Package ‘mice’

July 22, 2025

Type Package

Version 3.18.0

Title Multivariate Imputation by Chained Equations

Date 2025-05-27

Maintainer Stef van Buuren `<stef.vanbuuren@tno.nl>`

Depends R (>= 2.10.0)

Imports broom, dplyr, glmnet, graphics, grDevices, lattice, mitml, nnet, Rcpp, rpart, stats, tidyr, utils

Suggests broom.mixed, future, furrr, haven, knitr, literanger, lme4, MASS, miceadds, pan, parallelly, purrr, ranger, randomForest, rmarkdown, rstan, survival, testthat

Description Multiple imputation using Fully Conditional Specification (FCS) implemented by the MICE algorithm as described in Van Buuren and Groothuis-Oudshoorn (2011) <doi:10.18637/jss.v045.i03>. Each variable has its own imputation model. Built-in imputation models are provided for continuous data (predictive mean matching, normal), binary data (logistic regression), unordered categorical data (polytomous logistic regression) and ordered categorical data (proportional odds). MICE can also impute continuous two-level data (normal model, pan, second-level variables). Passive imputation can be used to maintain consistency between variables. Various diagnostic plots are available to inspect the quality of the imputations.

Encoding UTF-8

LazyLoad yes

LazyData yes

URL `https://github.com/amices/mice`, `https://amices.org/mice/`, `https://stefvanbuuren.name/fimd/`

BugReports `https://github.com/amices/mice/issues`

LinkingTo cpp11, Rcpp

License GPL (>= 2)

RoxygenNote 7.3.2

NeedsCompilation yes

Author Stef van Buuren [aut, cre],
Karin Groothuis-Oudshoorn [aut],
Gerko Vink [ctb],
Rianne Schouten [ctb],
Alexander Robitzsch [ctb],
Patrick Rockenschaub [ctb],
Lisa Doove [ctb],
Shahab Jolani [ctb],
Margarita Moreno-Betancur [ctb],
Ian White [ctb],
Philipp Gaffert [ctb],
Florian Meinfelder [ctb],
Bernie Gray [ctb],
Vincent Arel-Bundock [ctb],
Mingyang Cai [ctb],
Thom Volker [ctb],
Edoardo Costantini [ctb],
Caspar van Lissa [ctb],
Hanne Oberman [ctb],
Stephen Wade [ctb]

Repository CRAN

Date/Publication 2025-05-27 10:40:02 UTC

Contents

`.pmm.match` *Finds an imputed value from matches in the predictive metric (deprecated)*

Description

This function finds matches among the observed data in the predictive mean metric. It selects the `donors` closest matches, randomly samples one of the donors, and returns the observed value of the match.

Usage

```
.pmm.match(z, yhat = yhat, y = y, donors = 5, ...)
```

Arguments

`z`	A scalar containing the predicted value for the current case to be imputed.
`yhat`	A vector containing the predicted values for all cases with an observed outcome.
`y`	A vector of `length(yhat)` elements containing the observed outcome
`donors`	The size of the donor pool among which a draw is made. The default is `donors = 5`. Setting `donors = 1` always selects the closest match. Values between 3 and 10 provide the best results. Note: This setting was changed from 3 to 5 in version 2.19, based on simulation work by Tim Morris (UCL).
`...`	Other parameters (not used).

Details

This function is included for backward compatibility. It was used up to `mice 2.21`. The current `mice.impute.pmm()` function calls the faster `C` function `matcher` instead of `.pmm.match()`.

Value

A scalar containing the observed value of the selected donor.

Author(s)

Stef van Buuren

References

Schenker N & Taylor JMG (1996) Partially parametric techniques for multiple imputation. *Computational Statistics and Data Analysis*, 22, 425-446.

Little RJA (1988) Missing-data adjustments in large surveys (with discussion). *Journal of Business Economics and Statistics*, 6, 287-301.

`ampute` *Generate missing data for simulation purposes*

Description

This function generates multivariate missing data under a MCAR, MAR or MNAR missing data mechanism. Imputation of data sets containing missing values can be performed with `mice`.

Usage

```
ampute(
  data,
  prop = 0.5,
  patterns = NULL,
  freq = NULL,
  mech = "MAR",
  weights = NULL,
  std = TRUE,
  cont = TRUE,
  type = NULL,
  odds = NULL,
  bycases = TRUE,
  run = TRUE
)
```

Arguments

`data`	A complete data matrix or data frame. Values should be numeric. Categorical variables should have been transformed to dummies.
`prop`	A scalar specifying the proportion of missingness. Should be a value between 0 and 1. Default is a missingness proportion of 0.5.
`patterns`	A matrix or data frame of size #patterns by #variables where `0` indicates that a variable should have missing values and `1` indicates that a variable should remain complete. The user may specify as many patterns as desired. One pattern (a vector) is possible as well. Default is a square matrix of size #variables where each pattern has missingness on one variable only (created with `ampute.default.patterns`). After the amputation procedure, `md.pattern` can be used to investigate the missing data patterns in the data.
`freq`	A vector of length #patterns containing the relative frequency with which the patterns should occur. For example, for three missing data patterns, the vector could be `c(0.4, 0.4, 0.2)`, meaning that of all cases with missing values, 40 percent should have pattern 1, 40 percent pattern 2 and 20 percent pattern 3. The vector should sum to 1. Default is an equal probability for each pattern, created with `ampute.default.freq`.
`mech`	A string specifying the missingness mechanism, either "MCAR" (Missing Completely At Random), "MAR" (Missing At Random) or "MNAR" (Missing Not At Random). Default is a MAR missingness mechanism.
`weights`	A matrix or data frame of size #patterns by #variables. The matrix contains the weights that will be used to calculate the weighted sum scores. For a MAR mechanism, the weights of the variables that will be made incomplete should be zero. For a MNAR mechanism, these weights could have any possible value. Furthermore, the weights may differ between patterns and between variables. They may be negative as well. Within each pattern, the relative size of the values are of importance. The default weights matrix is made with `ampute.default.weights` and returns a matrix with equal weights for all variables. In case of MAR, variables that will be amputed will be weighted with `0`. For MNAR, variables that will be observed will be weighted with `0`. If the mechanism is MCAR, the weights matrix will not be used.
`std`	Logical. Whether the weighted sum scores should be calculated with standardized data or with non-standardized data. The latter is especially advised when making use of train and test sets in order to prevent leakage.
`cont`	Logical. Whether the probabilities should be based on a continuous or a discrete distribution. If TRUE, the probabilities of being missing are based on a continuous logistic distribution function. `ampute.continuous` will be used to calculate and assign the probabilities. These probabilities will then be based on the argument `type`. If FALSE, the probabilities of being missing are based on a discrete distribution (`ampute.discrete`) based on the `odds` argument. Default is TRUE.
`type`	A string or vector of strings containing the type of missingness for each pattern. Either `"LEFT"`, `"MID"`, `"TAIL"` or ’`"RIGHT"`. If a single missingness type is given, all patterns will be created with the same type. If the missingness types

