Package 'hmi'

April 26, 2017

Type Package

Title Hierarchical Multiple Imputation

Version 0.7.4

Maintainer Matthias Speidel <matthias.speidel@googlemail.com>

Description Runs single level and multilevel imputation models. The user just has to pass the data to the main function and, optionally, his analysis model. Basically the package then translates this analysis model into commands to impute the data according to it with functions from 'mice', 'MCMCglmm' or routines build for this package.

BugReports http://github.com/matthiasspeidel/hmi/issues

Imports boot (>= 1.3-18), linLIR (>= 1.1), lme4 (>= 1.1-12), MASS (>= 7.3-45), Matrix (>= 1.2), MCMCglmm (>= 2.22.1), mice (>= 2.25), msm (>= 1.6.4), mvtnorm (>= 1.0-5), nnet (>= 7.3-12), pbivnorm (>= 0.6.0), stats (>= 3.0.0), tmvtnorm (>= 1.4-10), utils

Suggests knitr, rmarkdown

Depends R (>= 3.0.0)

License GPL-3

LazyData TRUE

RoxygenNote 6.0.1

VignetteBuilder knitr

URL http://github.com/matthiasspeidel/hmi

NeedsCompilation no

Author Matthias Speidel [aut, cre] (Institute for Employment Research, Nuremberg, Germany),
Joerg Drechsler [aut] (Institute for Employment Research, Nuremberg, Germany),
Shahab Jolani [aut] (Maastricht University, Maastricht, The Netherlands)

Repository CRAN

Date/Publication 2017-04-26 19:50:27 UTC

R topics documented:

*.interval *Multiplication function*

Description

Function to multiply single elements or vectors (of correct dimension) to the interval object

Usage

```
## S3 method for class 'interval'
interval * x
```

Arguments

interval	an object from class interval
x	an single element or vector for multplication

Value

an interval object

+.interval *Adding function*

Description

Function to add single elements or vectors (of correct dimension) to the interval object

Usage

```
## S3 method for class 'interval'
interval + x
```

Arguments

interval	an object from class interval
x	an single element or vector to add to the interval object

Value

an interval object

-.interval *Subtraction function*

Description

Function to subtract single elements or vectors (of correct dimension) from the interval object

Usage

```
## S3 method for class 'interval'
interval - x
```

Arguments

interval	an object from class interval
x	an single element or vector to subtract to the interval object

Value

an interval object

/.interval *Dividing function*

Description

Function to divide single elements or vectors (of correct dimension) to the interval object

Usage

```
## S3 method for class 'interval'
interval / x
```

Arguments

interval	an object from class interval
x	an single element or vector for division

Value

an interval object

as_data_frame_interval

Transform interval objects into data.frames

Description

This function transforms interval objects into data.frames. This is not only relevant on its own, it is also needed whenever a function need objects as a data.frame (e.g. `View` or `cbind`).

Usage

```
as_data_frame_interval(x, ...)
```

Arguments

`x`	an object of class interval.
`...`	arguments to be passed to `as.data.frame`.

Value

a data.frame containing x as a character

as_interval

Function to transform numeric (or character) vectors into an interval object

Description

Function to transform numeric (or character) vectors into an interval object

Usage

```
as_interval(lower, upper)
```

Arguments

`lower`	a vector with the lower values of a variable
`upper`	a vector with the upper values of a variable

Value

a character vector with the lower and upper bound values.

components *Function to get the likelihood contribution of different rounding degrees*

Description

It is needed in the imputation routine for rounded income. See equation (5) in Drechsler, Kiesel & Speidel (2015)

Usage

```
components(ki, kj, mean1_obs, mean2_obs, sigma1, sigma2, rho, y_obs,
  mean_y_precise, sd_y_precise, half_divisor)
```

Arguments

ki	An integer for the i-th threshold (or "cutpoint")
kj	An integer for the j-th threshold (or "cutpoint") (ki < kj)
mean1_obs	A vector with the expected value of G for the observed data
mean2_obs	A vector with the expected value of log(Y) for the observed data
sigma1	A scalar for the variance of the G
sigma2	A scalar for the variance of log(Y)
rho	A scalar from [-1, 1] specifying the correlation between G and log(Y)
y_obs	The vector of the target variable (with all NAs removed)
mean_y_precise	A scalar specifying the mean of the target variable
sd_y_precise	A scalar specifying the standard deviation of the target variable
half_divisor	A scalar needed to find the bounds of possible rounding. E.g. if rounding happens to the closest multiple of 500, the half.divisor is 250.

