Package: JointAI (via r-universe)

October 30, 2026

Version 1.1.0

Title Joint Analysis and Imputation of Incomplete Data

Description Joint analysis and imputation of incomplete data in the Bayesian framework, using (generalized) linear (mixed) models and extensions there of, survival models, or joint models for longitudinal and survival data, as described in Erler, Rizopoulos and Lesaffre (2021) <doi:10.18637/jss.v100.i20>. Incomplete covariates, if present, are automatically imputed. The package performs some preprocessing of the data and creates a 'JAGS' model, which will then automatically be passed to 'JAGS' <https://mcmc-jags.sourceforge.io/> with the help of the package 'rjags'.

URL https://nerler.github.io/JointAI/

License GPL (>= 2)

BugReports https://github.com/nerler/JointAI/issues/

LazyData TRUE

RoxygenNote 7.3.3

Imports rjags, mcmcse, coda, rlang, future, mathjaxr, survival, MASS

SystemRequirements JAGS (https://mcmc-jags.sourceforge.io/)

Suggests knitr, rmarkdown, bookdown, foreign, ggplot2, corrplot, ggpubr, svglite, testthat, covr

VignetteBuilder knitr

Encoding UTF-8

RdMacros mathjaxr

Config/testthat/edition 3

Language en-GB

NeedsCompilation no

Author Nicole S. Erler [aut, cre] (ORCID: <https://orcid.org/0000-0002-9370-6832>)

Maintainer Nicole S. Erler <n.s.erler@umcutrecht.nl>

Config/pak/sysreqs jags

Repository https://cran.r-universe.dev

Date/Publication 2026-01-30 10:40:07 UTC

RemoteUrl https://github.com/cran/JointAI

RemoteRef HEAD

RemoteSha 47db87e16a65ae5fe40b5ecb95bbe35fc2855333

Contents

add_samples . . . 3
auto_corr . . . 4
clean_survname . . . 5
cross_corr . . . 5
default_hyperpars . . . 6
densplot . . . 8
extract_state . . . 9
get_MIdat . . . 10
get_missinfo . . . 11
GR_crit . . . 12
internal_clean_survname . . . 13
JointAI . . . 14
JointAIObject . . . 16
list_models . . . 18
longDF . . . 19
MC_error . . . 20
md_pattern . . . 22
model_imp . . . 23
NHANES . . . 36
parameters . . . 38
PBC . . . 38
plot.JointAI . . . 40
plot_all . . . 40
plot_imp_distr . . . 41
predict.JointAI . . . 42
print.Dmat . . . 44
rd_vcov . . . 46
residuals.JointAI . . . 46
set_refcat . . . 47
sharedParams . . . 48
simLong . . . 50
sum_duration . . . 51
traceplot . . . 52
wideDF . . . 54

Index **55**

add_samples *Continue sampling from an object of class JointAI*

Description

This function continues the sampling from the MCMC chains of an existing object of class 'JointAI'.

Usage

```
add_samples(object, n.iter, add = TRUE, thin = NULL,
  monitor_params = NULL, progress.bar = "text", mess = TRUE)
```

Arguments

object	object inheriting from class 'JointAI'
n.iter	the number of additional iterations of the MCMC chain
add	logical; should the new MCMC samples be added to the existing samples (TRUE; default) or replace them? If samples are added the arguments monitor_params and thin are ignored.
thin	thinning interval (see window.mcmc); ignored when add = TRUE.
monitor_params	named list or vector specifying which parameters should be monitored. For details, see *_imp and the vignette Parameter Selection. Ignored when add = TRUE.
progress.bar	character string specifying the type of progress bar. Possible values are "text" (default), "gui", and "none" (see update). Note: when sampling is performed in parallel it is not possible to display a progress bar.
mess	logical; should messages be given? Default is TRUE.

See Also

*_imp

The vignette Parameter Selection contains some examples on how to specify the argument monitor_params.

Examples

```
# Example 1:
# Run an initial JointAI model:
mod <- lm_imp(y ~ C1 + C2, data = wideDF, n.iter = 100)

# Continue sampling:
mod_add <- add_samples(mod, n.iter = 200, add = TRUE)


# Example 2:
```

```
# Continue sampling, but additionally sample imputed values.
# Note: Setting different parameters to monitor than in the original model
# requires add = FALSE.
imps <- add_samples(mod, n.iter = 200, monitor_params = c("imps" = TRUE),
                    add = FALSE)
```

---

auto_corr *Autocorrelation of MCMC samples*

Description

This function obtains the autocorrelation of the MCMC samples in an JointAI object via `coda::autocorr.diag()`. `autocorr_plot()` visualizes the results using **ggplot2**.

Usage

```
auto_corr(object, lags = 0:30, by_chain = TRUE, outcome = 1L,
  start = NULL, end = NULL, thin = NULL)

auto_corr_plot(object, lags = 0:30, by_chain = TRUE, outcome = 1L,
  start = NULL, end = NULL, thin = NULL)
```

Arguments

`object`	an object of class JointAI
`lags`	a numeric vector indicating the lags to consider
`by_chain`	logical; should the autocorrelation be computed for each chain separately?
`outcome`	integer; index of the outcome model for which the autocorrelation should be plotted
`start`	the first iteration of interest (see `window.mcmc`)
`end`	the last iteration of interest (see `window.mcmc`)
`thin`	thinning interval (integer; see `window.mcmc`). For example, `thin = 1` (default) will keep the MCMC samples from all iterations; `thin = 5` would only keep every 5th iteration.

Value

a `matrix` or a `list` of `matrix` objects if `by_chain = TRUE`, or a `ggplot()` object for `autocorr_plot()`.

Examples

```
fit <- lm_imp(y ~ C1 + C2 + B2, data = wideDF, n.iter = 200)
auto_corr(fit)
auto_corr_plot(fit)
```

clean_survname *Convert a survival outcome to a model name*

Description

A helper function that converts the "name of a survival model" (the "Surv(time, status)" specification) into a valid variable name so that it can be used in the JAGS model syntax.

Usage

```
clean_survname(x)
```

Arguments

x	a character string or vector of character strings

cross_corr *Cross-correlation of MCMC samples*

Description

These functions compute the cross-correlations of the MCMC samples in an JointAI object via coda::crosscorr() and plot them using either the **corrplot** package or coda::crosscorr.plot().

Usage

```
cross_corr(object, outcome = 1L, start = NULL, end = NULL, thin = NULL)

cross_corr_plot(object, outcome = 1L, start = NULL, end = NULL,
  thin = NULL, type = "corrplot")
```

Arguments

object	an object of class JointAI
outcome	integer; index of the outcome model for which the correlations should be plotted
start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
type	character; type of plot to be produced. Either "corrplot" (default) or "coda".

Value

a matrix (or a plot)

Examples

```
fit <- lm_imp(y ~ C1 + C2 + B2, data = wideDF, n.iter = 200)
cross_corr(fit)
cross_corr_plot(fit, type = "coda")
```

`default_hyperpars` *Get the default values for hyper-parameters*

Description

This function returns a list of default values for the hyper-parameters.

Usage

```
default_hyperpars()
```

Details

norm: hyper-parameters for normal and log-normal models

`mu_reg_norm`	mean in the priors for regression coefficients
`tau_reg_norm`	precision in the priors for regression coefficients
`shape_tau_norm`	shape parameter in Gamma prior for the precision of the (log-)normal distribution
`rate_tau_norm`	rate parameter in Gamma prior for the precision of the (log-)normal distribution

gamma: hyper-parameters for Gamma models

`mu_reg_gamma`	mean in the priors for regression coefficients
`tau_reg_gamma`	precision in the priors for regression coefficients
`shape_tau_gamma`	shape parameter in Gamma prior for the precision of the Gamma distribution
`rate_tau_gamma`	rate parameter in Gamma prior for the precision of the Gamma distribution

beta: hyper-parameters for beta models

`mu_reg_beta`	mean in the priors for regression coefficients
`tau_reg_beta`	precision in the priors for regression coefficients
`shape_tau_beta`	shape parameter in Gamma prior for the precision of the beta distribution
`rate_tau_beta`	rate parameter in Gamma prior for precision of the of the beta distribution

binom: hyper-parameters for binomial models

`mu_reg_binom`	mean in the priors for regression coefficients
`tau_reg_binom`	precision in the priors for regression coefficients

poisson: hyper-parameters for poisson models

mu_reg_poisson	mean in the priors for regression coefficients
tau_reg_poisson	precision in the priors for regression coefficients

multinomial: hyper-parameters for multinomial models

mu_reg_multinomial	mean in the priors for regression coefficients
tau_reg_multinomial	precision in the priors for regression coefficients

ordinal: hyper-parameters for ordinal models

mu_reg_ordinal	mean in the priors for regression coefficients
tau_reg_ordinal	precision in the priors for regression coefficients
mu_delta_ordinal	mean in the prior for the intercepts
tau_delta_ordinal	precision in the priors for the intercepts

ranef: hyper-parameters for the random effects variance-covariance matrices (when there is only one random effect a Gamma distribution is used instead of the Wishart distribution)

shape_diag_RinvD	shape parameter in Gamma prior for the diagonal elements of RinvD
rate_diag_RinvD	rate parameter in Gamma prior for the diagonal elements of RinvD
KinvD_expr	a character string that can be evaluated to calculate the number of degrees of freedom in the Wishart dist

surv: parameters for survival models (survreg, coxph and JM)

mu_reg_surv	mean in the priors for regression coefficients
tau_reg_surv	precision in the priors for regression coefficients

Note

From the JAGS user manual on the specification of the Wishart distribution:
For KinvD larger than the dimension of the variance-covariance matrix the prior on the correlation between the random effects is concentrated around 0, so that larger values of KinvD indicate stronger prior belief that the elements of the multivariate normal distribution are independent. For KinvD equal to the number of random effects the Wishart prior puts most weight on the extreme values (correlation 1 or -1).

Examples

```
default_hyperpars()

# To change the hyper-parameters:
hyp <- default_hyperpars()
hyp$norm['rate_tau_norm'] <- 1e-3
mod <- lm_imp(y ~ C1 + C2 + B1, data = wideDF, hyperpars = hyp, mess = FALSE)
```

densplot *Plot the posterior density from object of class JointAI*

Description

The function plots a set of densities (per chain and coefficient) from the MCMC sample of an object of class "JointAI".

Usage

```
densplot(object, ...)

## S3 method for class 'JointAI'
densplot(object, start = NULL, end = NULL, thin = NULL,
  subset = c(analysis_main = TRUE), outcome = NULL,
  exclude_chains = NULL, vlines = NULL, nrow = NULL, ncol = NULL,
  joined = FALSE, use_ggplot = FALSE, warn = TRUE, mess = TRUE, ...)
```

Arguments

object	object inheriting from class 'JointAI'
...	additional parameters passed to plot()
start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
subset	subset of parameters/variables/nodes (columns in the MCMC sample). Follows the same principle as the argument monitor_params in *_imp.
outcome	optional; vector identifying a subset of sub-models included in the output, either by specifying their indices (using the order used in the list of model formulas), or their names (LHS of the respective model formula as character string)
exclude_chains	optional vector of the index numbers of chains that should be excluded
vlines	list, where each element is a named list of parameters that can be passed to graphics::abline() to create vertical lines. Each of the list elements needs to contain at least v = <x location> where <x location> is a vector of the same length as the number of plots (see examples).
nrow	optional; number of rows in the plot layout; automatically chosen if unspecified
ncol	optional; number of columns in the plot layout; automatically chosen if unspecified
joined	logical; should the chains be combined before plotting?
use_ggplot	logical; Should ggplot be used instead of the base graphics?
warn	logical; should warnings be given? Default is TRUE.
mess	logical; should messages be given? Default is TRUE.

