Package 'mdmb'

January 21, 2021

Type Package

Title Model Based Treatment of Missing Data

Version 1.5-8

Date 2021-01-21 13:00:54

Author Alexander Robitzsch [aut, cre], Oliver Luedtke [aut]

Maintainer Alexander Robitzsch <robitzsch@leibniz-ipn.de>

Description Contains model-based treatment of missing data for regression models with missing values in covariates or the dependent variable using maximum likelihood or Bayesian estimation (Ibrahim et al., 2005; <doi:10.1198/016214504000001844>; Luedtke, Robitzsch, & West, 2020a, 2020b; <doi:10.1080/00273171.2019.1640104><doi:10.1037/met0000233>). The regression model can be nonlinear (e.g., interaction effects, quadratic effects or B-spline functions). Multilevel models with missing data in predictors are available for Bayesian estimation. Substantive-model compatible multiple imputation can be also conducted.

Depends R (>= 3.1)

Imports CDM, coda, graphics, miceadds (>= 3.2-23), Rcpp, sirt, stats, utils

Suggests MASS

LinkingTo miceadds, Rcpp, RcppArmadillo

Enhances JointAI, jomo, mice, smcfcs

URL https://github.com/alexanderrobitzsch/mdmb, https://sites.google.com/site/alexanderrobitzsch2/software

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-01-21 15:10:06 UTC

R topics documented:

`mdmb-package` *Model Based Treatment of Missing Data*

Description

Contains model-based treatment of missing data for regression models with missing values in covariates or the dependent variable using maximum likelihood or Bayesian estimation (Ibrahim et al., 2005; <doi:10.1198/016214504000001844>; Luedtke, Robitzsch, & West, 2020a, 2020b; <doi:10.1080/00273171.2019.1640104><doi:10.1037/met0000233>). The regression model can be nonlinear (e.g., interaction effects, quadratic effects or B-spline functions). Multilevel models with missing data in predictors are available for Bayesian estimation. Substantive-model compatible multiple imputation can be also conducted.

Details

- The maximum likelihood estimation of regression models with missing values in covariates is implemented in `frm_em`. Available regression models are linear regression, logistic regression, ordinal probit regression and models with Box-Cox or Yeo-Johnson transformed normally distributed outcomes. The factorization based regression model also allow the inclusion of latent variables and measurement error prone covariates.
- Bayesian estimation and multiple imputation of regression models with missing values in covariates is implemented in `frm_fb`. The same regression models like in `frm_em` can be specified. Moreover, multilevel models can also be specified with Bayesian estimation. The function `frm_fb` allows substantive model compatible multiple imputation.

Author(s)

Alexander Robitzsch [aut, cre], Oliver Luedtke [aut]

Maintainer: Alexander Robitzsch <robitzsch@leibniz-ipn.de>

References

Ibrahim, J. G., Chen, M. H., Lipsitz, S. R., & Herring, A. H. (2005). Missing-data methods for generalized linear models: A comparative review. *Journal of the American Statistical Association, 100*, 332-346.

Luedtke, O., Robitzsch, A., & West, S. (2020a). Analysis of interactions and nonlinear effects with missing data: A factored regression modeling approach using maximum likelihood estimation. *Multivariate Behavioral Research, 55*(3), 361-381. doi: 10.1080/00273171.2019.1640104

Luedtke, O., Robitzsch, A., & West, S. (2020b). Regression models involving nonlinear effects with missing data: A sequential modeling approach using Bayesian estimation. *Psychological Methods, 25*(2), 157-181. doi: 10.1037/met0000233

See Also

The EM algorithm for the multivariate normal model is implemented in `norm2::emNorm` in the **norm2** package. A corresponding MCMC algorithm can be found in the `norm2::mcmcNorm` function.

See the **lavaan**, **OpenMx** or **sem** packages for full information maximum likelihood approaches for handling missing data for multivariate normal distributions, linear regression models, and, more generally, structural equation modeling with missing data.

Structural equation models with missing data can be also estimated with a two-stage procedure. In a first stage, a mean vector and a covariance matrix is estimated (possibly with auxiliary variables) and in the second stage, the structural equation model is estimated on the previously obtained mean vector and covariance matrix. The procedure is implemented in the `semTools::twostage` function in the **semTools** package.

Examples

```
##
##  |\  /||~~\ |\  /||~~\
##  | \/ ||   || \/ ||--<
##  |    ||__/ |    ||__/
##

##
##  > library(mdmb)
##  * mdmb 0.0-13 (2017-01-15)
##
```

`data.mb` *Example Datasets for* **mdmb** *Package*

Description

Example datasets for **mdmb** package.

Usage

```
data(data.mb01)
data(data.mb02)
data(data.mb03)
data(data.mb04)
data(data.mb05)
```

Format

- Dataset data.mb01. Simulated dataset with missing values. Variables Y, X and Z are continuous.

  ```
  List of 2
  $ complete:'data.frame': 4000 obs. of 3 variables:
  ..$ X: num [1:4000] -1.08 0.57 -0.32 0.34 1.21 -0.44 -1.07 -0.29 0.76 -1.75 ...
  ..$ Z: num [1:4000] -0.02 0.26 -1.45 1.24 0.98 -2.36 0.84 -1.08 -0.15 -1.36 ...
  ..$ Y: num [1:4000] 0.88 1.75 -0.82 -1.81 -1.58 -3.34 -3.35 -0.29 1.47 0.23 ...
  $ missing :'data.frame': 4000 obs. of 3 variables:
  ..$ X: num [1:4000] -1.08 0.57 NA NA 1.21 NA -1.07 -0.29 0.76 NA ...
  ..$ Z: num [1:4000] -0.02 0.26 -1.45 1.24 0.98 -2.36 0.84 -1.08 -0.15 -1.36 ...
  ..$ Y: num [1:4000] 0.88 1.75 -0.82 -1.81 -1.58 -3.34 -3.35 -0.29 1.47 0.23 ...
  ```

- Dataset data.mb02. Simulated dataset with missing values. The variables Z and Y are dichotomous.

  ```
  List of 2
  $ complete:'data.frame': 2000 obs. of 3 variables:
  ..$ X: num [1:2000] -0.93 0.3 -0.93 0.7 0.52 -1.38 -0.14 0.09 0.23 -1.64 ...
  ..$ Z: num [1:2000] 1 0 1 0 0 1 1 1 1 1 ...
  ..$ Y: num [1:2000] 1 1 0 1 1 0 1 0 0 1 ...
  $ missing :'data.frame': 2000 obs. of 3 variables:
  ..$ X: num [1:2000] -0.93 0.3 -0.93 0.7 0.52 NA -0.14 0.09 0.23 -1.64 ...
  ..$ Z: num [1:2000] 1 0 1 NA NA 1 NA 1 1 1 ...
  ..$ Y: num [1:2000] 1 1 0 1 1 0 1 0 0 1 ...
  ```

- Dataset data.mb03. This dataset is from Enders, Baraldi & Cham (2014) and contains three variables primary school reading (x), primary school learning problems (z) and middle school reading (y) which all have missing values.

  ```
  'data.frame': 74 obs. of 3 variables:
  $ x: num NA NA NA NA NA 8.34 NA 8.36 6.89 8.56 ...
  $ z: num 8.81 4.5 6.31 4.7 5.1 4 6.11 3.7 6.81 6.31 ...
  $ y: num 5 5.1 6.3 9 9 9.3 NA 10.7 6.2 NA ...
  ```

- Dataset data.mb04. This multilevel dataset contains three variables: level-1 variables codey, x and the level-2 variable w.

  ```
  'data.frame': 500 obs. of 4 variables:
  $ idcluster: int 1 1 1 1 1 2 2 2 2 2 ...
  $ x : num NA NA -1.15 -1.65 0.25 ...
  $ w : num -0.552 -0.552 -0.552 -0.552 -0.552 ...
  ```

```
$ y : num NA NA -0.0711 0.7165 -0.1917 ...
```

- Dataset `data.mb05`. This dataset contains selected (and transformed) variables of the German PISA 2012 data.

```
'data.frame': 5001 obs. of 13 variables:
$ idschool : num 1001 1001 1001 1001 1001 ...
$ idstud : num 1e+05 1e+05 1e+05 1e+05 1e+05 ...
$ female : num 1 1 0 0 0 1 1 1 0 0 ...
$ books : num NA 3 3 1 NA 2 NA 1 NA 2 ...
$ hisced : num NA 6 6 2 NA 2 NA 2 NA 2 ...
$ hisei : num NA 30.6 57.7 26.9 NA ...
$ hisei10 : num NA 0.257 0.596 0.211 NA ...
$ native : num NA NA 1 0 NA 0 NA 1 NA 1 ...
$ ANCINTMAT: num NA 0.644 -0.096 1.057 NA ...
$ MATHEFF : num NA 0.34 0.54 -0.18 NA 0.15 NA NA NA NA ...
$ READ : num -0.25 -0.503 0.421 -1.664 -0.894 ...
$ MATH : num -0.565 -0.854 0.384 -0.896 -0.534 ...
$ W_FSTUWT : num 140 140 140 140 140 ...
```