Value

A vector with the contribution to the likelihood of every individual with an observed target variable value

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions", Austrian Journal of Statistics, Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

contributions4intervals *get the likelihood contributions of interval data*

Description

This function calculates the likelihood contributions of interval data

Usage

```
contributions4intervals(lower, upper, mymean, mysd)
```

Arguments

lower	a vector with the lower bounds of an interval covariate.
upper	a vector with the upper bounds of an interval covariate.
mymean	a numeric for the expected value of the normal distribution (which is one of the parameters trying to be optimized so that the likelihood becomes maximized)
mysd	a numeric for the standard deviation of the normal distribution (which is one of the parameters trying to be optimized so that the likelihood becomes maximized)

Value

a vector giving the likelihood contributions of the interval data.

decompose_interval *decompose up intervals*

Description

This function decomposes an interval object up into precise observations (e.g. "1850.23;1850.23" into 1850.23), imprecise observations (e.g. "1800;1900") and missing observations ("-Inf;Inf" into NA)

Usage

```
decompose_interval(interval)
```

Arguments

interval	an interval object of length n (if it is something else, it is returned unchanged)

Value

a data.frame with the precise observations, the lower and the upper bounds as covariates.

doubleintegral *Function to calculate double integrals*

Description

This function is primarily build to make the function `components` neater.

Usage

```
doubleintegral(lower_inner, upper_inner, lower_outer, upper_outer, cdf, ...)
```

Arguments

`lower_inner`	The vector for the lower bound for the inner integral
`upper_inner`	The vector for the upper bound for the inner integral
`lower_outer`	The vector for the lower bound for the outer integral
`upper_outer`	The vector for the upper bound for the outer integral
`cdf`	the cumulative density function (from the class "function")
`...`	Further arguments passed to the cdf.

Value

a vector with the value of the double integral for each observation (with an observed target variable)

extract_varnames *Function to extract the different elements of a formula*

Description

The function searches for the target variable, fixed effects variables, if there is a cluster ID: this and the random effects variables.
The names of the fixed and random intercepts variable (if existent) are explicitly labelled.

Usage

```
extract_varnames(model_formula = NULL, constant_variables,
  variable_names_in_data)
```

Arguments

`model_formula`	A formula (from class `formula`)
`constant_variables`	A Boolean-vector of length equal to the number of columns in the data set specifying whether a variable is a constant variable (eg. an intercept variable) or not.
`variable_names_in_data`	A character-vector with the column names of the data set.

Value

A list with the names of the target variable, the intercept variable, the fixed and random effects covariates and the cluster id variable.
If some of them don't exist, they get the value "".

`get_type` *Get the type of variables.*

Description

Function checks whether a variable is: ...

- continuous (numeric values),
- semicontinuous (numeric values with more than 5% of them are 0),
- rounded continuous (if continuous values are rounded to the closest multiple of 5, 10, 50, 100, 500 or 1000. We see this to be the case if more than 50% of the observations are devisable by 5)
- count data (if all values are integers).
- an intercept (the same value for all observations),
- binary (two different values - like 0s and 1s or "m" and "f"),
- categorical (the variable is a factor or has more than 3 different values)
- ordered categorical (the categorical variable is ordered.)

Usage

```
get_type(variable)
```

Arguments

`variable`	A variable (vector) from your data set.

Value

A character denoting the type of `variable`.

Examples

```
get_type(iris$Sepal.Length); get_type(iris$Species)
```

hmi *hmi: Hierarchical Multilevel Imputation.*

Description

The user has to pass his data to the function. Optionally he passes his analysis model formula so that hmi runs the imputation model in line with his analysis model formula.
And of course he can specify some parameters for the imputation routine (like the number of imputations and iterations) including Gibbs-sampler parameters (number of iterations, burnin and thinning.