See Also

The vignette Parameter Selection contains some examples how to specify the argument `subset`.

Examples

```
## Not run:
# fit a JointAI object:
mod <- lm_imp(y ~ C1 + C2 + M1, data = wideDF, n.iter = 100)

# Example 1: basic densityplot
densplot(mod)
densplot(mod, exclude_chains = 2)


# Example 2: use vlines to mark zero
densplot(mod, col = c("darkred", "darkblue", "darkgreen"),
         vlines = list(list(v = rep(0, nrow(summary(mod)$res$y$regcoef)),
                            col = grey(0.8))))


# Example 3: use vlines to visualize posterior mean and 2.5%/97.5% quantiles
res <- rbind(summary(mod)$res$y$regcoef[, c('Mean', '2.5%', '97.5%')],
             summary(mod)$res$y$sigma[, c('Mean', '2.5%', '97.5%'),
             drop = FALSE]
             )
densplot(mod, vlines = list(list(v = res[, "Mean"], lty = 1, lwd = 2),
                            list(v = res[, "2.5%"], lty = 2),
                            list(v = res[, "97.5%"], lty = 2)))


# Example 4: ggplot version
densplot(mod, use_ggplot = TRUE)


# Example 5: change how the ggplot version looks
library(ggplot2)

densplot(mod, use_ggplot = TRUE) +
  xlab("value") +
  theme(legend.position = 'bottom') +
  scale_color_brewer(palette = 'Dark2', name = 'chain')

## End(Not run)
```

`extract_state` *Return the current state of a 'JointAI' model*

Description

Return the current state of a 'JointAI' model

Usage

```
extract_state(object, pattern = paste0("^", c("RinvD", "invD", "tau", "b"),
  "_"))
```

Arguments

object	an object of class 'JointAI'
pattern	vector of patterns to be matched with the names of the nodes

Value

A list with one element per chain of the MCMC sampler, containing the Returns the current state of the MCMC sampler (values of the last iteration) for the subset of nodes identified based on the pattern the user has specified.

---

get_MIdat *Extract multiple imputed datasets from an object of class JointAI*

Description

This function returns a dataset containing multiple imputed datasets stacked onto each other (i.e., long format; optionally including the original, incomplete data).
These data can be automatically exported to SPSS (as a .txt file containing the data and a .sps file containing syntax to generate a .sav file). For the export function the **foreign** package needs to be installed.

Usage

```
get_MIdat(object, m = 10, include = TRUE, start = NULL, minspace = 50,
  seed = NULL, export_to_SPSS = FALSE, resdir = NULL, filename = NULL)
```

Arguments

object	object inheriting from class 'JointAI'
m	number of imputed datasets
include	should the original, incomplete data be included? Default is TRUE.
start	the first iteration of interest (see window.mcmc)
minspace	minimum number of iterations between iterations to be chosen as imputed values (to prevent strong correlation between imputed datasets in the case of high autocorrelation of the MCMC chains).
seed	optional seed value
export_to_SPSS	logical; should the completed data be exported to SPSS?
resdir	optional; directory for results. If unspecified and export_to_SPSS = TRUE the current working directory is used.
filename	optional; file name (without ending). If unspecified and export_to_SPSS = TRUE a name is generated automatically.

Value

A data.frame in which the original data (if include = TRUE) and the imputed datasets are stacked onto each other.
The variable Imputation_ indexes the imputation, while .rownr links the rows to the rows of the original data. In cross-sectional datasets the variable .id is added as subject identifier.

Note

In order to be able to extract (multiple) imputed datasets the imputed values must have been monitored, i.e., imps = TRUE had to be specified in the argument monitor_params in *_imp.

See Also

plot_imp_distr

Examples

```
## Not run:
# fit a model and monitor the imputed values with
# monitor_params = c(imps = TRUE)

mod <- lm_imp(y ~ C1 + C2 + M2, data = wideDF,
              monitor_params = c(imps = TRUE), n.iter = 100)

# Example 1: without export to SPSS
MIs <- get_MIdat(mod, m = 3, seed = 123)



# Example 2: with export for SPSS
# (here: to the temporary directory "temp_dir")

temp_dir <- tempdir()
MIs <- get_MIdat(mod, m = 3, seed = 123, resdir = temp_dir,
                 filename = "example_imputation",
                 export_to_SPSS = TRUE)


## End(Not run)
```

---

get_missinfo *Obtain a summary of the missing values involved in an object of class JointAI*

Description

This function returns a data.frame or a list of data.frames per grouping level. Each of the data.frames has columns variable, #NA (number of missing values) and %NA (proportion of missing values in percent).

Usage

```
get_missinfo(object)
```

Arguments

object	object inheriting from class JointAI

Examples

```
mod <-  lm_imp(y ~ C1 + B2 + C2, data = wideDF, n.iter = 100)
get_missinfo(mod)
```

GR_crit *Gelman-Rubin criterion for convergence*

Description

Calculates the Gelman-Rubin criterion for convergence (uses `gelman.diag` from package **coda**).

Usage

```
GR_crit(object, confidence = 0.95, transform = FALSE, autoburnin = TRUE,
  multivariate = TRUE, subset = NULL, exclude_chains = NULL,
  start = NULL, end = NULL, thin = NULL, warn = TRUE, mess = TRUE,
  ...)
```

Arguments

object	object inheriting from class 'JointAI'
confidence	the coverage probability of the confidence interval for the potential scale reduction factor
transform	a logical flag indicating whether variables in `x` should be transformed to improve the normality of the distribution. If set to TRUE, a log transform or logit transform, as appropriate, will be applied.
autoburnin	a logical flag indicating whether only the second half of the series should be used in the computation. If set to TRUE (default) and `start(x)` is less than `end(x)/2` then start of series will be adjusted so that only second half of series is used.
multivariate	a logical flag indicating whether the multivariate potential scale reduction factor should be calculated for multivariate chains
subset	subset of parameters/variables/nodes (columns in the MCMC sample). Follows the same principle as the argument `monitor_params` in `*_imp`.
exclude_chains	optional vector of the index numbers of chains that should be excluded

start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
warn	logical; should warnings be given? Default is TRUE.
mess	logical; should messages be given? Default is TRUE.
...	currently not used

References

Gelman, A and Rubin, DB (1992) Inference from iterative simulation using multiple sequences, *Statistical Science*, **7**, 457-511.

Brooks, SP. and Gelman, A. (1998) General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, **7**, 434-455.

See Also

The vignette Parameter Selection contains some examples how to specify the argument subset.

Examples

```
mod1 <- lm_imp(y ~ C1 + C2 + M2, data = wideDF, n.iter = 100)
GR_crit(mod1)
```

---

internal_clean_survname

Convert a survival outcome to a model name

Description

A helper function that converts the "name of a survival model" (the "Surv(time, status)" specification) into a valid variable name so that it can be used in the JAGS model syntax.

Usage

```
internal_clean_survname(x)
```

Arguments

x	a character string or vector of character strings

JointAI *JointAI: Joint Analysis and Imputation of Incomplete Data*

Description

The **JointAI** package performs simultaneous imputation and inference for incomplete or complete data under the Bayesian framework. Models for incomplete covariates, conditional on other covariates, are specified automatically and modelled jointly with the analysis model. MCMC sampling is performed in 'JAGS' via the R package **rjags**.

Main functions

JointAI provides the following main functions that facilitate analysis with different models:

- `lm_imp` for linear regression
- `glm_imp` for generalized linear regression
- `betareg_imp` for regression using a beta distribution
- `lognorm_imp` for regression using a log-normal distribution
- `clm_imp` for (ordinal) cumulative logit models
- `mlogit_imp` for multinomial models
- `lme_imp` or `lmer_imp` for linear mixed models
- `glme_imp` or `glmer_imp` for generalized linear mixed models
- `betamm_imp` for mixed models using a beta distribution
- `lognormmm_imp` for mixed models using a log-normal distribution
- `clmm_imp` for (ordinal) cumulative logit mixed models
- `survreg_imp` for parametric (Weibull) survival models
- `coxph_imp` for (Cox) proportional hazard models
- `JM_imp` for joint models of longitudinal and survival data

As far as possible, the specification of these functions is analogous to the specification of widely used functions for the analysis of complete data, such as `lm`, `glm`, `lme` (from the package **nlme**), `survreg` (from the package **survival**) and `coxph` (from the package **survival**).

Computations can be performed in parallel to reduce computational time, using the package **future**, the argument `shrinkage` allows the user to impose a penalty on the regression coefficients of some or all models involved, and hyper-parameters can be changed via the argument `hyperpars`.

To obtain summaries of the results, the functions `summary()`, `coef()` and `confint()` are available, and results can be visualized with the help of `traceplot()` or `densplot()`.

The function `predict()` allows prediction (including credible intervals) from `JointAI` models.

Evaluation and export

Two criteria for evaluation of convergence and precision of the posterior estimate are available:

- `GR_crit` implements the Gelman-Rubin criterion ('potential scale reduction factor') for convergence
- `MC_error` calculates the Monte Carlo error to evaluate the precision of the MCMC sample

Imputed data can be extracted (and exported to SPSS) using `get_MIdat()`. The function `plot_imp_distr()` allows visual comparison of the distribution of observed and imputed values.

Other useful functions

- `parameters` and `list_models` to gain insight in the specified model
- `plot_all` and `md_pattern` to visualize the distribution of the data and the missing data pattern

Vignettes

The following vignettes are available

- *Minimal Example*:
  A minimal example demonstrating the use of `lm_imp`, `summary.JointAI`, `traceplot` and `densplot`.
- *Visualizing Incomplete Data*:
  Demonstrations of the options in `plot_all` (plotting histograms and bar plots for all variables in the data) and `md_pattern` (plotting or printing the missing data pattern).
- *Model Specification*:
  Explanation and demonstration of all parameters that are required or optional to specify the model structure in `lm_imp`, `glm_imp` and `lme_imp`. Among others, the functions `parameters`, `list_models` and `set_refcat` are used.
- *Parameter Selection*:
  Examples on how to select the parameters/variables/nodes to follow using the argument `monitor_params` and the parameters/variables/nodes displayed in the `summary`, `traceplot`, `densplot` or when using `GR_crit` or `MC_error`.
- *MCMC Settings*:
  Examples demonstrating how to set the arguments controlling settings of the MCMC sampling, i.e., `n.adapt`, `n.iter`, `n.chains`, `thin`, `inits`.
- *After Fitting*:
  Examples on the use of functions to be applied after the model has been fitted, including `traceplot`, `densplot`, `summary`, `GR_crit`, `MC_error`, `predict`, `predDF` and `get_MIdat`.
- *Theoretical Background*:
  Explanation of the statistical method implemented in **JointAI**.

Author(s)

Maintainer: Nicole S. Erler <n.s.erler@umcutrecht.nl> (ORCID)

References

Erler NS, Rizopoulos D, Lesaffre EMEH (2021). "JointAI: Joint Analysis and Imputation of Incomplete Data in R." *Journal of Statistical Software*, *100*(20), 1-56. doi:10.18637/jss.v100.i20.