References

Enders, C. K., Baraldi, A. N., & Cham, H. (2014). Estimating interaction effects with incomplete predictor variables. *Psychological Methods, 19*(1), 39-55. doi: 10.1037/a0035314

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Linear interaction example from Enders et al. (2014)
#############################################################################

# load packages
library(mdmb)
library(mice)
library(mitools)
library(sandwich)

#--- attach example dataset (Enders et al., 2014) from mdmb package
data( data.mb03, package="mdmb")
dat <- data.mb03

#--- center data which speeds convergence of Bayesian estimation
#--- of the imputation model
for (vv in 1:3){
    M_vv <- mean( dat[,vv], na.rm=TRUE )
    dat[,vv] <- dat[,vv] - M_vv
}

#--- generate initial imputed values withj mice package
```

```
imp <- mice::mice( dat, m=, maxit=20 )
data_init <- mice::complete(imp, action=1)

#--- define number of iterations and number of imputed datasets
iter <- 50000; burnin <- 5000
Nimp <- 100

#******* imputation model M3 with quadratic effects

# model for dependent variable
dep <- list("model"="linreg", "formula"=y ~ x*z + I(x^2) + I(z^2) )
# covariate models
ind_x <- list( "model"="linreg", "formula"=x ~ z + I(z^2) )
ind_z <- list( "model"="linreg", "formula"=z ~ 1 )
ind <- list( x=ind_x, z=ind_z)

#generate imputations
imp <- mdmb::frm_fb(dat=dat, dep=dep, ind=ind, burnin=burnin, iter=iter,
             data_init=data_init, Nimp=Nimp)

#--- create list of multiply imputed datasets
datlist <- mdmb::frm2datlist(imp)

#-------------------------------
#--- analyze imputed datasets with mice package

# convert into object of class mids
imp2 <- miceadds::datlist2mids(datlist)
# estimate linear model on multiply imputed datasets
mod1 <- with(imp2, stats::lm( y ~ x*z ) )
summary( mice::pool(mod1) )

#-------------------------------
#--- analyze imputed datasets using sandwich standard errors

results <- list()
variances <- list()
Nimp <- length(datlist)
for (ii in 1:Nimp){
    mod_ii <- stats::lm( y ~ x*z, data=datlist[[ii]] )
    variances[[ii]] <- sandwich::vcovHC(mod_ii)
    results[[ii]] <- coef(mod_ii)
}

mod2 <- mitools::MIcombine(results=results,variances=variances,df.complete=69)
summary(mod2)

## End(Not run)
```

---

eval_prior_list *Evaluates Several Prior Distributions*

Description

The function `eval_prior_list` evaluates several prior distributions specified in a list. The function `eval_prior_list_sumlog` computes the sum of the logarithms of all prior values.

Usage

```
eval_prior_list(par, par_prior, log=FALSE, eps=1e-50)

eval_prior_list_sumlog(par, par_prior, use_grad=1)
```

Arguments

`par`	Parameter vector
`par_prior`	List of prior distributions specified in a list (see Examples)
`log`	Logical indicating whether the logarithm of the prior should be computed
`eps`	Decimal to be added to the prior to avoid computation of the logarithm for values of zero
`use_grad`	Integer value for computation value for gradient

Value

Vector or a numeric value

See Also

`sirt::prior_model_parse`

Examples

```
#############################################################################
# EXAMPLE 1: Evaluation of prior values
#############################################################################

# normal distribution
b0 <- list( "dnorm", list(mean=0,sd=100) )
# t distribution with one degree of freedom (Cauchy distribution)
b1 <- list( "dt", list(df=1) )
# define list of priors
beta_prior <- list( b0, b1 )
# parameter value
beta <- c( 0.3, 1 )

#-- evaluate priors
mdmb::eval_prior_list(par=beta, par_prior=beta_prior)
mdmb::eval_prior_list_sumlog(par=beta, par_prior=beta_prior)
```

frm *Factored Regression Model: Generalized Linear Regression Model with Missing Covariates*

Description

The factored regression model (FRM) allows the estimation of the linear regression model (with normally distributed residuals) and the generalized logistic regression model (logistic regression for dichotomous outcomes). Missing values in covariates are handled by posing a conditional univariate distribution for each covariate. The approach follows Ibrahim (1990), Ibrahim, Chen, Lipsitz and Herring (2005), Lee and Mitra (2016), and Zhang and Wang (2017) and is applied in Luedtke, Robitzsch, and West (2020a, 2020b). Latent variables and covariates with measurement error or multiple indicators can also be handled within this framework (see Examples 3, 4 and 5).

Missing values are handled by numerical integration in `frm_em` (see also Allison, 2012). The user has to specify an integration grid for each variable (defined in argument `nodes` for each model).

Standard error estimates in `frm_em` are obtained by a numerical differentiation of the Fisher score function (see Jamshidian & Jennrich, 2000).

The function `frm_fb` employs a fully Bayesian approach with noninformative prior distribution. This function imputes missing values in the models from the posterior distributions. Imputed datasets can be extracted by the function `frm2datlist`.

The current functionality only support missing values on continuous covariates (accommodating skewness and only positive values), dichotomous covariates and ordinal covariates.

Multilevel models (using `model="mlreg"`) for normally distributed (`outcome="normal"`) and ordinal variables (`outcome="probit"`) as well as variables at higher levels (using argument `variable_level`) are accommodated.

The handling of nominal covariates will be included in future **mdmb** package versions.

Usage

```
# Factored regression model: Numerical integration with EM algorithm
frm_em(dat, dep, ind, weights=NULL, verbose=TRUE, maxiter=500, conv_dev=1e-08,
         conv_parm=1e-05, nodes_control=c(11,5), h=1e-04, use_grad=2)

## S3 method for class 'frm_em'
coef(object, ...)
## S3 method for class 'frm_em'
logLik(object, ...)
## S3 method for class 'frm_em'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'frm_em'
vcov(object, ...)

# Factored regression model: Fully Bayesian estimation
frm_fb(dat, dep, ind, weights=NULL, verbose=TRUE, data_init=NULL, iter=500,
        burnin=100, Nimp=10, Nsave=3000, refresh=25, acc_bounds=c(.45,.50),
```

```
        print_iter=10, use_gibbs=TRUE, aggregation=TRUE)

## S3 method for class 'frm_fb'
coef(object, ...)
## S3 method for class 'frm_fb'
plot(x, idparm=NULL, ask=TRUE, ... )
## S3 method for class 'frm_fb'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'frm_fb'
vcov(object, ...)