Usage

```
hmi(data, model_formula = NULL, M = 10, maxit = 5, list_of_types = NULL,
  pool_with_mice = FALSE, nitt = 3000, thin = 100, burnin = 1000,
  mn = 6)
```

Arguments

data	A data.frame with all variables appearing in model_formula.
model_formula	A formula used for the analysis model. Currently the package is designed to handle formula used in lm, glm and lmer.
M	An integer defining the number of imputations that should be made.
maxit	An integer defining the number of times the imputation cycle (imputing $x_1\|x_{-1}$ then $x_2\|x_{-2}$, ... and finally $x_p\|x_{-p}$) shall be repeated. The task of checking convergence is left to the user, by evaluating the chainMean and chainVar!
list_of_types	a list where each list element has the name of a variable in the data.frame. The elements have to contain a single character denoting the type of the variable. See get_type for details about the variable types. With the function list_of_types_maker, the user can get the framework for this object. In most scenarios this is shouldn't be necessary. One example where it might be necessary is when only two observations of a continouos variable are left - because in this case get_type interprete is variable to be binary. Wrong is it in no case.
pool_with_mice	A Boolean indicating whether the user wants to pool the M data sets by mice using his model_formula. The default value is FALSE because this tampers the mids object as it adds an argument pooling not found in "normal" mids objects generated by mice.
nitt	An integer defining number of MCMC iterations (see MCMCglmm).
thin	An integer defining the thinning interval (see MCMCglmm).
burnin	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).
mn	An interger defining the minimum number of individuals per cluster.

Value

The function returns a mids object. See mice for further information.

Examples

```
## Not run:
my.formula <- Reaction ~ Days + (1 + Days|Subject)
my_analysis <- function(complete_data){
 # In this list, you can write all the parameters you are interested in.
 # Those will be averaged.
 # So make sure that averaging makes sense and that you only put in single numeric values.
 parameters_of_interest <- list()

 # ---- write in the following lines, what you are interetest in to do with your complete_data
 # the following lines are an example where the analyst is interested in the fixed intercept
 # and fixed slope and the random intercepts variance,
 # the random slopes variance and their covariance
 my_model <- lmer(my.formula, data = complete_data)

 parameters_of_interest[[1]] <- fixef(my_model)[1]
 parameters_of_interest[[2]] <- fixef(my_model)[2]
 parameters_of_interest[[3]] <- VarCorr(my_model)[[1]][1, 1]
 parameters_of_interest[[4]] <- VarCorr(my_model)[[1]][1, 2]
 parameters_of_interest[[5]] <- VarCorr(my_model)[[1]][2, 2]
 names(parameters_of_interest) <- c("beta_intercept", "beta_Days", "sigma0", "sigma01", "sigma1")

 # ---- do change this function below this line.
 return(parameters_of_interest)
}
require("lme4")
require("mice")
data(sleepstudy, package = "lme4")
test <- sleepstudy
test$Intercept <- 1
test[sample(1:nrow(test), size = 20), "Reaction"] <- NA
hmi_imp <- hmi(data = test, model_formula = my.formula, M = 5)
hmi_pool(mids = hmi_imp, analysis_function = my_analysis)
#if you are interested in fixed effects only, consider pool from mice:
pool(with(data = hmi_imp, expr = lmer(Reaction ~ Days + (1 + Days | Subject))))

## End(Not run)
require("lme4")
test <- sleepstudy
test$Intercept <- 1
test[sample(1:nrow(test), size = 20), "Reaction"] <- NA
hmi(data = test, model_formula = Reaction ~ Days + (1 + Days|Subject), M = 2, maxit = 1)
```

---

hmi_pool *Averages the results of the imputation function* hmi.

Description

This function applies the analysis the user is interested in, on all different imputed dataset. Then the results are pooled by simply averaging the results. So the user has to make sure that his analysis produces results with a meaningful average. And furthermore has to accept that no variance is calculated for these parameters.

Usage

```
hmi_pool(mids, analysis_function)
```

Arguments

`mids`	A `mids` (multiply imputed data set) object. Either from the `hmi` imputation function or `mice`.
`analysis_function`	A user generated function that contains the model and the model parameters he is interested in. See examples.