	should differ between patterns, a vector of missingness types should be given. Default is RIGHT for all patterns and is the result of `ampute.default.type`.
`odds`	A matrix where #patterns defines the #rows. Each row should contain the odds of being missing for the corresponding pattern. The number of odds values defines in how many quantiles the sum scores will be divided. The odds values are relative probabilities: a quantile with odds value 4 will have a probability of being missing that is four times higher than a quantile with odds 1. The number of quantiles may differ between the patterns, specify NA for cells remaining empty. Default is 4 quantiles with odds values 1, 2, 3 and 4 and is created by `ampute.default.odds`.
`bycases`	Logical. If TRUE, the proportion of missingness is defined in terms of cases. If FALSE, the proportion of missingness is defined in terms of cells. Default is TRUE.
`run`	Logical. If TRUE, the amputations are implemented. If FALSE, the return object will contain everything except for the amputed data set.

Details

This function generates missing values in complete data sets. Amputation of complete data sets is useful for the evaluation of imputation techniques, such as multiple imputation (performed with function `mice` in this package).

The basic strategy underlying multivariate imputation was suggested by Don Rubin during discussions in the 90's. Brand (1997) created one particular implementation, and his method found its way into the FCS paper (Van Buuren et al, 2006).

Until recently, univariate amputation procedures were used to generate missing data in complete, simulated data sets. With this approach, variables are made incomplete one variable at a time. When more than one variable needs to be amputed, the procedure is repeated multiple times.

With the univariate approach, it is difficult to relate the missingness on one variable to the missingness on another variable. A multivariate amputation procedure solves this issue and moreover, it does justice to the multivariate nature of data sets. Hence, `ampute` is developed to perform multivariate amputation.

The idea behind the function is the specification of several missingness patterns. Each pattern is a combination of variables with and without missing values (denoted by `0` and `1` respectively). For example, one might want to create two missingness patterns on a data set with four variables. The patterns could be something like: `0,0,1,1` and `1,0,1,0`. Each combination of zeros and ones may occur.

Furthermore, the researcher specifies the proportion of missingness, either the proportion of missing cases or the proportion of missing cells, and the relative frequency each pattern occurs. Consequently, the data is split into multiple subsets, one subset per pattern. Now, each case is candidate for a certain missingness pattern, but whether the case will have missing values eventually depends on other specifications.

The first of these specifications is the missing mechanism. There are three possible mechanisms: the missingness depends completely on chance (MCAR), the missingness depends on the values of the observed variables (i.e. the variables that remain complete) (MAR) or on the values of the variables that will be made incomplete (MNAR).

When the user specifies the missingness mechanism to be `"MCAR"`, the candidates have an equal probability of becoming incomplete. For a `"MAR"` or `"MNAR"` mechanism, weighted sum scores are calculated. These scores are a linear combination of the variables.

In order to calculate the weighted sum scores, the data is standardized. For this reason, the data has to be numeric. Second, for each case, the values in the data set are multiplied with the weights, specified by argument `weights`. These weighted scores will be summed, resulting in a weighted sum score for each case.

The weights may differ between patterns and they may be negative or zero as well. Naturally, in case of a MAR mechanism, the weights corresponding to the variables that will be made incomplete, have a 0. Note that this may be different for each pattern. In case of MNAR missingness, especially the weights of the variables that will be made incomplete are of importance. However, the other variables may be weighted as well.

It is the relative difference between the weights that will result in an effect in the sum scores. For example, for the first missing data pattern mentioned above, the weights for the third and fourth variables could be set to 2 and 4. However, weight values of 0.2 and 0.4 will have the exact same effect on the weighted sum score: the fourth variable is weighted twice as much as variable 3.

Based on the weighted sum scores, either a discrete or continuous distribution of probabilities is used to calculate whether a candidate will have missing values.

For a discrete distribution of probabilities, the weighted sum scores are divided into subgroups of equal size (quantiles). Thereafter, the user specifies for each subgroup the odds of being missing. Both the number of subgroups and the odds values are important for the generation of missing data. For example, for a RIGHT-like mechanism, scoring in one of the higher quantiles should have high missingness odds, whereas for a MID-like mechanism, the central groups should have higher odds. Again, not the size of the odds values are of importance, but the relative distance between the values.

The continuous distributions of probabilities are based on the logistic distribution function. The user can specify the type of missingness, which, again, may differ between patterns.

For an example and more explanation about how the arguments interact with each other, we refer to the vignette: Generate missing values with ampute.

Value

Returns an S3 object of class `mads` (multivariate amputed data set)

Author(s)

Rianne Schouten, Gerko Vink, Peter Lugtig, 2016

References

Brand, J.P.L. (1999) *Development, implementation and evaluation of multiple imputation strategies for the statistical analysis of incomplete data sets.* pp. 110-113. Dissertation. Rotterdam: Erasmus University.

Schouten, R.M., Lugtig, P and Vink, G. (2018) Generating missing values for simulation purposes: A multivariate amputation procedure. *Journal of Statistical Computation and Simulation*, 88(15): 1909-1930. doi:10.1080/00949655.2018.1491577

Schouten, R.M. and Vink, G. (2018) The Dance of the Mechanisms: How Observed Information Influences the Validity of Missingness Assumptions. *Sociological Methods and Research*, 50(3): 1243-1258. doi:10.1177/0049124118799376

Van Buuren, S., Brand, J.P.L., Groothuis-Oudshoorn, C.G.M., Rubin, D.B. (2006) Fully conditional specification in multivariate imputation. *Journal of Statistical Computation and Simulation*, 76(12): 1049-1064. doi:10.1080/10629360600810434

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Vink, G. (2016) Towards a standardized evaluation of multiple imputation routines.