frm2datlist(object, as_mids=FALSE)  # create list of imputed datasets
```

Arguments

dat	Data frame
dep	List containing model specification for dependent variable. The list has arguments (see Examples) model: String indicating the model type. Options are "linreg" (wrapper to stats::lm or stats::lm.wfit), "logistic" for dichotomous variables (wrapper to logistic_regression), "oprobit" for ordinal variables (wrapper to oprobit_regression), "yjtreg" for continuous variables and bounded variables on $(0, 1)$ (wrapper to yjt_regression), "bctreg" for positive continuous variables (wrapper to bct_regression), "mlreg" for multilevel models with normally distributed and ordinal data (wrapper to miceadds::ml_mcmc) formula: An R formula object. nodes (for frm_em): Vector containing the integration nodes nodes_weights (for frm_em): Optional vector containing initial probabilities for each node coef_inits: Optional vector containing initial coefficient for the model sigma_fixed: Fixed standard deviations in case of model="linreg". Heterogeneous standard deviations are allowed. R_args: Arguments for estimation functions. sampling_level: Variable name for cluster identifiers for level for sampling values for multilevel data. Sampling at level of clusters can be beneficials if derived variables from cluster members (e.g., group means) occur in multilevel models. The option is only applicable for frm_fb. variable_level: Cluster identifier indicating level of variable for imputations of higher-level variables
ind	List containing a list of univariate conditional models for covariates. See dep for more details on specification.
weights	Optional vector of sampling weights
verbose	Logical indicating whether convergence progress should be displayed.
maxiter	Maximum number of iterations
conv_dev	Convergence criterion for deviance
conv_parm	Convergence criterion for regression coefficients

nodes_control	Control arguments if nodes are not provided by the user. The first value denote the number of nodes, while the second value denotes the spread of the node distribution defined as the factor of the standard deviation of the observed data.
h	Step width for numerical differentiation for calculating the covariance matrix
use_grad	Computation method for gradient in yjt_regression, bct_regression or logistic_regression. It can be 0 (compatible with **mdmb** $\leq$0.3), 1 or 2 (most efficient one).
data_init	Initial values for dataset
iter	Number of iterations
burnin	Number of burnin iterations
Nimp	Number of imputed datasets
Nsave	(Maximum) Number of values to be saved for MCMC chain
refresh	Number of imputations after which proposal distribution should be updated in Metropolis-Hastings step
acc_bounds	Bounds for acceptance rates for parameters
print_iter	Number of imputation after which iteration progress should be displayed
use_gibbs	Logical indicating whether Gibbs sampling instead of Metropolis-Hastings sampling should be used. Can be only applied for linreg.
aggregation	Logical indicating whether complete dataset should be used for computing the predictive distribution of missing values. argument=TRUE is often needed for multilevel data in which cluster means are included in regression models.
object	Object of corresponding class
x	Object of corresponding class
digits	Number of digits in summary
file	File to which the summary should be linked
idparm	Indices for parameters to be plotted
ask	Logical indicating whether the user is asked before new plot
as_mids	Logical indicating whether multiply imputed datasets should be converted into objects of class mids
...	Further arguments to be passed

Details

The function allows for fitting a factored regression model. Consider the case of three variables Y, X and Z. A factored regression model consists of a sequence of univariate conditional models $P(Y|X,Z)$, $P(X|Z)$ and $P(Z)$ such that the joint distribution can be factorized as

$$P(Y,X,Z) = P(Y|X,Z)P(X|Z)P(Z)$$

Each of the three variables can contain missing values. Missing values are integrated out by posing a distributional assumption for each variable with missing values.

Value

For `frm_em` it is a list containing the following values

`coefs`	Estimated coefficients
`vcov`	Covariance matrix
`partable`	Summary parameter table
`all_coefs`	List with all estimated coefficients
`ll`	Log likelihood value
`like`	Individual likelihood
`dat`	Data frame with included latent values for each variable with missing values
`se`	Standard errors for coefficients
`info`	Information matrix
`conv`	Convergence indicator
`iter`	Number of iterations
`ic`	Information criteria
`ind0`	List with model specifications including `dep` and `ind`
`predictorMatrix`	Predictor matrix
`variablesMatrix`	Matrix containing all variables appearing in statistical models
`desc_vars`	Descriptive statistics of variables
`model_results`	Results from fitted models

The output for `frm_fb` contains particular additional values

`tech_summary`	Summary table with informations about MCMC algorithm
`values_coda`	Sampled parameter values saved as class `mcmc` for analysis in **coda** package
`parms_mcmc`	Object containing informations of sampled parameters
`imputations_mcmc`	Object containing informations of imputed datasets

Note

The `coef` and `vcov` methods can be used to extract coefficients and the corresponding covariance matrix, respectively. Standard errors for a fitted object `mod` can be extracted by making use of the **survey** package and the statement `survey::SE(mod)`.

Author(s)

Alexander Robitzsch

References

Allison, P. D. (2012). *Handling missing data by maximum likelihood*. SAS Global Forum 2012.

Bartlett, J. W., & Morris, T. P. (2015) Multiple imputation of covariates by substantive-model compatible fully conditional specification. *Stata Journal, 15*(2), 437-456.

Bartlett, J. W., Seaman, S. R., White, I. R., Carpenter, J. R., & Alzheimer's Disease Neuroimaging Initiative (2015). Multiple imputation of covariates by fully conditional specification: Accommodating the substantive model. *Statistical Methods in Medical Research, 24*(4), 462-487. doi: 10.1177/0962280214521348

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

Ibrahim, J. G. (1990). Incomplete data in generalized linear models. *Journal of the American Statistical Association, 85*(411), 765-769. doi: 10.1080/01621459.1990.10474938

Ibrahim, J. G., Chen, M. H., Lipsitz, S. R., & Herring, A. H. (2005). Missing-data methods for generalized linear models: A comparative review. *Journal of the American Statistical Association, 100*(469), 332-346. doi: 10.1198/016214504000001844

Jamshidian, M., & Jennrich, R. I. (2000). Standard errors for EM estimation. *Journal of the Royal Statistical Society (Series B), 62*(2), 257-270. doi: 10.1111/14679868.00230

Keller, B. T., & Enders, C. K. (2018). *Blimp user's manual*. Los Angeles, CA. *http://www.appliedmissingdata.com/multilevel-imputation.html*

Lee, M. C., & Mitra, R. (2016). Multiply imputing missing values in data sets with mixed measurement scales using a sequence of generalised linear models. *Computational Statistics & Data Analysis, 95*(24), 24-38. doi: 10.1016/j.csda.2015.08.004

Luedtke, O., Robitzsch, A., & West, S. (2020a). Analysis of interactions and nonlinear effects with missing data: A factored regression modeling approach using maximum likelihood estimation. *Multivariate Behavioral Research, 55*(3), 361-381. doi: 10.1080/00273171.2019.1640104

Luedtke, O., Robitzsch, A., & West, S. (2020b). Regression models involving nonlinear effects with missing data: A sequential modeling approach using Bayesian estimation. *Psychological Methods, 25*(2), 157-181. doi: 10.1037/met0000233

Zhang, Q., & Wang, L. (2017). Moderation analysis with missing data in the predictors. *Psychological Methods, 22*(4), 649-666. doi: 10.1037/met0000104

See Also

See also the **icdGLM** package for estimation of generalized linear models with incomplete discrete covariates.

The imputation of covariates in substantive models with interactions or nonlinear terms can be also conducted with the **JointAI** package which is a wrapper to the JAGS software (see Erler et al., 2016). This package is also based on a sequential modelling approach.

The **jomo** package also accommodates substantive models (`jomo::jomo.lm`) based on a joint modeling framework.

Substantive model compatible imputation based on fully conditional specification can be found in the **smcfcs** package (see Bartlett et al., 2015; Bartlett & Morris, 2015) or the Blimp stand-alone software (Keller & Enders, 2018).

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Simulated example linear regression with interaction effects
#############################################################################

# The interaction model stats::lm( Y ~ X + Z + X:Z) is of substantive interest.
# There can be missing values in all variables.

data(data.mb01)
dat <- data.mb01$missing

#****************************************
# Model 1: ML approach

#--- specify models

# define integration nodes
xnodes <- seq(-4,4,len=11)        # nodes for X
ynodes <- seq(-10,10,len=13)
  # nodes for Y. These ynodes are not really necessary for this dataset because
  # Y has no missing values.

# define model for dependent variable Y
dep <- list("model"="linreg", "formula"=Y ~  X*Z, "nodes"=ynodes )

# model P(X|Z)
ind_X <- list( "model"="linreg", "formula"=X ~ Z, "nodes"=xnodes )
# all models for covariates
ind <- list( "X"=ind_X )

#--- estimate model with numerical integration
mod1 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind )
summary(mod1)

# extract some informations
coef(mod1)
