo Maraschin, Miguel Rocha.
- spectrino** Spectra Visualization, Organizer and Data Preparation. Author: Teodor Krastev.
- spinyReg** Sparse Generative Model and Its EM Algorithm. Authors: Charles Bouveyron, Julien Chiquet, Pierre Latouche, Pierre-Alexandre Mattei.
- spm12r** Wrapper Functions for SPM (Statistical Parametric Mapping) Version 12 from the Wellcome Trust Centre for Neuroimaging. Author: John Muschelli.
- spocutils** Utilities for Use with ‘spocc’. Author: Scott Chamberlain [aut, cre].
- spsann** Optimization of Sample Configurations using Spatial Simulated Annealing. Authors: Alessandro Samuel-Rosa [aut, cre], Lucia Helena Cunha dos Anjos [ths], Gustavo de Mattos Vasques [ths], Gerard B M Heuvelink [ths], Edzer Pebesma [ctb], Jon Skoien [ctb], Joshua French [ctb], Pierre Roudier [ctb], Dick Brus [ctb], Murray Lark [ctb].

- spsi** Shape-Preserving Uni-Variate and Bi-Variate Spline Interpolation. Authors: Szymon Sacher & Andrew Clausen. Excerpts adapted from Fortran code Copyright (C) Paolo Costantini.
- sscor** Robust Correlation Estimation and Testing Based on Spatial Signs. Authors: Alexander Duerre [aut, cre], Daniel Vogel [aut].
- starma** Modelling Space Time AutoRegressive Moving Average (STARMA) Processes. Author: Felix Cheysson.
- stationaRy** Get Hourly Meteorological Data from Global Stations. Author: Richard Iannone [aut, cre].
- statnetWeb** A Graphical User Interface for Network Modeling with ‘Statnet’. Authors: Emily Beylerian [cre, aut], Kirk Li [ctb], Samuel Jenness [ctb], Martina Morris [ctb].
- steadyICA** ICA and Tests of Independence via Multivariate Distance Covariance. Authors: Benjamin B. Risk and Nicholas A. James and David S. Matteson.
- stmBrowser** Structural Topic Model Brower. Authors: Michael K. Freeman, Jason Chuang, Margaret E. Roberts, Brandon M. Stewart and Dustin Tingley.
- stplanr** Sustainable Transport Planning. Authors: Robin Lovelace [aut, cre], Richard Ellison [aut] (Author of various functions), Barry Rowlingson [aut] (Author of overline), Nick Bearman [aut] (Co-author of gclip).
- stringgaussnet** PPI and Gaussian Network Construction from Transcriptomic Analysis Results Integrating a Multilevel Factor. Authors: Emmanuel Chaplais, Henri-Jean Garchon.
- subscore** SubScore Computing Functions in Classical Test Theory. Authors: Shenghai Dai [aut, cre], Xiaolin Wang [aut], Dubravka Svetina [aut].
- subspace** Interface to OpenSubspace. Authors: Marwan Hassani [aut, cre], Matthias Hansen [aut], Emmanuel Müller [ctb], Ira Assent [ctb], Stephan Günemann [ctb], Timm Jansen [ctb], Thomas Seidl [ctb], University of Waikato [ctb, cph].
- surveyeditor** Generate a Survey that can be Completed by Survey Respondents. Author: Char Leung.
- surveyplanning** Survey Planning Tools. Authors: Juris Breidaks [aut, cre], Martins Liberts [aut], Janis Jukams [aut].
- svgPanZoom** R ‘Htmlwidget’ to Add Pan and Zoom to Almost any R Graphic. Authors: Anders Rium et. al. [aut, cph] (svg-pan-zoom.js BSD-licensed library in htmlwidgets/lib), Jorik Tangelder [aut, cph] (hammer.js MIT-licensed touch library in htmlwidgets/lib), Kent Russell [aut, cre] (R interface to svg-pan-zoom.js).
- svs** Tools for Semantic Vector Spaces. Author: Koen Plevoets [aut, cre].
- swirlify** A Toolbox for Writing ‘swirl’ Courses. Authors: Sean Kross [aut, cre], Nick Carchedi [aut], Chih-Cheng Liang [ctb], Wush Wu [ctb].
- switchr** Installing, Managing, and Switching Between Distinct Sets of Installed Packages. Author: Gabriel Becker[aut, cre].
- switchrGist** Publish Package Manifests to GitHub Gists. Author: Gabriel Becker [aut, cre].
- taber** Split and Recombine Your Data. Author: Seth Wenzel [aut, cre, cph].
- tablaxlsx** Write Formatted Tables in Excel Workbooks. Author: Jesus Maria Rodriguez Rodriguez.