See Also

`mads`, `bwplot.mads`, `xyplot.mads`

Examples

```
# start with a complete data set
compl_boys <- cc(boys)[1:3]

# Perform amputation with default settings
mads_boys <- ampute(data = compl_boys)
mads_boys$amp

# Change default matrices as desired
my_patterns <- mads_boys$patterns
my_patterns[1:3, 2] <- 0

my_weights <- mads_boys$weights
my_weights[2, 1] <- 2
my_weights[3, 1] <- 0.5

# Rerun amputation
my_mads_boys <- ampute(
  data = compl_boys, patterns = my_patterns, freq =
    c(0.3, 0.3, 0.4), weights = my_weights, type = c("RIGHT", "TAIL", "LEFT")
)
my_mads_boys$amp
```

`anova.mira` *Compare several nested models*

Description

Compare several nested models

Usage

```
## S3 method for class 'mira'
anova(object, ..., method = "D1", use = "wald")
```

Arguments

`object`	Two or more objects of class `mira`
`...`	Other parameters passed down to `D1()`, `D2()`, `D3()` and `mitml::testModels`.
`method`	Either `"D1"`, `"D2"` or `"D3"`
`use`	An character indicating the test statistic

Value

Object of class `mice.anova`

`appendbreak` *Appends specified break to the data*

Description

A custom function to insert rows in long data with new pseudo-observations that are being done on the specified break ages. There should be a column called `first` in `data` with logical data that codes whether the current row is the first for subject `id`. Furthermore, the function assumes that columns `age`, `occ`, `hgt.z`, `wgt.z` and `bmi.z` are available. This function is used on the `tbc` data in FIMD chapter 9. Check that out to see it in action.

Usage

```
appendbreak(data, brk, warp.model = warp.model, id = NULL, typ = "pred")
```

Arguments

`data`	A data frame in the long long format
`brk`	A vector of break ages
`warp.model`	A time warping model
`id`	The subject identifier
`typ`	Label to signal that this is a newly added observation

Value

A long data frame with additional rows for the break ages

`as.mids` *Converts an imputed dataset (long format) into a* `mids` *object*

Description

This function converts imputed data stored in long format into an object of class `mids`. The original incomplete dataset needs to be available so that we know where the missing data are. The function is useful to convert back operations applied to the imputed data back in a `mids` object. It may also be used to store multiply imputed data sets from other software into the format used by `mice`.

Usage

```
as.mids(long, where = NULL, .imp = ".imp", .id = ".id")
```

Arguments

`long`	A multiply imputed data set in long format, for example produced by a call to `complete(..., action = 'long', include = TRUE)`, or by other software.
`where`	A data frame or matrix with logicals of the same dimensions as `data` indicating where in the data the imputations should be created. The default, `where = is.na(data)`, specifies that the missing data should be imputed. The `where` argument may be used to overimpute observed data, or to skip imputations for selected missing values. Note: Imputation methods that generate imptutations outside of `mice`, like `mice.impute.panImpute()` may depend on a complete predictor space. In that case, a custom `where` matrix can not be specified.
`.imp`	An optional column number or column name in `long`, indicating the imputation index. The values are assumed to be consecutive integers between 0 and `m`. Values `1` through `m` correspond to the imputation index, value `0` indicates the original data (with missings). By default, the procedure will search for a variable named `".imp"`.
`.id`	An optional column number or column name in `long`, indicating the subject identification. If not specified, then the function searches for a variable named `".id"`. If this variable is found, the values in the column will define the row names in the `data` element of the resulting `mids` object.

Value

An object of class `mids`

Note

The function expects the input data `long` to be sorted by imputation number (variable `".imp"` by default), and in the same sequence within each imputation block.

Author(s)

Gerko Vink

Examples

```
# impute the nhanes dataset
imp <- mice(nhanes, print = FALSE)
# extract the data in long format
X <- complete(imp, action = "long", include = TRUE)
# create dataset with .imp variable as numeric
X2 <- X

# nhanes example without .id
test1 <- as.mids(X)
is.mids(test1)
identical(complete(test1, action = "long", include = TRUE), X)

# nhanes example without .id where .imp is numeric
test2 <- as.mids(X2)
is.mids(test2)
identical(complete(test2, action = "long", include = TRUE), X)

# nhanes example, where we explicitly specify .id as column 2
test3 <- as.mids(X, .id = ".id")
is.mids(test3)
identical(complete(test3, action = "long", include = TRUE), X)

# nhanes example with .id where .imp is numeric
test4 <- as.mids(X2, .id = 6)
is.mids(test4)
identical(complete(test4, action = "long", include = TRUE), X)

# example without an .id variable
# variable .id not preserved
X3 <- X[, -6]
test5 <- as.mids(X3)
is.mids(test5)
identical(complete(test5, action = "long", include = TRUE)[, -6], X[, -6])

# where argument copies also observed data into $imp element
where <- matrix(TRUE, nrow = nrow(nhanes), ncol = ncol(nhanes))
colnames(where) <- colnames(nhanes)
test11 <- as.mids(X, where = where)
identical(complete(test11, action = "long", include = TRUE), X)
```

`as.mira` *Create a `mira` object from repeated analyses*

Description

The `as.mira()` function takes the results of repeated complete-data analysis stored as a list, and turns it into a `mira` object that can be pooled.

Usage

```
as.mira(fitlist)
```

Arguments

`fitlist`	A list containing m fitted analysis objects

Value

An S3 object of class `mira`.

Author(s)

Stef van Buuren

See Also

`mira`

`as.mitml.result` *Converts into a* `mitml.result` *object*

Description

The `as.mitml.result()` function takes the results of repeated complete-data analysis stored as a list, and turns it into an object of class `mitml.result`.

Usage

```
as.mitml.result(x)
```

Arguments

`x`	An object of class `mira`

Value

An S3 object of class `mitml.result`, a list containing m fitted analysis objects.

Author(s)

Stef van Buuren

See Also

`with.mitml.list`

`boys` *Growth of Dutch boys*

Description

Height, weight, head circumference and puberty of 748 Dutch boys.

Format

A data frame with 748 rows on the following 9 variables:

age Decimal age (0-21 years)

hgt Height (cm)

wgt Weight (kg)

bmi Body mass index

hc Head circumference (cm)

gen Genital Tanner stage (G1-G5)

phb Pubic hair (Tanner P1-P6)

tv Testicular volume (ml)

reg Region (north, east, west, south, city)

Details

Random sample of 10\ Dutch growth references 1997. Variables `gen` and `phb` are ordered factors. `reg` is a factor.