Value

A vector with all averaged results.

Examples

```
## Not run:
my.formula <- Reaction ~ Days + (1 + Days|Subject)
my_analysis <- function(complete_data){
 # In this list, you can write all the parameters you are interested in.
 # Those will be averaged.
 # So make sure that averaging makes sense and that you only put in single numeric values.
 parameters_of_interest <- list()

 # ---- write in the following lines, what you are interested in to do with your complete_data
 # the following lines are an example where the analyst is interested in the fixed intercept
 # and fixed slope and the random intercepts variance,
 # the random slopes variance and their covariance
 my_model <- lmer(my.formula, data = complete_data)

 parameters_of_interest[[1]] <- fixef(my_model)[1]
 parameters_of_interest[[2]] <- fixef(my_model)[2]
 parameters_of_interest[[3]] <- VarCorr(my_model)[[1]][1, 1]
 parameters_of_interest[[4]] <- VarCorr(my_model)[[1]][1, 2]
 parameters_of_interest[[5]] <- VarCorr(my_model)[[1]][2, 2]
 names(parameters_of_interest) <- c("beta_intercept", "beta_Days", "sigma0", "sigma01", "sigma1")

 # ---- do not change this function below this line.
 return(parameters_of_interest)
}
require("lme4")
require("mice")
data(sleepstudy, package = "lme4")
```

```
test <- sleepstudy
test$Intercept <- 1
test[sample(1:nrow(test), size = 20), "Reaction"] <- NA
hmi_imp <- hmi(data = test, model_formula = my.formula, M = 5)
hmi_pool(mids = hmi_imp, analysis_function = my_analysis)
#if you are interested in fixed effects only, consider using \code{pool} from \code{mice}.
pool(with(data = hmi_imp, expr = lmer(Reaction ~ Days + (1 + Days | Subject))))

## End(Not run)
require("lme4")
test <- sleepstudy
test$Intercept <- 1
test[sample(1:nrow(test), size = 20), "Reaction"] <- NA
hmi(data = test, model_formula = Reaction ~ Days + (1 + Days|Subject), M = 2, maxit = 1)
```

`idf2interval` *Transform interval data frames into data.frames with interval variables*

Description

This function is the path from the linLIR package (Wiencierz, 2012) to this hmi package.

Usage

```
idf2interval(idf)
```

Arguments

`idf`	an interval data frame (idf-object).

Value

interval an `interval`.

`imp_binary_multi` *The function for hierarchical imputation of binary variables.*

Description

The function is called by the wrapper.

Usage

```
imp_binary_multi(y_imp, X_imp, Z_imp, clID, model_formula, nitt = 3000,
  thin = 10, burnin = 1000)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.
Z_imp	A data.frame with the random effects variables.
clID	A vector with the cluster ID.
model_formula	A formula used for the analysis model.
nitt	An integer defining number of MCMC iterations (see MCMCglmm).
thin	An integer defining the thinning interval (see MCMCglmm).
burnin	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).

Value

A n x 1 data.frame with the original and imputed values.

---

imp_binary_single *The function for imputation of binary variables.*

Description

The function is called by the wrapper.

Usage

```
imp_binary_single(y_imp, X_imp)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.

Value

A n x 1 data.frame with the original and imputed values.

imp_cat_multi *The function for hierarchical imputation of categorical variables.*

Description

The function is called by the wrapper and relies on MCMCglmm.
While in the single level function (imp_cat_single) we used regression trees to impute data, here we run a multilevel multinomial model. The basic idea is that for each category of the target variable (expect the reference category) an own formula is set up, saying for example that the chances to end up in category j increase with increasing X5. So there is an own regression coefficient $beta_{5,j}$ present. In a multilevel setting, this regression coefficient $beta_{5,j}$ might be different for different clusters: for cluster 27 it would be $beta_{5,j,27} = beta_{5,j} + u_{5,27}$. This also leads to own random effect covariance matrices for each category. All those random effect variance parameters can be collected in one (quite large) covariance matrix where (for example) not only the random intercepts variance and random slopes variance and their covariance is present. Instead, there is even a covariance between the random slopes in category s and the random intercepts in category p. Beside the difficulties in interpretation, these covariances have shown to be numerically instable so they are set to be 0.