Erler, N.S., Rizopoulos, D., Rosmalen, J., Jaddoe, V.W.V., Franco, O. H., & Lesaffre, E.M.E.H. (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

Erler, N.S., Rizopoulos D., Jaddoe, V.W.V., Franco, O.H. & Lesaffre, E.M.E.H. (2019). Bayesian imputation of time-varying covariates in linear mixed models. *Statistical Methods in Medical Research*, 28(2), 555–568. doi:10.1177/0962280217730851

See Also

Useful links:

- https://nerler.github.io/JointAI/
- Report bugs at https://github.com/nerler/JointAI/issues/

`JointAIObject` *Fitted object of class 'JointAI'*

Description

An object returned by one of the main functions `*_imp`.

Value

`analysis_type`	`lm`, `glm`, `clm`, `lme`, `glme`, `clmm`, `survreg` or `coxph` (with attributes `family` and `link` for GLM-type models
`formula`	The formula used in the (analysis) model.
`data`	original (incomplete, but pre-processed) data
`models`	named vector specifying the the types of all sub-models
`fixed`	a list of the fixed effects formulas of the sub-model(s) for which the use had specified a formula
`random`	a list of the random effects formulas of the sub-model(s) for which the use had specified a formula
`Mlist`	a list (for internal use) containing the data and information extracted from the data and model formulas, split up into • a named vector identifying the levels (in the hierarchy) of all variables (`Mlvls`) • a vector of the id variables that were extracted from the random effects formulas (`idvar`) • a list of grouping information for each grouping level of the data (`groups`)

- a named vector identifying the hierarchy of the grouping levels (`group_lvls`)
- a named vector giving the number of observations on each level of the hierarchy (`N`)
- the name of the time variable (only for survival models with time-varying covariates) (`timevar`)
- a formula of auxiliary variables (`auxvars`)
- a list specifying the reference categories and dummy variables for all factors involved in the models (`refs`)
- a list of linear predictor information (column numbers per design matrix) for all sub-models (`lp_cols`)
- a list identifying information for interaction terms found in the model formulas (`interactions`)
- a `data.frame` containing information on transformations of incomplete variables (`trafos`)
- a `data.frame` containing information on transformations of all variables (`fcts_all`)
- a logical indicator if parameter for posterior predictive checks should be monitored (`ppc`; not yet used)
- a vector specifying if shrinkage of regression coefficients should be performed, and if so for which models and what type of shrinkage (`shrinkage`)
- the number of degrees of freedom to be used in the spline specification of the baseline hazard in proportional hazards survival models (`df_basehaz`)
- a list of matrices, one per level of the data, specifying centring and scaling parameters for the data (`scale_pars`)
- a list containing information on the outcomes (mostly relevant for survival outcomes; `outcomes`)
- a list of terms objects, needed to be able to build correct design matrices for the Gauss-Kronrod quadrature when, for example, splines are used to model time in a joint model (`terms_list`)

`par_index_main` a list of matrices specifying the indices of the regression coefficients for each of the main models per design matrix

`par_index_other`
a list of matrices specifying the indices of regression coefficients for each covariate model per design matrix

`jagsmodel` The JAGS model as character string.

`mcmc_settings` a list containing MCMC sampling related information with elements

`modelfile`: path and name of the JAGS model file

`n.chains`: number of MCMC chains

`n.adapt`: number of iterations in the adaptive phase

`n.iter`: number of iterations in the MCMC sample

`variable.names`: monitored nodes

`thin`: thinning interval of the MCMC sample

`inits`: a list containing the initial values that were passed to **rjags**

`monitor_params` the named list of parameter groups to be monitored

data_list	list with data that was passed to **rjags**
hyperpars	a list containing the values of the hyper-parameters used
info_list	a list with information used to write the imputation model syntax
coef_list	a list relating the regression coefficient vectors used in the JAGS model to the names of the corresponding covariates
model	the JAGS model (an object of class 'jags', created by **rjags**)
sample	MCMC sample on the sampling scale (included only if keep_scaled_sample = TRUE)
MCMC	MCMC sample, scaled back to the scale of the data
comp_info	a list with information on the computational setting (start_time: date and time the calculation was started, duration: computational time of the model adaptive and sampling phase, JointAI_version: package version, R_version: the R.version.string, parallel: whether parallel computation was used, workers: if parallel computation was used, the number of workers)
fitted.values	fitted/predicted values (if available)
residuals	residuals (if available)
call	the original call

---

list_models *List model details*

Description

This function prints information on all models, those explicitly specified by the user and those specified automatically by JointAI for (incomplete) covariates in a JointAI object.

Usage

```
list_models(object, predvars = TRUE, regcoef = TRUE, otherpars = TRUE,
  priors = TRUE, refcat = TRUE)
```

Arguments

object	object inheriting from class 'JointAI'
predvars	logical; should information on the predictor variables be printed? (default is TRUE)
regcoef	logical; should information on the regression coefficients be printed? (default is TRUE)
otherpars	logical; should information on other parameters be printed? (default is TRUE)
priors	logical; should information on the priors (and hyper-parameters) be printed? (default is TRUE)
refcat	logical; should information on the reference category be printed? (default is TRUE)

Note

The models listed by this function are not the actual imputation models, but the conditional models that are part of the specification of the joint distribution. Briefly, the joint distribution is specified as a sequence of conditional models

$$p(y|x_1, x_2, x_3, ..., \theta)p(x_1|x_2, x_3, ..., \theta)p(x_2|x_3, ..., \theta)...$$

The actual imputation models are the full conditional distributions $p(x_1|\cdot)$ derived from this joint distribution. Even though the conditional distributions do not contain the outcome and all other covariates in their linear predictor, outcome and other covariates are taken into account implicitly, since imputations are sampled from the full conditional distributions. For more details, see Erler et al. (2016) and Erler et al. (2019).

The function `list_models` prints information on the conditional distributions of the covariates (since they are what is specified; the full-conditionals are automatically derived within JAGS). The outcome is, thus, not part of the printed linear predictor, but is still included during imputation.

References

Erler, N.S., Rizopoulos, D., Rosmalen, J.V., Jaddoe, V.W., Franco, O.H., & Lesaffre, E.M.E.H. (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.

Erler NS, Rizopoulos D, Lesaffre EMEH (2021). "JointAI: Joint Analysis and Imputation of Incomplete Data in R." *Journal of Statistical Software*, *100*(20), 1-56. doi:10.18637/jss.v100.i20.

Examples

```
# (set n.adapt = 0 and n.iter = 0 to prevent MCMC sampling to save time)
mod1 <- lm_imp(y ~ C1 + C2 + M2 + O2 + B2, data = wideDF, n.adapt = 0,
               n.iter = 0, mess = FALSE)

list_models(mod1)
```

---

longDF *Longitudinal example dataset*

Description

A simulated longitudinal dataset.

Usage

```
data(longDF)
```

Format

A simulated data frame with 329 rows and 21 variables with data from 100 subjects:

C1 continuous, complete baseline variable

C2 continuous, incomplete baseline variable

B1 binary, complete baseline variable

B2 binary, incomplete baseline variable

M1 unordered factor; complete baseline variable

M2 unordered factor; incomplete baseline variable

O1 ordered factor; complete baseline variable

O2 ordered factor; incomplete baseline variable

P1 count variable; complete baseline variable

P2 count variable; incomplete baseline variable

c1 continuous, complete longitudinal variable

c2 continuous incomplete longitudinal variable

b1 binary, complete longitudinal variable

b2 binary incomplete longitudinal variable

o1 ordered factor; complete longitudinal variable

o2 ordered factor; incomplete longitudinal variable

p1 count variable; complete longitudinal variable

p2 count variable; incomplete longitudinal variable

id id (grouping) variable

time continuous complete longitudinal variable

y continuous, longitudinal (outcome) variable

MC_error *Calculate and plot the Monte Carlo error*

Description

Calculate, print and plot the Monte Carlo error of the samples from a 'JointAI' model, combining the samples from all MCMC chains.

Usage

```
MC_error(x, subset = NULL, exclude_chains = NULL, start = NULL,
  end = NULL, thin = NULL, digits = 2, warn = TRUE, mess = TRUE, ...)

## S3 method for class 'MCElist'
plot(x, data_scale = TRUE, plotpars = NULL,
  ablinepars = list(v = 0.05), minlength = 20, ...)
```

Arguments

x	object inheriting from class 'JointAI'
subset	subset of parameters/variables/nodes (columns in the MCMC sample). Follows the same principle as the argument monitor_params in *_imp.
exclude_chains	optional vector of the index numbers of chains that should be excluded
start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
digits	number of digits for the printed output
warn	logical; should warnings be given? Default is TRUE.
mess	logical; should messages be given? Default is TRUE.
...	Arguments passed on to mcmcse::mcse.mat
	size represents the batch size in "bm" and the truncation point in "bartlett" and "tukey". Default is NULL which implies that an optimal batch size is calculated using the batchSize function. Can take character values of "sqroot" and "cuberoot" or any numeric value between 1 and n/2. "sqroot" means size is $\lfloor n^{1/2} \rfloor$ and "cuberoot" means size is $\lfloor n^{1/3} \rfloor$.
	g a function such that $E(g(x))$ is the quantity of interest. The default is NULL, which causes the identity function to be used.
	method any of "bm","obm","bartlett", "tukey". "bm" represents batch means estimator, "obm" represents overlapping batch means estimator with, "bartlett" and "tukey" represents the modified-Bartlett window and the Tukey-Hanning windows for spectral variance estimators.
	r The lugsail parameters (r) that converts a lag window into its lugsail equivalent. Larger values of r will typically imply less underestimation of "cov", but higher variability of the estimator. Default is r = 3 and r = 1,2 are also good choices although may lead to underestimates of the variance. r > 5 is not recommended.
data_scale	logical; show the Monte Carlo error of the sample transformed back to the scale of the data (TRUE) or on the sampling scale (this requires the argument keep_scaled_mcmc = TRUE to be set when fitting the model)
plotpars	optional; list of parameters passed to plot()
ablinepars	optional; list of parameters passed to abline()
minlength	number of characters the variable names are abbreviated to

Value

An object of class MCElist with elements unscaled, scaled and digits. The first two are matrices with columns est (posterior mean), MCSE (Monte Carlo error), SD (posterior standard deviation) and MCSE/SD (Monte Carlo error divided by post. standard deviation.)

Functions

- `plot(MCElist)`: plot Monte Carlo error

Note

Lesaffre & Lawson (2012; p. 195) suggest the Monte Carlo error of a parameter should not be more than 5% of the posterior standard deviation of this parameter (i.e., $MCSE/SD \leq 0.05$).

Long variable names:
The default plot margins may not be wide enough when variable names are longer than a few characters. The plot margin can be adjusted (globally) using the argument `"mar"` in `par`.

References

Lesaffre, E., & Lawson, A. B. (2012). *Bayesian Biostatistics*. John Wiley & Sons.

See Also

The vignette Parameter Selection provides some examples how to specify the argument `subset`.

Examples

```
## Not run:

mod <- lm_imp(y ~ C1 + C2 + M2, data = wideDF, n.iter = 100)

MC_error(mod)

plot(MC_error(mod), ablinepars = list(lty = 2),
     plotpars = list(pch = 19, col = 'blue'))

## End(Not run)
```

`md_pattern` *Missing data pattern*

Description

Obtain a plot of the pattern of missing data and/or return the pattern as a matrix.

