Journal of Statistical Software

November 2021, Volume 100, Issue 20. doi: 10.18637/jss.v100.i20

# JointAI: Joint Analysis and Imputation of Incomplete Data in R

**Nicole S. Erler**
Erasmus Medical Center

**Dimitris Rizopoulos**
Erasmus Medical Center

**Emmanuel M. E. H. Lesaffre**
KU Leuven

**Abstract**

Missing data occur in many types of studies and typically complicate the analysis. Multiple imputation, either using joint modeling or the more flexible fully conditional specification approach, are popular and work well in standard settings. In settings involving nonlinear associations or interactions, however, incompatibility of the imputation model with the analysis model is an issue often resulting in bias. Similarly, complex outcomes such as longitudinal or survival outcomes cannot be adequately handled by standard implementations. In this paper, we introduce the R package **JointAI**, which utilizes the Bayesian framework to perform simultaneous analysis and imputation in regression models with incomplete covariates. Using a fully Bayesian joint modeling approach it overcomes the issue of uncongeniality while retaining the attractive flexibility of fully conditional specification multiple imputation by specifying the joint distribution of analysis and imputation models as a sequence of univariate models that can be adapted to the type of variable. **JointAI** provides functions for Bayesian inference with generalized linear and generalized linear mixed models and extensions thereof as well as survival models and joint models for longitudinal and survival data, that take arguments analogous to the corresponding well known functions for the analysis of complete data from base R and other packages. Usage and features of **JointAI** are described and illustrated using various examples and the theoretical background is outlined.

*Keywords*: imputation, Bayesian, missing covariate, nonlinear, interaction, multi-level, survival, joint model, R, **JAGS**.

# 1. Introduction

Missing data are a challenge common to the analysis of data from virtually all kinds of studies. Especially when many variables are measured, as in large cohort studies, or when data are obtained retrospectively, e.g., from registries, large proportions of missing values in some variables are not uncommon.

Multiple imputation, which appears to be the gold standard to handle incomplete data, as indicated by its widespread use, has its origin in the 1970s and was primarily developed for survey data (Deng, Chang, Ido, and Long 2016; Treiman 2009; Rubin 1987, 2004). One of its first implementations in R (R Core Team 2021) is the package **norm** (Novo and Schafer 2013), which performs multiple imputation under the joint modeling framework using a multivariate normal distribution (Schafer 1997). Nowadays multiple imputation using a fully conditional specification (FCS) is more frequently used, also known as multiple imputation using chained equations (MICE) with its seminal implementation in the R package **mice** (Van Buuren and Groothuis-Oudshoorn 2011; Van Buuren 2012).

Since the introduction of multiple imputation, datasets have gotten more complex. Therefore, more sophisticated methods that can adequately handle the features of modern data and do not rely on assumptions that are likely violated by such data are required. Modern studies do not only record univariate outcomes, measured in a cross-sectional setting, but also outcomes that consist of two or more measurements, for instance, repeated measures or survival outcomes. Furthermore, nonlinear effects, introduced by functions of covariates, such as transformations, polynomials or splines, or interactions between variables are considered in the analysis and, hence, need to be taken into account during imputation.

Standard multiple imputation, either using FCS or a joint modeling approach, e.g., under a multivariate normal distribution, assumes linear associations between all variables. It is possible to include nonlinear associations using transformations of variables and passive imputation (Van Buuren 2012); however, this does not generally solve the issue of uncongenial and/or incompatible imputation models. Moreover, FCS requires the outcome to be explicitly specified in each of the linear predictors of the full conditional distributions. In settings where the outcome is more complex than just univariate (for instance, for a survival outcome that typically is represented by the observed event or censoring time and a censoring indicator, or a longitudinal outcome consisting of multiple, correlated measurements) this is not straightforward and not generally possible without information loss, leading to misspecified imputation models and, likely, to bias.

Some extensions of standard multiple imputation have been developed and are implemented in R packages and other software, e.g., Stata (StataCorp 2021), but the greater part of the software for imputation is restricted to standard settings such as cross-sectional survey data. The Comprehensive R Archive Network (CRAN) task view on missing data (Josse, Tierney, and Vialaneix 2021) gives an overview of available R packages that deal with missing data in different contexts, using various approaches.

The R packages reported here below are relevant in our context, i.e., in settings where potentially complex models (such as models with nonlinear associations, survival outcomes or multi-level structure) are estimated on data with missing values in covariates.

The R package **mice** itself provides limited options to perform multi-level imputation, restricted to conditionally normal and binary level-1 covariates (e.g., repeated measurements) and the use of a linear model or predictive mean matching for level-2 covariates (e.g., patient-specific characteristics). The packages **micemd** (Audigier and Resche-Rigon 2021) and **miceadds** (Robitzsch, Grund, and Henke 2021) provide extensions to Poisson models and predictive mean matching for level-1 covariates.

The R package **smcfcs** (Bartlett and Keogh 2021), short for "substantive model compatible fully conditional specification", uses Bayesian methodology to extend standard multiple imputation using FCS to ensure compatibility between analysis model and imputation models. It can handle linear, logistic and Poisson models, as well as parametric (Weibull) and Cox proportional hazards survival models, and competing risk models. Additionally, it provides functionality for case cohort and nested case control studies. The model specification is similar to the **mice** package, however less automated.

The R package **jomo** (Quartagno and Carpenter 2020) performs joint model multiple imputation in the Bayesian framework using a multivariate normal distribution and includes an extension to the standard approach to assure compatibility between analysis model and imputation models. It can handle generalized linear (mixed) models, cumulative link mixed models, proportional odds probit regression and Cox proportional hazards models. Unfortunately, no functions are available to facilitate the evaluation of convergence of the Markov chain Monte Carlo (MCMC) algorithm. The R package **mitml** (Grund, Robitzsch, and Luedtke 2021) provides an interface to the R packages **pan** (imputation of continuous level-1 covariates only) and **jomo** and includes functions that make the analysis and evaluation of the imputed data more convenient.

**hmi** (hierarchical multi-level imputation, Speidel, Drechsler, and Jolani 2020) combines functionality of the packages **mice** and **MCMCglmm** (Hadfield 2010) to perform multiple imputation in single- and multi-level models, but it assumes all incomplete covariates in multi-level models to be level-1 covariates. Similarly, **mlmmm** (Yucel 2010), which uses the EM algorithm to perform multi-level imputation, does not consider incomplete level-2 variables.

**mdmb** (Robitzsch and Luedtke 2021) implements model-based treatment of missing data using likelihood or Bayesian methods in linear and logistic regression and linear and ordinal multilevel models. Under the Bayesian framework, substantive model compatible imputation is available. A drawback is that the specification does not follow the specification of well-known R functions, which complicates usage especially for new users and makes the specification of more complex models more challenging.

Depending on the type of model outcome (survival, multi-level or single-level), whether nonlinear effects are involved (which need substantive model compatible imputation), the measurement level of incomplete covariates and whether missingness occurs in level-1 (e.g., repeated measurements) as well as in level-2 covariates (e.g., baseline covariates), the user has to work with different software and packages. This requires users to be familiar with the usage and underlying statistical methodology of a number of packages and approaches. Since for several packages the documentation is rather inscrutable and vague, it is unclear what precisely these packages can and cannot do and what the underlying assumptions are. Choosing an appropriate software package and applying it correctly may, thus, become quite a daunting challenge.

The R package **JointAI** (Erler 2021), which is presented in this paper, aims to facilitate the correct analysis of incomplete data by providing a unified framework for both simple and more complex models, using a consistent specification that most users will be familiar with from commonly used (base) R functions.

Most of the packages named above perform multiple imputation, i.e., create multiple imputed datasets, which are then analyzed in a second step, followed by pooling of the results. While the separation of imputation and analysis is often considered an advantage, especially when large databases are analyzed by multiple researchers, this separation permits the use of analysis models that are incompatible with the imputation models. **JointAI** follows a different, fully Bayesian approach (used in **mdmb** as well). By modeling the analysis model of interest jointly with the incomplete covariates, analysis and imputation can be performed simultaneously while assuring compatibility between all sub-models (Erler, Rizopoulos, Van Rosmalen, Jaddoe, Franco, and Lesaffre 2016; Erler, Rizopoulos, Jaddoe, Franco, and Lesaffre 2019). In this joint modeling approach, the added uncertainty due to the missing values is automatically taken into account in the posterior distribution of the parameters of interest, and no pooling of results from repeated analyses is necessary. The joint distribution is specified conveniently, using a sequence of conditional distributions that can be specified flexibly according to each type of variable. Since the analysis model of interest defines the first distribution in the sequence, the outcome is included in the joint distribution without the need for it to enter the linear predictor of any of the other models. Moreover, nonlinear associations that are part of the analysis model are automatically taken into account for the imputation of missing values. This directly enables our approach to handle complicated models, with complex outcomes and flexible linear predictors. Another feature that distinguishes **JointAI** from the other packages named above is that it can handle hierarchical settings with more than two levels.

In this paper, we introduce the R package **JointAI**, which performs joint analysis and imputation of regression models with incomplete covariates under the missing at random (MAR) assumption (Rubin 1976), and explain how data with incomplete covariate information can be analyses and imputed with it. The package is available for download from CRAN at `https://CRAN.R-project.org/package=JointAI`. Section 2 briefly describes the theoretical background of the method. An outline of the general structure of **JointAI** is given in Section 3, followed by an introduction of the example datasets that are used throughout the paper in Section 4. Details about model specification, settings controlling the MCMC sampling, and summary, plotting and other functions that can be applied after fitting the model are given in Sections 5 through 7. We conclude the paper with an outlook of planned extensions and discuss the limitations that are introduced by the assumptions made in the fully Bayesian approach.

## 2. Theoretical background

Consider the general setting of a regression model where interest lies in a set of parameters $\boldsymbol{\theta}$ that describe the association between a univariate outcome $\mathbf{y}$ and a set of covariates $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_p)$. In the Bayesian framework, inference over $\boldsymbol{\theta}$ is obtained by estimation of the posterior distribution of $\boldsymbol{\theta}$, which is proportional to the product of the likelihood of the data $(\mathbf{y}, \mathbf{X})$ and the prior distribution of $\boldsymbol{\theta}$,

$$p(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{X}) \propto p(\mathbf{y}, \mathbf{X} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}).$$

When some of the covariates are incomplete, $\mathbf{X}$ consists of two parts, the completely observed variables $\mathbf{X}_{obs}$ and those variables that are incomplete, $\mathbf{X}_{mis}$. If $\mathbf{y}$ had missing values (and this missingness was ignorable), the only necessary change in the formulas below would be to write $\mathbf{y}_{mis}$ instead of $\mathbf{y}$, however the model itself would not change, since the conditional distribution for $\mathbf{y}$ is already part of the model specification. Here, we will, therefore, consider $\mathbf{y}$ to be completely observed. In the implementation in the R package **JointAI**, however, missing values in the outcome are allowed and are imputed automatically.

The likelihood of the complete data, i.e., observed and unobserved data, can be factorized in the following convenient way:

$$p(\mathbf{y}, \mathbf{X}_{obs}, \mathbf{X}_{mis} \mid \boldsymbol{\theta}) = p(\mathbf{y} \mid \mathbf{X}_{obs}, \mathbf{X}_{mis}, \boldsymbol{\theta}_{y|x})\, p(\mathbf{X}_{mis} \mid \mathbf{X}_{obs}, \boldsymbol{\theta}_x),$$

where the first factor constitutes the analysis model of interest, described by a vector of parameters $\boldsymbol{\theta}_{y|x}$, and the second factor is the joint distribution of the incomplete variables, i.e., the imputation part of the model, described by parameters $\boldsymbol{\theta}_x$, and $\boldsymbol{\theta} = (\boldsymbol{\theta}_{y|x}^\top, \boldsymbol{\theta}_x^\top)^\top$.

Explicitly specifying the joint distribution of all data is one of the major advantages of the Bayesian approach, since this facilitates the use of all available information of the outcome in the imputation of the incomplete covariates (Erler *et al.* 2016), which becomes especially relevant for more complex outcomes like repeatedly measured variables (see Section 2.2.1).

In complex models the posterior distribution cannot usually be derived analytically but MCMC methods are used to obtain samples from the posterior distribution. The MCMC sampling in **JointAI** is done using the Gibbs method, which iteratively samples from the full conditional distributions of the unknown parameters and missing values.

In the following sections we describe in detail each of the three parts of the model: the analysis model, the imputation part and the prior distributions.

## 2.1. Analysis model

The analysis model of interest is described by the probability density function $p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}_{y|x})$. The R package **JointAI** can currently handle analysis models that are generalized linear regression models (GLMs) or generalized linear mixed models (GLMMs) or extensions thereof (using either a log-normal or a beta distribution), cumulative and multinomial logit (mixed) models, parametric (Weibull) or proportional hazards survival models. Moreover, it is possible to fit joint models for longitudinal and survival data.

In a multi-level setting, we use level-1 to refer to the lowest level of the hierarchy (for instance, repeated measurements of a biomarker), level-2 to the next higher level (e.g., patient-specific information), and so on. **JointAI** allows for models with more than two levels, but, to facilitate notation, we focus here on settings with two levels.

### *Generalized linear (mixed) models*

For a GLM the probability density function is chosen from the exponential family and has the linear predictor

$$g\{E(y_i \mid \mathbf{X}, \boldsymbol{\theta}_{y|x})\} = \mathbf{x}_i^\top \boldsymbol{\beta},$$

where $g(\cdot)$ is a link function, $y_i$ the value of the outcome variable for subject $i$, and $\mathbf{x}_i$ is a column vector containing the row of $\mathbf{X}$ that contains the covariate information for $i$.

For a GLMM the linear predictor is of the form

$$g\{E(y_{ij} \mid \mathbf{X}, \mathbf{b}_i, \boldsymbol{\theta}_{y|x})\} = \mathbf{x}_{ij}^\top \boldsymbol{\beta} + \mathbf{z}_{ij}^\top \mathbf{b}_i,$$

where $y_{ij}$ is the $j$-th outcome of subject $i$, $\mathbf{x}_{ij}$ is the corresponding vector of covariate values, $\mathbf{b}_i$ a vector of random effects pertaining to subject $i$, and $\mathbf{z}_{ij}$ a column vector containing the row of the design matrix of the random effects, $\mathbf{Z}$, that corresponds to the $j$-th measurement of subject $i$. $\mathbf{Z}$ typically contains a subset of the variables in $\mathbf{X}$, and $\mathbf{b}_i$ follows a normal distribution with mean zero and covariance matrix $\mathbf{D}$.

In both cases the parameter vector $\boldsymbol{\theta}_{y|x}$ contains the regression coefficients $\boldsymbol{\beta}$, and potentially additional variance parameters (e.g., for linear (mixed) models), for which prior distributions will be specified in Section 2.3.

As mentioned, the package allows for extensions of the GLMM using a log-normal and beta distribution, according to the data at hand. In the log-normal model, a log-normal distribution is assumed for the outcome $\mathbf{y}$. This distribution is parametrized in terms of the log scale, i.e., $E(\log(y_i)) = \mathbf{x}_{ij}^\top\boldsymbol{\beta}$ or, in case of a log-normal mixed model $E(\log(y_{ij})) = \mathbf{x}_{ij}^\top\boldsymbol{\beta} + \mathbf{z}_{ij}^\top\mathbf{b}_i$.

The beta distribution is parametrized as follows:

$$\begin{aligned}
y_{ij} &\sim Beta(a_{ij}, b_{ij}),\\
a_{ij} &= \mu_{ij}\tau,\\
b_{ij} &= (1-\mu_{ij})\tau,\\
\text{logit}(\mu_{ij}) &= \mathbf{x}_{ij}^\top\boldsymbol{\beta} + \mathbf{z}_{ij}^\top\mathbf{b}_i,
\end{aligned}$$

where $\mu_{ij}$ is the expected value of subject $i$ at measurement occasion $j$, $\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$, and $\tau$ follows a Gamma distribution.

### *Cumulative logit (mixed) models*

Cumulative logit mixed models are of the form

$$\begin{aligned}
y_{ij} &\sim \text{Mult}(\pi_{ij,1}, \ldots, \pi_{ij,K}),\\
\pi_{ij,1} &= 1 - \sum_{k=2}^{K}\pi_{ij,k},\\
\pi_{ij,k} &= P(y_{ij} > k-1) - P(y_{ij} > k), \quad k \in \{2, \ldots, K-1\},\\
\pi_{ij,K} &= P(y_{ij} \geq k-1),\\
\text{logit}(P(y_{ij} > k)) &= \gamma_k + \eta_{ij}, \quad k \in \{1, \ldots, K\},\\
\eta_{ij} &= \mathbf{x}_{ij}^\top\boldsymbol{\beta} + \mathbf{z}_{ij}^\top\mathbf{b}_i,\\
\gamma_1, \delta_1, \ldots, \delta_{K-1} &\overset{iid}{\sim} N(\mu_\gamma, \sigma_\gamma^2),\\
\gamma_k &\sim \gamma_{k-1} + \exp(\delta_{k-1}), \quad k = 2, \ldots, K,
\end{aligned}$$

where $\pi_{ij,k} = P(y_{ij} = k)$. A cumulative logit regression model for a univariate outcome $y_i$ can be obtained by dropping the index $j$ and omitting $\mathbf{z}_{ij}^\top\mathbf{b}_i$. In cumulative logit (mixed) models, the design matrix $\mathbf{X}$ does not contain an intercept, since outcome category specific intercepts $\gamma_1, \ldots, \gamma_K$ are specified. Here, the parameter vector $\boldsymbol{\theta}_{y|x}$ includes the regression coefficients $\boldsymbol{\beta}$, the first intercept $\gamma_1$ and increments $\delta_1, \ldots, \delta_{K-1}$.

Note that this implementation assumes proportional odds, i.e., that the linear predictors for the different categories of the outcome only differ in the intercepts, but that covariates have the same effect on the probability to be in the respective next category. This assumption can be relaxed for some or all of the regression coefficients by extending the linear predictor to $\gamma_k + \eta_{ij,k}$ with $\eta_{ij,k} = \mathbf{x}_{ij}^\top \boldsymbol{\beta}_k + \mathbf{z}_{ij}^\top \mathbf{b}_i$.

### *Multinomial logit (mixed) models*

Multinomial logit mixed models are implemented as

$$\begin{aligned}
y_{ij} &\sim \text{Mult}(\pi_{ij,1}, \ldots, \pi_{ij,K}),\\
\pi_{ij,k} &= \phi_{i,k} / \sum_{q=1}^{K} \phi_{i,q}, \quad k\{\in 1, \ldots, K\},\\
\log(\phi_{ij,1}) &= 0,\\
\log(\phi_{ij,k}) &= \mathbf{x}_{ij}^\top \boldsymbol{\beta}_k + \mathbf{z}_{ij}^\top \mathbf{b}_i, \quad k \in \{2, \ldots, K\},
\end{aligned}$$

where $\pi_{ij,k} = P(y_{ij} = k)$ is the probability to observe category $k$ for subject $i$ at measurement occasion $j$.