Source

Fredriks, A.M,, van Buuren, S., Burgmeijer, R.J., Meulmeester JF, Beuker, R.J., Brugman, E., Roede, M.J., Verloove-Vanhorick, S.P., Wit, J.M. (2000) Continuing positive secular growth change in The Netherlands 1955-1997. *Pediatric Research*, **47**, 316-323.

Fredriks, A.M., van Buuren, S., Wit, J.M., Verloove-Vanhorick, S.P. (2000). Body index measurements in 1996-7 compared with 1980. *Archives of Disease in Childhood*, **82**, 107-112.

Examples

```
# create two imputed data sets
imp <- mice(boys, m = 1, maxit = 2)
z <- complete(imp, 1)

# create imputations for age <8yrs
plot(z$age, z$gen,
  col = mdc(1:2)[1 + is.na(boys$gen)],
  xlab = "Age (years)", ylab = "Tanner Stage Genital"
)
```

```
# figure to show that the default imputation method does not impute BMI
# consistently
plot(z$bmi, z$wgt / (z$hgt / 100)^2,
  col = mdc(1:2)[1 + is.na(boys$bmi)],
  xlab = "Imputed BMI", ylab = "Calculated BMI"
)

# also, BMI distributions are somewhat different
oldpar <- par(mfrow = c(1, 2))
MASS::truehist(z$bmi[!is.na(boys$bmi)],
  h = 1, xlim = c(10, 30), ymax = 0.25,
  col = mdc(1), xlab = "BMI observed"
)
MASS::truehist(z$bmi[is.na(boys$bmi)],
  h = 1, xlim = c(10, 30), ymax = 0.25,
  col = mdc(2), xlab = "BMI imputed"
)
par(oldpar)

# repair the inconsistency problem by passive imputation
meth <- imp$meth
meth["bmi"] <- "~I(wgt/(hgt/100)^2)"
pred <- imp$predictorMatrix
pred["hgt", "bmi"] <- 0
pred["wgt", "bmi"] <- 0
imp2 <- mice(boys, m = 1, maxit = 2, meth = meth, pred = pred)
z2 <- complete(imp2, 1)

# show that new imputations are consistent
plot(z2$bmi, z2$wgt / (z2$hgt / 100)^2,
  col = mdc(1:2)[1 + is.na(boys$bmi)],
  ylab = "Calculated BMI"
)

# and compare distributions
oldpar <- par(mfrow = c(1, 2))
MASS::truehist(z2$bmi[!is.na(boys$bmi)],
  h = 1, xlim = c(10, 30), ymax = 0.25, col = mdc(1),
  xlab = "BMI observed"
)
MASS::truehist(z2$bmi[is.na(boys$bmi)],
  h = 1, xlim = c(10, 30), ymax = 0.25, col = mdc(2),
  xlab = "BMI imputed"
)
par(oldpar)
```

`brandsma` *Brandsma school data used Snijders and Bosker (2012)*

Description

Dataset with raw data from Snijders and Bosker (2012) containing data from 4106 pupils attending 216 schools. This dataset includes all pupils and schools with missing data.

Format

`brandsma` is a data frame with 4106 rows and 14 columns:

`sch`	School number
`pup`	Pupil ID
`iqv`	IQ verbal
`iqp`	IQ performal
`sex`	Sex of pupil
`ses`	SES score of pupil
`min`	Minority member 0/1
`rpg`	Number of repeated groups, 0, 1, 2
`lpr`	language score PRE
`lpo`	language score POST
`apr`	Arithmetic score PRE
`apo`	Arithmetic score POST
`den`	Denomination classification 1-4 - at school level
`ssi`	School SES indicator - at school level

Note

This dataset is constructed from the raw data. There are a few differences with the data set used in Chapter 4 and 5 of Snijders and Bosker:

1. All schools are included, including the five school with missing values on `langpost`.
2. Missing `denomina` codes are left as missing.
3. Aggregates are undefined in the presence of missing data in the underlying values. Variables `ses`, `iqv` and `iqp` are in their original scale, and not globally centered. No aggregate variables at the school level are included.
4. There is a wider selection of original variables. Note however that the source data contain an even wider set of variables.

Source

Constructed from `MLbook_2nded_total_4106-99.sav` from `https://www.stats.ox.ac.uk/~snijders/mlbook.htm` by function `data-raw/R/brandsma.R`

References

Brandsma, HP and Knuver, JWM (1989), Effects of school and classroom characteristics on pupil progress in language and arithmetic. International Journal of Educational Research, 13(7), 777 - 788.

Snijders, TAB and Bosker RJ (2012). Multilevel Analysis, 2nd Ed. Sage, Los Angeles, 2012.

`bwplot.mads` *Box-and-whisker plot of amputed and non-amputed data*

Description

Plotting method to investigate the relation between the data variables and the amputed data. The function shows how the amputed values are related to the variable values.

Usage

```
## S3 method for class 'mads'
bwplot(
  x,
  data,
  which.pat = NULL,
  standardized = TRUE,
  descriptives = TRUE,
  layout = NULL,
  ...
)
```

Arguments

`x`	A `mads` (`mads`) object, typically created by `ampute`.
`data`	A string or vector of variable names that needs to be plotted. As a default, all variables will be plotted.
`which.pat`	A scalar or vector indicating which patterns need to be plotted. As a default, all patterns are plotted.
`standardized`	Logical. Whether the box-and-whisker plots need to be created from standardized data or not. Default is TRUE.
`descriptives`	Logical. Whether the mean, variance and n of the variables need to be printed. This is useful to examine the effect of the amputation. Default is TRUE.
`layout`	A vector of two values indicating how the boxplots of one pattern should be divided over the plot. For example, `c(2, 3)` indicates that the boxplots of six variables need to be placed on 3 rows and 2 columns. Default is 1 row and an amount of columns equal to #variables. Note that for more than 6 variables, multiple plots will be created automatically.
`...`	Not used, but for consistency with generic

Value

A list containing the box-and-whisker plots. Note that a new pattern will always be shown in a new plot.

Note

The `mads` object contains all the information you need to make any desired plots. Check `mads` or the vignette *Multivariate Amputation using Ampute* to understand the contents of class object `mads`.