Usage

```
md_pattern(data, color = c(grDevices::grey(0.1), grDevices::grey(0.7)),
  border = grDevices::grey(0.5), plot = TRUE, pattern = FALSE,
  print_xaxis = TRUE, ylab = "Number of observations per pattern",
  print_yaxis = TRUE, legend.position = "bottom", sort_columns = TRUE,
  ...)
```

Arguments

data	data frame
color	vector of length two, that specifies the colour used to indicate observed and missing values (in that order)
border	colour of the grid
plot	logical; should the missing data pattern be plotted? (default is TRUE)
pattern	logical; should the missing data pattern be returned as matrix? (default is FALSE)
print_xaxis, print_yaxis	logical; should the x-axis (below the plot) and y-axis (on the right) be printed?
ylab	y-axis label
legend.position	the default position of legends ("none", "left", "right", "bottom", "top", "inside")
sort_columns	logical; should the columns be sorted by number of missing values? (default is TRUE)
...	optional additional parameters, currently not used

Note

This function requires the **ggplot2** package to be installed.

See Also

See the vignette Visualizing Incomplete Data for more examples.

Examples

```
op <- par(mar = c(3, 1, 1.5, 1.5), mgp = c(2, 0.6, 0))
md_pattern(wideDF)
par(op)
```

---

model_imp *Joint Analysis and Imputation of incomplete data*

Description

Main analysis functions to estimate different types of models using MCMC sampling, while imputing missing values.

Usage

```
lm_imp(formula, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, models = NULL, no_model = NULL, shrinkage = FALSE,
  ppc = TRUE, seed = NULL, inits = NULL, warn = TRUE, mess = TRUE,
  ...)

glm_imp(formula, family, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, models = NULL, no_model = NULL, shrinkage = FALSE,
  ppc = TRUE, seed = NULL, inits = NULL, warn = TRUE, mess = TRUE,
  ...)

clm_imp(formula, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, nonprop = NULL, rev = NULL, models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

lognorm_imp(formula, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, models = NULL, no_model = NULL, shrinkage = FALSE,
  ppc = TRUE, seed = NULL, inits = NULL, warn = TRUE, mess = TRUE,
  ...)

betareg_imp(formula, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, models = NULL, no_model = NULL, shrinkage = FALSE,
  ppc = TRUE, seed = NULL, inits = NULL, warn = TRUE, mess = TRUE,
  ...)

mlogit_imp(formula, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, models = NULL, no_model = NULL, shrinkage = FALSE,
  ppc = TRUE, seed = NULL, inits = NULL, warn = TRUE, mess = TRUE,
  ...)

lme_imp(fixed, data, random, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

lmer_imp(fixed, data, random, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)
```

```
glme_imp(fixed, data, random, family, n.chains = 3, n.adapt = 100,
  n.iter = 0, thin = 1, monitor_params = c(analysis_main = TRUE),
  auxvars = NULL, refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

glmer_imp(fixed, data, random, family, n.chains = 3, n.adapt = 100,
  n.iter = 0, thin = 1, monitor_params = c(analysis_main = TRUE),
  auxvars = NULL, refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

betamm_imp(fixed, random, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

lognormmm_imp(fixed, random, data, n.chains = 3, n.adapt = 100,
  n.iter = 0, thin = 1, monitor_params = c(analysis_main = TRUE),
  auxvars = NULL, refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

clmm_imp(fixed, data, random, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, nonprop = NULL, rev = NULL, rd_vcov = "blockdiag",
  models = NULL, no_model = NULL, shrinkage = FALSE, ppc = TRUE,
  seed = NULL, inits = NULL, warn = TRUE, mess = TRUE, ...)

mlogitmm_imp(fixed, data, random, n.chains = 3, n.adapt = 100,
  n.iter = 0, thin = 1, monitor_params = c(analysis_main = TRUE),
  auxvars = NULL, refcats = NULL, rd_vcov = "blockdiag", models = NULL,
  no_model = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)

survreg_imp(formula, data, n.chains = 3, n.adapt = 100, n.iter = 0,
  thin = 1, monitor_params = c(analysis_main = TRUE), auxvars = NULL,
  refcats = NULL, models = NULL, no_model = NULL, shrinkage = FALSE,
  ppc = TRUE, seed = NULL, inits = NULL, warn = TRUE, mess = TRUE,
  ...)

coxph_imp(formula, data, df_basehaz = 6, n.chains = 3, n.adapt = 100,
  n.iter = 0, thin = 1, monitor_params = c(analysis_main = TRUE),
  auxvars = NULL, refcats = NULL, models = NULL, no_model = NULL,
  shrinkage = FALSE, ppc = TRUE, seed = NULL, inits = NULL,
  warn = TRUE, mess = TRUE, ...)
```

```
JM_imp(formula, data, df_basehaz = 6, n.chains = 3, n.adapt = 100,
  n.iter = 0, thin = 1, monitor_params = c(analysis_main = TRUE),
  auxvars = NULL, timevar = NULL, refcats = NULL,
  rd_vcov = "blockdiag", models = NULL, no_model = NULL,
  assoc_type = NULL, shrinkage = FALSE, ppc = TRUE, seed = NULL,
  inits = NULL, warn = TRUE, mess = TRUE, ...)
```

Arguments

formula	a two sided model formula (see formula) or a list of such formulas; (more details below).
data	a data.frame containing the original data (more details below)
n.chains	number of MCMC chains
n.adapt	number of iterations for adaptation of the MCMC samplers (see adapt)
n.iter	number of iterations of the MCMC chain (after adaptation; see coda.samples)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
monitor_params	named list or vector specifying which parameters should be monitored (more details below)
auxvars	optional; one-sided formula of variables that should be used as predictors in the imputation procedure (and will be imputed if necessary) but are not part of the analysis model(s). For more details with regards to the behaviour with non-linear effects see the vignette on Model Specification
refcats	optional; either one of "first", "last", "largest" (which sets the category for all categorical variables) or a named list specifying which category should be used as reference category per categorical variable. Options are the category label, the category number, or one of "first" (the first category), "last" (the last category) or "largest" (chooses the category with the most observations). Default is "first". If reference categories are specified for a subset of the categorical variables the default will be used for the remaining variables. (See also set_refcat)
models	optional; named vector specifying the types of models for (incomplete) covariates. This arguments replaces the argument meth used in earlier versions. If NULL (default) models will be determined automatically based on the class of the respective columns of data.
no_model	optional; vector of names of variables for which no model should be specified. Note that this is only possible for completely observed variables and implies the assumptions of independence between the excluded variable and the incomplete variables.
shrinkage	optional; either a character string naming the shrinkage method to be used for regression coefficients in all models or a named vector specifying the type of shrinkage to be used in the models given as names.
ppc	logical: should monitors for posterior predictive checks be set? (not yet used)

`seed` optional; seed value (for reproducibility)

`inits` optional; specification of initial values in the form of a list or a function (see `jags.model`). If omitted, starting values for the random number generator are created by **JointAI**, initial values are then generated by JAGS. It is an error to supply an initial value for an observed node.

`warn` logical; should warnings be given? Default is `TRUE`.

`mess` logical; should messages be given? Default is `TRUE`.

`...` additional, optional arguments

`trunc` named list specifying limits of truncation for the distribution of the named incomplete variables (see the vignette ModelSpecification)

`hyperpars` list of hyper-parameters, as obtained by `default_hyperpars()`

`scale_vars` named vector of (continuous) variables that will be centred and scaled (such that mean = 0 and sd = 1) when they enter a linear predictor to improve convergence of the MCMC sampling. Default is that all numeric variables and integer variables with >20 different values will be scaled. If set to `FALSE` no scaling will be done.

`custom` named list of JAGS model chunks (character strings) that replace the model for the given variable.

`append_data_list` list that will be appended to the list containing the data that is passed to **rjags** (`data_list`). This may be necessary if additional data / variables are needed for custom (covariate) models. Note: since version 1.0.7 elements of `append_data_list` will overwrite existing elements of the `data_list` with the same name.

`progress.bar` character string specifying the type of progress bar. Possible values are "text" (default), "gui", and "none" (see `update`). Note: when sampling is performed in parallel it is not possible to display a progress bar.

`quiet` logical; if `TRUE` then messages generated by **rjags** during compilation as well as the progress bar for the adaptive phase will be suppressed, (see `jags.model`)

`keep_scaled_mcmc` should the "original" MCMC sample (i.e., the scaled version returned by `coda.samples()`) be kept? (The MCMC sample that is re-scaled to the scale of the data is always kept.)

`modelname` character string specifying the name of the model file (including the ending, either .R or .txt). If unspecified a random name will be generated.

`modeldir` directory containing the model file or directory in which the model file should be written. If unspecified a temporary directory will be created.

`overwrite` logical; whether an existing model file with the specified `<modeldir>/<modelname>` should be overwritten. If set to `FALSE` and a model already exists, that model will be used. If unspecified (`NULL`) and a file exists, the user is asked for input on how to proceed.

`keep_model` logical; whether the created JAGS model file should be saved or removed from (`FALSE`; default) when the sampling has finished.

`family` only for `glm_imp` and `glmm_imp`/`glmer_imp`: a description of the distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (For more details see below and `family`.)

nonprop	optional named list of one-sided formulas specifying covariates that have non-proportional effects in cumulative logit models. These covariates should also be part of the regular model formula, and the names of the list should be the names of the ordinal response variables.
rev	optional character vector; vector of ordinal outcome variable names for which the odds should be reversed, i.e., $logit(y \leq k)$ instead of $logit(y > k)$.
fixed	a two sided formula describing the fixed-effects part of the model (see formula)
random	only for multi-level models: a one-sided formula of the form ~x1 + ... + xn | g, where x1 + ... + xn specifies the model for the random effects and g the grouping variable
rd_vcov	character string or list specifying the structure of the random effects variance covariance matrix, see details below.
df_basehaz	degrees of freedom for the B-spline used to model the baseline hazard in proportional hazards models (coxph_imp and JM_imp)
timevar	name of the variable indicating the time of the measurement of a time-varying covariate in a proportional hazards survival model (also in a joint model). The variable specified in "timevar" will automatically be added to "no_model".
assoc_type	named vector specifying the type of the association used for a time-varying covariate in the linear predictor of the survival model when using a "JM" model. Implemented options are "underl.value" (linear predictor; default for covariates modelled using a Gaussian, Gamma, beta or log-normal distribution) covariates) and "obs.value" (the observed/imputed value; default for covariates modelled using other distributions).

Value

An object of class JointAI.

Model formulas

Random effects:

It is possible to specify multi-level models as it is done in the package **nlme**, using `fixed` and `random`, or as it is done in the package **lme4**, using `formula` and specifying the random effects in brackets:

```
formula = y ~ x1 + x2 + x3 + (1 | id)
```

is equivalent to

```
fixed = y ~ x1 + x2 + x3, random = ~ 1|id
```

Multiple levels of grouping:

For multiple levels of grouping the specification using `formula` should be used. There is no distinction between nested and crossed random effects, i.e., `... + (1 | id) + (1 | center)` is treated the same as `... + (1 | center/id)`.

Nested vs crossed random effects:

The distinction between nested and crossed random effects should come from the levels of the grouping variables, i.e., if `id` is nested in `center`, then there cannot be observations with the same `id` but different values for `center`.

Modelling multiple models simultaneously & joint models:

To fit multiple main models at the same time, a `list` of `formula` objects can be passed to the argument `formula`. Outcomes of one model may be contained as covariates in another model and it is possible to combine models for variables on different levels, for example:

```
formula = list(y ~ x1 + x2 + x3 + x4 + time + (time | id),
                    x2 ~ x3 + x4 + x5)
```

This principle is also used for the specification of a joint model for longitudinal and survival data. Note that it is not possible to specify multiple models for the same outcome variable.