### *Survival models*

Survival data are typically characterized by the observed event or censoring times, $T_i$, and the event indicator, $D_i$, which is one if the event was observed and zero otherwise. **JointAI** provides two types of models to analyze right censored survival data: a parametric model which assumes a Weibull distribution for the true (but partially unobserved) survival times $T^*$, and a semi-parametric proportional hazards model.

The parametric survival model is implemented as,

$$\begin{aligned}
T_i^* &\sim \text{Weibull}(1, r_i, s),\\
D_i &\sim \mathbb{1}\left(T_i^* \geq C_i\right),\\
\log(r_i) &= -\mathbf{x}_i^\top \boldsymbol{\beta}\\
s &\sim \text{Exp}(0.01),
\end{aligned}$$

where $\mathbb{1}(T_i^* \geq C_i)$ is the indicator function which is one if $T_i^* \geq C_i$, and zero otherwise.

The proportional hazards model can be written as

$$h_i(t) = h_0(t) \exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right),$$

where $h_0(t)$ is the baseline hazard function, which, in **JointAI**, is modeled using a B-spline approach with $Q$ degrees of freedom, i.e., $\log h_0(t) = \sum_{q=1}^{\mathrm{Q}} \gamma_{Bq} B_q(t)$, where $B_q$ denotes the $q$-th basis function and $\gamma_{Bq}$ the corresponding regression coefficient.

The survival function of the proportional hazards model with time-constant covariates is

$$S(t \mid \boldsymbol{\theta}) = \exp\left\{-\int_0^t h_0(s) \exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right) ds\right\} = \exp\left\{-\exp\left(\mathbf{x}_i^\top \boldsymbol{\beta}\right) \int_0^t h_0(s) ds\right\},$$

where $\boldsymbol{\theta}$ includes the regression coefficients $\boldsymbol{\beta}$ (which do not include an intercept) and the coefficients $\boldsymbol{\gamma}_B$ used in the specification of the baseline hazard. Since the integral over the baseline hazard does not have a closed-form solution, in **JointAI** it is approximated using Gauss-Kronrod quadrature with 15 evaluation points.

*Joint models*

Joint models for longitudinal and survival data are implemented using a semi-parametric proportional hazards model for the time-to-event outcome and mixed models for the longitudinal outcomes. The linear predictor of the proportional hazards model is then

$$\exp\left(\mathbf{x}_i^\top \boldsymbol{\beta} + f(s_i(t))\beta_s\right),$$

where $f(s_i(t))$ denotes a function that describes the association the hazard has with the longitudinal variable and $\beta_s$ is the regression coefficient associated with it. In the simplest case, this could be the observed or imputed value, i.e., $f(s_i(t)) = \widehat{s}_i$, or the expected value (i.e., the value of the linear predictor), $f(s_i(t)) = E(s_i \mid t, \mathbf{X}, \mathbf{b}_i, \boldsymbol{\theta})$.

To take into account potential correlation between multiple time-varying covariates, an association structure between them can be specified explicitly by including the time-varying covariates in each other's linear predictors in a sequential manner, or their random effects can be modeled jointly.

## 2.2. Imputation part

A convenient way to specify the joint distribution of the incomplete covariates $\mathbf{X}_{mis} = (\mathbf{x}_{mis_1}, \ldots, \mathbf{x}_{mis_q})$ is to use a sequence of conditional univariate distributions (Ibrahim, Chen, and Lipsitz 2002; Erler *et al.* 2016):

$$\begin{aligned} p(\mathbf{x}_{mis_1}, \ldots, \mathbf{x}_{mis_q} \mid \mathbf{X}_{obs}, \boldsymbol{\theta}_x) \;&=\; p(\mathbf{x}_{mis_1} \mid \mathbf{X}_{obs}, \boldsymbol{\theta}_{x_1}) \\ &\quad \prod_{\ell=2}^{q} p(\mathbf{x}_{mis_\ell} \mid \mathbf{X}_{obs}, \mathbf{x}_{mis_1}, \ldots, \mathbf{x}_{mis_{\ell-1}}, \boldsymbol{\theta}_{x_\ell}), \end{aligned} \tag{1}$$

with $\boldsymbol{\theta}_x = (\boldsymbol{\theta}_{x_1}^\top, \ldots, \boldsymbol{\theta}_{x_q}^\top)^\top$. Each of the conditional distributions is a member of the exponential family, extended with distributions for categorical variables, beta and log-normal models, and chosen according to the type of the respective variable. Its linear predictor is

$$g_\ell\left\{E\left(x_{i,mis_\ell} \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis_{<\ell}}, \boldsymbol{\theta}_{x_\ell}\right)\right\} = (\mathbf{x}_{i,obs}^\top, x_{i,mis_1}, \ldots, x_{i,mis_{\ell-1}})\boldsymbol{\alpha}_\ell, \quad \ell = 1, \ldots, q,$$

where $\mathbf{x}_{i,mis_{<\ell}} = (x_{i,mis_1}, \ldots, x_{i,mis_{\ell-1}})^\top$ and $\mathbf{x}_{i,obs}$ is the vector of values for subject $i$ of those covariates that are observed for all subjects.

Factorization of the joint distribution of the covariates in such a sequence yields a straightforward specification of the joint distribution, even when the covariates are of mixed type. Missing values in the covariates are sampled from their full conditional distribution that can be derived from the full joint distribution of outcome and covariates. When, for instance, the analysis model is a GLM, the full conditional distribution of an incomplete covariate $x_{i,mis_\ell}$ can be written as

$$\begin{aligned} p(x_{i,mis_\ell} \mid \mathbf{y}_i, \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis_{-\ell}}, \boldsymbol{\theta}) \;&\propto\; p(y_i \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{y|x}) p(\mathbf{x}_{i,mis} \mid \mathbf{x}_{i,obs}, \boldsymbol{\theta}_x)\, p(\boldsymbol{\theta}_{y|x})\, p(\boldsymbol{\theta}_x) \\ &\propto\; p(y_i \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{y|x}) \\ &\quad \times p(x_{i,mis_\ell} \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis_{<\ell}}, \boldsymbol{\theta}_{x_\ell}) \\ &\quad \times \left\{\prod_{k=\ell+1}^{q} p(x_{i,mis_k} \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis_{<k}}, \boldsymbol{\theta}_{x_k})\right\} \\ &\quad \times p(\boldsymbol{\theta}_{y|x}) p(\boldsymbol{\theta}_{x_\ell}) \prod_{k=\ell+1}^{p} p(\boldsymbol{\theta}_{x_k}), \end{aligned} \tag{2}$$

where $\boldsymbol{\theta}_{x_\ell}$ is the vector of parameters describing the model for the $\ell$-th covariate, and contains the vector of regression coefficients $\boldsymbol{\alpha}_\ell$ and potentially additional (e.g., variance) parameters. The product of distributions enclosed by curly brackets represents the distributions of those covariates that have $x_{mis_\ell}$ as a predictive variable in the specification of the sequence in (1).

Note that the imputed values for $x_{i,mis_\ell}$ are sampled from (2), which is the actual imputation model, and that the conditional distributions of $x_{i,mis_\ell}$ from (1) are the models that are explicitly specified in the product that forms the joint distribution.

### *Imputation in multi-level settings*

Factorizing the joint distribution into analysis model and imputation part also facilitates extensions to settings with more complex outcomes, such as repeatedly measured outcomes. In the case where the analysis model is a mixed model with two levels, the conditional distribution of the outcome in (2), $p\left(y_i \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{y|x}\right)$, has to be replaced by

$$\left\{\prod_{j=1}^{n_i} p\left(y_{ij} \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \mathbf{b}_i, \boldsymbol{\theta}_{y|x}\right)\right\}. \tag{3}$$

Since $\mathbf{y}$ does not appear in any of the other terms in (2) and (3), it can be chosen to be a model that is appropriate for the outcome at hand. The thereby specified full conditional distribution of $x_{i,mis_\ell}$ allows us to draw valid imputations that use all available information on the outcome.

This is an important difference to standard FCS, where the full conditional distributions used to impute missing values are specified directly, usually as regression models, and require the outcome to be explicitly included into the linear predictor of the imputation model. In settings with complex outcomes it is not clear how this should be done and simplifications may lead to biased results (Erler *et al.* 2016). The joint model specification utilized in **JointAI** overcomes this difficulty.

When some covariates are repeatedly measured, it is convenient to specify models for these variables at the beginning of the sequence of covariate models, so that models for lower level variables (e.g., level-1) have variables of the same or higher levels (e.g., level-1, level-2, level-3, ...) in their linear predictor, but lower level covariates do not enter the predictors of higher level covariates. Note that, whenever there are incomplete higher level covariates it is necessary to specify models for all lower level variables, even completely observed ones, while models for completely observed covariates on the highest level of the hierarchy can be omitted. This becomes clear when we explicitly extend the factorized joint distribution from above with completely and incompletely observed level-1 covariates $\mathbf{s}_{obs}$ and $\mathbf{s}_{mis}$:

$$\begin{aligned}
&p\left(y_{ij}, \mathbf{s}_{ij,obs}, \mathbf{s}_{ij,mis}, \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{y|x}, \boldsymbol{\theta}_{s_{mis}}, \boldsymbol{\theta}_{s_{obs}}, \boldsymbol{\theta}_{x_{mis}}, \boldsymbol{\theta}_{x_{obs}}\right) = \\
&\qquad p\left(y_{ij} \mid \mathbf{s}_{ij,obs}, \mathbf{s}_{ij,mis}, \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{y|x}\right) \\
&\qquad \times p(\mathbf{s}_{ij,mis} \mid \mathbf{s}_{ij,obs}, \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{s_{mis}})\, p(\mathbf{s}_{ij,obs} \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{s_{obs}}) \\
&\qquad \times p(\mathbf{x}_{i,mis} \mid \mathbf{x}_{i,obs}, \boldsymbol{\theta}_{x_{mis}})\, p(\mathbf{x}_{i,obs} \mid \boldsymbol{\theta}_{x_{obs}})\, p(\boldsymbol{\theta}_{y|x})\, p(\boldsymbol{\theta}_{s_{mis}})\, p(\boldsymbol{\theta}_{s_{obs}})\, p(\boldsymbol{\theta}_{x_{mis}})\, p(\boldsymbol{\theta}_{x_{obs}}).
\end{aligned}$$

Given that the parameter vectors $\boldsymbol{\theta}_{x_{obs}}$, $\boldsymbol{\theta}_{x_{mis}}$, $\boldsymbol{\theta}_{s_{obs}}$ and $\boldsymbol{\theta}_{s_{mis}}$ are a priori independent, and $p(\mathbf{x}_{i,obs} \mid \boldsymbol{\theta}_{x_{obs}})$ is independent of both $\mathbf{x}_{mis}$ and $\mathbf{s}_{mis}$, it can be omitted.

Since $p(\mathbf{s}_{ij,obs} \mid \mathbf{x}_{i,obs}, \mathbf{x}_{i,mis}, \boldsymbol{\theta}_{s_{obs}})$, however, has $\mathbf{x}_{i,mis}$ in its linear predictor and will, hence, be part of the full conditional distribution of $\mathbf{x}_{i,mis}$, it cannot be omitted from the model, unless it is reasonable to assume that $\mathbf{x}_{i,mis}$ and $\mathbf{s}_{ij,obs}$ are independent.

*Nonlinear associations and interactions*

Other settings in which the fully Bayesian approach employed in **JointAI** has an advantage over standard FCS is when we have interaction terms that involve incomplete covariates or when the association of the outcome with an incomplete covariate is nonlinear. In standard FCS such settings lead to incompatible imputation models (White, Royston, and Wood 2011; Bartlett, Seaman, White, and Carpenter 2015). This becomes clear when considering the following simple example where the analysis model of interest is the linear regression $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$ and $x_i$ is imputed using $x_i = \alpha_0 + \alpha_1 y_i + \tilde{\varepsilon}_i$. While the analysis model assumes a quadratic relationship, the imputation model assumes a linear association between $\mathbf{x}$ and $\mathbf{y}$ and there cannot be a joint distribution that has the imputation and analysis model as its full conditional distributions. Because, in **JointAI**, the conditional distribution of the response is a factor in the specification of the full conditional distribution that is used to impute $x_i$, the nonlinear association is taken into account. Furthermore, since it is the joint distribution that is specified, and the full conditional then derived from it, the joint distribution is ensured to exist.

### 2.3. Prior distributions

Prior distributions have to be specified for all (hyper)parameters. A common prior choice for the regression coefficients is the normal distribution with mean zero and large variance. In **JointAI**, variance parameters are specified as, by default vague, inverse-gamma distributions. The covariance matrix of the random effects in a mixed model, $\mathbf{D}$, is assumed to follow an inverse Wishart distribution where the degrees of freedom are, by default, chosen to be the dimension of the random effects plus one, and the scale matrix is diagonal. Since the magnitude of the diagonal elements relates to the variance of the random effects, the choice of suitable values depends on the scale of the variable the random effect is associated with. Therefore, **JointAI** uses independent gamma hyper-priors for each of the diagonal elements. More details about the default hyper-parameters and how to change them are given in Section 5.8 and Appendix A.

## 3. Package structure

The package **JointAI** has several main functions, `lm_imp()`, `glm_imp()`, `clm_imp()`, ..., generally abbreviated as `*_imp()`, that perform regression of continuous and categorical, univariate or multi-level data as well as right-censored survival data. The model specification is similar to the specification of standard regression models in R and described in detail in Section 5.

Based on the specified model formula and other function arguments, **JointAI** does some pre-processing of the data. It checks which variables are incomplete and identifies their measurement level and level in the hierarchical structure in order to specify appropriate (imputation) models. Interactions and functional forms of variables are detected in the model formula, and the design matrices for the various parts of the model are created.

MCMC sampling is performed by the program **JAGS** (Plummer 2003). The **JAGS** model, data list (containing all necessary parts of the data) and user-specified settings for the MCMC sampling (see Section 6) are passed to **JAGS** via the R package **rjags** (Plummer 2021).

All the main functions `*_imp()` return an object of class '`JointAI`'. Summary and plotting methods for `JointAI` objects, as well as functions to evaluate convergence and precision of the MCMC samples, to predict from `JointAI` objects and to export imputed values are discussed in Section 7.

Currently, the package works under the assumption of a missing at random (MAR) missingness process (Rubin 1976, 1987). When this assumption holds, observations with missing outcome may be excluded from the analysis in the Bayesian framework. Hence, missing values in the outcome do not require special treatment in this setting, and, therefore, our focus here is on missing values in covariates. Nevertheless, **JointAI** can handle missing values in the outcome; they are automatically imputed using the specified analysis model.

# 4. Example data

To illustrate the functionality of **JointAI**, we use three datasets that are part of this package. The `NHANES` data contain measurements from a cross-sectional cohort study, whereas the `simLong` data is a simulated dataset based on a longitudinal cohort study in toddlers. The third dataset (`PBC`) is the well known data on primary biliary cirrhosis from the Mayo clinic.

## 4.1. The NHANES data

The `NHANES` dataset is a subset of observations from the 2011 – 2012 wave of the National Health and Nutrition Examination Survey (National Center for Health Statistics (NCHS) 2011–2012) and contains information on 186 men and women between 20 and 80 years of age. The variables contained in this dataset are:

- `SBP`: Systolic blood pressure in mmHg; complete.
- `gender`: `male` vs `female`; complete.
- `age`: In years; complete.
- `race`: 5 unordered categories; complete.
- `WC`: Waist circumference in cm; 1.1% missing.
- `alc`: Weekly alcohol consumption; binary; 18.3% missing.
- `educ`: Educational level; binary; complete.
- `creat`: Creatinine concentration in mg/dL; 4.5% missing.
- `albu`: Albumin concentration in g/dL; 4.3% missing.
- `uricacid`: Uric acid concentration in mg/dL; 4.3% missing.
- `bili`: Bilirubin concentration in mg/dL; 4.3% missing.

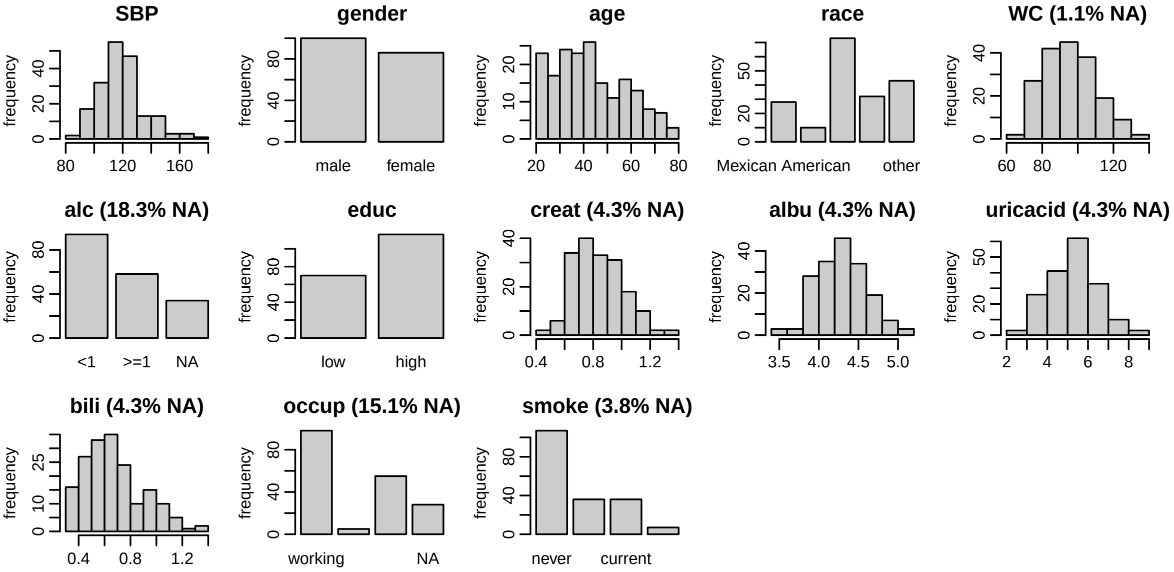

Figure 1: Distribution of the variables in the `NHANES` data (with percentage of missing values given for incomplete variables).

- `occup`: Occupational status; 3 unordered categories; 15.1% missing.
- `smoke`: Smoking status; 3 ordered categories; 3.8% missing.