Author(s)

Rianne Schouten, 2016

See Also

`ampute`, `bwplot`, `mads`

`bwplot.mids` *Box-and-whisker plot of observed and imputed data*

Description

Plotting methods for imputed data using **lattice**. `bwplot()` produces box-and-whisker plots. The function automatically separates the observed and imputed data. The functions extend the usual features of **lattice**.

Usage

```
## S3 method for class 'mids'
bwplot(
  x,
  data,
  na.groups = NULL,
  groups = NULL,
  as.table = TRUE,
  theme = mice.theme(),
  mayreplicate = TRUE,
  allow.multiple = TRUE,
  outer = TRUE,
  drop.unused.levels = lattice::lattice.getOption("drop.unused.levels"),
  ...,
  subscripts = TRUE,
  subset = TRUE
)
```

Arguments

`x`	A `mids` object, typically created by `mice()` or `mice.mids()`.
`data`	Formula that selects the data to be plotted. This argument follows the **lattice** rules for *formulas*, describing the primary variables (used for the per-panel display) and the optional conditioning variables (which define the subsets plotted in different panels) to be used in the plot. The formula is evaluated on the complete data set in the `long` form. Legal variable names for the formula include `names(x$data)` plus the two administrative factors `.imp` and `.id`. **Extended formula interface:** The primary variable terms (both the LHS `y` and RHS `x`) may consist of multiple terms separated by a '+' sign, e.g., `y1 + y2 ~ x | a * b`. This formula would be taken to mean that the user wants to plot both `y1 ~ x | a * b` and `y2 ~ x | a * b`, but with the `y1 ~ x` and `y2 ~ x` in *separate panels*. This behavior differs from standard **lattice**. *Only combine terms of the same type*, i.e. only factors or only numerical variables. Mixing numerical and categorical data occasionally produces odds labeling of vertical axis. For convenience, in `stripplot()` and `bwplot()` the formula `y~.imp` may be abbreviated as `y`. This applies only to a single `y`, and does not work for `y1+y2~.imp`.
`na.groups`	An expression evaluating to a logical vector indicating which two groups are distinguished (e.g. using different colors) in the display. The environment in which this expression is evaluated in the response indicator `is.na(x$data)`. The default `na.group = NULL` contrasts the observed and missing data in the LHS `y` variable of the display, i.e. groups created by `is.na(y)`. The expression `y` creates the groups according to `is.na(y)`. The expression `y1 & y2` creates groups by `is.na(y1) & is.na(y2)`, and `y1 | y2` creates groups as `is.na(y1) | is.na(y2)`, and so on.
`groups`	This is the usual `groups` arguments in **lattice**. It differs from `na.groups` because it evaluates in the completed data `data.frame(complete(x, "long", inc=TRUE))` (as usual), whereas `na.groups` evaluates in the response indicator. See `xyplot` for more details. When both `na.groups` and `groups` are specified, `na.groups` takes precedence, and `groups` is ignored.
`as.table`	See `xyplot`.
`theme`	A named list containing the graphical parameters. The default function `mice.theme` produces a short list of default colors, line width, and so on. The extensive list may be obtained from `trellis.par.get()`. Global graphical parameters like `col` or `cex` in high-level calls are still honored, so first experiment with the global parameters. Many setting consists of a pair. For example, `mice.theme` defines two symbol colors. The first is for the observed data, the second for the imputed data. The theme settings only exist during the call, and do not affect the trellis graphical parameters.
`mayreplicate`	A logical indicating whether color, line widths, and so on, may be replicated. The graphical functions attempt to choose "intelligent" graphical parameters. For example, the same color can be replicated for different element, e.g. use all reds for the imputed data. Replication may be switched off by setting the flag to `FALSE`, in order to allow the user to gain full control.

`allow.multiple`	See `xyplot`.
`outer`	See `xyplot`.
`drop.unused.levels`	See `xyplot`.
`...`	Further arguments, usually not directly processed by the high-level functions documented here, but instead passed on to other functions.
`subscripts`	See `xyplot`.
`subset`	See `xyplot`.

Details

The argument `na.groups` may be used to specify (combinations of) missingness in any of the variables. The argument `groups` can be used to specify groups based on the variable values themselves. Only one of both may be active at the same time. When both are specified, `na.groups` takes precedence over `groups`.

Use the `subset` and `na.groups` together to plots parts of the data. For example, select the first imputed data set by by `subset=.imp==1`.

Graphical parameters like `col`, `pch` and `cex` can be specified in the arguments list to alter the plotting symbols. If `length(col)==2`, the color specification to define the observed and missing groups. `col[1]` is the color of the 'observed' data, `col[2]` is the color of the missing or imputed data. A convenient color choice is `col=mdc(1:2)`, a transparent blue color for the observed data, and a transparent red color for the imputed data. A good choice is `col=mdc(1:2), pch=20, cex=1.5`. These choices can be set for the duration of the session by running `mice.theme()`.

Value

The high-level functions documented here, as well as other high-level Lattice functions, return an object of class `"trellis"`. The `update.trellis` method can be used to subsequently update components of the object, and the `print.trellis` method (usually called by default) will plot it on an appropriate plotting device.

Note

The first two arguments (`x` and `data`) are reversed compared to the standard Trellis syntax implemented in **lattice**. This reversal was necessary in order to benefit from automatic method dispatch.

In **mice** the argument `x` is always a `mids` object, whereas in **lattice** the argument `x` is always a formula.

In **mice** the argument `data` is always a formula object, whereas in **lattice** the argument `data` is usually a data frame.

All other arguments have identical interpretation.

Generates a box-and-whisker plot for each numerical variable in a `mids` object. This extends the `bwplot` generic.

Author(s)

Stef van Buuren

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer.

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

See Also

`mice`, `bwplot`

Examples

```
imp <- mice(boys, maxit = 1)

### box-and-whisker plot per imputation of all numerical variables
bwplot(imp)

### tv (testicular volume), conditional on region
bwplot(imp, tv ~ .imp | reg)

### same data, organized in a different way
bwplot(imp, tv ~ reg | .imp, theme = list())
```

`cbind` *Combine R objects by rows and columns*

Description

Functions `cbind()` and `rbind()` are defined in the `mice` package in order to enable dispatch to `cbind.mids()` and `rbind.mids()` when one of the arguments is a `data.frame`.