vcov(mod1)
logLik(mod1)

#****************************************
# Model 2: Fully Bayesian approach / Multiple Imputation

#--- define models
dep <- list("model"="linreg", "formula"=Y ~  X*Z )
ind_X <- list( "model"="linreg", "formula"=X ~ Z  )
ind_Z <- list( "model"="linreg", "formula"=Z ~ 1  )
ind <- list( "X"=ind_X, Z=ind_Z)

#--- estimate model
mod2 <- mdmb::frm_fb(dat, dep, ind, burnin=200, iter=1000)
summary(mod2)
```

```
#* plot parameters
plot(mod2)

#--- create list of multiply imputed datasets
datlist <- mdmb::frm2datlist(mod2)
# convert into object of class mids
imp2 <- miceadds::datlist2mids(datlist)
# estimate linear model on multiply imputed datasets
mod2c <- with(imp2, stats::lm( Y ~ X*Z ) )
summary( mice::pool(mod2c) )

#*****************************************
# Model 3: Multiple imputation in jomo package

library(jomo)

# impute with substantive model containing interaction effects
formula <- Y ~ X*Z
imp <- jomo::jomo.lm( formula=formula, data=dat, nburn=100, nbetween=100)

# convert to object of class mids
datlist <- miceadds::jomo2mids( imp )
# estimate linear model
mod3 <- with(datlist, lm( Y ~ X*Z ) )
summary( mice::pool(mod3) )

################################################################################
# EXAMPLE 2: Simulated example logistic regression with interaction effects
################################################################################

# Interaction model within a logistic regression Y ~ X + Z + X:Z
# Y and Z are dichotomous variables.

# attach data
data(data.mb02)
dat <- data.mb02$missing

#*****************************************
# Model 1: ML approach

#--- specify model

# define nodes
xnodes <- seq(-5,5,len=15)  # X - normally distributed variable
ynodes <- c(0,1)                # Y and Z dichotomous variable

# model P(Y|X,Z) for dependent variable
dep <- list("model"="logistic", "formula"=Y ~  X*Z, "nodes"=ynodes )
# model P(X|Z)
ind_X <- list( "model"="linreg", "formula"=X ~ Z, "nodes"=xnodes )
# model P(Z)
ind_Z <- list( "model"="logistic", "formula"=Z ~ 1, "nodes"=ynodes )
ind <- list(  "Z"=ind_Z, "X"=ind_X )
```

```
#--- estimate model with numerical integration
mod1 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind )
summary(mod1)

#****************************************
# Model 2: Fully Bayesian approach

#--- specify model
dep <- list("model"="logistic", "formula"=Y ~  X*Z  )
ind_X <- list( "model"="linreg", "formula"=X ~ Z )
ind_Z <- list( "model"="logistic", "formula"=Z ~ 1 )
ind <- list(  "Z"=ind_Z, "X"=ind_X )

#--- Bayesian estimation
mod2 <- mdmb::frm_fb(dat=dat, dep=dep, ind=ind, burnin=500, iter=1000 )
summary(mod2)

##############################################################################
# EXAMPLE 3: Confirmatory factor analysis
##############################################################################

# A latent variable can be considered as missing data and the 'frm_em' function
# is used to estimate the latent variable model.

#--- simulate data
N <- 1000
set.seed(91834)
# latent variable
theta <- stats::rnorm(N)
# simulate items
y1 <- 1.5 + 1*theta + stats::rnorm(N, sd=.7 )
y2 <- 1.9 + .7*theta + stats::rnorm(N, sd=1 )
y3 <- .9 + .7*theta + stats::rnorm(N, sd=.2 )
dat <- data.frame(y1,y2,y3)
dat$theta <- NA

#****************************************
# Model 1: ML approach

#--- define model
nodes <- seq(-4,4,len=21)
ind_y1 <- list("model"="linreg", "formula"=y1 ~  offset(1*theta),
               "nodes"=nodes )
ind_y2 <- list( "model"="linreg", "formula"=y2 ~ theta, "nodes"=nodes,
               "coef_inits"=c(NA,1) )
ind_y3 <- list( "model"="linreg", "formula"=y3 ~ theta, "nodes"=nodes,
                "coef_inits"=c(1,1) )
dep <- list( "model"="linreg", "formula"=theta ~ 0, "nodes"=nodes )
ind <- list( "y1"=ind_y1,  "y2"=ind_y2, "y3"=ind_y3)

#*** estimate model with mdmb::frm_em
mod1 <- mdmb::frm_em(dat, dep, ind)
```

```
summary(mod1)

#*** estimate model in lavaan
library(lavaan)
lavmodel <- "
     theta=~ 1*y1 + y2 + y3
     theta ~~ theta
     "
mod1b <- lavaan::cfa( model=lavmodel, data=dat )
summary(mod1b)

# compare likelihood
logLik(mod1)
logLik(mod1b)

#############################################################################
# EXAMPLE 4: Rasch model
#############################################################################

#--- simulate data
set.seed(91834)
N <- 500
# latent variable
theta0 <- theta <- stats::rnorm(N)
# number of items
I <- 7
dat <- sirt::sim.raschtype( theta, b=seq(-1.5,1.5,len=I) )
colnames(dat) <- paste0("I",1:I)
dat$theta <- NA

#****************************************
# Model 1: ML approach

#--- define model
nodes <- seq(-4,4,len=13)
dep <- list("model"="linreg", "formula"=theta ~  0, "nodes"=nodes )
ind <- list()
for (ii in 1:I){
    ind_ii <- list( "model"="logistic", formula=
                  stats::as.formula( paste0("I",ii, " ~ offset(1*theta)") ) )
    ind[[ii]] <- ind_ii
}
names(ind) <- colnames(dat)[-(I+1)]

#--- estimate Rasch model with mdmb::frm_em
mod1 <- mdmb::frm_em(dat, dep, ind )
summary(mod1)

#--- estmate Rasch model with sirt package
library(sirt)
mod2 <- sirt::rasch.mml2( dat[,-(I+1)], theta.k=nodes, use.freqpatt=FALSE)
summary(mod2)
```

```
#** compare estimated parameters
round(cbind(coef(mod1), c( mod2$sd.trait, -mod2$item$thresh[ seq(I,1)] ) ), 3)

#############################################################################
# EXAMPLE 5: Regression model with measurement error in covariates
#############################################################################

#--- simulate data
set.seed(768)
N <- 1000
# true score
theta <- stats::rnorm(N)
# heterogeneous error variance
var_err <- stats::runif(N, .5, 1)
# simulate observed score
x <- theta + stats::rnorm(N, sd=sqrt(var_err) )
# simulate outcome
y <- .3 + .7 * theta + stats::rnorm( N, sd=.8 )
dat0 <- dat <- data.frame( y=y, x=x, theta=theta )

#*** estimate model with true scores (which are unobserved in real datasets)
mod0 <- stats::lm( y ~ theta, data=dat0 )
summary(mod0)

#****************************************
# Model 1: Model-based approach

#--- specify model
dat$theta <- NA
nodes <- seq(-4,4,len=15)
dep <- list( "model"="linreg", "formula"=y ~ theta, "nodes"=nodes,
                "coef_inits"=c(NA, .4 ) )
ind <- list()
ind[["theta"]] <- list( "model"="linreg", "formula"=theta ~ 1,
                         "nodes"=nodes  )
ind[["x"]] <- list( "model"="linreg", "formula"=x ~ 0 + offset(theta),
                        "nodes"=nodes )
# assumption of heterogeneous known error variance
ind[["x"]]$sigma_fixed <- sqrt( var_err )

#--- estimate regression model
mod1 <- mdmb::frm_em(dat, dep, ind )
summary(mod1)

#****************************************
# Model 2: Fully Bayesian estimation

#--- specify model
dep <- list( "model"="linreg", "formula"=y ~ theta )
ind <- list()
ind[["theta"]] <- list( "model"="linreg", "formula"=theta ~ 1  )
ind[["x"]] <- list( "model"="linreg", "formula"=x ~ 0 + offset(theta) )
# assumption of heterogeneous known error variance
```

```
ind[["x"]]$sigma_fixed <- sqrt( var_err )
data_init <- dat
data_init$theta <- dat$x

# estimate model
mod2 <- mdmb::frm_fb(dat, dep, ind, burnin=200, iter=1000, data_init=data_init)
summary(mod2)
plot(mod2)

#############################################################################
# EXAMPLE 6: Non-normally distributed covariates:
#            Positive values with Box-Cox transformation
#############################################################################

# simulate data with chi-squared distributed covariate from
# regression model Y ~ X
set.seed(876)
n <- 1500
df <- 2
x <- stats::rchisq( n, df=df )
x <- x / sqrt( 2*df)
y <- 0 + 1*x
R2 <- .25    # explained variance
y <- y + stats::rnorm(n, sd=sqrt( (1-R2)/R2 * 1) )
dat0 <- dat <- data.frame( y=y, x=x )

# simulate missing responses
prop_miss <- .5
cor_miss <- .7
resp_tend <- cor_miss*(dat$y-mean(y) )/ stats::sd(y) +
                 stats::rnorm(n, sd=sqrt( 1 - cor_miss^2) )
dat[ resp_tend < stats::qnorm( prop_miss ), "x" ] <- NA
summary(dat)

#-- complete data
mod0 <- stats::lm( y ~ x, data=dat0 )
summary(mod0)
#-- listwise deletion
mod1 <- stats::lm( y ~ x, data=dat )
summary(mod1)

# normal distribution assumption for frm

# define models
dep <- list("model"="linreg", "formula"=y ~  x )
# normally distributed data
ind_x1 <- list( "model"="linreg", "formula"=x ~ 1 )
# Box-Cox normal distribution
ind_x2 <- list( "model"="bctreg", "formula"=x ~ 1,
              nodes=c( seq(0.05, 3, len=31), seq( 3.5, 9, by=.5 ) ) )
ind1 <- list( "x"=ind_x1 )
ind2 <- list( "x"=ind_x2 )
```

```
#--- incorrect normal distribution assumption
mod1 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind1 )
summary(mod1)

#--- model chi-square distribution of predictor with Box-Cox transformation
mod2 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind2 )
summary(mod2)

#############################################################################
# EXAMPLE 7: Latent interaction model
#############################################################################

# A latent interaction model Y ~ FX + FZ is of interest. Y is directly observed,
# FX and FZ are both indirectly observed by three items

#--- simulate data
N <- 1000
set.seed(987)
# latent variable
FX <- stats::rnorm(N)
FZ <- stats::rnorm(N)
# simulate items
x1 <- 1.5 + 1*FX + stats::rnorm(N, sd=.7 )
x2 <- 1.9 + .7*FX + stats::rnorm(N, sd=1 )
x3 <- .9 + .7*FX + stats::rnorm(N, sd=.2 )
z1 <- 1.5 + 1*FZ + stats::rnorm(N, sd=.7 )
z2 <- 1.9 + .7*FZ + stats::rnorm(N, sd=1 )
z3 <- .9 + .7*FZ + stats::rnorm(N, sd=.2 )
dat <- data.frame(x1,x2,x3,z1,z2,z3)
dat$FX <- NA
dat$FZ <- NA