Figure 1 shows histograms and bar plots of all continuous and categorical variables, respectively, together with the proportion of missing values for incomplete variables. Such a plot can be obtained with the function `plot_all()`. Arguments `fill` and `border` allow the user to change colors, while the number of rows and columns can be adapted using `nrow` and/or `ncol`, and additional arguments can be passed to `hist()` and `barplot()` via `"..."`. The pattern of missing values in the `NHANES` data is shown in Figure 2. This plot can be obtained using the function `md_pattern()`. Again, arguments `color` and `border` allow the user to change colors, while arguments such as `legend.position`, `print_xaxis` and `print_yaxis` permit further customization. Each row represents a pattern of observed/missing values, where observed (missing) values are depicted with dark (light) color. The frequency with which each of the patterns is observed is given on the right margin, the number of missing values in each variable is given underneath the plot. Rows and columns are ordered by number of cases per pattern (decreasing) and number of missing values (increasing). The first row, for instance, shows that there are 116 complete cases, the second row that there are 29 cases for which only `alc` is missing. Furthermore, it is apparent that `creat`, `albu`, `uricacid` and `bili` are always missing together. Since these variables are all measured in serum, this is not surprising.

The function `md_pattern()` returns also the missing data pattern in matrix representation (`pattern = TRUE`), where missing and observed values are represented with a `0` and `1`, respectively.

### *Missing data visualization and exploration*

There are several R packages that provide functionality for a more in-depth exploration of

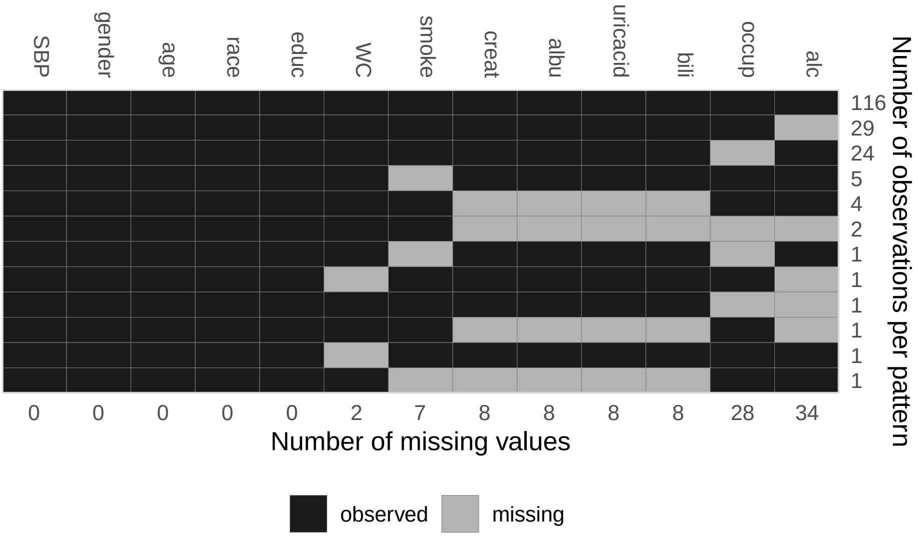

Figure 2: Missing data pattern of the NHANES data.

incomplete data, see for example the ones listed in the CRAN task view on missing data (Josse *et al.* 2021). Particularly useful may be the packages **naniar** (Tierney and Cook 2020) and **VIM** (Kowarik and Templ 2016).

## 4.2. The simLong data

The simLong dataset is a simulated dataset mimicking a longitudinal cohort study of 200 mother-child pairs. It contains the following baseline (i.e., not time-varying) covariates

- GESTBIR: Gestational age at birth in weeks; complete.
- ETHN: Ethnicity; binary; 2.8% missing.
- AGE_M: Age of the mother at intake; complete.
- HEIGHT_M: Height of the mother in cm; 2.0% missing.
- PARITY: Number of times the mother has given birth; binary; 2.4% missing.
- SMOKE: Smoking status of the mother during pregnancy; 3 ordered categories; 12.2% missing.
- EDUC: Educational level of the mother; 3 ordered categories; 7.8% missing.
- MARITAL: Marital status; 3 unordered categories; 7.0% missing.
- ID: Subject identifier.

and seven longitudinal variables:

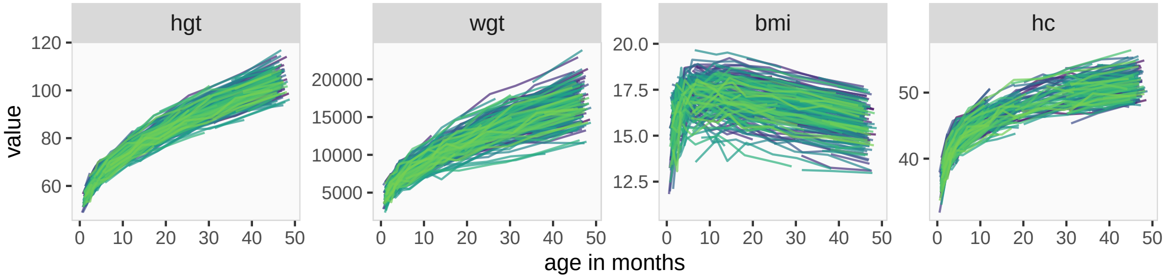

Figure 3: Trajectories of the continuous time-varying variables in the `simLong` data.

- `time`: Measurement occasion/visit (by design, children should have been measured at/around 1, 2, 3, 4, 7, 11, 15, 20, 26, 32, 40 and 50 months of age).
- `age`: Child's age at measurement time in months.
- `hgt`: Child's height in cm; 20% missing.
- `wgt`: Child's weight in gram; 8.8% missing.
- `bmi`: Child's BMI (body mass index) in kg/m$^2$; 21.6% missing.
- `hc`: Child's head circumference in cm; 23.6% missing.
- `sleep`: Child's sleeping behavior; 3 ordered categories; 24.7% missing.

Figure 3 shows the longitudinal profiles of `hgt`, `wgt`, `bmi` and `hc` over age. All four variables clearly have a nonlinear pattern over time. Histograms and bar plots of all the variables in the `simLong` data are displayed in Figure 4. Here, the argument `idvar` of the function `plot_all()` is used to display baseline (level-2) covariates on the subject level instead of the observation level:

```
R> plot_all(simLong, use_level = TRUE, idvar = "ID", ncol = 5)
```

The missing data pattern of the `simLong` data is shown in Figure 5. For readability, the pattern is given separately for the level-1 (left) and level-2 (right) variables. It is non-monotone and does not have any distinctive features.

## 4.3. The PBC data

For demonstration of the use of **JointAI** for the analysis of survival data we use the dataset `PBC` which is a re-coded version of the PBC data in the **survival** package. It contains baseline and follow-up data of 312 patients with primary biliary cirrhosis and includes the following variables:

Baseline covariates:

- `id`: Patient identifier; complete.
- `futime`: Time until death, transplantation or censoring in days; complete.

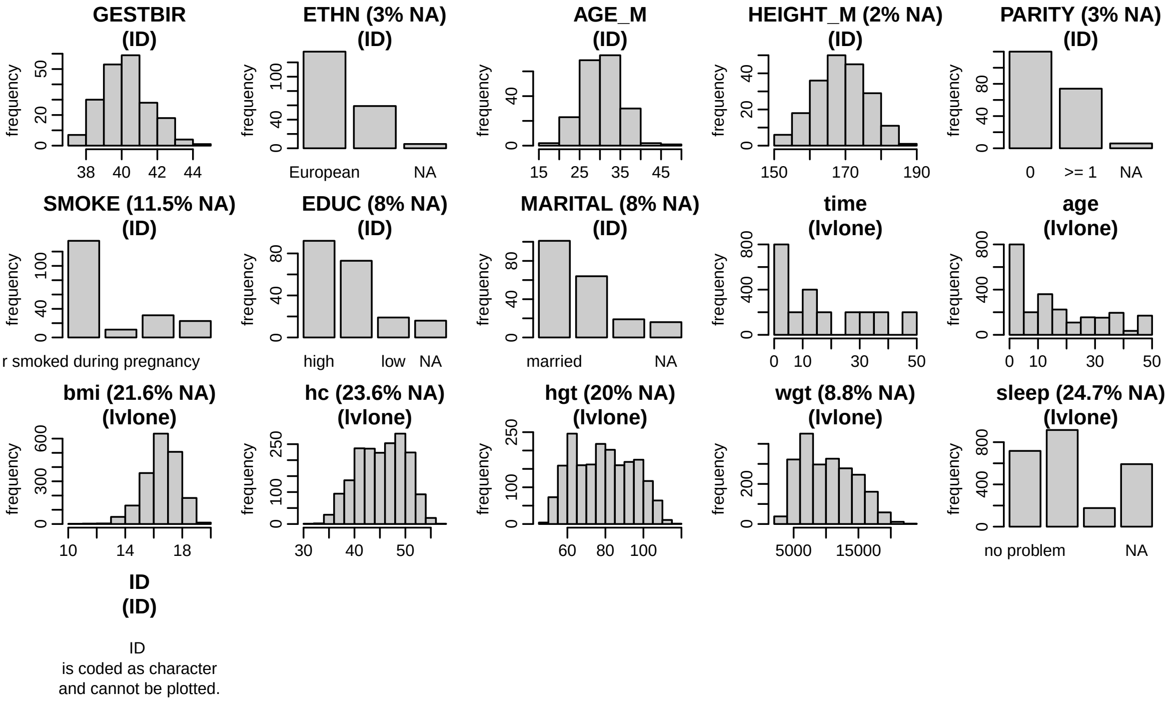

Figure 4: Distribution of the variables in the `simLong` data.

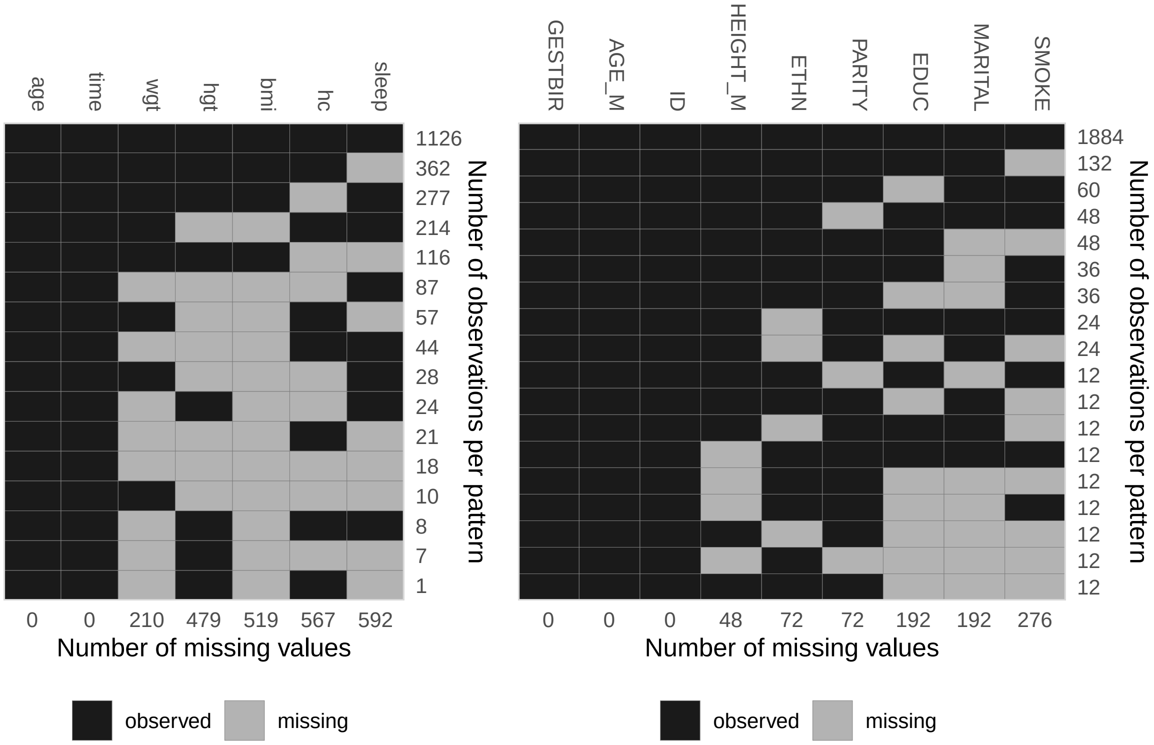

Figure 5: Missing data pattern of the `simLong` data (left: level-1 variables, right: level-2 variables).

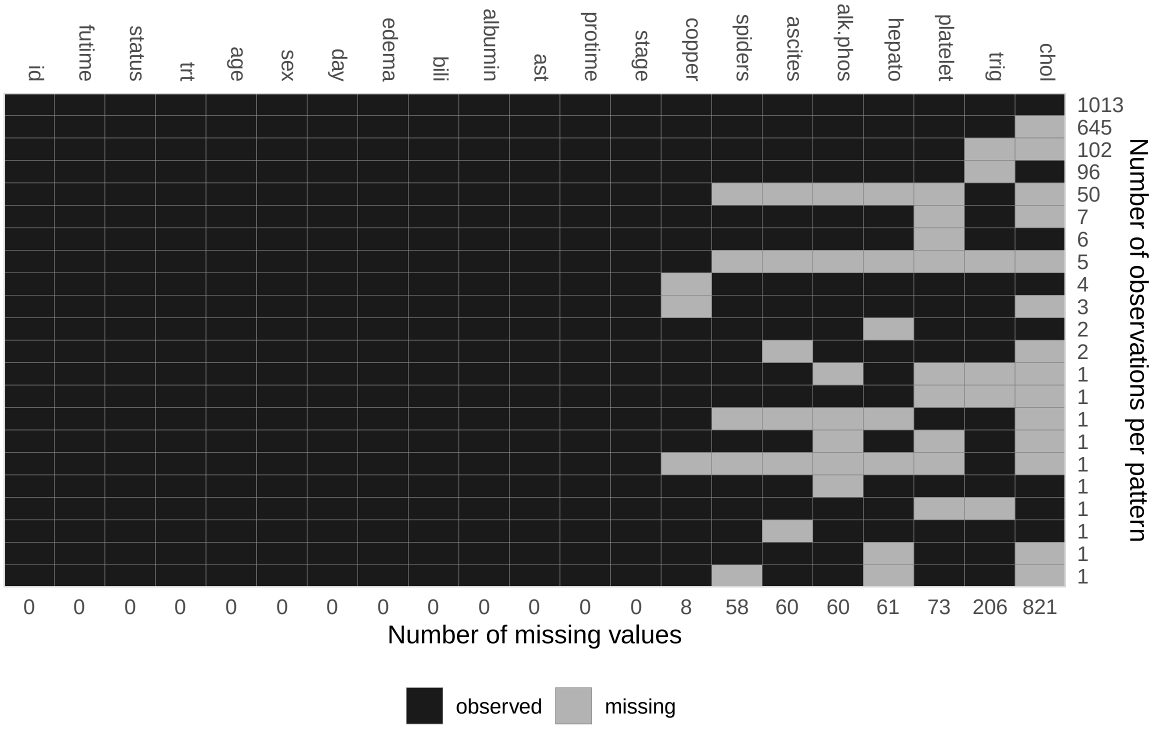

Figure 6: Missing data pattern of the PBC data.

- `status`: Event indicator (`censored`, `transplant` or `dead`); complete.
- `trt`: Treatment (D-penicillamine or placebo); complete.
- `age`: Patient's age in years; complete.
- `sex`: Patient's sex; complete.
- `copper`: Urine copper ($\mu$g/day); 0.6% missing.
- `trig`: Triglyceride (mg/dl); 9.6% missing.

Time-varying covariates:

- `day`: Number of days between enrollment and this visit date (time variable for the laboratory measurements); complete.
- `albumin`: Serum albumin (mg/dl); complete.
- `alk.phos`: Alkaline phosphatase (U/litre); 3.1% missing.
- `ascites`: Presence of ascites; 3.1% missing.
- `ast`: Aspartate aminotransferase (U/ml); complete.
- `bili`: Serum bilirubin (mg/dl); complete.

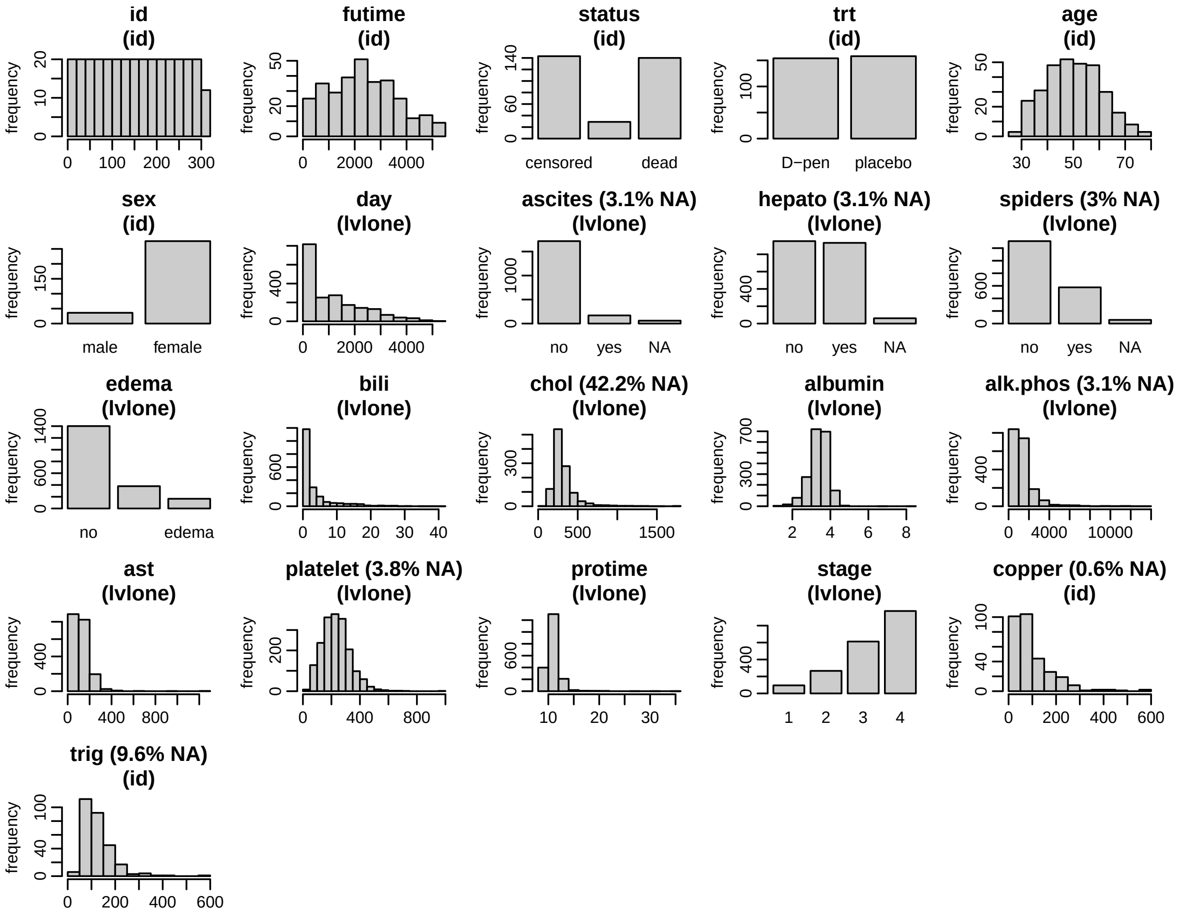

Figure 7: Distribution of the variables in the PBC data (with percentage of missing values given for incomplete variables).