Usage

```
cbind(...)

rbind(...)
```

Arguments

`...`	Arguments passed on to `base::cbind` `deparse.level` integer controlling the construction of labels in the case of non-matrix-like arguments (for the default method): `deparse.level = 0` constructs no labels; the default `deparse.level = 1` typically and `deparse.level = 2` always construct labels from the argument names, see the 'Value' section below.

Details

The standard `base::cbind()` and `base::rbind()` always dispatch to `base::cbind.data.frame()` or `base::rbind.data.frame()` if one of the arguments is a `data.frame`. The versions defined in the `mice` package intercept the user command and test whether the first argument has class `"mids"`. If so, function calls `cbind.mids()`, respectively `rbind.mids()`. In all other cases, the call is forwarded to standard functions in the `base` package.

The `cbind.mids()` function combines two `mids` objects columnwise into a single object of class `mids`, or combines a single `mids` object with a `vector`, `matrix`, `factor` or `data.frame` columnwise into a `mids` object.

If both arguments of `cbind.mids()` are `mids`-objects, the `data` list components should have the same number of rows. Also, the number of imputations (`m`) should be identical. If the second argument is a `matrix`, `factor` or `vector`, it is transformed into a `data.frame`. The number of rows should match with the `data` component of the first argument.

The `cbind.mids()` function renames any duplicated variable or block names by appending `".1"`, `".2"` to duplicated names.

The `rbind.mids()` function combines two `mids` objects rowwise into a single `mids` object, or combines a `mids` object with a vector, matrix, factor or data frame rowwise into a `mids` object.

If both arguments of `rbind.mids()` are `mids` objects, then `rbind.mids()` requires that both have the same number of multiple imputations. In addition, their `data` components should match.

If the second argument of `rbind.mids()` is not a `mids` object, the columns of the arguments should match. The `where` matrix for the second argument is set to `FALSE`, signalling that any missing values in that argument were not imputed. The `ignore` vector for the second argument is set to `FALSE`. Rows inherited from the second argument will therefore influence the parameter estimation of the imputation model in any future iterations.

Value

An S3 object of class `mids`

Note

The `cbind.mids()` function constructs the elements of the new `mids` object as follows:

`data`	Columnwise combination of the data in `x` and `y`
`imp`	Combines the imputed values from `x` and `y`
`m`	Taken from `x$m`
`where`	Columnwise combination of `x$where` and `y$where`
`blocks`	Combines `x$blocks` and `y$blocks`
`call`	Vector, `call[1]` creates `x`, `call[2]` is call to `cbind.mids()`
`nmis`	Equals `c(x$nmis, y$nmis)`
`method`	Combines `x$method` and `y$method`
`predictorMatrix`	Combination with zeroes on the off-diagonal blocks
`visitSequence`	Combined as `c(x$visitSequence, y$visitSequence)`
`formulas`	Combined as `c(x$formulas, y$formulas)`
`post`	Combined as `c(x$post, y$post)`
`blots`	Combined as `c(x$blots, y$blots)`

`ignore`	Taken from `x$ignore`
`seed`	Taken from `x$seed`
`iteration`	Taken from `x$iteration`
`lastSeedValue`	Taken from `x$lastSeedValue`
`chainMean`	Combined from `x$chainMean` and `y$chainMean`
`chainVar`	Combined from `x$chainVar` and `y$chainVar`
`loggedEvents`	Taken from `x$loggedEvents`
`version`	Current package version
`date`	Current date

The `rbind.mids()` function constructs the elements of the new `mids` object as follows:

`data`	Rowwise combination of the (incomplete) data in `x` and `y`
`imp`	Equals `rbind(x$imp[[j]], y$imp[[j]])` if `y` is `mids` object; otherwise the data of `y` will be copied
`m`	Equals `x$m`
`where`	Rowwise combination of `where` arguments
`blocks`	Equals `x$blocks`
`call`	Vector, `call[1]` creates `x`, `call[2]` is call to `rbind.mids`
`nmis`	`x$nmis` + `y$nmis`
`method`	Taken from `x$method`
`predictorMatrix`	Taken from `x$predictorMatrix`
`visitSequence`	Taken from `x$visitSequence`
`formulas`	Taken from `x$formulas`
`post`	Taken from `x$post`
`blots`	Taken from `x$blots`
`ignore`	Concatenate `x$ignore` and `y$ignore`
`seed`	Taken from `x$seed`
`iteration`	Taken from `x$iteration`
`lastSeedValue`	Taken from `x$lastSeedValue`
`chainMean`	Set to `NA`
`chainVar`	Set to `NA`
`loggedEvents`	Taken from `x$loggedEvents`
`version`	Taken from `x$version`
`date`	Taken from `x$date`

Author(s)

Karin Groothuis-Oudshoorn, Stef van Buuren

References

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

See Also

`cbind`, `ibind`, `mids`

Examples

```
# --- cbind ---
# impute four variables at once (default)
imp <- mice(nhanes, m = 1, maxit = 1, print = FALSE)
imp$predictorMatrix

# impute two by two
data1 <- nhanes[, c("age", "bmi")]
data2 <- nhanes[, c("hyp", "chl")]
imp1 <- mice(data1, m = 2, maxit = 1, print = FALSE)
imp2 <- mice(data2, m = 2, maxit = 1, print = FALSE)

# Append two solutions
imp12 <- cbind(imp1, imp2)

# This is a different imputation model
imp12$predictorMatrix

# Append the other way around
imp21 <- cbind(imp2, imp1)
imp21$predictorMatrix

# Append 'forgotten' variable chl
data3 <- nhanes[, 1:3]
imp3 <- mice(data3, maxit = 1, m = 2, print = FALSE)
imp4 <- cbind(imp3, chl = nhanes$chl)

# Of course, chl was not imputed
head(complete(imp4))

# Combine mids object with data frame
imp5 <- cbind(imp3, nhanes2)
head(complete(imp5))

# --- rbind ---
imp1 <- mice(nhanes[1:13, ], m = 2, maxit = 1, print = FALSE)
imp5 <- mice(nhanes[1:13, ], m = 2, maxit = 2, print = FALSE)
mylist <- list(age = NA, bmi = NA, hyp = NA, chl = NA)

nrow(complete(rbind(imp1, imp5)))
nrow(complete(rbind(imp1, mylist)))

nrow(complete(rbind(imp1, data.frame(mylist))))
nrow(complete(rbind(imp1, complete(imp5))))
```

`cc` *Select complete cases*

Description

Extracts the complete cases, also known as *listwise deletion*. `cc(x)` is similar to `na.omit(x)`, but returns an object of the same class as the input data. Dimensions are not dropped. For extracting incomplete cases, use `ici`.