Random effects variance-covariance structure:

(Note: This feature is new and has not been fully tested yet.)

By default, a block-diagonal structure is assumed for the variance-covariance matrices of the random effects in models with random effects. This means that per outcome and level random effects are assumed to be correlated, but random effects of different outcomes are modelled as independent. The argument `rd_vcov` allows the user specify different assumptions about these variance-covariance matrices. Implemented structures are `full`, `blockdiag` and `indep` (all off-diagonal elements are zero).

If `rd_vcov` is set to one of these options, the structure is assumed for all random effects variance-covariance matrices. Alternatively, it is possible to specify a named list of vectors, where the names are the structures and the vectors contain the names of the response variables which are included in this structure.

For example, for a multivariate mixed model with five outcomes `y1`, ..., `y5`, the specification could be:

```
rd_vcov = list(blockdiag = c("y1", "y2"),
               full = c("y3", "y4"),
               indep = "y5")
```

This would entail that the random effects for `y3` and `y4` are assumed to be correlated (within and across outcomes), random effects for `y1` and `y2` are assumed to be correlated within each outcome, and the random effects for `y5` are assumed to be independent.

It is possible to have multiple sets of response variables for which separate full variance-covariance matrices are used, for example:

```
rd_vcov = list(full = c("y1", "y2", "y5"),
               full = c("y3", "y4"))
```

In models with multiple levels of nesting, separate structures can be specified per level:

```
rd_vcov = list(id = list(blockdiag = c("y1", "y2"),
                         full = c("y3", "y4"),
                         indep = "y5"),
              center = "indep")
```

Survival models with frailties or time-varying covariates:

Random effects specified in brackets can also be used to indicate a multi-level structure in survival models, as would, for instance be needed in a multi-centre setting, where patients are from multiple hospitals.

It also allows to model time-dependent covariates in a proportional hazards survival model (using `coxph_imp`), also in combination with additional grouping levels.

In time-dependent proportional hazards models, last-observation-carried-forward is used to fill in missing values in the time-varying covariates, and to determine the value of the covariate at the event time. Preferably, all time-varying covariates should be measured at baseline (`timevar = 0`). If a value for a time-varying covariate needs to be filled in and there is no previous observation, the next observation will be carried backward.

Differences to basic regression models:

It is not possible to specify transformations of outcome variables, i.e., it is not possible to use a model formula like

```
log(y) ~ x1 + x2 + ...
```

In the specific case of a transformation with the natural logarithm, a log-normal model can be used instead of a normal model.

Moreover, it is not possible to use `.` to indicate that all variables in a `data.frame` other than the outcome variable should be used as covariates. I.e., a formula `y ~ .` is not valid in **JointAI**.

Data structure

For multi-level settings, the data must be in long format, so that repeated measurements are recorded in separate rows.

For survival data with time-varying covariates (`coxph_imp` and `JM_imp`) the data should also be in long format. The survival/censoring times and event indicator variables must be stored in separate variables in the same data and should be constant across all rows referring to the same subject.

During the pre-processing of the data the survival/censoring times will automatically be merged with the observation times of the time-varying covariates (which must be supplied via the argument `timevar`).

It is possible to have multiple time-varying covariates, which do not have to be measured at the same time points, but there can only be one `timevar`.

Distribution families and link functions

`gaussian`	with links: `identity`, `log`
`binomial`	with links: `logit`, `probit`, `log`, `cloglog`
`Gamma`	with links: `inverse`, `identity`, `log`
`poisson`	with links: `log`, `identity`

Imputation methods / model types

Implemented model types that can be chosen in the argument `models` for baseline covariates (not repeatedly measured) are:

`lm`	linear (normal) model with identity link (alternatively: `glm_gaussian_identity`); default for con

glm_gaussian_log	linear (normal) model with log link
glm_gaussian_inverse	linear (normal) model with inverse link
glm_logit	logistic model for binary data (alternatively: glm_binomial_logit); default for binary variables
glm_probit	probit model for binary data (alternatively: glm_binomial_probit)
glm_binomial_log	binomial model with log link
glm_binomial_cloglog	binomial model with complementary log-log link
glm_gamma_inverse	gamma model with inverse link for skewed continuous data
glm_gamma_identity	gamma model with identity link for skewed continuous data
glm_gamma_log	gamma model with log link for skewed continuous data
glm_poisson_log	Poisson model with log link for count data
glm_poisson_identity	Poisson model with identity link for count data
lognorm	log-normal model for skewed continuous data
beta	beta model (with logit link) for skewed continuous data in (0, 1)
mlogit	multinomial logit model for unordered categorical variables; default for unordered factors with >2
clm	cumulative logit model for ordered categorical variables; default for ordered factors

For repeatedly measured variables the following model types are available:

lmm	linear (normal) mixed model with identity link (alternatively: glmm_gaussian_identity); defau
glmm_gaussian_log	linear (normal) mixed model with log link
glmm_gaussian_inverse	linear (normal) mixed model with inverse link
glmm_logit	logistic mixed model for binary data (alternatively: glmm_binomial_logit); default for binary v
glmm_probit	probit model for binary data (alternatively: glmm_binomial_probit)
glmm_binomial_log	binomial mixed model with log link
glmm_binomial_cloglog	binomial mixed model with complementary log-log link
glmm_gamma_inverse	gamma mixed model with inverse link for skewed continuous data
glmm_gamma_identity	gamma mixed model with identity link for skewed continuous data
glmm_gamma_log	gamma mixed model with log link for skewed continuous data
glmm_poisson_log	Poisson mixed model with log link for count data
glmm_poisson_identity	Poisson mixed model with identity link for count data
glmm_lognorm	log-normal mixed model for skewed covariates
glmm_beta	beta mixed model for continuous data in (0, 1)
mlogitmm	multinomial logit mixed model for unordered categorical variables; default for unordered factors
clmm	cumulative logit mixed model for ordered factors; default for ordered factors

When models are specified for only a subset of the variables for which a model is needed, the default model choices (as indicated in the tables) are used for the unspecified variables.

Parameters to follow (monitor_params)

See also the vignette: Parameter Selection

Named vector specifying which parameters should be monitored. This can be done either directly by specifying the name of the parameter or indirectly by one of the key words selecting a set of parameters. Except for other, in which parameter names are specified directly, parameter (groups) are just set as TRUE or FALSE.

Models are divided into two groups, the main models, which are the models for which the user has explicitly specified a formula (via `formula` or `fixed`), and all other models, for which models were specified automatically.

If left unspecified, `monitor_params = c("analysis_main" = TRUE)` will be used.

name/key word	what is monitored
`analysis_main`	`betas` and `sigma_main`, `tau_main` (for beta regression) or `shape_main` (for parametric survival models)
`analysis_random`	`ranef_main`, `D_main`, `invD_main`, `RinvD_main`
`other_models`	`alphas`, `tau_other`, `gamma_other`, `delta_other`
`imps`	imputed values
`betas`	regression coefficients of the main analysis model
`tau_main`	precision of the residuals from the main analysis model(s)
`sigma_main`	standard deviation of the residuals from the main analysis model(s)
`gamma_main`	intercepts in ordinal main model(s)
`delta_main`	increments of ordinal main model(s)
`ranef_main`	random effects from the main analysis model(s) `b`
`D_main`	covariance matrix of the random effects from the main model(s)
`invD_main`	inverse(s) of `D_main`
`RinvD_main`	matrices in the priors for `invD_main`
`alphas`	regression coefficients in the covariate models
`tau_other`	precision parameters of the residuals from covariate models
`gamma_other`	intercepts in ordinal covariate models
`delta_other`	increments of ordinal intercepts
`ranef_other`	random effects from the other models `b`
`D_other`	covariance matrix of the random effects from the other models
`invD_other`	inverses of `D_other`
`RinvD_other`	matrices in the priors for `invD_other`
`other`	additional parameters

For example:
`monitor_params = c(analysis_main = TRUE, tau_main = TRUE, sigma_main = FALSE)` would monitor the regression parameters `betas` and the residual precision `tau_main` instead of the residual standard deviation `sigma_main`.

For a linear model, `monitor_params = c(imps = TRUE)` would monitor `betas`, and `sigma_main` (because `analysis_main = TRUE` by default) as well as the imputed values.

Cumulative logit (mixed) models

In the default setting for cumulative logit models, i.e, `rev = NULL`, the odds for a variable y with K ordered categories are defined as

$$\log\left(\frac{P(y_i > k)}{P(y_i \leq k)}\right) = \gamma_k + \eta_i, \quad k = 1, \ldots, K-1,$$

where γ_k is a category specific intercept and η_i the subject specific linear predictor.

To reverse the odds to

$$\log\left(\frac{P(y_i \leq k)}{P(y_i > k)}\right) = \gamma_k + \eta_i, \quad k = 1, \ldots, K-1,$$

the name of the response variable has to be specified in the argument `rev`, e.g., `rev = c("y")`.

By default, proportional odds are assumed and only the intercepts differ per category of the ordinal response. To allow for non-proportional odds, i.e.,

$$\log\left(\frac{P(y_i > k)}{P(y_i \leq k)}\right) = \gamma_k + \eta_i + \eta_{ki}, \quad k = 1, \ldots, K-1,$$

the argument `nonprop` can be specified. It takes a one-sided formula or a list of one-sided formulas. When a single formula is supplied, or a unnamed list with just one element, it is assumed that the formula corresponds to the main model. To specify non-proportional effects for linear predictors in models for ordinal covariates, the list has to be named with the names of the ordinal response variables.

For example, the following three specifications are equivalent and assume a non-proportional effect of `C1` on `O1`, but `C1` is assumed to have a proportional effect on the incomplete ordinal covariate `O2`:

```
clm_imp(O1 ~ C1 + C2 + B2 + O2, data = wideDF, nonprop = ~ C1)
clm_imp(O1 ~ C1 + C2 + B2 + O2, data = wideDF, nonprop = list(~ C1))
clm_imp(O1 ~ C1 + C2 + B2 + O2, data = wideDF, nonprop = list(O1 = ~ C1))
```

To specify non-proportional effects on `O2`, a named list has to be provided:

```
clm_imp(O1 ~ C1 + C2 + B2 + O2 + B1, data = wideDF,
        nonprop = list(O1 = ~ C1,
                       O2 = ~ C1 + B1))
```

The variables for which a non-proportional effect is assumed also have to be part of the regular model formula.

Custom model parts

(Note: This feature is experimental and has not been fully tested yet.)

Via the argument `custom` it is possible to provide custom sub-models that replace the sub-models that are automatically generated by **JointAI**.

Using this feature it is, for instance, possible to use the value of a repeatedly measured variable at a specific time point as covariate in another model. An example would be the use of "baseline" cholesterol (`chol` at `day = 0`) as covariate in a survival model.