- `chol`: Serum cholesterol (mg/dl); 42.2% missing.
- `edema`: `no`: no edema, `(un)treated`: untreated or successfully treated 1 edema, `edema`: edema despite diuretic therapy; complete.
- `hepato`: Presence of hepatomegaly (enlarged liver); 3.1% missing.
- `platelet`: Platelet count; 3.8% missing.
- `protime`: Standardized blood clotting time; complete.
- `spiders`: Blood vessel malformations in the skin; 3.0% missing.
- `stage`: Histologic stage of disease (4 levels); complete.

The missing data pattern and distribution of the observed values of the PBC data is shown in Figures 6 and 7.

# 5. Model specification

The main analysis functions in **JointAI** are `lm_imp()`, `glm_imp()`, `lognorm_imp()`, `betareg_imp()`, `clm_imp()`, `mlogit_imp()`, `lme_imp()`, `glme_imp()`, `lognormmm_imp()`, `betamm_imp()`, `clmm_imp()`, `mlogitmm_imp()`, `survreg_imp()`, `coxph_imp()` and `JM_imp()`. The main arguments of these functions, i.e., `formula`, `data`, `family`, `fixed`, and `random`, are used analogously to the specification in the standard complete data functions `lm()` and `glm()` from package **stats**, `lme()` from package **nlme** (Pinheiro, Bates, DebRoy, Sarkar, and R Core Team 2021) and `survreg()` and `coxph()` from package **survival** (Therneau and Grambsch 2000; Therneau 2021). For example, the usage of these functions with their most relevant arguments is:

```
lm_imp(formula, data,
  n.chains = 3, n.adapt = 100, n.iter = 0, thin = 1, ...)

glm_imp(formula, family, data,
  n.chains = 3, n.adapt = 100, n.iter = 0, thin = 1, ...)

lme_imp(fixed, data, random,
  n.chains = 3, n.adapt = 100, n.iter = 0, thin = 1, ...)

glme_imp(fixed, data, random, family,
  n.chains = 3, n.adapt = 100, n.iter = 0, thin = 1, ...)

survreg_imp(formula, data,
  n.chains = 3, n.adapt = 100, n.iter = 0, thin = 1, ...)
```

The specification for `lognorm_imp()`, `betareg_imp()`, and `mlogit_imp()` is the same as for `lm_imp()`.

The functions `lme_imp()` and `glme_imp()` have aliases `lmer_imp()` and `glmer_imp()`, and all mixed model functions accept specification of a combined fixed and random effects formula (like in the package **lme4**; Bates, Mächler, Bolker, and Walker 2015) using `fixed` and `random`.

The arguments `formula` and `fixed` take a standard two-sided `formula` object, where an intercept is added automatically (except in ordinal and proportional hazards models). For the specification of random effects formulas, see Section 5.2.

The functions `clm_imp()` and `clmm_imp()` have additional optional arguments `nonprop` and `rev`. The input `nonprop` expects a one-sided formula containing those terms of `formula` or `fixed` that should have non-proportional effects; `rev` can be set to `TRUE` to indicate that the odds should be reversed, i.e., to model $\frac{P(y\leq k)}{P(y>k)}$ instead of $\frac{P(y>k)}{P(y\leq k)}$.

Survival models expect the left hand side of `formula` to be a survival object (created with the function `Surv()` from package **survival**, see Section 5.3).

The argument `family` enables the choice of a distribution and link function from a range of options when using `glm_imp()` or `glme_imp()`. The implemented options are given in Table 1. For the description of the remaining arguments see below and Section 6.

| Distribution | Link function |
|---|---|
| `gaussian` | `identity`, `log`, `inverse` |
| `binomial` | `logit`, `probit`, `log`, `cloglog` |
| `gamma` | `inverse`, `identity`, `log` |
| `poisson` | `log`, `identity` |

Table 1: Possible choices for the `family` (distribution) and `link` (link function) arguments in `glm_imp()` and `glme_imp()`.

## 5.1. Specification of the model formula

*Interactions*

In **JointAI** interactions between any type of variables (observed, incomplete, variables from different hierarchical levels) can be handled. When an incomplete variable is involved, the interaction term is re-calculated within each iteration of the MCMC sampling, using the imputed values from the current iteration. Interaction terms involving incomplete variables should, hence, not be pre-calculated as an additional variable since this would lead to inconsistent imputed values of main effect and interaction term.

Interactions between multiple variables can be specified using parentheses; for higher lever interactions the `^` operator can be used:

```
R> mod1a <- glm_imp(educ ~ gender * (age + smoke + creat),
+    data = NHANES, family = binomial())
R> mod1b <- glm_imp(educ ~ gender + (age + smoke + creat)^3,
+    data = NHANES, family = binomial())
```

In `mod1a` the interaction between `gender` and each category of `age`, `smoke` and `creat` is included, while model `mod1b` includes all pairwise interactions between `age`, `smoke` and `creat` as well as the 3-way interaction between these variables.

*Nonlinear functional forms*

In practice, associations between outcome and covariates do not always meet the standard assumption of linearity. Often, assuming a logarithmic, quadratic or other nonlinear effect is more appropriate.

For completely observed covariates, **JointAI** can handle any common type of function implemented in R, including splines, e.g., using `ns()` or `bs()` from the package **splines**. Since functions involving variables that have missing values need to be re-calculated in each iteration of the MCMC sampling, currently, only functions that are available in **JAGS** can be used for incomplete variables. Those functions include:

- `log()`, `exp()`.
- `sqrt()`, polynomials (using `I()`).
- `abs()`.

- `sin()`, `cos()`.
- Algebraic operations (wrapped in `I()`) involving one or multiple (in)complete variables, as long as the formula can be interpreted by **JAGS**.

The list of functions implemented in **JAGS** can be found in the **JAGS** user manual (Plummer 2017). Some examples (that do not necessarily have a meaningful interpretation or good model fit) are:

```
R> mod2a <- lm_imp(SBP ~ age + gender + abs(bili - creat), data = NHANES)
R> mod2b <- lm_imp(SBP ~ ns(age, df = 2) + gender + I(bili^2) + I(bili^3),
+    data = NHANES)
R> mod2c <- lm_imp(SBP ~ age + gender + I(creat/albu^2), data = NHANES,
+    trunc = list(albu = c(1e-5, NA)))
R> mod2d <- lm_imp(SBP ~ bili + sin(creat) + cos(albu), data = NHANES)
```

It is also possible to nest a function in another function:

```
R> mod2e <- lm_imp(SBP ~ age + gender + sqrt(exp(creat)/2), data = NHANES)
```

### *Functions with restricted support*

When a function of an incomplete variable has restricted support (e.g., $\log(x)$ is only defined for $x > 0$, in `mod2c` defined above `I(creat/albu^2)` can not be calculated for `albu = 0`) the model specified for that incomplete variable needs to comply with these restrictions. This can either be achieved by truncating the distribution, using the argument `trunc`, or by selecting a distribution that meets the restrictions.

Note that truncation should be used with care. Its intended use here is to prevent issues when a variable takes a value that would result in an invalid mathematical expression. Truncation should not be used to use symmetric distributions, like the normal distribution, to fit skewed data (Von Hippel 2013; Rodwell, Lee, Romaniuk, and Carlin 2014; Geraci and McLain 2018).

**Example:** When using a log transformation for the covariate `uricacid`, we can use the default imputation method `"norm"` (a normal distribution) and truncate it by specifying `trunc = list(uricacid = c(<lower>, <upper>))`, where `<lower>` and `<upper>` are the smallest and largest values allowed:

```
R> mod3a <- lm_imp(SBP ~ age + gender + log(uricacid) + exp(creat),
+    trunc = list(uricacid = c(1e-5, NA)), data = NHANES)
```

One-sided truncation is possible by setting the limit that is not needed to `NA`.

Alternatively, we may choose a model for the incomplete variable (using the argument `models`; for more details see Section 5.5) that only imputes positive values such as a log-normal distribution or a gamma distribution:

```
R> mod3b <- lm_imp(SBP ~ age + gender + log(uricacid) + exp(creat),
+    models = c(uricacid = "lognorm"), data = NHANES)
R> mod3c <- lm_imp(SBP ~ age + gender + log(uricacid) + exp(creat),
+    models = c(uricacid = "glm_gamma_inverse"), data = NHANES)
```

### *Functions that are not available in R*

It is possible to use functions that have different names in R and **JAGS**, or that do exist in **JAGS**, but not in R, by defining a new function in R that has the name of the function in **JAGS**.

**Example:** In **JAGS** the inverse logit transformation is defined in the function `ilogit()`. In base R, there is no function `ilogit`, but the inverse logit is available as the distribution function of the logistic distribution `plogis()`. Thus, we can define the function `ilogit()` as

```
R> ilogit <- plogis
```

and use it in the model formula

```
R> mod4a <- lm_imp(SBP ~ age + gender + ilogit(creat), data = NHANES)
```

### *A note on what happens inside* ***JointAI***

When a function of a complete or incomplete variable is used in the model formula, the main effect of that variable is automatically added as an auxiliary variable (more on auxiliary variables in Section 5.6), and only the main effects are used as predictors in the imputation models.

In `mod2b` defined previously, for example, the spline of `age` is used as predictor for `SBP`, but in the imputation model for `bili`, `age` enters with a linear effect. This can be checked using the function `list_models()`, which prints a list of all sub-models used in a **JointAI** model. Here, we are only interested in the predictor variables, and, hence, suppress printing of information on prior distributions, regression coefficients and other parameters by setting `priors`, `regcoef` and `otherpars` to `FALSE`:

```
R> list_models(mod2b, priors = FALSE, regcoef = FALSE, otherpars = FALSE)

Linear model for "SBP"
   family: gaussian
   link: identity
* Predictor variables:
  (Intercept), ns(age, df = 2)1, ns(age, df = 2)2, genderfemale,
  I(bili^2), I(bili^3)

Linear model for "bili"
   family: gaussian
   link: identity
* Predictor variables:
  (Intercept), age, genderfemale
```

When a function of a variable is specified as auxiliary variable, this function is used in the imputation models. For example, in the following `mod4b` waist circumference (`WC`) is not part of the model for `SBP`, and the quadratic term `I(WC^2)` is used in the linear predictor of the imputation model for `bili`:

```
R> mod4b <- lm_imp(SBP ~ age + gender + bili, auxvars = ~ I(WC^2),
+    data = NHANES)
R> list_models(mod4b, priors = FALSE, regcoef = FALSE, otherpars = FALSE)

Linear model for "SBP"
   family: gaussian
   link: identity
* Predictor variables:
  (Intercept), age, genderfemale, bili

Linear model for "bili"
   family: gaussian
   link: identity
* Predictor variables:
  (Intercept), age, genderfemale, I(WC^2)

Linear model for "WC"
   family: gaussian
   link: identity
* Predictor variables:
  (Intercept), age, genderfemale
```

Incomplete variables are always imputed on their original scale, i.e., in `mod2b` the variable `bili` is imputed and the quadratic and cubic versions are then calculated from the imputed values. Likewise, `creat` and `albu` in `mod2c` are imputed separately, and `I(creat/albu^2)` calculated from the imputed (and observed) values. To ensure consistency between variables, functions involving incomplete variables should be specified as part of the model formula and not be pre-calculated as separate variables.

## 5.2. Multi-level structure and longitudinal covariates

In multi-level models, in addition to the fixed effects structure specified by the argument `fixed`, a random effects structure needs to be provided, either via the argument `random` (as in the package **nlme**) or in round brackets (as in the package **lme4**).

The argument `random` takes a one-sided formula starting with a `~`, and the grouping variable is separated by `|`. A random intercept is added automatically and only needs to be specified in a random intercept only model.

A few examples:

- Random intercept only, with `id` as grouping variable:
  `random = ~ 1 | id` or `formula = <...> + (1 | id)`.
- Random intercept and slope for variable `time`:
  `random = ~ time | id` or `formula = <...> + (time | id)`.
- Random intercept, slope and quadratic random effect for `time`:
  `random = ~ time + I(time^2) | id` or
  `formula = <...> + (time + I(time^2) | id)`.

- Random intercept, random slope for `time` and random effect for variable `x`: `random = ~ time + x | id` or `formula = <...> + (time + x | id)`.

It is possible to use splines in the random effects structure if there are no missing values in the variables involved, e.g.:

```
R> mod5 <- lme_imp(bmi ~ GESTBIR + ETHN + HEIGHT_M + ns(age, df = 2),
+    random = ~ ns(age, df = 2) | ID, data = simLong)
```

To specify multiple levels of grouping, i.e., a hierarchical model with more than two levels, the specification via the argument `formula` should be used. Note that in **JointAI** there is no difference between `(1 | id) + (1 | center)` and `(1 | center/id)`. The distinction between nested and crossed random effects needs to be done via the coding of the two grouping variables: if `id` should be nested in `center` then all cases with the same `id` have to have the same value for `center`.

### 5.3. Survival models

**JointAI** provides two functions to analyze survival data with incomplete covariates: `survreg_imp()` and `coxph_imp()`. Analogously to the complete data versions of these functions from the package **survival**, the left hand side of the model formula has to be a survival object specified using the function `Surv()`.

**Example:** To analyze the `PBC` data (see Section 4.3), we can either use a parametric Weibull model (considering only time-constant covariates) or a proportional hazards model. Since the `PBC` data contains time-varying covariates, we use the subset of rows where `day == 0` to have only one observation per patient.

```
R> mod6a <- survreg_imp(Surv(futime, status != "alive") ~ age + sex +
+    copper + trig, models = c(copper = "lognorm", trig = "lognorm"),
+    data = subset(PBC, day == 0), n.iter = 250)
R> mod6b <- coxph_imp(Surv(futime, status != "alive") ~ age + sex +
+    copper + trig, models = c(copper = "lognorm", trig = "lognorm"),
+    data = subset(PBC, day == 0), n.iter = 250)
```

Currently only right-censored survival data can be handled and it is not yet possible to take into account strata (i.e., strata specific baseline hazards). To model clustered data, the model formula can be extended with a random effect specification of the form `formula = <...> + (1 | center)`. The specification of subject-specific random effects also allows the user to include time-varying covariates in proportional hazards models. This requires the specification of the name of the variable containing the timing of the repeated measurements via the additional argument `timevar`:

```
R> mod6c <- coxph_imp(Surv(futime, status != "alive") ~ age + sex + copper +
+    trig + platelet + (1 | id),
+    models = c(copper = "lognorm", trig = "lognorm"),
+    timevar = "day", data = PBC)
```

Time-varying covariates are modeled (and imputed) using the last-observation-carried-forward principle. The data should include a baseline measurement (where `timevar = 0`) of the time-varying covariates. If a value needs to be filled in and no previous measurement is available, the subsequent observation is "carried-backward".

## 5.4. Joint models

Joint models for longitudinal and survival data can be fitted using the function `JM_imp()`. The specification is analogue to the specification of a proportional hazards model with time-dependent covariates, but the longitudinal trajectories are assumed to follow a smooth trajectory over time (as modeled by a mixed model) and not a step-function.

If the models for time-varying covariates are not explicitly specified, random intercept models with the default fixed effects structure are used (including linear effects for all baseline variables and time-varying variables that are complete or imputed earlier in the sequence).

To specify models for time-dependent covariates, a list of models can be supplied to the argument `formula`:

```
R> PBC$logbili <- log(PBC$bili)
R> mod6d <- JM_imp(
+    list(Surv(futime, status != "alive") ~ age + sex + platelet + logbili +
+           (1 | id),
+         platelet ~ age + sex + day + logbili + (day | id),
+         logbili ~ age + sex + day + (day | id)
+    ),
+    timevar = "day", data = PBC, n.adapt = 10)
```

The use of a list of model formulas is not restricted to `JM_imp()` but possible in any of the main analysis functions `*_imp()`. This allows the user to fit multiple analyses simultaneously, or to explicitly specify the structure of a covariate model.

When there are multiple sub-models with random effects, the structure of the joint variance-covariance matrix of these random effects can be specified as independent (`"indep"`), block-diagonal (`"blockdiag"`) or unstructured (`"full"`) using the argument `rd_vcov`.

Joint analysis of multiple substantive models may be particularly desirable if they share incomplete covariates.

## 5.5. Covariate model types

**JointAI** automatically selects an (imputation) model type for each of the incomplete covariates (and sometimes also complete covariates, as detailed in Section 2.2.1) based on the `class` of the variable.

The automatically selected types for covariates on the highest level are:

- `lm`: Linear model (for continuous variables).
- `glm_binomial_logit`: Binary logistic model (for factors with two levels).
- `mlogit`: Multinomial logit model (for unordered factors with $> 2$ levels).

- `clm`: Cumulative logit model (for ordered factors with > 2 levels).

The default methods for covariates on lower levels are:

- `lmm`: Linear mixed model.
- `glmm_binomial_logit`: Logistic mixed model.
- `mlogitmm`: Multinomial logit mixed model.
- `clmm`: Cumulative logit mixed model.

When a continuous variable has only two different values, it is automatically converted to a factor and modeled using a logistic model, unless a different model type is specified by the user. Variables of type `logical` are also converted to binary factors.

The (imputation) models that are chosen by default may not necessarily be appropriate for the data at hand, especially for continuous variables, which often do not comply with the assumptions of (conditional) normality.

Therefore, the following alternative (imputation) model types are available:

- Gamma (mixed) models for right-skewed variables > 0:
  `glm_gamma_<link>` and `glmm_gamma_<link>`, where `<link>` should be one of `inverse`, `identity` or `log`.
- Poisson (mixed) models for count data:
  `glm_poisson_<link>` and `glmm_poisson_<link>`, where `<link>` should be `log` or `identity/`
- Beta (mixed) models for for continuous variables with values in $(0, 1)$:
  `beta` and `glmm_beta`.
- Log-normal (mixed) model for right-skewed variables > 0:
  `lognorm` and `glmm_lognorm`.