Usage

```
imp_cat_multi(y_imp, X_imp, Z_imp, clID, model_formula, nitt = 3000,
  thin = 10, burnin = 1000)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.
Z_imp	A data.frame with the random effects variables.
clID	A vector with the cluster ID.
model_formula	A formula used for the analysis model.
nitt	An integer defining number of MCMC iterations (see MCMCglmm).
thin	An integer defining the thinning interval (see MCMCglmm).
burnin	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).

Value

A n x 1 data.frame with the original and imputed values.

imp_cat_single *The function to impute unordered categorical variables*

Description

The function uses regression trees for imputation implemented in `mice`. The principle is the following: For each observation it is calculated at which leave it would end. Then one (randomly selected) observation of the other observations found on this leave functions as a donor.

Usage

```
imp_cat_single(y_imp, X_imp)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.

Value

A n x 1 data.frame with the original and imputed values as a factor.

imp_cont_multi *The function for hierarchical imputation of contious variables.*

Description

The function is called by the wrapper.

Usage

```
imp_cont_multi(y_imp, X_imp, Z_imp, clID, nitt = 3000, thin = 10,
  burnin = 1000)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`thin`	An integer defining the thinning interval (see MCMCglmm).
`burnin`	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).

Value

A n x 1 matrix with the original and imputed values.

`imp_cont_single` *The function for imputation of continuous variables.*

Description

The function is called by the wrapper (hmi). It uses `mice` with the method "norm".

Usage

```
imp_cont_single(y_imp, X_imp)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.

Value

A n x 1 data.frame with the original and imputed values.

`imp_count_multi` *The function for hierarchical imputation of variables with count data.*

Description

The function is called by the wrapper.

Usage

```
imp_count_multi(y_imp, X_imp, Z_imp, clID, nitt = 3000, thin = 10,
  burnin = 1000)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`Z_imp`	A data.frame with the random effects variables.
`clID`	A vector with the cluster ID.
`nitt`	An integer defining number of MCMC iterations (see MCMCglmm).
`thin`	An integer defining the thinning interval (see MCMCglmm).
`burnin`	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).

Value

A n x 1 data.frame with the original and imputed values.

`imp_count_single` *The function for imputation of binary variables.*

Description

The function is called by the wrapper.

Usage

```
imp_count_single(y_imp, X_imp)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.

Value

A n x 1 data.frame with the original and imputed values.

`imp_interval` *The function to impute interval data variables*

Description

This functions imputes interval data variables. Those are variables, that consists of a lower and upper (numeric) boundary. Technically those boundaries are cointained in a string, separated by a semi colon. E.g. if a person reports there income to be something between 3000 and 4000 dollars, its value in the interval covariate would be `"3000;4000"`. Left (resp. right) censored data can be denoted by `"-Inf;x"` (resp. `"x;Inf"`), with `x` being the (numeric) observed value.

Usage

```
imp_interval(y_imp, X_imp)
```

Arguments

`y_imp`	A Vector from the class `interval` with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.

Value

A n x 1 data.frame with the original and imputed values. Note that this function won't return `interval` data as its purpose is to "break" the interval answers into precise answers.

imp_orderedcat_multi *The function for hierarchical imputation of categorical variables.*

Description

The function is called by the wrapper.

Usage

```
imp_orderedcat_multi(y_imp, X_imp, Z_imp, clID, model_formula, nitt = 3000,
  thin = 10, burnin = 1000)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.
Z_imp	A data.frame with the random effects variables.
clID	A vector with the cluster ID.
model_formula	A formula used for the analysis model.
nitt	An integer defining number of MCMC iterations (see MCMCglmm).
thin	An integer defining the thinning interval (see MCMCglmm).
burnin	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).

Value

A n x 1 data.frame with the original and imputed values as a factor.

imp_orderedcat_single *The function to impute ordered categorical variables*

Description

The function uses the proportional odds logistic regression (polr) approach, implemented in mice.

Usage

```
imp_orderedcat_single(y_imp, X_imp)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.