- tailloss** Estimate the Probability in the Upper Tail of the Aggregate Loss Distribution.
Authors: Isabella Gollini [aut, cre], Jonathan Rougier [ctb].
- textruse** Detect Text Reuse and Document Similarity. Author: Lincoln Mullen [aut, cre]. In view: *NaturalLanguageProcessing*.
- tgcd** Thermoluminescence Glow Curve Deconvolution. Authors: Jun Peng [aut, cre], Jorge More [ctb], Burton Garbow [ctb], Kenneth Hillstrom [ctb], John Burkardt [ctb], Linda R. Petzold [ctb], Alan C. Hindmarsh [ctb], R. Woodrow Setzer [ctb].
- tglm** Binary Regressions under Independent Student-t Priors. Author: Yingbo Li.
- threewords** Represent Precise Coordinates in Three Words. Author: Oliver Keyes.
- tigris** Load Census TIGER/Line Shapefiles into R. Authors: Kyle Walker [aut, cre], Bob Rudis [ctb].
- timedelay** Time Delay Estimation for Stochastic Time Series of Gravitationally Lensed Quasars. Authors: Hyungsuk Tak, Kaisey Mandel, David A. van Dyk, Vinay L. Kashyap, Xiao-Li Meng, and Aneta Siemiginowska.
- titanic** Titanic Passenger Survival Data Set. Author: Paul Hendricks [aut, cre].
- tmlenet** Targeted Maximum Likelihood Estimation for Network Data. Authors: Oleg Sofrygin [aut, cre], Mark J. van der Laan [aut].
- tmod** Transcriptional Module Analysis. Author: January Weiner.
- tmvnsim** Truncated Multivariate Normal Simulation. Author: Samsiddhi Bhattacharjee.
- tnam** Temporal Network Autocorrelation Models (TNAM). Authors: Philip Leifeld [aut, cre], Skyler J. Cranmer [ctb].
- tolBasis** Fundamental Definitions and Utilities of the Time Oriented Language (TOL).
Author: Pedro Gea.
- tracheideR** Standardize Tracheidograms. Author: Filipe Campelo.
- traits** Species Trait Data from Around the Web. Authors: Scott Chamberlain [aut, cre], Zachary Foster [aut], Ignasi Bartomeus [aut], David LeBauer [aut], David Harris [aut].
- transcribeR** Automated Transcription of Audio Files Through the HP IDOL API. Authors: Christopher Lucas, Dean Knox, Dustin Tingley, Thomas Scanlan, Shiv Sunil, Michael May, Angela Su.
- translateSPSS2R** Toolset for Translating SPSS-Syntax to R-Code. Authors: Andreas Wygrabek [aut, cre], Bastian Wiessner [aut], eoda GmbH [cph].
- translation.ko** R Manuals Literally Translated in Korean. Authors: Chel Hee Lee [aut, cre], Edward Kang [ctb], Sunyoung Kim [ctb], Heather Kim [ctb].
- treescape** Statistical Exploration of Landscapes of Phylogenetic Trees. Authors: Thibaut Jombart [aut], Michelle Kendall [aut, cre], Jacob Almagro-Garcia [aut], Caroline Colijn [aut].
- trib** Analysing and Visualizing Tribology Measurements. Authors: Gokce Mehmet Ay [aut, cre], Osman Nuri Celik [ths].
- trimr** An Implementation of Common Response Time Trimming Methods. Author: James Grange [aut, cre].
- tsPI** Improved Prediction Intervals for ARIMA Processes and Structural Time Series. Author: Jouni Helske. In view: *TimeSeries*.