All model types are implemented as described in Section 2.1.

*Specification of covariate model types*

In models `mod3b` and `mod3c` introduced in Section 5.1.3, we have already seen two examples in which the imputation model type was changed using the argument `models`. This argument takes a named vector of (imputation) model types, where the names are given by the names of covariates. When the vector supplied to `models` only contains specifications for a subset of the covariates for which a model is needed, default models are used for the remaining ones. As explained in Section 2.2.1, models for completely observed covariates may need to be specified in multi-level settings.

```
R> mod7a <- lm_imp(SBP ~ age + gender + WC + alc + bili + occup + smoke,
+    models = c(WC = "glm_gamma_inverse", bili = "lognorm"), data = NHANES,
+    n.adapt = 0)
R> mod7a$models
```

```
                    SBP                     alc                   occup 
"glm_gaussian_identity"   "glm_binomial_logit"                "mlogit" 
                   bili                   smoke                      WC 
              "lognorm"                   "clm"   "glm_gamma_inverse" 
```

When there is a "time" variable in the model, such as `age` in our example (which is the age of the child at the time of the measurement), it may not be meaningful to specify a model for that variable. Especially when the "time" variable is pre-specified by the design of the study it can usually be assumed to be independent of the covariates and a model for it has no useful interpretation. The argument `no_model` allows the user to avoid specifying models for such variables (as long as they are completely observed):

```
R> mod7b <- lme_imp(bmi ~ GESTBIR + ETHN + HEIGHT_M + SMOKE + hc +
+    MARITAL + ns(age, df = 2), random = ~ ns(age, df = 2) | ID,
+    data = simLong, no_model = "age", n.adapt = 0)
R> mod7b$models
```

```
                     bmi                      hc                   SMOKE 
"glmm_gaussian_identity"                   "lmm"                   "clm" 
                 MARITAL                    ETHN                HEIGHT_M 
                "mlogit"    "glm_binomial_logit"                    "lm" 
```

Note that by excluding the model for `age` we implicitly assume that incomplete baseline variables are independent of `age`.

*Order of the sequence of imputation models*

In multi-level models, the sequence of models for covariates is sorted by the variables level, so that variables of a higher level enter the linear predictor of variables of lower levels, but not vice versa. Within each level, models are ordered by the number of missing values (decreasing), so that the model for the variable with the largest amount of missing values has most of the variables in its linear predictor.

## 5.6. Auxiliary variables

Auxiliary variables are variables that are not part of the analysis model but should be considered as predictor variables in the imputation models because they can inform the imputation of unobserved values. Good auxiliary variables are (Van Buuren 2012):

- associated with an incomplete variable of interest, or are associated with the missingness of that variable,
- do not have too many missing values themselves. Importantly, they should be observed for a large proportion of the cases that have a missing value in the variable to be imputed.

In the main functions `*_imp()`, auxiliary variables can be specified with the argument `auxvars`, which takes a one-sided formula.

**Example:** We might consider the variables `educ` and `smoke` as predictors for the imputation of `occup`:

```
R> mod8a <- lm_imp(SBP ~ gender + age + occup, auxvars = ~ educ + smoke,
+     data = NHANES, n.iter = 100)
```

The variables `educ` and `smoke` are not included in the analysis model. They are, however, used as predictors in the imputation for `occup` and imputed themselves if they have missing values:

```
R> list_models(mod8a, priors = FALSE, regcoef = FALSE, otherpars = FALSE,
+     refcat = FALSE)


Linear model for "SBP"
   family: gaussian
   link: identity
* Predictor variables:
  (Intercept), genderfemale, age, occuplooking for work, occupnot
  working

Multinomial logit model for "occup"
* Predictor variables:
  (Intercept), genderfemale, age, educhigh, smokeformer,
  smokecurrent

Cumulative logit model for "smoke"
* Predictor variables:
  genderfemale, age, educhigh
```

*Functions of variables as auxiliary variables*

As shown above in `mod4b`, it is possible to specify functions of auxiliary variables. In that case, the auxiliary variable is not considered as a linear effect but as specified by the function.

Note that omitting auxiliary variables from the analysis model implies that the outcome is independent of these variables, conditional on the other variables in the model. If this is not true, the model is misspecified which may lead to biased results (similar to leaving a confounding variable out of a model).

## 5.7. Reference values for categorical covariates

In **JointAI**, contrasts for incomplete categorical variables need to be derived from the imputed values in each iteration of the MCMC sampling. Currently, this is only implemented for dummy and effect coding, i.e., `contr.treatment` and `contr.sum`. If a model contains an incomplete ordered factor as covariate, and R's default `contr.poly` (orthogonal polynomials) for ordered factors is set in the global `options()`, a warning is printed and dummy coding is used instead.

By default, the first category of a categorical variable (ordered or unordered) is used as reference, however, this may not always allow the desired interpretation of the regression coefficients. Moreover, when categories are unbalanced, setting the largest group as reference may result in better mixing of the MCMC chains. Therefore, **JointAI** allows the user to specify the reference category separately for each variable, via the argument `refcats`. Changes in `refcats` will not impact the imputation of the respective variable, but the definition of the contrasts, which affects the linear predictor of the analysis model or other covariate models.

*Setting reference categories for all variables*

To specify globally the choice of the reference category for all the variables in the model, `refcats` can be set as

- `refcats = "first"`
- `refcats = "last"`
- `refcats = "largest"`

For example:

```
R> mod9a <- lm_imp(SBP ~ gender + age + race + educ + occup + smoke,
+    refcats = "largest", data = NHANES)
```

*Setting reference categories for individual variables*

Alternatively, `refcats` can take a named vector, in which the reference category for each variable can be specified either by its number or its name, or one of the three global types: `"first"`, `"last"` or `"largest"`. For variables for which no reference category is specified in the list the default is used.

```
R> mod9b <- lm_imp(SBP ~ gender + age + race + educ + occup + smoke,
+    refcats = list(occup = "not working", race = 3, educ = "largest"),
+    data = NHANES)
```

To facilitate specification of the reference categories, the function `set_refcat()` can be used. It prints the names of the categorical variables that are selected by:

- a specified model formula (using the argument `formula`) and/or
- a one-sided formula specifying auxiliary variables (using the argument `auxvars`), or
- a vector naming covariates (using the argument `covars`)

or all categorical variables in the data if only the argument `data` is provided. In the latter case some questions are asked to which the user needs to reply to via input of a number:

```
R> refs_mod9 <- set_refcat(NHANES, formula = formula(mod9b))
```

```
The categorical variables are:
- "gender"
- "race"
- "educ"
- "occup"
- "smoke"

How do you want to specify the reference categories?

1: Use the first category for each variable.
2: Use the last category for each variable.
3: Use the largest category for each variable.
4: Specify the reference categories individually.
```

When option 4 is chosen, a question for each categorical variable is asked, for example:

```
The reference category for "race" should be

1: Mexican American
2: Other Hispanic
3: Non-Hispanic White
4: Non-Hispanic Black
5: other
```

After specification of the reference categories for all the categorical variables, the determined specification for the argument `refcats` is printed:

```
In the JointAI model specify:
refcats = c(gender = "female", race = "Non-Hispanic White",
           educ = "low", occup = "not working", smoke = "never")
or use the output of this function.
```

`set_refcat()` also returns a named vector that can be passed to the argument `refcats`:

```
R> mod9c <- lm_imp(SBP ~ gender + age + race + educ + occup + smoke,
+    refcats = refs_mod9, data = NHANES)
```

### 5.8. Hyper-parameters

In a Bayesian framework, parameters are random variables for which a distribution needs to be specified. These distributions depend on parameters themselves, i.e., on hyper-parameters.

The function `default_hyperpars()` returns a list containing the default hyper-parameters used in a `JointAI` model (see Appendix A).

`mu_reg_*` and `tau_reg_*` refer to the mean and precision of the prior distribution for regression coefficients. `shape_tau_*` and `rate_tau_*` are the shape and rate parameters of a gamma distribution that is used as prior for precision parameters. `RinvD` is the scale matrix in the Wishart prior for the inverse of the random effects covariance matrix `D`, and `KinvD` is the number of degrees of freedom in that distribution. `shape_diag_RinvD` and `rate_diag_RinvD` are the shape and rate parameters of the gamma prior of the diagonal elements of `RinvD`. In random effects models with only one random effect, a gamma prior is used instead of the Wishart distribution for the inverse of `D`.

The hyper-parameters `mu_reg_surv` and `tau_reg_surv` are used in `survreg_imp()`, `coxph_imp()` and `JM_imp()`.

To change hyper-parameters in a `JointAI` model, the default values can be obtained from `default_hyperpars()`, and then be adjusted and passed to the argument `hyperpars`:

```
R> hyp <- default_hyperpars()
R> hyp$norm["shape_tau_norm"] <- 0.5
R> mod9d <- lm_imp(SBP ~ gender + age + race + educ + occup + smoke,
+    data = NHANES, hyperpars = hyp)
```

### 5.9. Scaling

When variables are measured on very different scales this can result in slow convergence and bad mixing. Therefore, **JointAI** automatically scales continuous variables to approximately have mean zero and standard deviation one when they enter a linear predictor. Results are transformed back to the original scale. To prevent scaling, the argument `scale_vars` in `*_imp()` can be set to `FALSE`. When a vector of the names of model terms is supplied to `scale_vars`, only those terms are scaled. By default, only the MCMC samples that are scaled back to the scale of the data are stored in a `JointAI` object. When the argument `keep_scaled_mcmc = TRUE`, the scaled sample is also kept.

### 5.10. Shrinkage priors

Using the argument `shrinkage` it is possible to impose a penalty on the regression coefficients of all or some sub-models. If `shrinkage = "ridge"`, a ridge penalty is imposed on the regression coefficients of all sub-models by specifying a $\text{Gamma}(0.01, 0.01)$ prior for the precision of the regression coefficients instead of setting it to a fixed (small) value. It is also possible to provide a named vector to `shrinkage`, where the names should be the names of the response variables of models on which the penalty should be imposed, together with the type of shrinkage (e.g., `shrinkage = c(SBP = "ridge")`).

### 5.11. JAGS model file

Using the user-specified or default settings described above, **JointAI** writes the **JAGS** model. By default, the model is written to a temporary file and deleted when the MCMC sampling has finished. When the argument `keep_model` is set to `TRUE` the model file will be kept. In any case, the **JAGS** model is stored in the `JointAI` object as a character string. Arguments `modelname` and `modeldir` allow the user to specify the name of the file (including the ending, e.g., `.R` or `.txt`) and the file location. When a file with that same name already exists in the given location, a question is prompted giving the user the option to use the existing file or to overwrite it. To prevent the question, the argument `overwrite` can be set to `TRUE` or `FALSE`.

The functionality of using an existing **JAGS** model file enables the user to make changes to the **JAGS** model that is created automatically by **JointAI**, for example to change the type of prior distribution used for a particular parameter.

# 6. MCMC settings

The main functions `*_imp()` have a number of arguments that specify settings for the MCMC sampling:

- `n.chains`: Number of MCMC chains.
- `n.adapt`: Number of iterations in the adaptive phase.
- `n.iter`: Number of iterations in the sampling phase.
- `thin`: Thinning degree.
- `monitor_params`: Parameters/nodes to be monitored.
- `seed`: Optional seed value for reproducibility.
- `inits`: Initial values.
- `quiet`: Should printing of information be suppressed?
- `progress.bar`: Type of progress bar (`"text"`, `"gui"` or `"none"`).

The first four and last two arguments are passed directly to functions from the R package **rjags**, `monitor_params` and `seed` refer to additional functionality provided by **JointAI**.

In the following sections, the arguments listed above are explained in more detail and examples are given.

## 6.1. Number of chains, iterations and samples

### *Number of chains*

To evaluate convergence of MCMC chains it is helpful to create multiple chains that have different starting values. More information on how to evaluate convergence and the specification of initial values can be found in Sections 6.3 and , respectively.

The argument `n.chains` selects the number of chains (by default `n.chains = 3`). For calculating the model summary, multiple chains are merged.

### *Adaptive phase*

**JAGS** has an adaptive mode, in which samplers are optimized (for example the step size is adjusted). Samples obtained during the adaptive mode do not form a Markov chain and are discarded. The argument `n.adapt` controls the length of this adaptive phase.

The default value for `n.adapt` is 100, which works well in many of the examples considered here. Complex models may require longer adaptive phases. If the adaptive phase is not sufficient for **JAGS** to optimize the samplers, a warning message will be printed (see example below).

*Sampling iterations*

`n.iter` specifies the number of iterations in the sampling phase, i.e., the length of the MCMC chain. How many samples are required to reach convergence and to have sufficient precision (see also Section 7.3.2) depends on the complexity of data and model, and may range from as few as 100 to several million.

*Thinning*

In settings with high autocorrelation, it may take many iterations before a sample is created that sufficiently represents the whole range of the posterior distribution. Processing of such long chains can be slow and may cause memory issues. The parameter `thin` allows the user to specify if and how much the MCMC chains should be thinned before storing them. By default `thin = 1` is used, which corresponds to keeping all values. A value `thin = 10` would result in keeping every 10th value and discarding all other values.

**Example: default settings** Using the default settings `n.adapt = 100` and `thin = 1`, and 100 sampling iterations, a simple model would be specified as follows:

```
R> mod10a <- lm_imp(SBP ~ alc, data = NHANES, n.iter = 100)
```

The relevant part of the model summary (obtained with `summary()`) shows that the first 100 iterations (adaptive phase) were discarded, the 100 iterations that follow form the posterior sample, thinning was set to 1, and that there are three chains.

```
[...]
MCMC settings:
Iterations = 101:200
Sample size per chain = 100
Thinning interval = 1
Number of chains = 3
```

**Example: insufficient adaptation phase**

```
R> mod10b <- lm_imp(SBP ~ alc, data = NHANES, n.adapt = 10, n.iter = 100)
```

```
Warning in rjags::jags.model(file = modelfile, data = data_list, inits =
inits, : Adaptation incomplete
```

```
NOTE: Stopping adaptation
```

Specifying `n.adapt = 10` results in a warning message. The relevant part of the model summary from the resulting model is:

```
[...]
MCMC settings:
Iterations = 11:110
Sample size per chain = 100
Thinning interval = 1
Number of chains = 3
```

**Example: thinning**

```
R> mod10c <- lm_imp(SBP ~ alc, data = NHANES, n.iter = 500, thin = 10)
```

Here, iterations 110 until 600 are used in the output, but due to a thinning interval of ten, the resulting MCMC chains contain only 50 samples instead of 500, that is, the samples from iteration 110, 120, 130, …

```
[...]
MCMC settings:
Iterations = 110:600
Sample size per chain = 50
Thinning interval = 10
Number of chains = 3
```

## 6.2. Parameters to follow

Since **JointAI** uses **JAGS** (Plummer 2003) for performing the MCMC sampling, and **JAGS** only saves the values of MCMC chains for those nodes for which the user has specified that they should be monitored, this is also the case in **JointAI**.

For this purpose, the main functions `*_imp()` have an argument `monitor_params`, which takes a named list (or a named vector) with possible entries given in Table 6.2. This table contains a number of keywords that refer to (groups of) nodes. Each of the keywords works as a switch and can be specified as `TRUE` or `FALSE` (with the exception of `other`). The default setting is `monitor_params = c(analysis_main = TRUE)`, i.e., only the main parameters of the analysis model are monitored, and monitoring is switched off for all the other parameters. To additionally monitor the parameters of covariate models and imputed values `monitor_params = c(other_models = TRUE, imps = TRUE)` would have to be specified.

It is possible to switch off sub-sets of the selected groups of nodes, for example, to monitor all random effects parameters of the main model(s), but not the random effects themselves: `monitor_params = c(analysis_random = TRUE, ranef_main = FALSE)`.

The element `other` in `monitor_params` allows the specification of one or multiple additional nodes to be monitored. When `other` is used with more than one element, `monitor_params` has to be a list. Here, as an example, we monitor the probability of being in the `alc>=1` group for subjects one through three and the expected value of the distribution of `creat` for the first subject.

```
R> mod11a <- lm_imp(SBP ~ gender + WC + alc + creat, data = NHANES,
+    monitor_params = list(other = c("p_alc[1:3]", "mu_creat[1]")))
```

| Name/keyword | What is monitored |
|---|---|
| `analysis_main` | `betas` and `sigma_main` (for models with a variance parameter), `tau_main` (for beta models), `gamma_main` (for cumulative logit models), `shape_main` (for parametric survival models), `D_main` (for multi-level models), `basehaz` (for proportional hazards models) |
| `betas` | regression coefficients of the main model(s) |
| `tau_main` | precision of the residuals from the main model(s) |
| `sigma_main` | standard deviation of the residuals from the main model(s) |
| `analysis_random` | `ranef_main`, `D_main`, `invD_main`, `RinvD_main` |
| `ranef_main` | random effects of the main model(s) |
| `D_main` | covariance matrix of the random effects from the main model(s) |
| `invD_main` | inverse of `D_main` |
| `RinvD_main` | scale matrix in Wishart prior(s) for `invD_main` |
| `other_models` | `alphas`, `tau_other`, `sigma_other`, `gamma_other`, `delta_other` |
| `alphas` | regression coefficients in the covariate model(s) |
| `tau_other` | precision parameters of the residuals from covariate model(s) |
| `gamma_other` | intercepts in ordinal imputation models |
| `delta_other` | increments of ordinal intercepts |
| `imps` | imputed values |
| `ranef_other` | random effects of the covariate model(s) |
| `D_other` | covariance matrix of the random effects from the covariate model(s) |
| `invD_other` | inverse of `D_other` |
| `RinvD_other` | scale matrix in Wishart prior(s) for `invD_other` |
| `other` | additional nodes |

Table 2: Keywords and names of (groups of) nodes that can be specified to be monitored using the argument `monitor_params`.

Even though this example may not be particularly meaningful, in cases of convergence issues it can be helpful to be able to monitor any node of the model, not just the ones that are typically of interest.

More examples are given in the package vignette (`https://nerler.github.io/JointAI/articles/SelectingParameters.html`).

## 6.3. Initial values

Initial values are the starting point for the MCMC sampler. Setting good initial values, i.e., values that are likely under the posterior distribution, can speed up convergence. By default `inits = NULL`, which means that initial values are generated automatically by **JAGS**. It is also possible to supply initial values directly as a list or as a function.