First, the variable `chol0` is added to the `PBC` data. For most patients the value of cholesterol at baseline is observed, but not for all. It is important that the data has a row with `day = 0` for each patient.

```
PBC <- merge(PBC,
             subset(PBC, day == 0, select = c("id", "chol")),
             by = "id", suffixes = c("", "0"))
```

Next, the custom piece of JAGS model syntax needs to be specified. We loop here only over the patients for which the baseline cholesterol is missing.

```
calc_chol0 <- "
for (ii in 1:28) {
  M_id[row_chol0_id[ii], 3] <- M_lvlone[row_chol0_lvlone[ii], 1]
  }"
```

To be able to run the model with the custom imputation "model" for baseline cholesterol we need to provide the numbers of the rows in the data matrices that contain the missing values of baseline cholesterol and the rows that contain the imputed cholesterol at `day = 0`:

```
row_chol0_lvlone <- which(PBC$day == 0 & is.na(PBC$chol0))
row_chol0_id <- match(PBC$id, unique(PBC$id))[row_chol0_lvlone]
```

Then we pass both the custom sub-model and the additional data to the analysis function `coxph_imp()`. Note that we explicitly need to specify the model for `chol`.

```
coxph_imp(list(Surv(futime, status != "censored") ~ age + sex + chol0,
               chol ~ age + sex + day + (day | id)),
          no_model = "day", data = PBC,
          append_data_list = list(row_chol0_lvlone = row_chol0_lvlone,
                                  row_chol0_id = row_chol0_id),
          custom = list(chol0 = calc_chol0))
```

Note

Coding of variables::

The default covariate (imputation) models are chosen based on the `class` of each of the variables, distinguishing between `numeric`, `factor` with two levels, unordered `factor` with >2 levels and ordered `factor` with >2 levels.

When a continuous variable has only two different values it is assumed to be binary and its coding and default (imputation) model will be changed accordingly. This behaviour can be overwritten specifying a model type via the argument `models`.

Variables of type `logical` are automatically converted to unordered factors.

Contrasts:

JointAI version $\geq$ 1.0.0 uses the globally (via `options("contrasts")`) specified contrasts. However, for incomplete categorical variables, for which the contrasts need to be re-calculated within the JAGS model, currently only `contr.treatment` and `contr.sum` are possible. Therefore, when an in complete ordinal covariate is used and the default contrasts (`contr.poly()`) are set to be used for ordered factors, a warning message is printed and dummy coding (`contr.treatment()`) is used for that variable instead.

Non-linear effects and transformation of variables::

JointAI handles non-linear effects, transformation of covariates and interactions the following way:

When, for instance, a model formula contains the function `log(x)` and `x` has missing values, `x` will be imputed and used in the linear predictor of models for which no formula was specified, i.e.,

it is assumed that the other variables have a linear association with x. The log() of the observed and imputed values of x is calculated and used in the linear predictor of the main analysis model.

If, instead of using log(x) in the model formula, a pre-calculated variable logx is used, this variable is imputed directly and used in the linear predictors of all models, implying that variables that have logx in their linear predictors have a linear association with logx but not with x.

When different transformations of the same incomplete variable are used in one model it is strongly discouraged to calculate these transformations beforehand and supply them as different variables. If, for example, a model formula contains both x and x2 (where x2 = x^2), they are treated as separate variables and imputed with separate models. Imputed values of x2 are thus not equal to the square of imputed values of x. Instead, x and I(x^2) should be used in the model formula. Then only x is imputed and x^2 is calculated from the imputed values of x internally.
The same applies to interactions involving incomplete variables.

Sequence of models::
Models generated automatically (i.e., not mentioned in formula or fixed are specified in a sequence based on the level of the outcome of the respective model in the multi-level hierarchy and within each level according to the number of missing values. This means that level-1 variables have all level-2, level-3, ... variables in their linear predictor, and variables on the highest level only have variables from the same level in their linear predictor. Within each level, the variable with the most missing values has the most variables in its linear predictor.

Not (yet) possible::

- prediction (using predict) conditional on random effects
- the use of splines for incomplete variables
- the use of (or equivalents for) pspline, or strata in survival models
- left censored or interval censored data

See Also

set_refcat, traceplot, densplot, summary.JointAI, MC_error, GR_crit, predict.JointAI, add_samples, JointAIObject, add_samples, parameters, list_models

Vignettes

- Minimal Example
- Model Specification
- Parameter Selection
- MCMC Settings
- After Fitting
- Theoretical Background

Examples

```
# Example 1: Linear regression with incomplete covariates
mod1 <- lm_imp(y ~ C1 + C2 + M1 + B1, data = wideDF, n.iter = 100)
```

```
# Example 2: Logistic regression with incomplete covariates
mod2 <- glm_imp(B1 ~ C1 + C2 + M1, data = wideDF,
                family = binomial(link = "logit"), n.iter = 100)

## Not run:

# Example 3: Linear mixed model with incomplete covariates
mod3 <- lme_imp(y ~ C1 + B2 + c1 + time, random = ~ time|id,
                data = longDF, n.iter = 300)


# Example 4: Parametric Weibull survival model
mod4 <- survreg_imp(Surv(time, status) ~ age + sex + meal.cal + wt.loss,
                    data = survival::lung, n.iter = 100)


# Example 5: Proportional hazards survival model
mod5 <- coxph_imp(Surv(time, status) ~ age + sex + meal.cal + wt.loss,
                    data = survival::lung, n.iter = 200)

# Example 6: Joint model for longitudinal and survival data
mod6 <- JM_imp(list(Surv(futime, status != 'censored') ~ age + sex +
                    albumin + copper + trig + (1 | id),
                    albumin ~ day + age + sex + (day | id)),
                    timevar = 'day', data = PBC, n.iter = 100)

# Example 7: Proportional hazards  model with a time-dependent covariate
mod7 <- coxph_imp(Surv(futime, status != 'censored') ~ age + sex + copper +
                  trig + stage + (1 | id),
                  timevar = 'day', data = PBC, n.iter = 100)



# Example 8: Parallel computation
# If no strategy how the "future" should be handled is specified, the
# MCMC chains are run sequentially.
# To run MCMC chains in parallel, a strategy can be specified using the
# package \pkg{future} (see ?future::plan), for example:
future::plan(future::multisession, workers = 4)
mod8 <- lm_imp(y ~ C1 + C2 + B2, data = wideDF, n.iter = 500, n.chains = 8)
mod8$comp_info$future
# To re-set the strategy to sequential computation, the sequential strategy
# can be specified:
future::plan(future::sequential)


## End(Not run)
```

NHANES *National Health and Nutrition Examination Survey (NHANES) Data*

Description

This data is a small subset of the data collected within the 2011-2012 wave of the NHANES study, a study designed to assess the health and nutritional status of adults and children in the United States, conduced by the National Center for Health Statistics.

Usage

```
data(NHANES)
```

Format

A data frame with 186 rows and 13 variables:

SBP systolic blood pressure

gender male or female

age in years

race race / Hispanic origin (5 categories)

WC waist circumference in cm

alc alcohol consumption (binary: <1 drink per week vs. >= 1 drink per week)

educ educational level (binary: low vs. high)

creat creatinine concentration in mg/dL

albu albumin concentration in g/dL

uricacid uric acid concentration in mg/dL

bili bilirubin concentration in mg/dL

occup occupational status (3 categories)

smoke smoking status (3 ordered categories)

Note

The subset provided here was selected and re-coded to facilitate demonstration of the functionality of the JointAI package, and no clinical conclusions should be derived from it.

Source

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

Examples

```
summary(NHANES)
```

parameters *Parameter names of an JointAI object*

Description

Returns the names of the parameters/nodes of an object of class 'JointAI' for which a monitor is set.

Usage

```
parameters(object, expand_ranef = FALSE, mess = TRUE, warn = TRUE, ...)
```

Arguments

object	object inheriting from class 'JointAI'
expand_ranef	logical; should all elements of the random effects vectors/matrices be shown separately?
mess	logical; should messages be given? Default is TRUE.
warn	logical; should warnings be given? Default is TRUE.
...	currently not used

Examples

```
# (This function does not need MCMC samples to work, so we will set
# n.adapt = 0 and n.iter = 0 to reduce computational time)
mod1 <- lm_imp(y ~ C1 + C2 + M2 + O2 + B2, data = wideDF, n.adapt = 0,
               n.iter = 0, mess = FALSE)

parameters(mod1)
```

PBC *PBC data*

Description

Data from the Mayo Clinic trial in primary biliary cirrhosis (PBC) of the liver. This dataset was obtained from the **survival** package: the variables copper and trig from survival::pbc were merged into survival::pbcseq and several categorical variables were re-coded.

Format

PBC: A data frame of 312 individuals in long format with 1945 rows and 21 variables.

Survival outcome and id

id case number

futime number of days between registration and the earlier of death, transplantation, or end of follow-up

status status at endpoint ("censored", "transplant" or "dead")

Baseline covariates

trt D-pen (D-penicillamine) vs placebo

age in years

sex male or female

copper urine copper (μg/day)

trig triglycerides (mg/dl)

Time-varying covariates

day number of days between enrolment and this visit date; all measurements below refer to this date

albumin serum albumin (mg/dl)

alk.phos alkaline phosphatase (U/liter)

ascites presence of ascites

ast aspartate aminotransferase (U/ml)

bili serum bilirubin (mg/dl)

chol serum cholesterol (mg/dl)

edema "no": no oedema, "(un)treated": untreated or successfully treated 1 oedema, "edema": oedema despite diuretic therapy

hepato presence of hepatomegaly (enlarged liver)

platelet platelet count

protime standardised blood clotting time

spiders blood vessel malformations in the skin

stage histologic stage of disease (4 levels)

Examples

```
summary(PBC)
```

plot.JointAI *Plot an object object inheriting from class 'JointAI'*

Description

Plot an object object inheriting from class 'JointAI'

Usage

```
## S3 method for class 'JointAI'
plot(x, ...)
```

Arguments

x	object inheriting from class 'JointAI'
...	currently not used

Note

Currently, `plot()` can only be used with (generalized) linear (mixed) models.

Examples

```
mod <- lm_imp(y ~ C1 + C2 + B1, data = wideDF, n.iter = 100)
plot(mod)
```

plot_all *Visualize the distribution of all variables in the dataset*

Description

This function plots a grid of histograms (for continuous variables) and bar plots (for categorical variables) and labels it with the proportion of missing values in each variable.

Usage

```
plot_all(data, nrow = NULL, ncol = NULL, fill = grDevices::grey(0.8),
  border = "black", allNA = FALSE, idvars = NULL, xlab = "",
  ylab = "frequency", ...)
```

Arguments

data	a data.frame (or a matrix)
nrow	optional; number of rows in the plot layout; automatically chosen if unspecified
ncol	optional; number of columns in the plot layout; automatically chosen if unspecified
fill	colour the histograms and bars are filled with
border	colour of the borders of the histograms and bars
allNA	logical; if FALSE (default) the proportion of missing values is only given for variables that have missing values, if TRUE it is given for all variables
idvars	name of the column that specifies the multi-level grouping structure
xlab, ylab	labels for the x- and y-axis
...	additional parameters passed to barplot and hist

See Also

Vignette: Visualizing Incomplete Data

Examples

```
op <- par(mar = c(2,2,3,1), mgp = c(2, 0.6, 0))
plot_all(wideDF)
par(op)
```

---

plot_imp_distr *Plot the distribution of observed and imputed values*

Description

Plots densities and bar plots of the observed and imputed values in a long-format dataset (multiple imputed datasets stacked onto each other).