dat$y <- 2 + .5*FX + .3*FZ + .4*FX*FZ + rnorm( N, sd=1 )

# estimate interaction model with ML

#--- define model
nodes <- seq(-4,4,len=11)
ind_x1 <- list("model"="linreg", "formula"=x1 ~  offset(1*FX),
                "nodes"=nodes )
ind_x2 <- list( "model"="linreg", "formula"=x2 ~ FX, "nodes"=nodes,
                "coef_inits"=c(NA,1) )
ind_x3 <- list( "model"="linreg", "formula"=x3 ~ FX, "nodes"=nodes,
                 "coef_inits"=c(1,1) )
ind_FX <- list( "model"="linreg", "formula"=FX ~ 0, "nodes"=nodes )
ind_z1 <- list("model"="linreg", "formula"=z1 ~  offset(1*FZ),
                "nodes"=nodes )
ind_z2 <- list( "model"="linreg", "formula"=z2 ~ FZ, "nodes"=nodes,
                "coef_inits"=c(NA,1) )
ind_z3 <- list( "model"="linreg", "formula"=z3 ~ FZ, "nodes"=nodes,
                 "coef_inits"=c(1,1) )
ind_FZ <- list( "model"="linreg", "formula"=FZ ~ 0 + FX, "nodes"=nodes )
ind <- list( "x1"=ind_x1,  "x2"=ind_x2, "x3"=ind_x3, "FX"=ind_FX,
           "z1"=ind_z1,  "z2"=ind_z2, "z3"=ind_z3, "FX"=ind_FZ )
```

```
dep <- list( "model"="linreg", formula=y ~ FX+FZ+FX*FZ, "coef_inits"=c(1,.2,.2,0) )

#*** estimate model with mdmb::frm_em
mod1 <- mdmb::frm_em(dat, dep, ind)
summary(mod1)

#############################################################################
# EXAMPLE 8: Non-ignorable data in Y
#############################################################################

# regression Y ~ X in which Y is missing depending Y itself

library(mvtnorm)
cor_XY <- .4        # correlation between X and Y
prop_miss <- .5     # missing proportion
cor_missY <- .7     # correlation with missing propensity
N <- 3000           # sample size

#----- simulate data
set.seed(790)
Sigma <- matrix( c(1, cor_XY, cor_XY, 1), 2, 2 )
mu <- c(0,0)
dat <- mvtnorm::rmvnorm( N, mean=mu, sigma=Sigma )
colnames(dat) <- c("X","Y")
dat <- as.data.frame(dat)

#-- generate missing responses on Y depending on Y itself
y1 <- dat$Y
miss_tend <- cor_missY * y1  + rnorm(N, sd=sqrt( 1 - cor_missY^2) )
dat$Y[ miss_tend < quantile( miss_tend, prop_miss ) ] <- NA

#--- ML estimation under assumption of ignorability
nodes <- seq(-5,5,len=15)
dep <- list("model"="linreg", "formula"=Y ~  X, "nodes"=nodes )
ind_X <- list( "model"="linreg", "formula"=X ~ 1, "nodes"=nodes )
ind <- list( "X"=ind_X )
mod1 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind)
summary(mod1)

#--- ML estimation under assumption with specifying a model for non-ignorability
#    for response indicator resp_Y
dat$resp_Y <- 1* ( 1 - is.na(dat$Y) )
dep <- list("model"="linreg", "formula"=Y ~  X, "nodes"=nodes )
ind_X <- list( "model"="linreg", "formula"=X ~ 1, "nodes"=nodes )
ind_respY <- list( "model"="logistic", "formula"=resp_Y ~ Y, "nodes"=nodes )
ind <- list( "X"=ind_X, "resp_Y"=ind_respY )
mod2 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind)
summary(mod2)

#############################################################################
# EXAMPLE 9: Ordinal variables: Graded response model
#############################################################################
```

```
#--- simulate data
N <- 2000
set.seed(91834)
# latent variable
theta <- stats::rnorm(N)
# simulate items
y1 <- 1*theta + stats::rnorm(N)
y2 <- .7*theta + stats::rnorm(N)
y3 <- .7*theta + stats::rnorm(N)
# discretize variables
y1 <- as.numeric( cut( y1, breaks=c(-Inf, -.5, 0.4, 1, Inf ) ) ) - 1
y2 <- as.numeric( cut( y2, breaks=c(-Inf, 0.3, 1, Inf ) ) ) - 1
y3 <- as.numeric( cut( y3, breaks=c(-Inf, .2, Inf ) ) ) - 1
# define dataset
dat <- data.frame(y1,y2,y3)
dat$theta <- NA

#*****************************************
# Model 1: Fully Bayesian estimation

#--- define model
ind_y1 <- list( "model"="oprobit", "formula"=y1 ~  offset(1*theta) )
ind_y2 <- list( "model"="oprobit", "formula"=y2 ~ theta )
ind_y3 <- list( "model"="oprobit", "formula"=y3 ~ theta )
dep <- list( "model"="linreg", "formula"=theta ~ 0  )
ind <- list( "y1"=ind_y1,  "y2"=ind_y2, "y3"=ind_y3)
# initial data
data_init <- dat
data_init$theta <- as.numeric( scale(dat$y1) ) + stats::rnorm(N, sd=.4 )

#-- estimate model
iter <- 3000; burnin <- 1000
mod1 <- mdmb::frm_fb(dat=dat, dep=dep, ind=ind, data_init=data_init,
                  iter=iter, burnin=burnin)
summary(mod1)
plot(mod1)

#############################################################################
# EXAMPLE 10: Imputation for missig predictors in models with interaction
#             effects in multilevel regression models
#############################################################################

library(miceadds)
data(data.mb04, package="mdmb")
dat <- data.mb04

#*** model specification
mcmc_iter <- 4   # number of MCMC iterations for model parameter sampling
model_formula <- y ~ cwc(x, idcluster) + gm(x, idcluster) + w + w*cwc(x, idcluster) +
                  w*gm(x, idcluster) + ( 1 + cwc(x, idcluster) | idcluster)
dep <- list("model"="mlreg", "formula"=model_formula,
               R_args=list(iter=mcmc_iter, outcome="normal") )
ind_x <- list( "model"="mlreg", "formula"=x ~ w + (1|idcluster), R_args=list(iter=mcmc_iter),
```

```
                  sampling_level="idcluster" )
# group means of x are involved in the outcome model. Therefore, Metropolis-Hastings
# sampling of missing values in x should be conducted at the level of clusters,
# i.e. specifying sampling_level
ind <- list("x"=ind_x)

# --- estimate model
mod1 <- mdmb::frm_fb(dat, dep, ind, aggregation=TRUE)
# argument aggregation is necessary because group means are involved in regression formulas

#-------------
#*** imputation of a continuous level-2 variable w

#  create artificially some missings on w
dat[ dat$idcluster %%3==0, "w" ] <- NA

# define level-2 model with argument variable_level
ind_w <- list( "model"="linreg", "formula"=w ~ 1, "variable_level"="idcluster" )
ind <- list( x=ind_x, w=ind_w)

#* conduct imputations
mod2 <- mdmb::frm_fb(dat, dep, ind, aggregation=TRUE)
summary(mod2)

#--- Model 1 with user-defined prior distributions for covariance matrices
model_formula <- y ~ cwc(x, idcluster) + gm(x, idcluster) + w + w*cwc(x, idcluster) +
                   w*gm(x, idcluster) + ( 1 + cwc(x, idcluster) | idcluster)

# define scale degrees of freedom (nu) and scale matrix (S) for inverse Wishart distribution
psi_nu0_list <- list( -3 )
psi_S0_list <- list( diag(0,2) )
dep <- list("model"="mlreg", "formula"=model_formula,
               R_args=list(iter=mcmc_iter, outcome="normal",
               psi_nu0_list=psi_nu0_list, psi_S0_list=psi_S0_list ) )

# define nu and S parameters for covariate model
psi_nu0_list <- list( .4 )
psi_S0_list <- list( matrix(.2, nrow=1, ncol=1) )
ind_x <- list( "model"="mlreg", "formula"=x ~ w + (1|idcluster),
                   R_args=list(iter=mcmc_iter, psi_nu0_list=psi_nu0_list,
                            psi_S0_list=psi_S0_list),
                  sampling_level="idcluster" )
ind <- list("x"=ind_x)

# --- estimate model
mod3 <- mdmb::frm_fb(dat, dep, ind, aggregation=TRUE)

#############################################################################
# EXAMPLE 11: Bounded variable combined with Yeo-Johnson transformation
#############################################################################

#*** simulate data
set.seed(876)
```

```
n <- 1500
x <- mdmb::ryjt_scaled( n, location=-.2, shape=.8, lambda=.9, probit=TRUE)
R2 <- .25    # explained variance
y <- 1*x + stats::rnorm(n, sd=sqrt( (1-R2)/R2 * stats::var(x)) )
dat0 <- dat <- data.frame( y=y, x=x )

# simulate missing responses
prop_miss <- .5
cor_miss <- .7
resp_tend <- cor_miss*(dat$y-mean(y) )/ stats::sd(y) +
                stats::rnorm(n, sd=sqrt( 1 - cor_miss^2) )
dat[ resp_tend < stats::qnorm(prop_miss), "x" ] <- NA
summary(dat)

#*** define models
dep <- list("model"="linreg", "formula"=y ~  x )
# distribution according to Yeo-Johnson transformation
ind_x1 <- list( "model"="yjtreg", "formula"=x ~ 1 )
# distribution according to Probit Yeo-Johnson transformation
ind_x2 <- list( "model"="yjtreg", "formula"=x ~ 1, R_args=list("probit"=TRUE ) )
ind1 <- list( "x"=ind_x1 )
ind2 <- list( "x"=ind_x2 )

#--- complete data
mod0 <- stats::lm( y~x, data=dat0)
summary(mod0)

#--- Yeo-Johnson normal distribution (for unbounded variables)
mod1 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind1 )
summary(mod1)

#--- Probit Yeo-Johnson normal distribution (for bounded variable on (0,1))
mod2 <- mdmb::frm_em(dat=dat, dep=dep, ind=ind2)
summary(mod2)

#--- same model, but MCMC estimation
mod3 <- mdmb::frm_fb(dat, dep, ind=ind2, burnin=2000, iter=5000)
summary(mod3)
plot(mod3)

############################################################################
# EXAMPLE 12: Yeo-Johnson transformation with estimated degrees of freedom
############################################################################

#*** simulate data
set.seed(876)
n <- 1500
x <- mdmb::ryjt_scaled( n, location=-.2, shape=.8, lambda=.9, df=10 )
R2 <- .25    # explained variance
y <- 1*x + stats::rnorm(n, sd=sqrt( (1-R2)/R2 * stats::var(x)) )
dat0 <- dat <- data.frame( y=y, x=x )

# simulate missing responses
```

```
prop_miss <- .5
cor_miss <- .7
resp_tend <- cor_miss*(dat$y-mean(y) )/ stats::sd(y) +
                 stats::rnorm(n, sd=sqrt( 1-cor_miss^2) )
dat[ resp_tend < stats::qnorm(prop_miss), "x" ] <- NA
summary(dat)

#*** define models
dep <- list("model"="linreg", "formula"=y ~  x )
# specify distribution with estimated degrees of freedom
ind_x <- list( "model"="yjtreg", "formula"=x ~ 1, R_args=list(est_df=TRUE ) )
ind <- list( "x"=ind_x )

#--- Yeo-Johnson t distribution
mod1 <- mdmb::frm_fb(dat=dat, dep=dep, ind=ind, iter=3000, burnin=1000 )
summary(mod1)

## End(Not run)
```