Initial values can be specified for every unobserved node, that is, parameters and missing values, and it is possible to specify initial values for only a subset of nodes.

When the initial values provided by the user do not have elements named `".RNG.name"` or

".RNG.seed", **JointAI** will add those elements, which specify the name and seed value of the random number generator used for each chain. The argument seed allows the specification of a seed value with which the starting values of the random number generator, and, hence, the values of the MCMC sample, can be reproduced.

### *Initial values in a list of lists*

A list of initial values should have the same length as the number of chains, where each element is a named list of initial values and initial values should differ between chains.

For example, to create initial values for three chains for the parameter vector beta and the precision parameter tau_SBP in mod11a the following syntax could be used:

```
R> init_list <- lapply(1:3, function(i) {
+    list(beta = rnorm(5), tau_SBP = rgamma(1, 1, 1))
+  })
R> mod12a <- lm_imp(SBP ~ gender + WC + alc + creat, data = NHANES,
+    inits = init_list)
```

The user provided lists of initial values (and starting values for the random number generator) are stored in the JointAI object and can be accessed via mod11a$mcmc_settings$inits.

### *Initial values as a function*

Initial values can be specified as a function. The function should either take no arguments or a single argument called chain, and return a named list that supplies values for one chain.

For example, to create initial values for the parameter vectors beta and alpha in mod11a:

```
R> inits_fun <- function() {
+    list(beta = rnorm(5), alpha = rnorm(9))
+ }
R> mod12b <- lm_imp(SBP ~ gender + WC + alc + creat, data = NHANES,
+    inits = inits_fun)
```

When a function is supplied, the function is evaluated by **JointAI** and the resulting list is stored in the JointAI object.

### *For which nodes can initial values be specified?*

Initial values can be specified for all unobserved stochastic nodes, i.e., parameters or unobserved data for which a distribution is specified in the **JAGS** model. They have to be supplied in the format of the parameter or unobserved value in the **JAGS** model. To find out which nodes there are in a model and in which form they have to be specified, the function coef() from package **rjags** can be used to obtain a list with the current values of the MCMC chains (by default the first chain) from a **JAGS** model object. This object is contained in a JointAI object under the name model (this requires at least one iteration in the adaptive phase). Elements of the initial values should have the same structure as the elements in this list of current values. For more details, see the package vignettes.

### 6.4. Parallel sampling

To reduce the computational time it is possible to perform sampling of multiple MCMC chains in parallel. The packages **future** (Bengtsson 2021b) and **doFuture** (Bengtsson 2021a) can be used to specify how parallel processes are handled. To specify that a model should be run as four different processes, the following specification can be used before fitting the model:

```
R> library("doFuture")
R> doFuture::registerDoFuture()
R> plan(multiprocess(workers = 4))
```

This setting will remain for the entire R session, unless it is explicitly re-set to sequential computation, for instance using the following syntax:

```
R> plan(sequential)
```

## 7. After fitting

Each of the main functions `*_imp()` will return an object of class '`JointAI`'. It contains the original data (`data`), information on the type of model (`call`, `analysis_type`, `models`, `fixed`, `random`, `hyperpars`) and MCMC sampling (`mcmc_settings`), the **JAGS** model (as object of class '`jags`' in the element `model` and as string in the element `jagsmodel`) and MCMC sample (`MCMC`; if a sample was generated), information on the setting the model was run with (`comp_info` containing the start time, computational time, **JointAI** version number; `future` containing information on the setting for parallel computation), and some additional elements that are used by methods for objects of class '`JointAI`' but are typically not of interest for the user.

In this section, we describe how the results from a `JointAI` model can be visualized, summarized and evaluated. The functions described here use, by default, the full MCMC sample and show only the parameters of the analysis model. Arguments `start`, `end`, `thin` and `exclude_chains` are available to select a subset of the iterations of the MCMC sample that is used to calculate the summary. The argument `subset` allows the user to control for which nodes the summary or visualization is returned and follows the same logic as the argument `monitor_params` in `*_imp()`. For `JointAI` objects that include multiple main models (i.e, when a `list` of formulas was supplied), the argument `outcome` can be used to provide a vector of integers to select for which of the analysis models the output should be shown. The use of these arguments is further explained in Section 7.4.

### 7.1. Visualizing the posterior sample

The posterior sample can be visualized by two commonly used plots: a trace plot, showing samples across iterations, and a plot of the empirical density of the posterior sample.

*Trace plot*

A trace plot shows the sampled values per chain and node across iterations. It allows the visual evaluation of convergence and mixing of the chains and can be obtained with the function `traceplot()`.

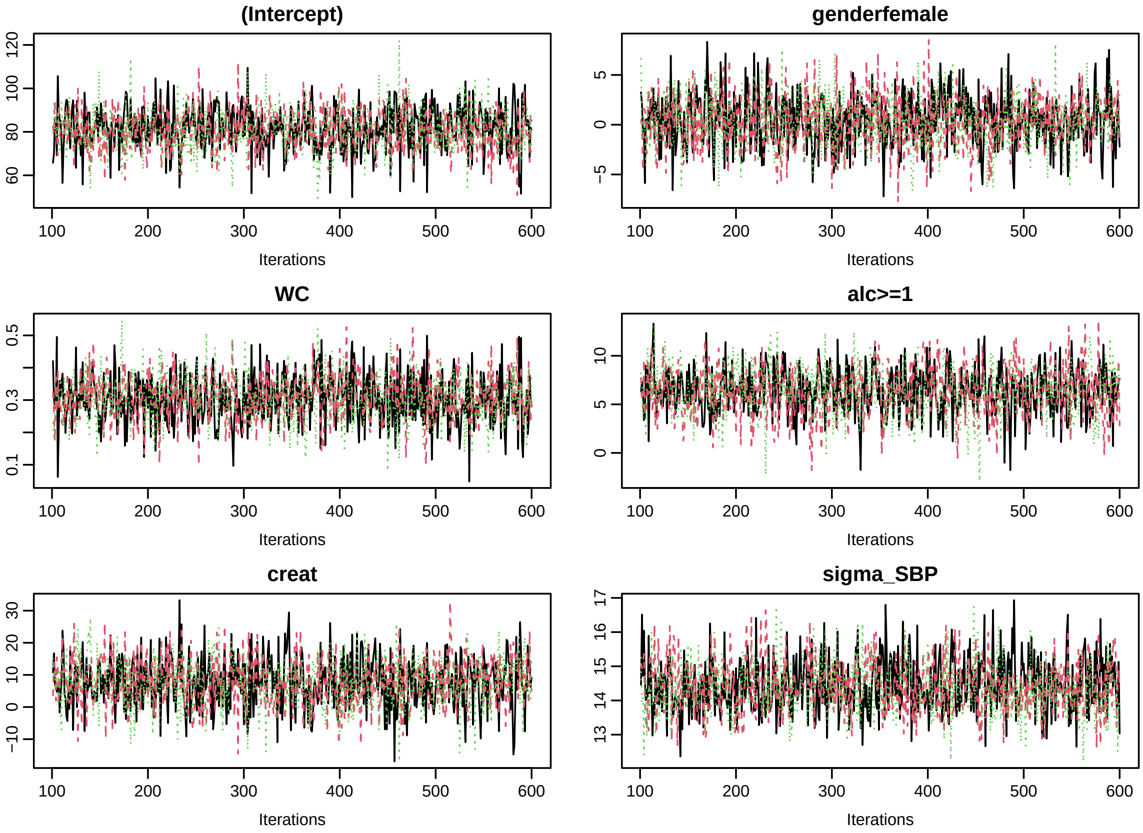

Figure 8: Traceplot for the output of `mod13a`.

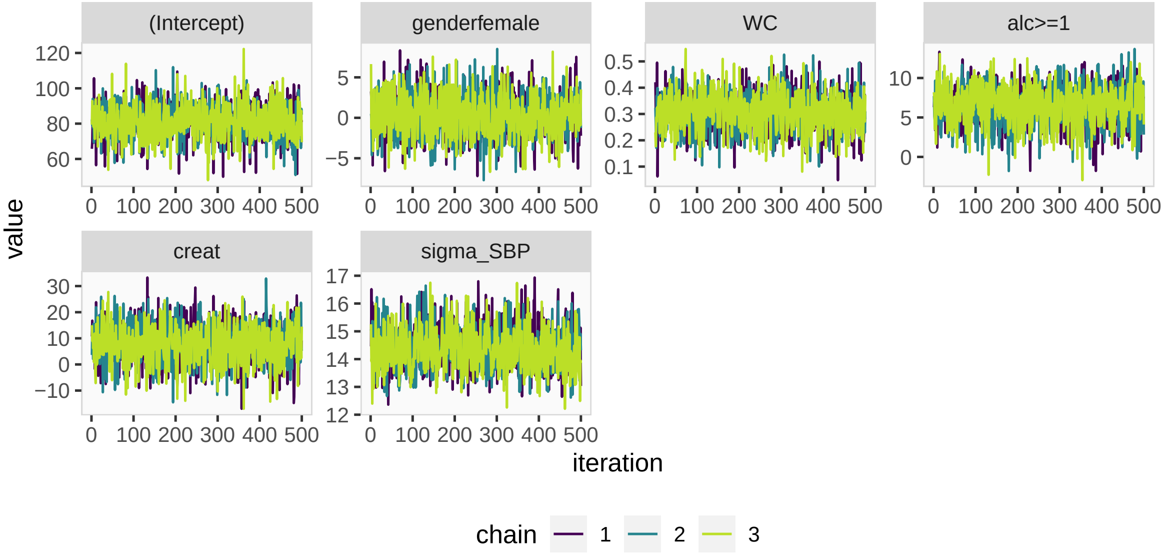

Figure 9: **ggplot2** version of the traceplot for model `mod13a`.

```
R> mod13a <- lm_imp(SBP ~ gender + WC + alc + creat, data = NHANES,
+    n.iter = 500, seed = 2020)
R> traceplot(mod13a)
```

When the sampler has converged the chains show one horizontal band, as in Figure 8. Consequently, when traces show a trend, convergence has not been reached and more iterations are necessary (e.g., using `add_samples()`).

Graphical aspects of the trace plot can be controlled by specifying standard graphical arguments via the dots argument `"..."`, which are passed to `matplot()` (which is part of base R). This allows the user to change color, line type and width, limits, and so on. Arguments `nrow` and/or `ncol` can be supplied to set specific numbers of rows and columns for the layout of the grid of plots.

With the argument `use_ggplot` it is possible to get a **ggplot2** (Wickham 2016) version of the trace plot. It can be extended using standard **ggplot2** syntax. The output of the following syntax is shown in Figure 9.

```
R> library("ggplot2")
R> traceplot(mod13a, ncol = 4, use_ggplot = TRUE) +
+    theme(legend.position = "bottom") +
+    scale_color_viridis_d(end = 0.9)
```

*Density plot*

The posterior distributions can also be visualized using the function `densplot()`, which plots the empirical density per node per chain, or combining chains (when `joined = TRUE`).

```
R> densplot(mod13a, ncol = 4,
+    vlines = list(list(v = coef(mod13a)$SBP, lwd = 2),
+                  list(v = confint(mod13a)$SBP[, "2.5%"], lty = 2),
+                  list(v = confint(mod13a)$SBP[, "97.5%"], lty = 2))
+  )
```

The argument `vlines` takes a list of lists, containing specifications passed to the function `abline()` (part of base R), and allows the addition of (vertical) lines to the plot, e.g., marking zero, or marking the posterior mean and 2.5% and 97.5% quantiles (Figure 10).

As with `traceplot()`, it is possible to use the **ggplot2** version of `densplot()` when setting `use_ggplot = TRUE`. Here, vertical lines can be added as additional layers. Figure 11 shows, as an example, the posterior density from `mod13a` to which vertical lines, representing the 95% credible interval and a 95% confidence interval from a complete case analysis, are added. The corresponding syntax is given in Appendix B.

## 7.2. Model Summary

A summary of the posterior distribution estimated in a `JointAI` model can be obtained using the function `summary()`.

The posterior summary consists of the mean, standard deviation and quantiles (by default the 2.5% and 97.5% quantiles) of the MCMC samples from all chains combined, as well as

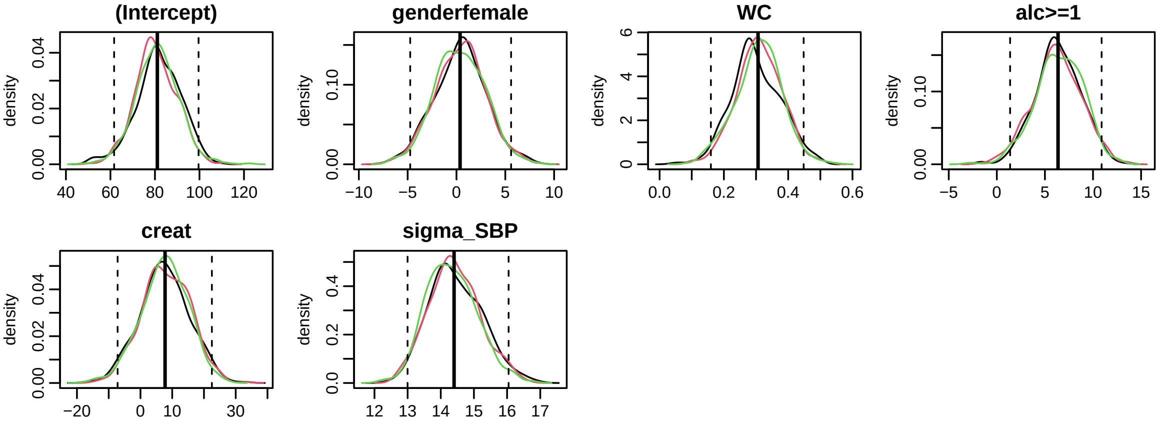

Figure 10: Empirical posterior densities for model `mod13a`.

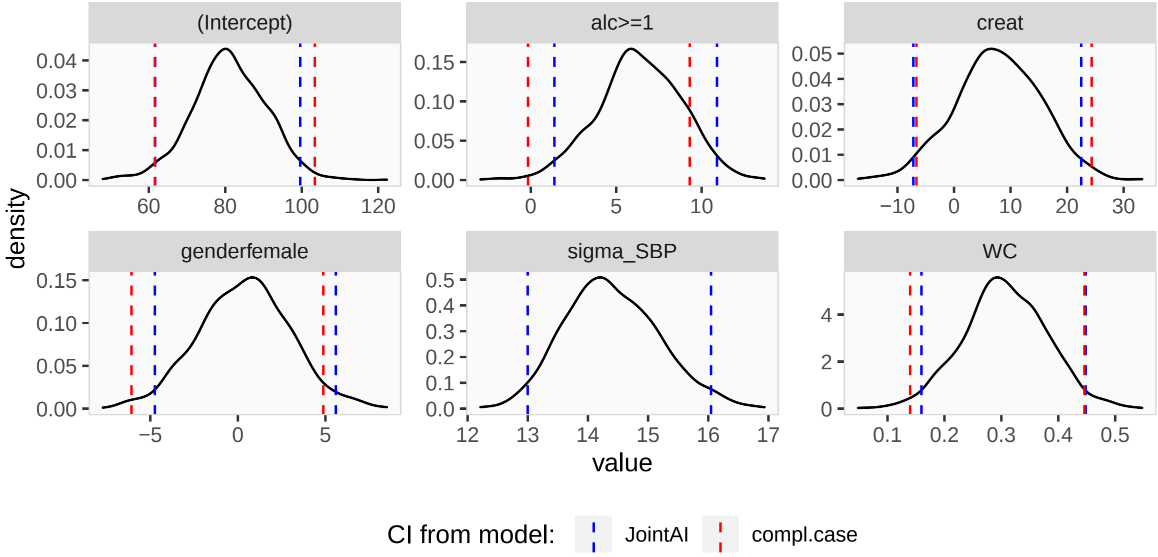

Figure 11: Density plots for model `mod13a`.

the tail probability (see below), Gelman-Rubin criterion (see Section 7.3.1) and Monte Carlo error to posterior standard deviation ratio (see Section 7.3.2).