Usage

```
plot_imp_distr(data, imp = "Imputation_", id = ".id", rownr = ".rownr",
  ncol = NULL, nrow = NULL, labeller = NULL)
```

Arguments

data	a data.frame containing multiple imputations and the original incomplete data stacked onto each other
imp	the name of the variable specifying the imputation indicator
id	the name of the variable specifying the subject indicator

rownr	the name of a variable identifying which rows correspond to the same observation in the original (un-imputed) data
ncol	optional; number of columns in the plot layout; automatically chosen if unspecified
nrow	optional; number of rows in the plot layout; automatically chosen if unspecified
labeller	optional labeller to be passed to ggplot2::facet_wrap() to change the facet labels

Examples

```
## Not run:
mod <- lme_imp(y ~ C1 + c2 + B2 + C2, random = ~ 1 | id, data = longDF,
               n.iter = 200, monitor_params = c(imps = TRUE), mess = FALSE)
impDF <- get_MIdat(mod, m = 5)
plot_imp_distr(impDF, id = "id", ncol = 3)

## End(Not run)
```

---

predict.JointAI *Predict values from an object of class JointAI*

Description

Obtains predictions and corresponding credible intervals from an object of class 'JointAI'.

Usage

```
## S3 method for class 'JointAI'
predict(object, outcome = 1L, newdata,
  quantiles = c(0.025, 0.975), type = "lp", start = NULL, end = NULL,
  thin = NULL, exclude_chains = NULL, mess = TRUE, warn = TRUE,
  return_sample = FALSE, ...)
```

Arguments

object	object inheriting from class 'JointAI'
outcome	vector of variable names or integers identifying for which outcome(s) the prediction should be performed.
newdata	optional new dataset for prediction. If left empty, the original data is used.
quantiles	quantiles of the predicted distribution of the outcome
type	the type of prediction. The default is on the scale of the linear predictor ("link" or "lp"). Additionally, for generalized linear (mixed) models (incl. beta and log-normal) type = "response" transforms the predicted values to the scale of the response, and for ordinal and multinomial (mixed) models type may be "prob" (to obtain probabilities per class), "class" to obtain the class with the highest posterior probability, or "lp". For parametric survival models type can be "lp" or "response", and for proportional hazards survival models the options are "lp", "risk" (= exp(lp)), "survival" or "expected" (= -log(survival)).

start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
exclude_chains	optional vector of the index numbers of chains that should be excluded
mess	logical; should messages be given? Default is TRUE.
warn	logical; should warnings be given? Default is TRUE.
return_sample	logical; should the full sample on which the summary (mean and quantiles) is calculated be returned?#'
...	currently not used

Details

A model.matrix X is created from the model formula (currently fixed effects only) and newdata. $X\beta$ is then calculated for each iteration of the MCMC sample in object, i.e., $X\beta$ has n.iter rows and nrow(newdata) columns. A subset of the MCMC sample can be selected using start, end and thin.

Value

A list with entries dat, fit and quantiles, where fit contains the predicted values (mean over the values calculated from the iterations of the MCMC sample), quantiles contain the specified quantiles (by default 2.5% and 97.5%), and dat is newdata, extended with fit and quantiles (unless prediction for an ordinal outcome is done with type = "prob", in which case the quantiles are an array with three dimensions and are therefore not included in dat).

Note

- So far, predict cannot calculate predicted values for cases with missing values in covariates. Predicted values for such cases are NA.
- For repeated measures models prediction currently only uses fixed effects.

Functionality will be extended in the future.

See Also

predDF.JointAI, *_imp

Examples

```
# fit model
mod <- lm_imp(y ~ C1 + C2 + I(C2^2), data = wideDF, n.iter = 100)

# calculate the fitted values
fit <- predict(mod)

# create dataset for prediction
```

```
newDF <- predDF(mod, vars = ~ C2)

# obtain predicted values
pred <- predict(mod, newdata = newDF)

# plot predicted values and 95% confidence band
matplot(newDF$C2, pred$fitted, lty = c(1, 2, 2), type = "l", col = 1,
xlab = 'C2', ylab = 'predicted values')
```

`print.Dmat` *Summarize the results from an object of class JointAI*

Description

Obtain and print the `summary`, (fixed effects) coefficients (`coef`) and credible interval (`confint`) for an object of class 'JointAI'.

Usage

```
## S3 method for class 'Dmat'
print(x, digits = getOption("digits"),
  scientific = getOption("scipen"), ...)

## S3 method for class 'JointAI'
summary(object, start = NULL, end = NULL, thin = NULL,
  quantiles = c(0.025, 0.975), subset = NULL, exclude_chains = NULL,
  outcome = NULL, missinfo = FALSE, warn = TRUE, mess = TRUE, ...)

## S3 method for class 'summary.JointAI'
print(x, digits = max(3, .Options$digits - 4), ...)

## S3 method for class 'JointAI'
coef(object, start = NULL, end = NULL, thin = NULL,
  subset = NULL, exclude_chains = NULL, warn = TRUE, mess = TRUE, ...)

## S3 method for class 'JointAI'
confint(object, parm = NULL, level = 0.95,
  quantiles = NULL, start = NULL, end = NULL, thin = NULL,
  subset = NULL, exclude_chains = NULL, warn = TRUE, mess = TRUE, ...)

## S3 method for class 'JointAI'
print(x, digits = max(4, getOption("digits") - 4), ...)
```

Arguments

`x`	an object of class `summary.JointAI` or `JointAI`
`digits`	the minimum number of significant digits to be printed in values.

scientific	A penalty to be applied when deciding to print numeric values in fixed or exponential notation, by default the value obtained from `getOption("scipen")`
...	currently not used
object	object inheriting from class 'JointAI'
start	the first iteration of interest (see `window.mcmc`)
end	the last iteration of interest (see `window.mcmc`)
thin	thinning interval (integer; see `window.mcmc`). For example, `thin = 1` (default) will keep the MCMC samples from all iterations; `thin = 5` would only keep every 5th iteration.
quantiles	posterior quantiles
subset	subset of parameters/variables/nodes (columns in the MCMC sample). Follows the same principle as the argument `monitor_params` in `*_imp`.
exclude_chains	optional vector of the index numbers of chains that should be excluded
outcome	optional; vector identifying for which outcomes the summary should be given, either by specifying their indices, or their names (LHS of the respective model formulas as character string).
missinfo	logical; should information on the number and proportion of missing values be included in the summary?
warn	logical; should warnings be given? Default is `TRUE`.
mess	logical; should messages be given? Default is `TRUE`.
parm	same as `subset` (for consistency with `confint` method for other types of objects)
level	confidence level (default is 0.95)

See Also

The model fitting functions `lm_imp`, `glm_imp`, `clm_imp`, `lme_imp`, `glme_imp`, `survreg_imp` and `coxph_imp`, and the vignette Parameter Selection for examples how to specify the parameter `subset`.

Examples

```
## Not run:
mod1 <- lm_imp(y ~ C1 + C2 + M2, data = wideDF, n.iter = 100)

summary(mod1, missinfo = TRUE)
coef(mod1)
confint(mod1)

## End(Not run)
```

rd_vcov *Extract the random effects variance covariance matrix*

Description

Returns the posterior mean of the variance-covariance matrix/matrices of the random effects in a fitted JointAI object.

Usage

```
rd_vcov(object, outcome = NULL, start = NULL, end = NULL, thin = NULL,
  exclude_chains = NULL, mess = TRUE, warn = TRUE)
```

Arguments

object	object inheriting from class 'JointAI'
outcome	optional; vector of integers giving the indices of the outcomes for which the random effects variance-covariance matrix/matrices should be returned.
start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
exclude_chains	optional vector of the index numbers of chains that should be excluded
mess	logical; should messages be given? Default is TRUE.
warn	logical; should warnings be given? Default is TRUE.

residuals.JointAI *Extract residuals from an object of class JointAI*

Description

Extract residuals from an object of class JointAI

Usage

```
## S3 method for class 'JointAI'
residuals(object, type = c("working", "pearson",
  "response"), warn = TRUE, ...)
```

Arguments

object	object inheriting from class 'JointAI'
type	type of residuals: "deviance", "response", "working"
warn	logical; should warnings be given? Default is TRUE.
...	currently not used

Note

- For mixed models residuals are currently calculated using the fixed effects only.
- For ordinal (mixed) models and parametric survival models only type = "response" is available.
- For Cox proportional hazards models residuals are not yet implemented.

Examples

```
mod <- glm_imp(B1 ~ C1 + C2 + O1, data = wideDF, n.iter = 100,
               family = binomial(), mess = FALSE)
summary(residuals(mod, type = 'response')[[1]])
summary(residuals(mod, type = 'working')[[1]])
```

---

set_refcat *Specify reference categories for all categorical covariates in the model*

Description

The function is a helper function that asks questions and, depending on the answers given by the user, returns the input for the argument refcats in the main analysis functions *_imp.

Usage

```
set_refcat(data, formula, covars, auxvars = NULL)
```

Arguments

data	a data.frame
formula	optional; model formula or a list of formulas (used to select subset of relevant columns of data)
covars	optional; vector containing the names of relevant columns of data
auxvars	optional; formula containing the names of relevant columns of data that should be considered additionally to the columns occurring in the formula

Details

The arguments `formula`, `covars` and `auxvars` can be used to specify a subset of the `data` to be considered. If non of these arguments is specified, all variables in `data` will be considered.

Examples

```
## Not run:
# Example 1: set reference categories for the whole dataset and choose
# answer option 3:
set_refcat(data = NHANES)
3

# insert the returned string as argument refcats
mod1 <- lm_imp(SBP ~ age + race + creat + educ, data = NHANES,
               refcats = 'largest')

# Example 2:
# specify a model formula
fmla <- SBP ~ age + gender + race + bili + smoke + alc

# write the output of set_refcat to an object
ref_mod2 <- set_refcat(data = NHANES, formula = fmla)
4
2
5
1
1

# enter the output in the model specification
mod2 <- lm_imp(formula = fmla, data = NHANES, refcats = ref_mod2,
               n.adapt = 0)

## End(Not run)
```

`sharedParams` *Parameters used by several functions in JointAI*

Description

Parameters used by several functions in JointAI

Arguments

`object`	object inheriting from class 'JointAI'
`no_model`	optional; vector of names of variables for which no model should be specified. Note that this is only possible for completely observed variables and implies the assumptions of independence between the excluded variable and the incomplete variables.

timevar	name of the variable indicating the time of the measurement of a time-varying covariate in a proportional hazards survival model (also in a joint model). The variable specified in "timevar" will automatically be added to "no_model".
assoc_type	named vector specifying the type of the association used for a time-varying covariate in the linear predictor of the survival model when using a "JM" model. Implemented options are "underl.value" (linear predictor; default for covariates modelled using a Gaussian, Gamma, beta or log-normal distribution) covariates) and "obs.value" (the observed/imputed value; default for covariates modelled using other distributions).
subset	subset of parameters/variables/nodes (columns in the MCMC sample). Follows the same principle as the argument monitor_params in *_imp.
exclude_chains	optional vector of the index numbers of chains that should be excluded
start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)
n.adapt	number of iterations for adaptation of the MCMC samplers (see adapt)
n.iter	number of iterations of the MCMC chain (after adaptation; see coda.samples)
n.chains	number of MCMC chains
quiet	logical; if TRUE then messages generated by **rjags** during compilation as well as the progress bar for the adaptive phase will be suppressed, (see jags.model)
progress.bar	character string specifying the type of progress bar. Possible values are "text" (default), "gui", and "none" (see update). Note: when sampling is performed in parallel it is not possible to display a progress bar.
thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
nrow	optional; number of rows in the plot layout; automatically chosen if unspecified
ncol	optional; number of columns in the plot layout; automatically chosen if unspecified
use_ggplot	logical; Should ggplot be used instead of the base graphics?
warn	logical; should warnings be given? Default is TRUE.
mess	logical; should messages be given? Default is TRUE.
xlab, ylab	labels for the x- and y-axis
idvars	name of the column that specifies the multi-level grouping structure
seed	optional; seed value (for reproducibility)
ppc	logical: should monitors for posterior predictive checks be set? (not yet used)
rd_vcov	optional character string or list (of lists or character strings) specifying the structure of the variance covariance matrix/matrices of the random effects for multivariate mixed models. Options are "full", "blockdiag" (default) and "indep". Different structures can be specified per grouping level (in multi-level models with more than two levels) by specifying a list with elements per grouping level. To specify different structures for different outcomes, a list (maybe nested in the list per grouping level) can be specified. This list should have the type of structure as names and contain vectors of variable names that belong to the respective structure.

simLong *Simulated Longitudinal Data in Long and Wide Format*

Description

This data was simulated to mimic data from a longitudinal cohort study following mothers and their child from birth until approximately 4 years of age. It contains 2400 observations of 200 mother-child pairs. Children's BMI and head circumference was measured repeatedly and their age in months was recorded at each measurement. Furthermore, the data contain several baseline variables with information on the mothers' demographics and socio-economic status.