Value

A n x 1 data.frame with the original and imputed values as a factor.

`imp_roundedcont` *The function to impute rounded continuous variables*

Description

For example the income in surveys is often reported rounded by the respondents. See Drechsler, Kiesl and Speidel (2015) for more details.

Usage

```
imp_roundedcont(y_imp, X_imp, intercept_varname = NULL)
```

Arguments

`y_imp`	A Vector with the variable to impute.
`X_imp`	A data.frame with the fixed effects variables.
`intercept_varname`	A character denoting the name of the intercept variable.

Value

A n x 1 data.frame with the original and imputed values.

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions". Austrian Journal of Statistics Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

`imp_semicont_multi` *The function for hierarchical imputation of semicontinuous variables.*

Description

The function is called by the wrapper. We consider data to be "semicontinuous" when more than 5% of the (non categorical) observations.
For example in surveys a certain portion of people, when asked for their income, report "0", which clearly violates the assumption of income to be (log-) normally distributed.

Usage

```
imp_semicont_multi(y_imp, X_imp, Z_imp, clID, model_formula, heap = 0,
  nitt = 3000, thin = 10, burnin = 1000)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.
Z_imp	A data.frame with the random effects variables.
clID	A vector with the cluster ID.
model_formula	A formula used for the analysis model.
heap	A numeric saying to which (single) values the data might be heaped.
nitt	An integer defining number of MCMC iterations (see MCMCglmm).
thin	An integer defining the thinning interval (see MCMCglmm).
burnin	An integer defining the percentage of draws from the gibbs sampler that should be discarded as burn in (see MCMCglmm).

Value

A n x 1 data.frame with the original and imputed values.

---

imp_semicont_single *The function for hierarchical imputation of semicontinous variables.*

Description

The function is called by the wrapper. We consider data to be "semicontinuous" when more than 5% of the (non categorical) observations.
For example in surveys a certain portion of people, when asked for their income, report "0", which clearly violates the assumption of income to be (log-) normally distributed.

Usage

```
imp_semicont_single(y_imp, X_imp, heap = 0)
```

Arguments

y_imp	A Vector with the variable to impute.
X_imp	A data.frame with the fixed effects variables.
heap	A scalar saying to which value the data might be heaped.

Value

A n x 1 data.frame with the original and imputed values.

interval2idf *Transform interval variables to an interval data frame*

Description

This function is the path from this hmi package to the linLIR package (Wiencierz, 2012).

Usage

```
interval2idf(interval)
```

Arguments

interval	an interval

Value

an interval data frame (idf-object) with one variable (having a lower and an upper bound).

is_interval *Function to check whether an object is an interval*

Description

If there are numerics separated by semicolons (;), this is considered to be an interval. intervals with 2.4e5 are not considered to be an interval.

Usage

```
is_interval(x)
```

Arguments

x	an object

Value

a boolean value indicting whether x is an interval or not

list_of_types_maker *Helps the user to make a list of types.*

Description

In hmi the user can add a list of types. This function gives him a framework with suggestions. Of course he can changes on his own afterwards. For example, if a continuous variable as only two observations left, then get_type interprete this as a binary variable and not a continuous.

Usage

```
list_of_types_maker(data)
```

Arguments

data	the data.frame also passed to hmi.

Value

a list with suggested types. Each list element has the name of a variable in the data.frame. The elements contain a single character denoting the type of the variable. See get_type for details about the variable types.

negloglik2 *calculate the likelihood contribution of the data*

Description

This function based on Drechsler, Kiesl & Speidel (2015) is needed in the imputation routine for rounded income. It calculates the likelihood contribution of the data (regardless whether they are observed precisely or presumably rounded).

Usage