Usage

```
cc(x)
```

Arguments

`x`	An `R` object. Methods are available for classes `mids`, `data.frame` and `matrix`. Also, `x` could be a vector.

Value

A `vector`, `matrix` or `data.frame` containing the data of the complete cases.

Author(s)

Stef van Buuren, 2017.

See Also

`na.omit`, `cci`, `ici`

Examples

```
# cc(nhanes)   # get the 13 complete cases
# cc(nhanes$bmi) # extract complete bmi
```

`cci` *Complete case indicator*

Description

The complete case indicator is useful for extracting the subset of complete cases. The function `cci(x)` calls `complete.cases(x)`. The companion function `ici()` selects the incomplete cases.

Usage

```
cci(x)
```

Arguments

`x`	An `R` object. Currently supported are methods for the following classes: `mids`.

Value

Logical vector indicating the complete cases.

Author(s)

Stef van Buuren, 2017.

See Also

`complete.cases`, `ici`, `cc`

Examples

```
cci(nhanes) # indicator for 13 complete cases
cci(mice(nhanes, maxit = 0))
f <- cci(nhanes[, c("bmi", "hyp")]) # complete data for bmi and hyp
nhanes[f, ] # obtain all data from those with complete bmi and hyp
```

`complete.mids` *Extracts the completed data from a* `mids` *object*

Description

Takes an object of class `mids`, fills in the missing data, and returns the completed data in a specified format.

Usage

```
## S3 method for class 'mids'
complete(
  data,
  action = 1L,
  include = FALSE,
  mild = FALSE,
  order = c("last", "first"),
  ...
)
```

Arguments

`data`	An object of class `mids` as created by the function `mice()`.
`action`	A numeric vector or a keyword. Numeric values between 1 and `data$m` return the data with imputation number `action` filled in. The value of `action = 0` return the original data, with missing values. `action` can also be one of the following keywords: `"all"`, `"long"`, `"broad"` and `"repeated"`. See the Details section for the interpretation. The default is `action = 1L` returns the first imputed data set.

`include`	A logical to indicate whether the original data with the missing values should be included.
`mild`	A logical indicating whether the return value should always be an object of class `mild`. Setting `mild = TRUE` overrides `action` keywords `"long"`, `"broad"` and `"repeated"`. The default is `FALSE`.
`order`	Either `"first"` or `"last"`. Only relevant when `action == "long"`. Writes the `".imp"` and `".id"` in columns 1 and 2. The default is `order = "last"`. Included for backward compatibility with `"< mice 3.16.0"`.
`...`	Additional arguments. Not used.

Details

The argument `action` can be length-1 character, which is matched to one of the following keywords:

- `"all"` produces a `mild` object of imputed data sets. When `include = TRUE`, then the original data are appended as the first list element;
- `"long"` produces a data set where imputed data sets are stacked vertically. The columns are added: 1) `.imp`, integer, referring the imputation number, and 2) `.id`, character, the row names of `data$data`;
- `"stacked"` same as `"long"` but without the two additional columns;
- `"broad"` produces a data set with where imputed data sets are stacked horizontally. Columns are ordered as in the original data. The imputation number is appended to each column name;
- `"repeated"` same as `"broad"`, but with columns in a different order.

Value

Complete data set with missing values replaced by imputations. A `data.frame`, or a list of data frames of class `mild`.

Note

Technical note: `mice 3.7.5` renamed the `complete()` function to `complete.mids()` and exported it as an S3 method of the generic `tidyr::complete()`. Name clashes between `mice::complete()` and `tidyr::complete()` should no longer occur.

See Also

`mice`, `mids`

Examples

```
# obtain first imputed data set
sum(is.na(nhanes2))
imp <- mice(nhanes2, print = FALSE, maxit = 1)
dat <- complete(imp)
sum(is.na(dat))

# obtain stacked third and fifth imputation
```

```
dat <- complete(imp, c(3, 5))

# obtain all datasets, with additional identifiers
head(complete(imp, "long"))

# same, but now as list, mild object
dslist <- complete(imp, "all")
length(dslist)

# same, but also include the original data
dslist <- complete(imp, "all", include = TRUE)
length(dslist)

# select original + 3 + 5, store as mild
dslist <- complete(imp, c(0, 3, 5), mild = TRUE)
names(dslist)
```

`construct.blocks` *Construct blocks from* `formulas` *and* `predictorMatrix`

Description

This helper function attempts to find blocks of variables in the specification of the `formulas` and/or `predictorMatrix` objects. Blocks specified by `formulas` may consist of multiple variables. Blocks specified by `predictorMatrix` are assumed to consist of single variables. Any duplicates in names are removed, and the formula specification is preferred. `predictorMatrix` and `formulas`. When both arguments specify models for the same block, the model for the `predictMatrix` is removed, and priority is given to the specification given in `formulas`.

Usage

```
construct.blocks(formulas = NULL, predictorMatrix = NULL)
```

Arguments

`formulas`	A named list of formula's, or expressions that can be converted into formula's by `as.formula`. List elements correspond to blocks. The block to which the list element applies is identified by its name, so list names must correspond to block names. The `formulas` argument is an alternative to the `predictorMatrix` argument that allows for more flexibility in specifying imputation models, e.g., for specifying interaction terms.
`predictorMatrix`	A numeric matrix of `length(blocks)` rows and `ncol(data)` columns, containing 0/1 data specifying the set of predictors to be used for each target column. Each row corresponds to a variable block, i.e., a set of variables to be imputed. A value of `1` means that the column variable is used as a predictor for the target block (in the rows). By default, the `predictorMatrix` is a square matrix of `ncol(data)` rows and columns with all 1's, except for the diagonal. Note: For two-level imputation models (which have `"2l"` in their names) other codes (e.g, `2` or `-2`) are also allowed.

Value

A `blocks` object.

See Also

`make.blocks`, `name.blocks`

Examples

```
form <- list(bmi + hyp ~ chl + age, chl ~ bmi)
pred <- make.predictorMatrix(nhanes[, c("age", "chl")])
construct.blocks(formulas = form, pred = pred)
```

`convergence` *Computes convergence diagnostics for a* `mids` *object*

Description

Takes an object of class `mids`, computes the autocorrelation and/or potential scale reduction factor, and returns a `data.frame` with the specified diagnostic(s) per iteration.