`mdmb_regression` *Several Regression Models with Prior Distributions and Sampling Weights*

Description

Several regression functions which allow for sampling weights and prior distributions.

The function `yjt_regression` performs a linear regression in which the response variable is transformed according to the Yeo-Johnson transformation (Yeo & Johnson, 2000; see `yjt_dist`) and the residuals are distributed following the scaled t distribution. The degrees of freedom of the t distribution can be fixed or estimated (`est_df=TRUE`). The function `bct_regression` has same functionality like the Yeo-Johnson transformation but employs a Box-Cox transformation of the outcome variable.

The Yeo-Johnson transformation can be extended by a probit transformation (`probit=TRUE`) to cover the case of bounded variables on $[0, 1]$.

The function `logistic_regression` performs logistic regression for dichotomous data.

The function `oprobit_regression` performs ordinal probit regression for ordinal polytomous data.

Usage

```
#---- linear regression with Yeo-Johnson transformed scaled t distribution
yjt_regression(formula, data, weights=NULL, beta_init=NULL, beta_prior=NULL,
     df=Inf, lambda_fixed=NULL, probit=FALSE, est_df=FALSE, df_min=0.5, df_max=100,
        use_grad=2, h=1e-5, optimizer="optim", maxiter=300, control=NULL)

## S3 method for class 'yjt_regression'
coef(object, ...)
## S3 method for class 'yjt_regression'
```

```
logLik(object, ...)
## S3 method for class 'yjt_regression'
predict(object, newdata=NULL, trafo=TRUE,  ...)
## S3 method for class 'yjt_regression'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'yjt_regression'
vcov(object, ...)

#---- linear regression with Box-Cox transformed scaled t distribution
bct_regression(formula, data, weights=NULL, beta_init=NULL, beta_prior=NULL,
        df=Inf, lambda_fixed=NULL, est_df=FALSE, use_grad=2, h=1e-5,
        optimizer="optim", maxiter=300, control=NULL)

## S3 method for class 'bct_regression'
coef(object, ...)
## S3 method for class 'bct_regression'
logLik(object, ...)
## S3 method for class 'bct_regression'
predict(object, newdata=NULL, trafo=TRUE, ...)
## S3 method for class 'bct_regression'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'bct_regression'
vcov(object, ...)

#---- logistic regression
logistic_regression(formula, data, weights=NULL, beta_init=NULL,
         beta_prior=NULL, use_grad=2, h=1e-5, optimizer="optim", maxiter=300,
         control=NULL)

## S3 method for class 'logistic_regression'
coef(object, ...)
## S3 method for class 'logistic_regression'
logLik(object, ...)
## S3 method for class 'logistic_regression'
predict(object, newdata=NULL, ...)
## S3 method for class 'logistic_regression'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'logistic_regression'
vcov(object, ...)

#---- ordinal probit regression
oprobit_regression(formula, data, weights=NULL, beta_init=NULL,
        use_grad=2, h=1e-5, optimizer="optim", maxiter=300,
        control=NULL, control_optim_fct=NULL)

## S3 method for class 'oprobit_regression'
coef(object, ...)
## S3 method for class 'oprobit_regression'
```

```
logLik(object, ...)
## S3 method for class 'oprobit_regression'
predict(object, newdata=NULL, ...)
## S3 method for class 'oprobit_regression'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'oprobit_regression'
vcov(object, ...)
```

Arguments

formula	Formula
data	Data frame. The dependent variable must be coded as 0 and 1.
weights	Optional vector of sampling weights
beta_init	Optional vector of initial regression coefficients
beta_prior	Optional list containing priors of all parameters (see Examples for definition of this list).
df	Fixed degrees of freedom for scaled t distribution
lambda_fixed	Optional fixed value for λ for scaled t distribution with Yeo-Johnson transformation
probit	Logical whether probit transformation should be employed for bounded outcome in `yjt_regression`
est_df	Logical indicating whether degrees of freedom in t distribution should be estimated.
df_min	Minimum value for estimated degrees of freedom
df_max	Maximum value for estimated degrees of freedom
use_grad	Computation method for gradients in `stats::optim`. The value `0` is the internal approximation of `stats::optim` and applies the settings in **mdmb** ($\leq$0.3). The specification `use_grad=1` uses the calculation of the gradient in `CDM::numerical_Hessian`. The value `2` is usually the most efficient calculation of the gradient.
h	Numerical differentiation parameter.
optimizer	Type of optimizer to be chosen. Options are `"nlminb"` (`stats::nlminb`) and the default `"optim"` (`stats::optim`)
maxiter	Maximum number of iterations
control	Optional arguments to be passed to optimization function (`stats::nlminb`) or `stats::optim`
control_optim_fct	Optional control argument for gradient in optimization
object	Object of class `logistic_regression`
newdata	Design matrix for `predict` function
trafo	Logical indicating whether fitted values should be on the transformed metric (`trafo=TRUE`) or the original metric (`trafo=FALSE`)
digits	Number of digits for rounding
file	File name if the `summary` output should be sunk into a file.
...	Further arguments to be passed.

Value

List containing values

`coefficients`	Estimated regression coefficients
`vcov`	Estimated covariance matrix
`partable`	Parameter table
`y`	Vector of values of dependent variable
`Xdes`	Design matrix
`weights`	Sampling weights
`fitted.values`	Fitted values in metric of probabilities
`linear.predictor`	Fitted values in metric of logits
`loglike`	Log likelihood value
`logprior`	Log prior value
`logpost`	Log posterior value
`deviance`	Deviance
`loglike_case`	Case-wise likelihood
`ic`	Information criteria
`R2`	Pseudo R-square value according to McKelvey and Zavoina

Author(s)

Alexander Robitzsch

References

McKelvey, R., & Zavoina, W. (1975). A statistical model for the analysis of ordinal level dependent variables. *Journal of Mathematical Sociology, 4*(1), 103-120. doi: 10.1080/0022250X.1975.9989847

Yeo, I.-K., & Johnson, R. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika, 87*(4), 954-959. doi: 10.1093/biomet/87.4.954

See Also

See `yjt_dist` or `car::yjPower` for functions for the Yeo-Johnson transformation.

See `stats::lm` and `stats::glm` for linear and logistic regression models.