```
negloglik2(para, X, y_in_negloglik, lower = NA, upper = NA, my_p,
  mean_y_precise, sd_y_precise)
```

Arguments

para	This is the vector Psi of parameters (see p. 62 in Drechsler, Kiesl & Speidel, 2015). With respect to them, the value returned by negloglik2 shall be maximized. The starting values are c(kstart, betastart2, gammastart, sigmastart2) (the 6 thresholds (or "cutting points") for the latent variable behind the rounding degree, the regression parameters explaining the logged income, the regression parameters explaining the rounding degree and the variance parameter).

X	the data.frame of covariates.
y_in_negloglik	the target variable (a vector).
lower	the lower bound of an interval variable.
upper	the upper bound of an interval variable.
my_p	This vector is the indicator of the (highest possible) rounding degree for an observation. This parameter comes directly from the data.
mean_y_precise	the scalar with the value of the mean of the target variable.
sd_y_precise	the scalar with the value equal to the standard deviation of the target variable.

Value

An integer equal to the (sum of the) negative log-likelihood contributions (of the observations)

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions", Austrian Journal of Statistics, Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

---

negloglik2_intervalsonly

calculate the likelihood contribution of the interval data only

---

Description

calculate the likelihood contribution of the interval data only

Usage

```
negloglik2_intervalsonly(para, X, lower, upper)
```

Arguments

para	This is the vector Psi of parameters (see p. 62 in Drechsler, Kiesl & Speidel, 2015). With respect to them, the value returned by negloglik2 shall be maximized. The starting values are c(betastart2, sigmastart2) (the regression parameters explaining the logged income, and the variance parameter).
X	the data.frame of covariates.
lower	the lower bound of an interval variable.
upper	the upper bound of an interval variable.

Value

An integer equal to the (sum of the) negative log-likelihood contributions (of the observations)

References

Joerg Drechsler, Hans Kiesl, Matthias Speidel (2015): "MI Double Feature: Multiple Imputation to Address Nonresponse and Rounding Errors in Income Questions", Austrian Journal of Statistics, Vol. 44, No. 2, http://dx.doi.org/10.17713/ajs.v44i2.77

`pbivnormX` *calculate probabilities from the cumulative distribution function of a standard bivariate normal distribution*

Description

A modfied version of pbivnorm() from package `pbivnorm`. It is needed in the imputation routine for rounded income.

Usage

```
pbivnormX(x, y, rho = 0)
```

Arguments

`x`	the vector (or a two columned matrix) with the values of the first random variable
`y`	the vector with the values of the second random variable
`rho`	the correlation (a scalar) between the two random variables.

Value

A vector with the values of the density distribution at the points (x, y).

`remove_excessives` *Removing excessive factors*

Description

Function to exclude variables that have too many levels as they may cause numerical problems

Usage

```
remove_excessives(X, k = 10)
```

Arguments

`X`	A n times p data.frame with p fixed (or random) effects variables.
`k`	An integer defining the allowed maximum of levels in a factor covariate.

Value

A n times (p-r) data.frame, with r beeing the number of variables with too many factors.

sample_imp *Sample imputation.*

Description

Function to sample values in a variable from other (observed) values in this variable. So this imputation doesn't use further covariates.

Usage

```
sample_imp(variable)
```

Arguments

variable	A vector of size n with missing values.

Value

A vector of size n without missing values.

Examples

```
set.seed(123)
sample_imp(c(1, NA, 3, NA, 5))
```

sna_interval *Get standard NAs from interval data*

Description

This function replaces observations with "-Inf;Inf" with the standard NAs (therefore 'sna')

Usage

```
sna_interval(x)
```

Arguments

x	can by any object, but the function was designed for interval-objects.

Value

In case of x being an interval-object, it returns a n times 2 matrix. The first column is the lower bound, the second the upper bound. Otherwise it returns just x.

split_interval *Split up intervals*

Description

This function splits an interval object up into the lower and upper bound

Usage

```
split_interval(interval)
```

Arguments

interval	an interval object of length n (if it is something else, it is returned unchanged)

Value

a n times 2 matrix. The first column is the lower bound, the second the upper bound.

stand *Standardizing function*

Description

Function to standardize variables that are numeric (continuous and count variables) but no rounded continuous, semicontinuous, intercepts or categorical variables.

Usage

```
stand(X)
```

Arguments

X	A n times p data.frame with p fixed (or random) effects variables.

Value

A n times p data.frame with the standardized versions of the numeric variables.

Index