- tseriesEntropy** Entropy Based Analysis and Tests for Time Series. Author: Simone Giannerini. In view: [TimeSeries](#).
- tsna** Tools for Temporal Social Network Analysis. Authors: Skye Bender-deMoll [aut, cre], Martina Morris [aut], James Moody [ctb].
- tuber** Client for the YouTube API. Author: Gaurav Sood [aut, cre].
- tumgr** Tumor Growth Rate Analysis. Author: Julia Wilkerson.
- tweet2r** Twitter Collector and Export to ‘postGIS’. Author: Pau Aragó.
- ukgasapi** API for UK Gas Market Information. Author: Timothy Wong [aut, cre].
- umx** Helper Functions for Structural Equation Modelling in OpenMx. Author: Timothy C Bates [aut, cre].
- unfoldr** Stereological Unfolding for Spheroidal Particles. Authors: Markus Baaske [aut, cre], Felix Ballani [ctb].
- uniqueAtomMat** Finding Unique or Duplicated Rows or Columns for Atomic Matrices. Author: Long Qu [aut, cre].
- unitedR** Assessment and Evaluation of Formations in United. Author: David Schindler [aut, cre].
- uptimeRobot** Access the UptimeRobot Ping API. Author: Gabriele Baldassarre [aut, cre].
- urlshorteneR** R Wrapper for the Bit.ly, Goo.gl and Is.gd URL Shortening Services. Author: John Malc.
- validate** Data Validation Infrastructure. Authors: Mark van der Loo [cre, aut], Edwin de Jonge [aut], Paul Hsieh [ctb].
- valottery** Results from the Virginia Lottery Draw Games. Author: Clay Ford [cre, aut].
- varhandle** Functions for Robust Variable Handling. Author: Mehrad Mahmoudian [aut, cre].
- varian** Variability Analysis in R. Authors: Joshua F. Wiley [aut, cre], Elkhart Group Limited [cph].
- versions** Query and Install Specific Versions of Packages on CRAN. Author: Nick Golding.
- vertexenum** Vertex Enumeration of Polytopes. Author: Robert Robere.
- vipor** Plot Categorical Data Using Quasirandom Noise and Density Estimates. Authors: Scott Sherrill-Mix, Erik Clarke.
- viridis** Matplotlib Default Color Map. Authors: Simon Garnier [aut, cre], Noam Ross [ctb, cph] (Continuous scale), Bob Rudis [ctb, cph] (Combined scales).
- viridisLite** Default Color Maps from ‘matplotlib’ (Lite Version). Authors: Simon Garnier [aut, cre], Noam Ross [ctb, cph], Bob Rudis [ctb, cph].
- visNetwork** Network Visualization using ‘vis.js’ Library. Authors: Almende B.V. [aut, cph] (vis.js library in htmlwidgets/lib), Benoit Thieurmel [aut, cre] (R interface).
- vtreat** Simple Variable Treatment. Authors: John Mount, Nina Zumel.
- wBoot** wBootstrap Routines. Author: Neil A. Weiss.
- wPerm** Permutation Tests. Author: Neil A. Weiss.
- walkr** Random Walks in the Intersection of Hyperplanes and the N-Simplex. Authors: Andy Yao, David Kane.

- warbleR** Streamline Bioacoustic Analysis. Authors: Marcelo Araya-Salas, Grace Smith Vidaurre, Hua Zhong.
- water** Actual Evapotranspiration with Energy Balance Models. Authors: Guillermo Federico Olmedo [aut, cre], Samuel Ortega-Farias [aut], David Fonseca-Luengo [aut], Daniel de la Fuente-Saiz [aut], Fernando Fuentes Peñailillo [aut].
- wbsts** Multiple Change-Point Detection for Nonstationary Time Series. Authors: Karolos Korkas and Piotr Fryzlewicz. In view: *TimeSeries*.
- weatherr** Tools for Handling and Scrapping Instant Weather Forecast Feeds. Author: Stan Yip [aut, cre].
- webreadr** Tools for Reading Formatted Access Log Files. Author: Oliver Keyes.
- webuse** Import Stata ‘webuse’ Datasets. Author: Thomas J. Leeper [aut, cre].
- wec** Weighted Effect Coding. Authors: Rense Nieuwenhuis, Manfred te Grotenhuis, Ben Pelzer, Alexander Schmidt, Ruben Konig, Rob Eisinga.
- weightTAPSPACK** Weight TAPS Data. Authors: David G. Carlson, Michelle Torres, and Taeyong Park.
- wellknown** Convert Between ‘WKT’ and ‘GeoJSON’. Author: Scott Chamberlain [aut, cre].
- whoami** Username, Full Name, Email Address, ‘GitHub’ Username of the Current User. Author: Gabor Csardi.
- whoapi** A ‘Whoapi’ API Client. Author: Oliver Keyes.
- wingui** Advanced Windows Functions. Author: Andrew Redd.
- wiod** World Input Output Database 1995–2011. Author: Bastiaan Quast [aut, cre].
- withr** Run Code With Temporarily Modified Global State. Authors: Jim Hester [aut, cre], Kirill Müller [aut], Hadley Wickham [aut], Winston Chang [aut], RStudio [cph].
- woe** Computes Weight of Evidence and Information Values. Author: Sudarson Mothilal Thoppay.
- wordbankr** Accessing the Wordbank Database. Authors: Mika Braginsky [aut, cre], Daniel Yurovsky [ctb], Michael Frank [ctb].
- wpp2015** World Population Prospects 2015. Authors: Population Division, Department of Economic and Social Affairs, United Nations.
- wqs** Weighted Quantile Sum Regression. Authors: Jenna Czarnota, David Wheeler.
- xergm.common** Common Infrastructure for Extensions of Exponential Random Graph Models. Author: Philip Leifeld [aut, cre].
- xmeta** A Toolbox for Multivariate Meta-Analysis. Authors: Yong Chen, Chuan Hong, Haitao Chu. In view: *MetaAnalysis*.
- xseq** Assessing Functional Impact on Gene Expression of Mutations in Cancer. Authors: Jiarui Ding, Sohrab Shah.
- yakmoR** A Simple Wrapper for the k-Means Library Yakmo. Author: Aydin Demircioglu.
- yummlyr** R Bindings for Yummly API. Author: Roman Tsegelskyi.
- zetadiv** Functions to Compute Compositional Turnover Using Zeta Diversity. Authors: Guillaume Latombe, Melodie A. McGeoch, Cang Hui.