Examples

```
#############################################################################
# EXAMPLE 1: Simulated example logistic regression
#############################################################################

#--- simulate dataset
set.seed(986)
N <- 500
```

```
x <- stats::rnorm(N)
y <- 1*( stats::runif(N) < stats::plogis( -0.8 + 1.2 * x ) )
data <- data.frame( x=x, y=y )

#--- estimate logistic regression with mdmb::logistic_regression
mod1 <- mdmb::logistic_regression( y ~ x, data=data )
summary(mod1)

## Not run:
#--- estimate logistic regression with stats::glm
mod1b <- stats::glm( y ~ x, data=data, family="binomial")
summary(mod1b)

#--- estimate logistic regression with prior distributions
b0 <- list( "dnorm", list(mean=0, sd=100) )  # first parameter
b1 <- list( "dcauchy", list(location=0, scale=2.5) )   # second parameter
beta_priors <- list( b0, b1 )  # order in list defines priors for parameters
#* estimation
mod2 <- mdmb::logistic_regression( y ~ x, data=data, beta_prior=beta_priors )
summary(mod2)

#############################################################################
# EXAMPLE 2: Yeo-Johnson transformed scaled t regression
#############################################################################

#*** create simulated data
set.seed(9865)
n <- 1000
x <- stats::rnorm(n)
y <- .5 + 1*x + .7*stats::rt(n, df=8 )
y <- mdmb::yj_antitrafo( y, lambda=.5 )
dat <- data.frame( y=y, x=x )
# display data
graphics::hist(y)

#--- Model 1: fit regression model with transformed normal distribution (df=Inf)
mod1 <- mdmb::yjt_regression( y ~ x, data=dat )
summary(mod1)

#--- Model 2: fit regression model with transformed scaled t distribution (df=10)
mod2 <- mdmb::yjt_regression( y ~ x, data=dat, df=10)
summary(mod2)

#--- Model 3: fit regression model with transformed normal distribution (df=Inf)
#             and fixed transformation parameter lambda of .5
mod3 <- mdmb::yjt_regression( y ~ x, data=dat, lambda_fixed=.5)
summary(mod3)

#--- Model 4: fit regression model with transformed normal distribution (df=Inf)
#             and fixed transformation parameter lambda of 1
#             -> This model corresponds to least squares regression
mod4 <- mdmb::yjt_regression( y ~ x, data=dat, lambda_fixed=1)
summary(mod4)
```

```
# fit with lm function
mod4b <- stats::lm( y ~ x, data=dat )
summary(mod4b)

#--- Model 5: fit regression model with estimated degrees of freedom
mod5 <- mdmb::yjt_regression( y ~ x, data=dat, est_df=TRUE)
summary(mod5)

#** compare log-likelihood values
logLik(mod1)
logLik(mod2)
logLik(mod3)
logLik(mod4)
logLik(mod4b)
logLik(mod5)

############################################################################
# EXAMPLE 3: Regression with Box-Cox and Yeo-Johnson transformations
############################################################################

#*** simulate data
set.seed(985)
n <- 1000
x <- stats::rnorm(n)
y <- .5 + 1*x + stats::rnorm(n, sd=.7 )
y <- mdmb::bc_antitrafo( y, lambda=.5 )
dat <- data.frame( y=y, x=x )

#--- Model 1: fit regression model with Box-Cox transformation
mod1 <- mdmb::bct_regression( y ~ x, data=dat )
summary(mod1)
#--- Model 2: fit regression model with Yeo-Johnson transformation
mod2 <- mdmb::yjt_regression( y ~ x, data=dat )
summary(mod2)
#--- compare fit
logLik(mod1)
logLik(mod2)

############################################################################
# EXAMPLE 4: Ordinal probit regression
############################################################################

#--- simulate data
set.seed(987)
N <- 1500
x <- stats::rnorm(N)
z <- stats::rnorm(N)
# regression coefficients
b0 <- -.5 ; b1 <- .6 ; b2 <- .1
# vector of thresholds
thresh <- c(-1, -.3, 1)
yast <- b0 + b1 * x + b2*z + stats::rnorm(N)
```

```
y <- as.numeric( cut( yast, c(-Inf,thresh,Inf) ) ) - 1
dat <- data.frame( x=x, y=y, z=z )

#--- probit regression
mod <- mdmb::oprobit_regression( formula=y ~ x + z + I(x*z), data=dat)
summary(mod)

## End(Not run)
```

`offset_values_extract` *Extracts Offset Values*

Description

Extracts offset values by applying a formula with an offset term to a data frame.

Usage

```
offset_values_extract(formula, data)
```

Arguments

`formula`	An R Formula
`data`	Data frame

Value

Vector containing offset values

See Also

`stats::offset`, `stats::model.offset`

Examples

```
#############################################################################
# EXAMPLE 1: Toy example for extraction of offset values
#############################################################################

data(data.ma01, package="miceadds")
dat <- data.ma01

dat1 <- mdmb::offset_values_extract( formula=~ migrant + offset(books), data=dat )
dat1[1:5]
  ##  [1] 6 6 5 2 6
```

oprobit_dist *Ordinal Probit Models*

Description

Fits and evaluates the ordinal probit model.

Usage

```
#---- ordinal probit model
doprobit(x, thresh, max_val=99)

fit_oprobit(x, par_init=NULL, weights=NULL)
## S3 method for class 'fit_oprobit'
coef(object, ...)
## S3 method for class 'fit_oprobit'
logLik(object, ...)
## S3 method for class 'fit_oprobit'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'fit_oprobit'
vcov(object, ...)
```

Arguments

x	Numeric vector
thresh	Vector of thresholds
max_val	Maximum value for computing thresholds
par_init	Optional vector of initial parameters
weights	Optional vector of sampling weights
object	Object of class fit_yjt_scaled or fit_t_scaled
digits	Number of digits used for rounding in summary
file	File name for the summary to be sunk into
...	Further arguments to be passed

Value

Vector or an object of fitted distribution depending on the called function

See Also

See oprobit_regression for fitting a regression model in which the response variable follows an ordinal probit model.

Examples

```
#############################################################################
# EXAMPLE 1: Fit an ordinal probit distribution
#############################################################################

#-- simulate data
set.seed(987)
N <- 1500
# define thresholds
thresh <- c(0,.3, .7, 1.6)
# latent continuous data
yast <- stats::rnorm(N)
# discretized ordinal data
x <- as.numeric( cut( yast, c(-Inf,thresh,Inf) ) ) - 1

#-- fit ordinal probit distribution
mod <- mdmb::fit_oprobit(x=x)
summary(mod)
logLik(mod)
vcov(mod)
```

`remove_NA_data_frame` *Removes Rows with Some Missing Entries in a Data Frame*

Description

Removes rows with some missing entries in a data frame for variables appearing in the R formula `formula`. This operation is also known as listwise deletion.

Usage

```
remove_NA_data_frame(formula, data, weights=NULL)
```

Arguments

`formula`	An R formula
`data`	Data frame
`weights`	Optional vector of sample weights

Value

Data frame with some rows removed according to missing data

See Also

`stats::model.matrix`

Examples

```
#############################################################################
# EXAMPLE 1: Removing rows in a data frame (listwise deletion)
#############################################################################

data(data.ma01, package="miceadds")
dat <- data.ma01

#*** remove rows with some missings according to a formula
dat1 <- mdmb::remove_NA_data_frame( formula=read ~ migrant + books, data=dat)

#*** remove rows according to two formulas
formula1 <- read ~ migrant + books
formula2 <- paredu ~ migrant + female
# create formula consisting of formula1 and formula2
formula3 <- paste( "~", deparse(formula1[[2]]), "+", deparse(formula1[[3]]),
             " + ", deparse(formula2[[2]]), "+", deparse(formula2[[3]])  )
dat2 <- mdmb::remove_NA_data_frame( formula=as.formula(formula3), data=dat)

dim(dat)
dim(dat1$data)
dim(dat2$data)
  ##  > dim(dat)
  ##  [1] 4073   11
  ##  > dim(dat1$data)
  ##  [1] 3371   11
  ##  > dim(dat2$data)
  ##  [1] 3090   11
```

---

yjt_dist *Scaled t Distribution with Yeo-Johnson and Box-Cox Transformations*

Description

Collection of functions for the Yeo-Johnson transformation (Yeo & Johnson, 2000) and the corresponding distribution family of scaled t distribution with and without Yeo-Johnson transformation (see Details). The Yeo-Johnson transformation can also be applied for bounded variables on $(0, 1)$ which uses a probit transformation (see Details; argument `probit`).

The Box-Cox transformation (`bc`; Sakia, 1992) can be applied for variables with positive values.

Usage

```
# Yeo-Johnson transformation and its inverse transformation
yj_trafo(y, lambda, use_rcpp=TRUE, probit=FALSE)
yj_antitrafo(y, lambda, probit=FALSE)

#---- scaled t distribution with Yeo-Johnson transformation
dyjt_scaled(x, location=0, shape=1, lambda=1, df=Inf, log=FALSE, probit=FALSE)
```

```
ryjt_scaled(n, location=0, shape=1, lambda=1, df=Inf, probit=FALSE)

fit_yjt_scaled(x, df=Inf, par_init=NULL, lambda_fixed=NULL, weights=NULL, probit=FALSE)
## S3 method for class 'fit_yjt_scaled'
coef(object, ...)
## S3 method for class 'fit_yjt_scaled'
logLik(object, ...)
## S3 method for class 'fit_yjt_scaled'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'fit_yjt_scaled'
vcov(object, ...)

# Box-Cox transformation and its inverse transformation
bc_trafo(y, lambda)
bc_antitrafo(y, lambda)

#---- scaled t distribution with Box-Cox transformation
dbct_scaled(x, location=0, shape=1, lambda=1, df=Inf, log=FALSE, check_zero=TRUE)
rbct_scaled(n, location=0, shape=1, lambda=1, df=Inf)

fit_bct_scaled(x, df=Inf, par_init=NULL, lambda_fixed=NULL, weights=NULL)
## S3 method for class 'fit_bct_scaled'
coef(object, ...)
## S3 method for class 'fit_bct_scaled'
logLik(object, ...)
## S3 method for class 'fit_bct_scaled'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'fit_bct_scaled'
vcov(object, ...)

#---- scaled t distribution
dt_scaled(x, location=0, shape=1, df=Inf, log=FALSE)
rt_scaled(n, location=0, shape=1, df=Inf)

fit_t_scaled(x, df=Inf, par_init=NULL, weights=NULL)
## S3 method for class 'fit_t_scaled'
coef(object, ...)
## S3 method for class 'fit_t_scaled'
logLik(object, ...)
## S3 method for class 'fit_t_scaled'
summary(object, digits=4, file=NULL, ...)
## S3 method for class 'fit_t_scaled'
vcov(object, ...)
```