Additionally, some important characteristics of the MCMC samples on which the summary is based, are given. This includes the range and number of iterations (`Sample size per chain`), thinning interval and number of chains. Furthermore, the number of observations (number of rows in the data) is printed.

```
R> summary(mod13a)

Bayesian linear model fitted with JointAI

Call:
lm_imp(formula = SBP ~ gender + WC + alc + creat, data = NHANES,
    n.iter = 500, seed = 2020)
```

```
Posterior summary:
               Mean     SD  2.5%  97.5% tail-prob. GR-crit MCE/SD
(Intercept)  81.077 9.6921 61.66 99.602      0.000   1.011 0.0258
genderfemale  0.368 2.6138 -4.74  5.594      0.871   0.999 0.0258
WC            0.306 0.0736  0.16  0.448      0.000   1.012 0.0259
alc>=1        6.365 2.4692  1.38 10.897      0.016   1.006 0.0291
creat         7.747 7.5949 -7.19 22.496      0.299   1.003 0.0264

Posterior summary of residual std. deviation:
          Mean    SD 2.5% 97.5% GR-crit MCE/SD
sigma_SBP 14.4 0.779   13    16    1.02 0.0278

MCMC settings:
Iterations = 101:600
Sample size per chain = 500
Thinning interval = 1
Number of chains = 3

Number of observations: 186
```

Depending on the type of model, the output shows additional sections with posterior summaries for model specific parameters, for example, the random effects variance-covariance matrix for multi-level models or the shape parameter of the Weibull distribution in a parametric survival model. Using the argument `missinfo` information on the number and proportion of complete cases and missing values per variable can be added:

```
R> mod13b <- lme_imp(bmi ~ GESTBIR + ETHN + HEIGHT_M + ns(age, df = 3),
+    random = ~ ns(age, df = 3) | ID, data = subset(simLong, !is.na(bmi)),
+    n.iter = 250, seed = 2020)
R> summary(mod13b, missinfo = TRUE)
```

```
Bayesian linear mixed model fitted with JointAI

Call:
lme_imp(fixed = bmi ~ GESTBIR + ETHN + HEIGHT_M + ns(age, df = 3),
    data = subset(simLong, !is.na(bmi)), random = ~ns(age, df = 3) |
        ID, n.iter = 250, seed = 2020)

Posterior summary:
                     Mean      SD    2.5%   97.5% tail-prob. GR-crit MCE/SD
(Intercept)       16.7480 2.40355 11.9615 21.0763      0.000    1.04 0.1031
GESTBIR           -0.0338 0.04718 -0.1249  0.0565      0.488    1.08 0.0876
ETHNother         -0.0358 0.14465 -0.3058  0.2289      0.813    1.01 0.1263
HEIGHT_M           0.0030 0.00923 -0.0141  0.0230      0.789    1.03 0.1240
ns(age, df = 3)1 -0.2917 0.07383 -0.4603 -0.1771      0.000    2.62 0.2736
ns(age, df = 3)2  1.6008 0.15233  1.1891  1.8777      0.000    3.22 0.2673
ns(age, df = 3)3 -1.3292 0.05014 -1.4198 -1.2334      0.000    1.30 0.2829
```

```
Posterior summary of random effects covariance matrix:
                Mean     SD   2.5%  97.5% tail-prob. GR-crit MCE/SD
D_bmi_ID[1,1]  1.438 0.1743  1.125  1.809                1.00 0.0485
D_bmi_ID[1,2] -0.756 0.1153 -1.017 -0.564          0    1.28 0.0727
D_bmi_ID[2,2]  0.716 0.1375  0.476  0.989                2.39 0.1567
D_bmi_ID[1,3] -2.554 0.3603 -3.380 -1.914          0    1.01 0.0494
D_bmi_ID[2,3]  2.389 0.3122  1.861  3.046          0    1.47 0.0782
D_bmi_ID[3,3]  8.240 0.9494  6.552 10.356                1.06 0.0453
D_bmi_ID[1,4] -0.719 0.1041 -0.951 -0.524          0    1.39 0.0761
D_bmi_ID[2,4]  0.593 0.0775  0.456  0.753          0    1.14 0.0766
D_bmi_ID[3,4]  2.023 0.2597  1.555  2.554          0    1.44 0.0723
D_bmi_ID[4,4]  0.526 0.0810  0.377  0.696                1.76 0.1109

Posterior summary of residual std. deviation:
           Mean      SD  2.5% 97.5% GR-crit MCE/SD
sigma_bmi 0.458 0.00852 0.442 0.475    1.03 0.0445

MCMC settings:
Iterations = 101:350
Sample size per chain = 250
Thinning interval = 1
Number of chains = 3

Number of observations: 1881
Number of groups:
 - ID: 200

Number and proportion of complete cases:
        level    #   %
ID         ID  190  95
lvlone lvlone 1881 100

Number and proportion of missing values:
     level # NA % NA
bmi lvlone    0    0
age lvlone    0    0

         level # NA % NA
GESTBIR     ID    0    0
ID          ID    0    0
HEIGHT_M    ID    4    2
ETHN        ID    6    3
```

*Tail probability*

The tail probability which is provided in the output is calculated as 2 × min $\{Pr(\theta > 0), Pr(\theta < 0)\}$, where $\theta$ is the parameter of interest. It is a measure of how likely the value 0 is

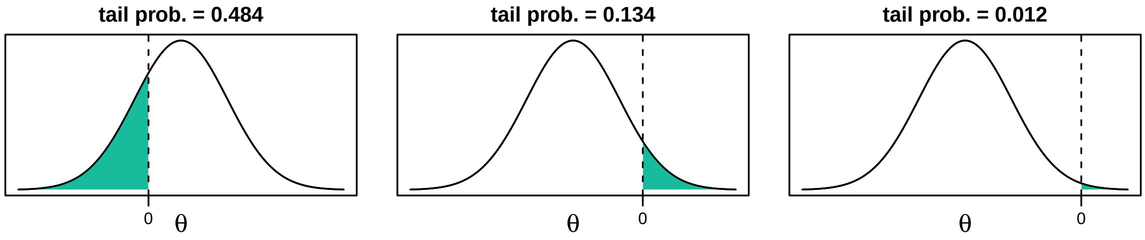

Figure 12: Visualization of the tail probability.

under the estimated posterior distribution. Figure 12 visualizes three examples of posterior distributions and the corresponding minimum between $Pr(\theta > 0)$ and $Pr(\theta < 0)$ (shaded area).

## 7.3. Evaluation criteria

Convergence of the MCMC chains and precision of the posterior sample can also be evaluated in a more formal manner. The Gelman-Rubin criterion for convergence (Gelman and Rubin 1992; Brooks and Gelman 1998) is implemented in **JointAI** together with a comparison of the Monte Carlo error with the posterior standard deviation.

### *Gelman-Rubin criterion for convergence*

The Gelman-Rubin criterion (Gelman and Rubin 1992; Brooks and Gelman 1998), also referred to as "potential scale reduction factor", evaluates convergence by comparing within and between chain variability and, thus, requires at least two MCMC chains to be calculated. It is implemented for `JointAI` objects in the function `GR_crit()`, which is based on the function `gelman.diag()` from the package **coda** (Plummer, Best, Cowles, and Vines 2006). The upper limit of the confidence interval should not be much larger than 1.

```
R> GR_crit(mod13a)
```

```
Potential scale reduction factors:

             Point est. Upper C.I.
(Intercept)        1.01       1.02
genderfemale       1.00       1.01
WC                 1.00       1.01
alc>=1             1.00       1.01
creat              1.00       1.00
sigma_SBP          1.02       1.05

Multivariate psrf

1.01
```

Besides the arguments `start`, `end`, `thin`, `exclude_chains` and `subset` (explained in Section 7.4) `GR_crit()` also takes the arguments `confidence`, `transform` and `autoburnin` inherited from `gelman.diag()`.

*Monte Carlo Error*

Precision of the MCMC sample can be checked with the function `MC_error)()`. It uses the function `mcse()` from the package **mcmcse** (Flegal, Hughes, Vats, and Dai 2021) to calculate the Monte Carlo error (the error that is made since the sample is finite) and compares it to the standard deviation of the posterior sample. A rule of thumb is that the Monte Carlo error should not be more than 5% of the standard deviation (Lesaffre and Lawson 2012). Besides the arguments explained in Section 7.4, `MC_error()` takes the arguments of `mcse()`.

```
R> MC_error(mod13a)
```

```
               est   MCSE    SD MCSE/SD
(Intercept)  81.08 0.2502 9.692   0.026
genderfemale  0.37 0.0675 2.614   0.026
WC            0.31 0.0019 0.074   0.026
alc>=1        6.37 0.0718 2.469   0.029
creat         7.75 0.2007 7.595   0.026
sigma_SBP    14.40 0.0217 0.779   0.028
```

`MC_error()` returns an object of class '`MCElist`', which is a list containing matrices with the posterior mean, estimated Monte Carlo error, posterior standard deviation and the ratio of the Monte Carlo error and posterior standard deviation, for the scaled (if this MCMC sample was included in the `JointAI` object) and unscaled (transformed back to the scale of the data) posterior samples. The associated `print` method prints only the latter.

To facilitate quick evaluation of the Monte Carlo error to posterior standard deviation ratio, plotting of an object of class '`MCElist`' using `plot()` shows this ratio for each (selected) node and automatically adds a vertical line at the desired cut-off (by default 5%; see Figure 13):

```
R> plot(MC_error(mod13a))
R> plot(MC_error(mod13a, end = 250))
```

## 7.4. Subset of the MCMC sample

By default, the functions `traceplot()`, `densplot()`, `summary()`, `predict()`, `GR_crit()` and `MC_error()` use all iterations of the MCMC sample and consider only the parameters of the analysis model (if they were monitored). In this section we describe how the set of iterations and parameters to display can be changed using the arguments `subset`, `start`, `end`, `thin` and `exclude_chains`.

*Subset of parameters*

When the main parameters of the main/analysis model(s) have been monitored in a `JointAI` object only these parameters are returned in the model summary, plots and criteria shown above. If the main parameters of the analysis model(s) were not monitored and the argument `subset` is not specified, all parameters that were monitored are displayed.

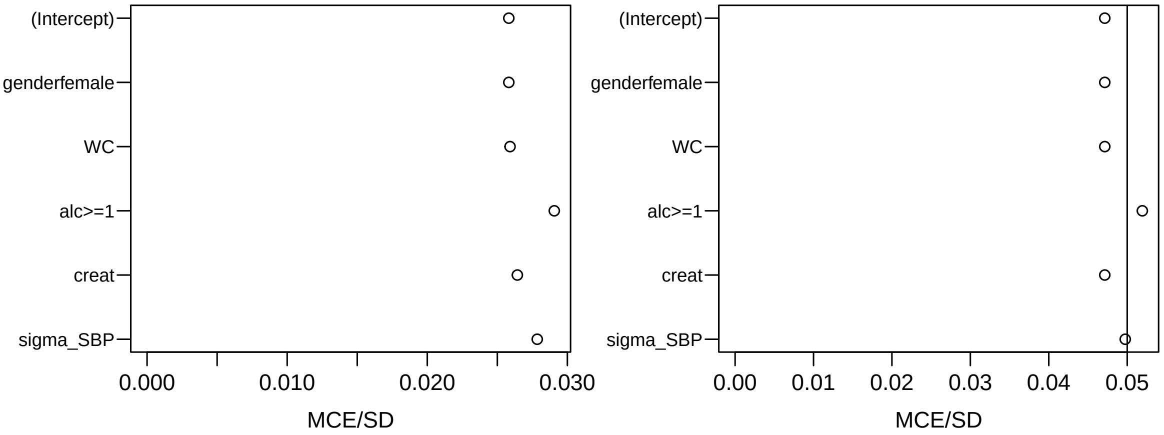

Figure 13: Plot of the 'MCElist' object from mod13a. Left: including all iterations, right: using only the first 250 iterations of the MCMC sample.

To display output for nodes other than the main parameters of the analysis model or for a subset of nodes, the argument `subset` needs to be specified. It follows the same logic as the argument `monitor_params` of `*_imp()` explained in Section 6.2.

**Example:** To display only the parameters of the covariate models, we re-estimate the model with the monitoring for these parameters switched on and set `subset = c(analysis_main = FALSE, other_models = TRUE)`:

```
R> mod13c <- update(mod13a, monitor_params = c(other_models = TRUE))
R> summary(mod13c, subset = c(analysis_main = FALSE, other_models = TRUE))

Bayesian joint model fitted with JointAI

Call:
lm_imp(formula = SBP ~ gender + WC + alc + creat, data = NHANES,
    n.iter = 500, monitor_params = c(other_models = TRUE), seed = 2020)


# --------------------------------------------------------------------- #
  Bayesian binomial model for "alc"
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

Posterior summary:
                 Mean     SD    2.5%   97.5% tail-prob. GR-crit MCE/SD
(Intercept)   0.51390 1.5325 -2.6068  3.4078     0.7080    1.01 0.0535
genderfemale -0.88236 0.3995 -1.6322 -0.0498     0.0373    1.04 0.0762
WC            0.00632 0.0115 -0.0169  0.0298     0.5627    1.01 0.0375
creat        -1.48151 1.2238 -3.9056  0.9785     0.2213    1.00 0.0603

# --------------------------------------------------------------------- #
```

```
  Bayesian linear model for "creat"
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

Posterior summary:
                  Mean       SD      2.5%    97.5% tail-prob. GR-crit MCE/SD
(Intercept)   0.844704 0.076409  0.694938  0.99127      0.000       1 0.0258
genderfemale -0.178815 0.022122 -0.223627 -0.13699      0.000       1 0.0258
WC            0.000877 0.000772 -0.000612  0.00243      0.256       1 0.0258

Posterior summary of residual std. deviation:
             Mean      SD  2.5% 97.5% GR-crit MCE/SD
sigma_creat 0.145 0.00769 0.132 0.161    1.01 0.0277

# --------------------------------------------------------------------- #
  Bayesian linear model for "WC"
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - #

Posterior summary:
              Mean   SD  2.5%   97.5% tail-prob. GR-crit MCE/SD
(Intercept)  97.41 1.52 94.48 100.469     0.0000    1.00 0.0258
genderfemale -5.16 2.21 -9.38  -0.971     0.0147    1.01 0.0258

Posterior summary of residual std. deviation:
         Mean    SD 2.5% 97.5% GR-crit MCE/SD
sigma_WC 14.5 0.785 13.2  16.3       1 0.0258

# ------------------------------------------------------------ #

MCMC settings:
Iterations = 101:600
Sample size per chain = 500
Thinning interval = 1
Number of chains = 3

Number of observations: 186
```

**Example:** To select only some of the parameters, they can be specified directly by name via the `other` element of `subset` (output not shown).

```
R> densplot(mod13a, nrow = 1,
+    subset = list(analysis_main = FALSE, other = c("beta[2]", "beta[4]")))
```

**Example:** This also works when a subset of the imputed values should be displayed. For example, re-fit the model and monitor the imputed values and select all imputed values for `WC` (4-th column of `M_lvlone`, the data matrix containing all level-1 variables):

```
R> mod13d <- update(mod13a, monitor_params = c(imps = TRUE))
R> sub3 <- grep("M_lvlone\\[[[:digit:]]+,4\\]", parameters(mod13d)$coef,
+    value = TRUE)
R> sub3

[1] "M_lvlone[33,4]"  "M_lvlone[150,4]"
```

The function `parameters()` returns a `data.frame` containing the names of all nodes monitored in a `JointAI object` and can help to identify the correct names of the nodes to be plotted.

Pass `sub3` to `subset` via `"other"`, for example in a `traceplot()`:

```
R> traceplot(mod13d, subset = list(other = sub3))
```

**Example:** When the number of imputed values is large or in order to check convergence of random effects, it may not be feasible to plot and inspect all trace plots. In that case, a random subset of, for instance, the random effects, can be selected.

Here below for example we re-fit the model monitoring the random effects, then obtain a vector with the names of all random effects and obtain the trace plots for a random subset (output not shown):

```
R> mod13e <- update(mod13b, monitor_params = c(ranef_main = TRUE))
R> rde <- grep("^b_bmi_ID\\[", colnames(mod13e$MCMC[[1]]), value = TRUE)
R> traceplot(mod13e, subset = list(analysis_main = FALSE,
+    other = sample(rde, size = 12)), ncol = 4)
```

*Subset of MCMC samples*

With the arguments `start`, `end` and `thin` it is possible to select which iterations from the MCMC sample are included in the summary. In particular, `start` and `end` specify the first and last iterations to be used, `thin` the thinning interval. Specification of `start` thus allows the user to discard a “burn-in”, i.e., the iterations before the MCMC chain had converged. If a particular chain has not converged it can be excluded from the result summary or plot using the argument `exclude_chains` which takes a numeric vector identifying chains to be excluded, e.g., `exclude_chains = c(1, 3)`.

## 7.5. Predicted values

Often the aim of an analysis is not only to estimate the association between outcome and covariates but to predict future outcomes or outcomes for new subjects.

The function `predict()` allows us to obtain predicted values and corresponding credible intervals from `JointAI` objects. Note that for mixed models, currently only prediction for an “average” subject is implemented, not prediction conditional on the random effects. A dataset containing data for which the prediction should be performed is specified via the argument `newdata`. If no `newdata` is given, the original data are used. The argument `quantiles` allows the specification of the quantiles of the posterior sample that are used to obtain the credible interval (by default the 2.5% and 97.5% quantile). Arguments `start`, `end`, `thin` and `exclude_chains` control the subset of MCMC samples that is used.

```
R> predict(mod13a, newdata = NHANES[27, ])

$newdata
         SBP gender age             race   WC alc educ creat albu uricacid
392 126.6667   male  32 Mexican American 94.1  <1  low  0.83  4.2      8.7
    bili occup  smoke      fit     2.5%    97.5%
392    1  <NA> former 116.3273 112.4343 120.1817

$fitted
       fit     2.5%    97.5%
1 116.3273 112.4343 120.1817
```

`predict()` returns a list with elements `newdata` (the provided data with the predicted values and quantiles appended) and `fit`, a `matrix` or `array` of the predicted values and the quantiles that form the credible interval.

Via the argument `type` the user can specify the scale of the predicted values. For generalized linear (mixed) models predicted values can be calculated on the scale of the linear predictor (`type = "link"` or `type = "lp"`) or the scale of the response (`type = "response"`). For ordinal and multinomial (mixed) models it is possible to return the posterior probability of each of the outcome categories (`type = "prob"`), the class with the highest mean posterior probability (`type = "class"`, or `type = "response"`) or the linear predictor (`type = "lp"`).

For parametric survival models `type = "lp"` is synonymous for `type = "link"` and `type = "linear"`, and `type = "response"` corresponds to `exp(lp)`. The options for proportional hazards models are `type = "lp"`, `type = "risk"` (for exp(`lp`)), `type = "survival"` and `type = "expected"` (for $-\log($`survival`$)$).

### *Prediction to visualize nonlinear effects*

Another reason to obtain predicted values is the visualization of nonlinear effects (see Figure 14). To facilitate the generation of a dataset for such a prediction, the function `predDF()` can be used. It generates a `data.frame` that contains a sequence of values through the range of observed values for the covariate specified by the argument `vars` which takes a one-sided formula. Median or reference values are used for all the other continuous and categorical variables, respectively.

The following code creates the dataset for prediction and obtain the predicted values

```
R> newDF <- predDF(mod13b, vars = ~ age)
R> pred <- predict(mod13b, newdata = newDF)
```

and then plot the predicted values and credible interval (see Figure 14):

```
R> matplot(pred$newdata$age, pred$newdata[, c("fit", "2.5%", "97.5%")],
+    lty = c(1,2,2), type = "l", col = 1, xlab = "age in months",
+    ylab = "predicted value")
```

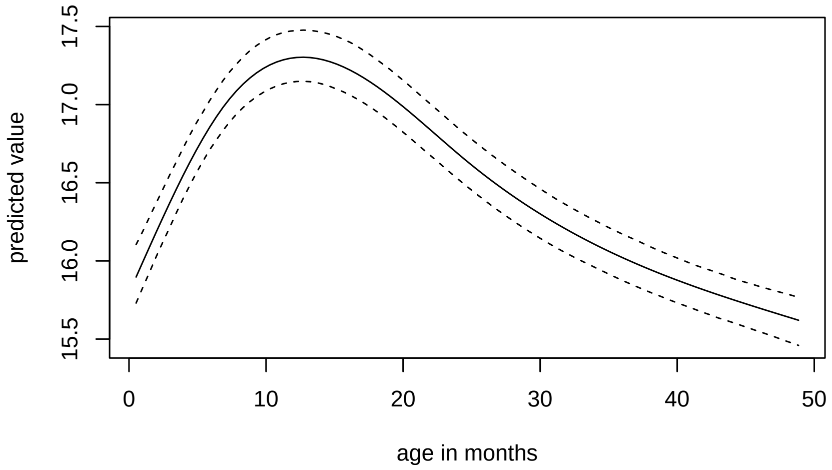

Figure 14: Predicted values of BMI and corresponding 95% credible interval from mod13b.