Other changes

The following packages were moved to the Archive: **ABCExtremes**, **ABCp2**, **AOfamilies**, **BHMSMAfMRI**, **Biograph**, **Brq**, **CALINE3**, **CARramps**, **CLAG**, **COP**, **CSS**, **Cladis**, **DBFTest**, **DPw**, **EMJumpDiffusion**, **ETAS**, **FluOMatic**, **GPlab**, **HWxtest**, **LGS**, **LVQTools**, **MADAM**, **MFDA**, **MGL**, **MMIX**, **McParre**, **Multiclasstesting**, **OPE**, **PBC**, **PERregress**, **PF**, **PIGShift**, **PKfit**, **RHive**, **RMediation**, **RcmdrPlugin.StatisticalURV**, **Rhh**, **SASxport**, **SE.IGE**, **SINGLE**, **SNPMClust**, **SPMS**, **Watersheds**, **WaveCD**, **WideLM**, **YourCast**, **ZeligGAM**, **ascrda**, **bayesclust**, **bdoc**, **bear**, **bigrfs**, **biopara**, **branchLars**, **clustergas**, **cmprskQR**, **cobs99**, **correlate**, **datalist**, **db.r**, **diskmemoiser**, **emdatr**, **epi2loc**, **esotericR**, **evora**, **fuzzyMM**, **gbs**, **glassomix**, **gooJSON**, **gstudio**, **hlr**, **iFes**, **indicoio**, **ivivc**, **lmmfit**, **loe**, **magma**, **mixlow**, **netweavers**, **ngramr**, **nlADG**, **pcrcoal**, **permGPU**, **permtest**, **phom**, **phyloTop**, **phylosim**, **polytomous**, **predfinitepop**, **pt**, **qlspack**, **qualityTools**, **rJavax**, **radiant**, **rawFasta**, **reccsim**, **repra**, **rgauges**, **ringbuffer**, **ringscale**, **rlme**, **rpud**, **rrules**, **rspear**, **signal.hsmm**, **spca**, **sqlshare**, **sse**, **stab**, **tdm**, **tslars**, **varSelectIP**, **vectoptim**, **waterfall**, **weightedKmeans**, **wombsoft**.

The following packages were resurrected from the Archive: **AdapEnetClass**, **BAS**, **Barnard**, **DESP**, **MPCI**, **MetaPath**, **NHMSAR**, **OneArmPhaseTwoStudy**, **RAHRS**, **RBerkeley**, **REGENT**, **RPPanalyzer**, **RSpincalc**, **Rcell**, **RcmdrPlugin.Export**, **Rgbp**, **Rgnuplot**, **Rlof**, **SigTree**, **ThreeGroups**, **anoint**, **cems**, **cgttools**, **colorscience**, **conicfit**, **cpm**, **cusp**, **delt**, **dprep**, **dynpred**, **edesign**, **edmr**, **flowr**, **geofd**, **googlePublicData**, **highriskzone**, **htmltab**, **imputeYn**, **ipw**, **jackknifeKME**, **locfdr**, **muRL**, **mansellinterpol**, **nordklimdata1**, **ocomposition**, **paperR**, **parfm**, **pgnorm**, **pmc**, **rbiouml**, **rknn**, **samplesize**, **skellam**, **spectral.methods**, **toaster**, **wccsom**, **wikipediatrend**, **xgboost**, **xml2**.

Kurt Hornik

WU Wirtschaftsuniversität Wien, Austria

Kurt.Hornik@R-project.org

Achim Zeileis

Universität Innsbruck, Austria

Achim.Zeileis@R-project.org