Arguments

y	Numeric vector
lambda	Transformation parameter λ for Yeo-Johnson transformation

use_rcpp	Logical indicating whether **Rcpp** package should be used
probit	Logical indicating whether probit transformation should be applied for bounded variables on $(0, 1)$
x	Numeric vector
location	Location parameter of (transformed) scaled t distribution
shape	Shape parameter of (transformed) scaled t distribution
df	Degrees of freedom of (transformed) scaled t distribution
log	Logical indicating whether logarithm of the density should be computed
check_zero	Logical indicating whether check for inadmissible values should be conducted
n	Number of observations to be simulated
par_init	Optional vector of initial parameters
lambda_fixed	Optional value for fixed λ parameter
weights	Optional vector of sampling weights
object	Object of class `fit_yjt_scaled` or `fit_t_scaled`
digits	Number of digits used for rounding in `summary`
file	File name for the `summary` to be sunk into
...	Further arguments to be passed

Details

Let g_λ be the Yeo-Johnson transformation. A random variable X is distribution as Scaled t with Yeo-Johnson transformation with location μ, scale σ and transformation parameter λ iff $X = g_\lambda(\mu + \sigma Z)$ and Z is t distributed with `df` degrees of freedom.

For a bounded variable X on $(0, 1)$, the probit transformation Φ is applied such that $X = \Phi(g_\lambda(\mu + \sigma Z))$ with a t distributed variable Z.

For a Yeo-Johnson normally distributed variable, a normally distributed variable results in case of $\lambda = 1$. For a Box-Cox normally distributed variable, a normally distributed variable results for $\lambda = 1$.

Value

Vector or an object of fitted distribution depending on the called function

References

Sakia, S. M. (1992). The Box-Cox transformation technique: A review. *The Statistician, 41*(2), 169-178. doi: 10.2307/2348250

Yeo, I.-K., & Johnson, R. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika, 87*(4), 954-959. doi: 10.1093/biomet/87.4.954

See Also

See `yjt_regression` for fitting a regression model in which the response variable is distributed according to the scaled t distribution with Yeo-Johnson transformation.

See `car::yjPower` for fitting the Yeo-Johnson transformation in the **car** package. See `car::bcPower` for the Box-Cox transformation.

The scaled t distribution can be also found in `metRology::dt.scaled` (**metRology** package).

See `stats::dt` for the t distribution.

See the **fitdistrplus** package or the general `stats4::mle` function for fitting several distributions in R.

Examples

```
##############################################################################
# EXAMPLE 1: Transforming values according to Yeo-Johnson transformation
##############################################################################

# vector of y values
y <- seq(-3,3, len=100)

# non-negative lambda values
plot( y, mdmb::yj_trafo( y, lambda=1 ), type="l", ylim=8*c(-1,1),
          ylab=expression( g[lambda] (y) ) )
lines( y, mdmb::yj_trafo( y, lambda=2 ), lty=2 )
lines( y, mdmb::yj_trafo( y, lambda=.5 ), lty=3 )
lines( y, mdmb::yj_trafo( y, lambda=0 ), lty=4 )

# non-positive lambda values
plot( y, mdmb::yj_trafo( y, lambda=-1 ), type="l", ylim=8*c(-1,1),
          ylab=expression(g[lambda] (y) ) )
lines( y, mdmb::yj_trafo( y, lambda=-2 ), lty=2 )
lines( y, mdmb::yj_trafo( y, lambda=-.5 ), lty=3 )
lines( y, mdmb::yj_trafo( y, lambda=0 ), lty=4 )

## Not run:
##############################################################################
# EXAMPLE 2: Density of scaled t distribution
##############################################################################

# define location and scale parameter
m0 <- 0.3
sig <- 1.5
#-- compare density of scaled t distribution with large degrees of freedom
#   with normal distribution
y1 <- mdmb::dt_scaled( y, location=m0, shape=sig, df=100 )
y2 <- stats::dnorm( y, mean=m0, sd=sig )
max(abs(y1-y2))

##############################################################################
# EXAMPLE 3: Simulating and fitting the scaled t distribution
##############################################################################
```

```
#-- simulate data with 10 degrees of freedom
set.seed(987)
df0 <- 10    # define degrees of freedom
x <- mdmb::rt_scaled( n=1E4, location=m0, shape=sig, df=df0 )
#** fit data with df=10 degrees of freedom
fit1 <- mdmb::fit_t_scaled(x=x, df=df0 )
#** compare with fit from normal distribution
fit2 <- mdmb::fit_t_scaled(x=x, df=Inf )  # df=Inf is the default

#-- some comparisons
coef(fit1)
summary(fit1)
logLik(fit1)
AIC(fit1)
AIC(fit2)

############################################################################
# EXAMPLE 4: Simulation and fitting of scaled t distribution with
#            Yeo-Johnson transformation
############################################################################

# define parameters of transformed scaled t distribution
m0 <- .5
sig <- 1.5
lam <- .5

# evaluate density
x <- seq( -5, 5, len=100 )
y <- mdmb::dyjt_scaled( x, location=m0, shape=sig, lambda=lam )
graphics::plot( x, y, type="l")

# transform original values
mdmb::yj_trafo( y=x, lambda=lam )

#** simulate data
set.seed(987)
x <- mdmb::ryjt_scaled(n=3000, location=m0, shape=sig, lambda=lam )
graphics::hist(x, breaks=30)

#*** Model 1: Fit data with lambda to be estimated
fit1 <- mdmb::fit_yjt_scaled(x=x)
summary(fit1)
coef(fit1)

#*** Model 2: Fit data with lambda fixed to simulated lambda
fit2 <- mdmb::fit_yjt_scaled(x=x, lambda_fixed=lam)
summary(fit2)
coef(fit2)

#*** Model 3: Fit data with lambda fixed to 1
fit3 <- mdmb::fit_yjt_scaled(x=x, lambda_fixed=1)
```

```
#-- compare log-likelihood values
logLik(fit1)
logLik(fit2)
logLik(fit3)

################################################################################
# EXAMPLE 5: Approximating the chi square distribution
#            with yjt and bct distribution
################################################################################

#-- simulate data
set.seed(987)
n <- 3000
df0 <- 5
x <- stats::rchisq( n=n, df=df0 )

#-- plot data
graphics::hist(x, breaks=30)

#-- fit data with yjt distribution
fit1 <- mdmb::fit_yjt_scaled(x)
summary(fit1)
c1 <- coef(fit1)

#-- fit data with bct distribution
fit2 <- mdmb::fit_bct_scaled(x)
summary(fit2)
c2 <- coef(fit2)
# compare log-likelihood values
logLik(fit1)
logLik(fit2)

#-- plot chi square distribution and approximating yjt distribution
y <- seq( .01, 3*df0, len=100 )
dy <- stats::dchisq( y, df=df0 )
graphics::plot( y, dy, type="l", ylim=c(0, max(dy) )*1.1 )
# approximation with scaled t distribution and Yeo-Johnson transformation
graphics::lines( y, mdmb::dyjt_scaled(y, location=c1[1], shape=c1[2], lambda=c1[3]),
                    lty=2)
# approximation with scaled t distribution and Box-Cox transformation
graphocs::lines( y, mdmb::dbct_scaled(y, location=c2[1], shape=c2[2], lambda=c2[3]),
                    lty=3)
# appoximating normal distribution
graphics::lines( y, stats::dnorm( y, mean=df0, sd=sqrt(2*df0) ), lty=4)
graphics::legend( .6*max(y), .9*max(dy), c("chi square", "yjt", "bct", "norm"),
                    lty=1:4)

################################################################################
# EXAMPLE 6: Bounded variable on (0,1) with Probit Yeo-Johnson transformation
################################################################################

set.seed(876)
n <- 1000
```

```
x <- stats::rnorm(n)
y <- stats::pnorm( 1*x + stats::rnorm(n, sd=sqrt(.5) ) )
dat <- data.frame( y=y, x=x )

#*** fit Probit Yeo-Johnson distribution
mod1 <- mdmb::fit_yjt_scaled(x=y, probit=TRUE)
summary(mod1)

#*** estimation using regression model
mod2 <- mdmb::yjt_regression( y ~ x, data=dat, probit=TRUE )
summary(mod2)

## End(Not run)
```

Index