Usage

```
simLong

simWide
```

Format

simLong: A data frame in long format with 2400 rows and 16 variables

simWide: A data frame in wide format with 200 rows and 81 variables

An object of class data.frame with 2400 rows and 16 columns.

An object of class data.frame with 200 rows and 81 columns.

Baseline covariates

(in simLong and simWide)

GESTBIR gestational age at birth (in weeks)

ETHN ethnicity (binary: European vs. other)

AGE_M age of the mother at intake

HEIGHT_M height of the mother (in cm)

PARITY number of times the mother has given birth (binary: 0 vs. >=1)

SMOKE smoking status of the mother during pregnancy (3 ordered categories: never smoked during pregnancy, smoked until pregnancy was known, continued smoking in pregnancy)

EDUC educational level of the mother (3 ordered categories: low, mid, high)

MARITAL marital status (3 categories)

ID subject identifier

Long-format variables

(only in `simLong`)

time measurement occasion/visit (by design, children should be measured at/around 1, 2, 3, 4, 7, 11, 15, 20, 26, 32, 40 and 50 months of age)

age child age at measurement time in months

bmi child BMI

hc child head circumference in cm

hgt child height in cm

wgt child weight in gram

sleep sleeping behaviour of the child (3 ordered categories)

Wide-format variables

(only in `simWide`)

age1, age2, age3, age4, age7, age11, age15, age20, age26, age32, age40, age50 child age at the repeated measurements in months

bmi1, bmi2, bmi3, bmi4, bmi7, bmi11, bmi15, bmi20, bmi26, bmi32, bmi40, bmi50 repeated measurements of child BMI

hc1, hc2, hc3, hc4, hc7, hc11, hc15, hc20, hc26, hc32, hc40, hc50 repeated measurements of child head circumference in cm

hgt1, hgt2, hgt3, hgt4, hgt7, hgt11, hgt15, hgt20, hgt26, hgt32, hgt40, hgt50 repeated measurements of child height in cm

wgt1, wgt2, wgt3, wgt4, wgt7, wgt11, wgt15, wgt20, wgt26, wgt32, wgt40, wgt50 repeated measurements of child weight in gram

sleep1, sleep2, sleep3, sleep4, sleep7, sleep11, sleep15, sleep20, sleep26, sleep32, sleep40, sleep50 repeated measurements of child sleep behaviour (3 ordered categories)

Examples

```
summary(simLong)
summary(simWide)
```

`sum_duration` *Calculate the sum of the computational duration of a JointAI object*

Description

Calculate the sum of the computational duration of a JointAI object

Usage

```
sum_duration(object, by = NULL)
```

Arguments

object	object of class JointAI
by	optional grouping information; options are NULL (default) to calculate the sum over all chains and runs and both the adaptive and sampling phase, "run" to get the duration per run, "phase" to get the sum over all chains and runs per phase, "chain" to get the sum per chain over both phases and all runs, "phase and run" to get the sum over all chains, separately per phase and run.

---

traceplot *Create traceplots for a MCMC sample*

Description

Creates a set of traceplots from the MCMC sample of an object of class 'JointAI'.

Usage

```
traceplot(object, ...)

## S3 method for class 'JointAI'
traceplot(object, start = NULL, end = NULL,
  thin = NULL, subset = c(analysis_main = TRUE), outcome = NULL,
  exclude_chains = NULL, nrow = NULL, ncol = NULL, use_ggplot = FALSE,
  warn = TRUE, mess = TRUE, ...)
```

Arguments

object	object inheriting from class 'JointAI'
...	Arguments passed on to graphics::matplot lty,lwd,lend vector of line types, widths, and end styles. The first element is for the first column, the second element for the second column, etc., even if lines are not plotted for all columns. Line types will be used cyclically until all plots are drawn. col vector of colors. Colors are used cyclically. cex vector of character expansion sizes, used cyclically. This works as a multiple of par("cex"). NULL is equivalent to 1.0. bg vector of background (fill) colors for the open plot symbols given by pch = 21:25 as in points. The default NA corresponds to the one of the underlying function plot.xy. add logical. If TRUE, plots are added to current one, using points and lines. verbose logical. If TRUE, write one line of what is done.
start	the first iteration of interest (see window.mcmc)
end	the last iteration of interest (see window.mcmc)

thin	thinning interval (integer; see window.mcmc). For example, thin = 1 (default) will keep the MCMC samples from all iterations; thin = 5 would only keep every 5th iteration.
subset	subset of parameters/variables/nodes (columns in the MCMC sample). Follows the same principle as the argument monitor_params in *_imp.
outcome	optional; vector identifying a subset of sub-models included in the output, either by specifying their indices (using the order used in the list of model formulas), or their names (LHS of the respective model formula as character string)
exclude_chains	optional vector of the index numbers of chains that should be excluded
nrow	optional; number of rows in the plot layout; automatically chosen if unspecified
ncol	optional; number of columns in the plot layout; automatically chosen if unspecified
use_ggplot	logical; Should ggplot be used instead of the base graphics?
warn	logical; should warnings be given? Default is TRUE.
mess	logical; should messages be given? Default is TRUE.

See Also

summary.JointAI, *_imp, densplot
The vignette Parameter Selection contains some examples how to specify the parameter subset.

Examples

```
# fit a JointAI model
mod <- lm_imp(y ~ C1 + C2 + M1, data = wideDF, n.iter = 100)


# Example 1: simple traceplot
traceplot(mod)


# Example 2: ggplot version of traceplot
traceplot(mod, use_ggplot = TRUE)


# Example 5: changing how the ggplot version looks (using ggplot syntax)
library(ggplot2)

traceplot(mod, use_ggplot = TRUE) +
  theme(legend.position = 'bottom') +
  xlab('iteration') +
  ylab('value') +
  scale_color_discrete(name = 'chain')
```

`wideDF` *Cross-sectional example dataset*

Description

A simulated cross-sectional dataset.

Usage

```
data(wideDF)
```

Format

A simulated data frame with 100 rows and 13 variables:

C1 continuous, complete variable

C2 continuous, incomplete variable

B1 binary, complete variable

B2 binary, incomplete variable

M1 unordered factor; complete variable

M2 unordered factor; incomplete variable

O1 ordered factor; complete variable

O2 ordered factor; incomplete variable

L1 continuous, complete variable

L2 continuous incomplete variable

id id (grouping) variable

time continuous complete variable

y continuous, complete variable

Index

∗ **datasets**
longDF, 19
NHANES, 37
PBC, 38
simLong, 50
wideDF, 54
∗_imp, *3*, *8*, *11*, *12*, *16*, *21*, *43*, *45*, *47*, *49*, *53*

abline, *21*
adapt, *26*, *49*
add_samples, 3, *35*
auto_corr, 4
auto_corr_plot *(*auto_corr*)*, 4

barplot, *41*
betamm_imp, *14*
betamm_imp *(*model_imp*)*, 23
betareg_imp, *14*
betareg_imp *(*model_imp*)*, 23

clean_survname, 5
clm_imp, *14*, *45*
clm_imp *(*model_imp*)*, 23
clmm_imp, *14*
clmm_imp *(*model_imp*)*, 23
coda.samples, *26*, *49*
coef(), *14*
coef.JointAI *(*print.Dmat*)*, 44
confint(), *14*
confint.JointAI *(*print.Dmat*)*, 44
coxph, *14*
coxph_imp, *14*, *45*
coxph_imp *(*model_imp*)*, 23
cross_corr, 5
cross_corr_plot *(*cross_corr*)*, 5

default_hyperpars, 6, *27*
densplot, 8, *15*, *35*, *53*
densplot(), *14*

extract_state, 9

family, *27*
formula, *26*, *28*

gelman.diag, *12*
get_MIdat, 10, *15*
get_MIdat(), *15*
get_missinfo, 11
ggplot2::facet_wrap(), *42*
glm, *14*
glm_imp, *14*, *15*, *45*
glm_imp *(*model_imp*)*, 23
glme_imp, *14*, *45*
glme_imp *(*model_imp*)*, 23
glmer_imp, *14*
glmer_imp *(*model_imp*)*, 23
GR_crit, 12, *15*, *35*
graphics::matplot, *52*

hist, *41*

internal_clean_survname, 13

jags.model, *27*, *49*
JM_imp, *14*
JM_imp *(*model_imp*)*, 23
JointAI, 14, *28*
JointAI-package *(*JointAI*)*, 14
JointAIObject, 16, *35*

lines, *52*
list_models, *15*, 18, *35*
lm, *14*
lm_imp, *14*, *15*, *45*
lm_imp *(*model_imp*)*, 23
lme, *14*
lme_imp, *14*, *15*, *45*
lme_imp *(*model_imp*)*, 23
lmer_imp, *14*
lmer_imp *(*model_imp*)*, 23
lognorm_imp, *14*
lognorm_imp *(*model_imp*)*, 23

lognormmm_imp, *14*
lognormmm_imp *(*model_imp*)*, 23
longDF, 19

MC_error, *15*, 20, *35*
mcmcse::mcse.mat, *21*
md_pattern, *15*, 22
mlogit_imp, *14*
mlogit_imp *(*model_imp*)*, 23
mlogitmm_imp *(*model_imp*)*, 23
model_imp, 23

NHANES, 36

par, *22*, *52*
parameters, *15*, *35*, 38
PBC, 38
plot.JointAI, 40
plot.MCElist *(*MC_error*)*, 20
plot.xy, *52*
plot_all, *15*, 40
plot_imp_distr, *11*, 41
plot_imp_distr(), *15*
points, *52*
predDF, *15*
predDF.JointAI, *43*
predict, *15*
predict(), *14*
predict.JointAI, *35*, 42
print.Dmat, 44
print.JointAI *(*print.Dmat*)*, 44
print.summary.JointAI *(*print.Dmat*)*, 44
pspline, *35*

rd_vcov, 46
residuals.JointAI, 46

set_refcat, *15*, *26*, *35*, 47
sharedParams, 48
simLong, 50
simWide *(*simLong*)*, 50
strata, *35*
sum_duration, 51
summary, *15*
summary(), *14*
summary.JointAI, *15*, *35*, *53*
summary.JointAI *(*print.Dmat*)*, 44
survreg, *14*
survreg_imp, *14*, *45*
survreg_imp *(*model_imp*)*, 23

traceplot, *15*, *35*, 52
traceplot(), *14*

update, *3*, *27*, *49*

wideDF, 54
window.mcmc, *3–5*, *8*, *10*, *13*, *21*, *26*, *43*, *45*, *46*, *49*, *52*, *53*