The optional "..." argument of predDF() allows the user to explicitly specify which values to be used for the variables given in vars, for example:

```
R> newDF2 <- predDF(mod13b, vars = ~ age + HEIGHT_M, HEIGHT_M = c(160, 175))
```

## 7.6. Export of imputed values

Imputed datasets can be extracted from a JointAI object (in which a monitor for the imputed values has been set, i.e., monitor_params = c(imps = TRUE)) with the function get_MIdat(). It creates completed datasets by taking the imputed values from randomly chosen iterations of the MCMC sample and filling them into copies of the original incomplete data.

```
R> impDF <- get_MIdat(mod13d, m = 10, seed = 2019)
```

The argument m specifies the number of imputed datasets to be created, include controls whether the original data are included in the long format data.frame (default is include = TRUE), start specifies the first iteration that may be used, and minspace is the minimum number of iterations between iterations eligible for selection. To make the selection of iterations reproducible, a seed value can be specified via the argument seed.

When export_to_SPSS = TRUE the imputed data are exported to SPSS (IBM Corporation 2017), i.e., a .txt file containing the data and a .sps file containing SPSS syntax to convert the data into an SPSS data file (with extension .sav) are written. Arguments filename and resdir allow specification of the name of the .txt and .sps file and the directory they are written to.

get_MIdat() returns a long-format data.frame containing the imputed datasets (and by default the original data) stacked on top of each other. The imputation number is given in the variable Imputation_, while column .id contains a newly created id variable for each observation in cross-sectional data (multi-level data should already contain an id variable) and the column .rownr identifies rows of the original data (relevant in multi-level data).

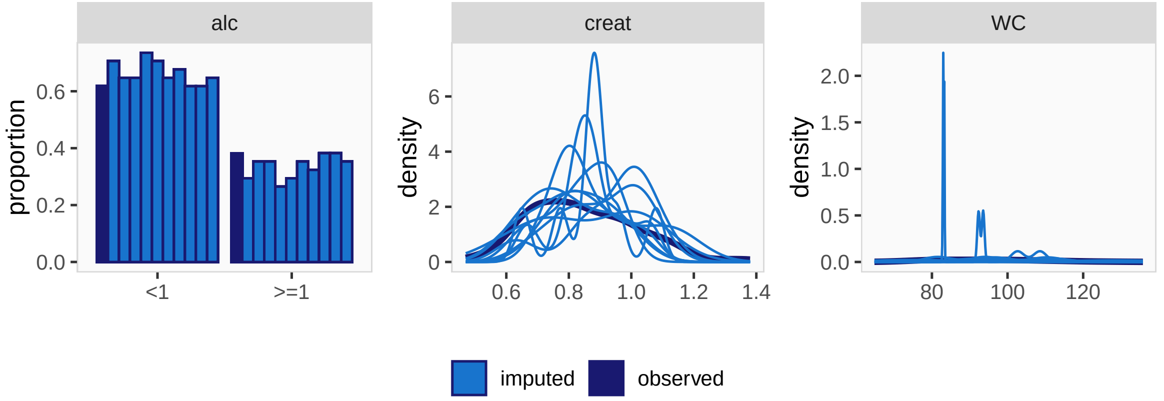

Figure 15: Distribution of observed and imputed values for model `mod13d`.

The function `plot_imp_distr()` allows visual comparison of the distributions of the observed and imputed values. The distribution of the observed values is shown in dark blue, the distribution of the imputed values per dataset in light blue (Figure 15):

```
R> plot_imp_distr(impDF, nrow = 1)
```

# 8. Assumptions and extensions

Like any statistical model, the approach followed in **JointAI** relies on assumptions that need to be satisfied in order to obtain valid results.

A commonly made assumption that is also required for **JointAI** is that the missing data mechanism is ignorable, i.e., that data are missing at random (MAR) or missing completely at random (MCAR) (Rubin 1976) and that parameters in the model of the missingness mechanism are independent of the parameters in the data model (Schafer 1997). It is the task of the researcher to critically evaluate whether this assumption is satisfied for a given dataset and model.

Furthermore, all models involved in the imputation and analysis need to be correctly specified. In current implementations of imputation procedures in software (e.g., the package **mice** in R or `proc mi` in SAS (SAS Institute Inc. 2013), imputation models are typically automatically specified, using standard assumptions like linear associations and default model types. In **JointAI** the arguments `models` and `auxvar` permit tailoring of the automatically chosen models to some extent, by allowing the user to choose non-normal imputation models for continuous variables and to include variables or functional forms of variables that are not used in the analysis model in the linear predictor of the imputation models. Moreover, it is possible to explicitly specify the linear predictor of covariate models by providing a list of model formulas instead of just the formula for the main analysis model.

When using auxiliary variables in **JointAI**, it should be noted that, due to the default ordering of the conditional distributions in the sequence of models, it is implied that the auxiliary variable is independent of the outcome, since neither the model for the auxiliary variable has the outcome in its linear predictor nor vice versa. In some settings it may be possible to avoid this assumption by providing a list of model formulas in which the model for the auxiliary variable is specified explicitly to include the outcome in its linear predictor.

Moreover, in order to make any statistical software usable, default values have to be chosen for various parameters. These default values are chosen to work well in certain settings, but can not be guaranteed to be appropriate in general and it is the task of the user to make the appropriate changes. In **JointAI** this concerns, for example, the choice of hyper-parameters and automatically chosen types of imputation models.

To expand the range of settings in which **JointAI** provides a valid and user-friendly way to simultaneously analyze and impute data, several extensions are planned. These include:

- Implementation of (penalized) splines for incompletely observed covariates.
- Evaluation of model fit by providing functionality to perform posterior predictive checks.
- Implementation of subject-specific prediction from mixed models.
- Implementation of additional choices of shrinkage priors (such as lasso and elastic net).
- Implementation of additional model types, for example, using zero-inflated or over-dispersed distributions.
- Extensions of joint models for longitudinal and survival data to other association structures such as slopes and cumulative effects.
- Extensions of survival models to other types of censoring, competing risks and stratified baseline hazards.

## Computational details

The results in this paper have been obtained with R 4.1.1, **JointAI** 1.0.3, **rjags** 4.12 and **JAGS** 4.3.0 on a Windows 10 system. The full replication code including random seeds is provided in the supplementary materials. Note, however, that for replicating the results exactly, the same operating system and R version 3.6.0 or newer would be required. In other setups the results will be very similar, however.

## References

Audigier V, Resche-Rigon M (2021). ***micemd**: Multiple Imputation by Chained Equations with Multilevel Data.* R package version 1.8.0, URL https://CRAN.R-project.org/package=micemd.

Bartlett J, Keogh R (2021). ***smcfcs**: Multiple Imputation of Covariates by Substantive Model Compatible Fully Conditional Specification.* R package version 1.6.0, URL https://CRAN.R-project.org/package=smcfcs.

Bartlett JW, Seaman SR, White IR, Carpenter JR (2015). "Multiple Imputation of Covariates by Fully Conditional Specification: Accommodating the Substantive Model." *Statistical Methods in Medical Research*, **24**(4), 462–487. doi:10.1177/0962280214521348.

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using **lme4**." *Journal of Statistical Software*, **67**(1), 1–48. doi:10.18637/jss.v067.i01.

Bengtsson H (2021a). ***doFuture**: A Universal Foreach Parallel Adapter Using the Future API of the **future** Package.* R package version 0.12.0, URL https://CRAN.R-project.org/package=doFuture.

Bengtsson H (2021b). "A Unifying Framework for Parallel and Distributed Processing in R using Futures." *The R Journal*. doi:10.32614/RJ-2021-048. Forthcoming.

Brooks SP, Gelman A (1998). "General Methods for Monitoring Convergence of Iterative Simulations." *Journal of Computational and Graphical Statistics*, **7**(4), 434–455. doi:10.1080/10618600.1998.10474787.

Deng Y, Chang C, Ido MS, Long Q (2016). "Multiple Imputation for General Missing Data Patterns in the Presence of High-dimensional Data." *Scientific reports*, **6**(1), 1–10. doi:10.1038/srep21689.

Erler NS (2021). ***JointAI**: Joint Analysis and Imputation of Incomplete Data.* R package version 1.0.3, URL https://CRAN.R-project.org/package=JointAI.

Erler NS, Rizopoulos D, Jaddoe VW, Franco OH, Lesaffre EMEH (2019). "Bayesian Imputation of Time-Varying Covariates in Linear Mixed Models." *Statistical Methods in Medical Research*, **28**(2), 555–568. doi:10.1177/0962280217730851.

Erler NS, Rizopoulos D, Van Rosmalen J, Jaddoe VWV, Franco OH, Lesaffre EMEH (2016). "Dealing with Missing Covariates in Epidemiologic Studies: A Comparison between Multiple Imputation and a Full Bayesian Approach." *Statistics in Medicine*, **35**(17), 2955–2974. doi:10.1002/sim.6944.

Flegal JM, Hughes J, Vats D, Dai N (2021). ***mcmcse**: Monte Carlo Standard Errors for MCMC.* R package version 1.5-0, URL https://CRAN.R-project.org/package=mcmcse.

Gelman A, Rubin DB (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4), 457–472. doi:10.1214/ss/1177011136.

Geraci M, McLain A (2018). "Multiple Imputation for Bounded Variables." *Psychometrika*, **83**(4), 919–940. doi:10.1007/s11336-018-9616-y.

Grund S, Robitzsch A, Luedtke O (2021). ***mitml**: Tools for Multiple Imputation in Multilevel Modeling.* R package version 0.4-3, URL https://CRAN.R-project.org/package=mitml.

Hadfield JD (2010). "MCMC Methods for Multi-Response Generalized Linear Mixed Models: The **MCMCglmm** R Package." *Journal of Statistical Software*, **33**(2), 1–22. doi:10.18637/jss.v033.i02.

IBM Corporation (2017). *IBM SPSS Statistics 25*. IBM Corporation, Armonk. URL https://www.ibm.com/software/analytics/spss/.

Ibrahim JG, Chen MH, Lipsitz SR (2002). "Bayesian Methods for Generalized Linear Models with Covariates Missing At Random." *Canadian Journal of Statistics*, **30**(1), 55–78. doi:10.2307/3315865.

Josse J, Tierney NJ, Vialaneix N (2021). *CRAN Task View: Missing Data*. Version 2021-11-09, URL https://CRAN.R-project.org/view=MissingData.

Kowarik A, Templ M (2016). "Imputation with the R Package **VIM**." *Journal of Statistical Software*, **74**(7), 1–16. doi:10.18637/jss.v074.i07.

Lesaffre EMEH, Lawson AB (2012). *Bayesian Biostatistics*. John Wiley & Sons. doi:10.1002/9781119942412.

National Center for Health Statistics (NCHS) (2011–2012). "National Health and Nutrition Examination Survey Data." URL https://www.cdc.gov/nchs/nhanes/.

Novo AA, Schafer JL (2013). ***norm**: Analysis of Multivariate Normal Datasets with Missing Values*. R package version 1.0-9.5, URL https://CRAN.R-project.org/package=norm.

Pinheiro J, Bates D, DebRoy S, Sarkar D, R Core Team (2021). ***nlme**: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-153, URL https://CRAN.R-project.org/package=nlme.

Plummer M (2003). "**JAGS**: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*. Technische Universität Wien, Vienna, Austria. URL https://www.R-project.org/conferences/DSC-2003/Proceedings/Plummer.pdf.

Plummer M (2017). ***JAGS** Version 4.3.0 User Manual*. URL https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/jags_user_manual.pdf/download.

Plummer M (2021). ***rjags**: Bayesian Graphical Models Using MCMC*. R package version 4-12, URL https://CRAN.R-project.org/package=rjags.

Plummer M, Best N, Cowles K, Vines K (2006). "**coda**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. URL https://CRAN.R-project.org/doc/Rnews/.

Quartagno M, Carpenter J (2020). ***jomo**: A Package for Multilevel Joint Modelling Multiple Imputation*. R package version 2.7-2, URL https://CRAN.R-project.org/package=jomo.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Robitzsch A, Grund S, Henke T (2021). ***miceadds**: Some Additional Multiple Imputation Functions, Especially for **mice***. R package version 3.11-6, URL https://CRAN.R-project.org/package=miceadds.

Robitzsch A, Luedtke O (2021). ***mdmb**: Model Based Treatment of Missing Data*. R package version 1.5-8, URL https://CRAN.R-project.org/package=mdmb.

Rodwell L, Lee KJ, Romaniuk H, Carlin JB (2014). "Comparison of Methods for Imputing Limited-range Variables: A Simulation Study." *BMC Medical Research Methodology*, **14**(1), 57. doi:10.1186/1471-2288-14-57.

Rubin DB (1976). "Inference and Missing Data." *Biometrika*, **63**(3), 581–592. doi:10.2307/2335739.

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys.* John Wiley & Sons.

Rubin DB (2004). "The Design of a General and Flexible System for Handling Nonresponse in Sample Surveys." *The American Statistician*, **58**, 298–302. doi:10.1198/000313004X6355.

SAS Institute Inc (2013). *SAS/STAT Software, Version 9.4.* Cary. URL https://www.sas.com/.

Schafer JL (1997). *Analysis of Incomplete Multivariate Data.* Chapman & Hall/CRC, New York.

Speidel M, Drechsler J, Jolani S (2020). ***hmi**: Hierarchical Multiple Imputation.* R package version 1.0.0, URL https://CRAN.R-project.org/package=hmi.

StataCorp (2021). *Stata Statistical Software: Release 17.* StataCorp LLC, College Station. URL https://www.stata.com/.

Therneau TM (2021). ***survival**: Survival Analysis.* R package version 3.2-13, URL https://CRAN.R-project.org/package=survival.

Therneau TM, Grambsch PM (2000). *Modeling Survival Data: Extending the Cox Model.* Springer-Verlag, New York.

Tierney NJ, Cook DH (2020). "Expanding Tidy Data Principles to Facilitate Missing Data Exploration, Visualization and Assessment of Imputations." *arXiv 1809.02264*, arXiv.org E-Print Archive. URL https://arxiv.org/abs/1809.02264.

Treiman D (2009). *Quantitative Data Analysis: Doing Social Research to Test Ideas.* Research Methods for the Social Sciences. John Wiley & Sons.

Van Buuren S (2012). *Flexible Imputation of Missing Data.* Taylor & Francis.

Van Buuren S, Groothuis-Oudshoorn K (2011). "**mice**: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software*, **45**(3), 1–67. doi:10.18637/jss.v045.i03.

Von Hippel PT (2013). "Should a Normal Imputation Model be Modified to Impute Skewed Variables?" *Sociological Methods & Research*, **42**(1), 105–138. doi:10.1177/0049124112464866.

White IR, Royston P, Wood AM (2011). "Multiple Imputation Using Chained Equations: Issues and Guidance for Practice." *Statistics in Medicine*, **30**(4), 377–399. doi:10.1002/sim.4067.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis.* Springer-Verlag, New York. URL https://ggplot2.tidyverse.org/.

Yucel R (2010). ***mlmmm**: ML Estimation under Multivariate Linear Mixed Models with Missing Values.* R package version 0.3-1.2, URL https://CRAN.R-project.org/package=mlmmm.

# A. Default hyper-parameters

```
R> default_hyperpars()

$norm
   mu_reg_norm   tau_reg_norm shape_tau_norm  rate_tau_norm
         0e+00          1e-04          1e-02          1e-02

$gamma
   mu_reg_gamma   tau_reg_gamma shape_tau_gamma  rate_tau_gamma
          0e+00           1e-04           1e-02           1e-02

$beta
   mu_reg_beta   tau_reg_beta shape_tau_beta  rate_tau_beta
         0e+00          1e-04          1e-02          1e-02

$binom
 mu_reg_binom tau_reg_binom
        0e+00         1e-04

$poisson
 mu_reg_poisson tau_reg_poisson
          0e+00           1e-04

$multinomial
 mu_reg_multinomial tau_reg_multinomial
              0e+00               1e-04

$ordinal
   mu_reg_ordinal   tau_reg_ordinal  mu_delta_ordinal tau_delta_ordinal
            0e+00             1e-04             0e+00             1e-04

$ranef
shape_diag_RinvD  rate_diag_RinvD       KinvD_expr
          "0.01"          "0.001"   "nranef + 1.0"

$surv
 mu_reg_surv tau_reg_surv
       0.000        0.001
```

# B. Density plot using ggplot2

This appendix shows the syntax to create the density plot for model `mod13a` shown in Figure 11 in Section 7.1.2. Analogously to what was shown previously for `traceplot()`, we can obtain a density plot using **ggplot2** by setting the argument `use_ggplot = TRUE`:

```
R> p13a <- densplot(mod13a, ncol = 3, use_ggplot = TRUE, joined = TRUE) +
+    theme(legend.position = "bottom")
```

It is also straightforward to add vertical lines for credible intervals and, for the purpose of comparison, also confidence intervals of from a complete case analysis.

To do this, we first fit the complete-case version of the model:

```
R> mod13a_cc <- lm(formula(mod13a), data = NHANES)
```

It is convenient to create a dataset containing the quantiles of the posterior sample and confidence intervals from the complete case analysis:

```
R> quantDF <- rbind(
+    data.frame(variable = names(coef(mod13a)$SBP),
+               type = "2.5%",
+               model = "JointAI",
+               value = confint(mod13a)$SBP[, c("2.5%")]),
+    data.frame(variable = names(coef(mod13a)$SBP),
+               type = "97.5%",
+               model = "JointAI",
+               value = confint(mod13a)$SBP[, c("97.5%")]),
+    data.frame(variable = names(coef(mod13a_cc)),
+               type = "2.5%",
+               model = "cc",
+               value = confint(mod13a_cc)[, "2.5 %"]),
+    data.frame(variable = names(coef(mod13a_cc)),
+               type = "97.5%",
+               model = "cc",
+               value = confint(mod13a_cc)[, "97.5 %"])
+  )
```

The vertical lines can then be added to the previously created plot `p13a` using the function `geom_vline()` from the package **ggplot2**:

```
R> p13a +
+    geom_vline(data = quantDF, aes(xintercept = value, color = model),
+               lty = 2) +
+    scale_color_manual(name = "CI from model: ", values = c("blue", "red"),
+                       limits = c("JointAI", "cc"),
+                       labels = c("JointAI", "compl.case"))
```

## Affiliation:

Nicole S. Erler
Erasmus Medical Center
Department of Biostatistics
Doctor Molewaterplein 40
3015 GD Rotterdam, The Netherlands
E-mail: n.erler@erasmusmc.nl
URL: https://www.nerler.com/

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2021, Volume 100, Issue 20 *Submitted:* 2019-06-24
doi:10.18637/jss.v100.i20 *Accepted:* 2020-11-18