Usage

```
convergence(data, diagnostic = "all", parameter = "mean", ...)
```

Arguments

`data`	An object of class `mids` as created by the function `mice()`.
`diagnostic`	A keyword. One of the following keywords: `"ac"`, `"all"`, `"gr"` and `"psrf"`. See the Details section for the interpretation. The default is `diagnostic = "all"` which returns both the autocorrelation and potential scale reduction factor per iteration.
`parameter`	A keyword. One of the following keywords: `"mean"` or `"sd"` to evaluate chain means or chain standard deviations, respectively.
`...`	Additional arguments. Not used.

Details

The argument `diagnostic` can be length-1 character, which is matched to one of the following keywords:

`"all"` computes both the lag-1 autocorrelation as well as the potential scale reduction factor (cf. Vehtari et al., 2021) per iteration of the MICE algorithm;

`"ac"` computes only the autocorrelation per iteration;

`"psrf"` computes only the potential scale reduction factor per iteration;

`"gr"` same as `psrf`, the potential scale reduction factor is colloquially called the Gelman-Rubin diagnostic.

In the unlikely event of perfect convergence, the autocorrelation equals zero and the potential scale reduction factor equals one. To interpret the convergence diagnostic(s) in the output of the function, it is recommended to plot the diagnostics (ac and/or psrf) against the iteration number (.it) per imputed variable (vrb). A persistently decreasing trend across iterations indicates potential non-convergence.

Value

A `data.frame` with the autocorrelation and/or potential scale reduction factor per iteration of the MICE algorithm.

References

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Burkner, P.-C. (2021). Rank-Normalization, Folding, and Localization: An Improved R for Assessing Convergence of MCMC. Bayesian Analysis, 1(1), 1-38. https://doi.org/10.1214/20-BA1221

See Also

`mice`, `mids`

Examples

```
## Not run:
# obtain imputed data set
imp <- mice(nhanes2, print = FALSE)
# compute convergence diagnostics
convergence(imp)

## End(Not run)
```

`D1` *Compare two nested models using D1-statistic*

Description

The D1-statistics is the multivariate Wald test.

Usage

```
D1(fit1, fit0 = NULL, dfcom = NULL, df.com = NULL)
```

Arguments

`fit1`	An object of class `mira`, produced by `with()`.
`fit0`	An object of class `mira`, produced by `with()`. The model in `fit0` is a nested within `fit1`. The default null model `fit0 = NULL` compares `fit1` to the intercept-only model.
`dfcom`	A single number denoting the complete-data degrees of freedom of model `fit1`. If not specified, it is set equal to `df.residual` of model `fit1`. If that cannot be done, the procedure assumes (perhaps incorrectly) a large sample.
`df.com`	Deprecated

Note

Warning: `D1()` assumes that the order of the variables is the same in different models. See `https://github.com/amices/mice/issues/420` for details.

References

Li, K. H., T. E. Raghunathan, and D. B. Rubin. 1991. Large-Sample Significance Levels from Multiply Imputed Data Using Moment-Based Statistics and an F Reference Distribution. *Journal of the American Statistical Association*, 86(416): 1065–73.

`https://stefvanbuuren.name/fimd/sec-multiparameter.html#sec:wald`

See Also

`testModels`

Examples

```
# Compare two linear models:
imp <- mice(nhanes2, seed = 51009, print = FALSE)
mi1 <- with(data = imp, expr = lm(bmi ~ age + hyp + chl))
mi0 <- with(data = imp, expr = lm(bmi ~ age + hyp))
D1(mi1, mi0)
## Not run:
# Compare two logistic regression models
imp <- mice(boys, maxit = 2, print = FALSE)
fit1 <- with(imp, glm(gen > levels(gen)[1] ~ hgt + hc + reg, family = binomial))
fit0 <- with(imp, glm(gen > levels(gen)[1] ~ hgt + hc, family = binomial))
D1(fit1, fit0)

## End(Not run)
```

D2 *Compare two nested models using D2-statistic*

Description

The D2-statistic pools test statistics from the repeated analyses. The method is less powerful than the D1- and D3-statistics.

Usage

```
D2(fit1, fit0 = NULL, use = "wald")
```

Arguments

`fit1`	An object of class `mira`, produced by `with()`.
`fit0`	An object of class `mira`, produced by `with()`. The model in `fit0` is a nested within `fit1`. The default null model `fit0 = NULL` compares `fit1` to the intercept-only model.
`use`	A character string denoting Wald- or likelihood-based based tests. Can be either `"wald"` or `"likelihood"`. Only used if `method = "D2"`.

Note

Warning: `D2()` assumes that the order of the variables is the same in different models. See `https://github.com/amices/mice/issues/420` for details.

References

Li, K. H., X. L. Meng, T. E. Raghunathan, and D. B. Rubin. 1991. Significance Levels from Repeated p-Values with Multiply-Imputed Data. *Statistica Sinica* 1 (1): 65–92.

`https://stefvanbuuren.name/fimd/sec-multiparameter.html#sec:chi`

See Also

`testModels`

Examples

```
# Compare two linear models:
imp <- mice(nhanes2, seed = 51009, print = FALSE)
mi1 <- with(data = imp, expr = lm(bmi ~ age + hyp + chl))
mi0 <- with(data = imp, expr = lm(bmi ~ age + hyp))
D2(mi1, mi0)
## Not run:
# Compare two logistic regression models
imp <- mice(boys, maxit = 2, print = FALSE)
fit1 <- with(imp, glm(gen > levels(gen)[1] ~ hgt + hc + reg, family = binomial))
fit0 <- with(imp, glm(gen > levels(gen)[1] ~ hgt + hc, family = binomial))
```

```
D2(fit1, fit0)

## End(Not run)
```

`D3` *Compare two nested models using D3-statistic*

Description

The D3-statistic is a likelihood-ratio test statistic.

Usage

```
D3(fit1, fit0 = NULL, dfcom = NULL, df.com = NULL)
```

Arguments

`fit1`	An object of class `mira`, produced by `with()`.
`fit0`	An object of class `mira`, produced by `with()`. The model in `fit0` is a nested within `fit1`. The default null model `fit0 = NULL` compares `fit1` to the intercept-only model.
`dfcom`	A single number denoting the complete-data degrees of freedom of model `fit1`. If not specified, it is set equal to `df.residual` of model `fit1`. If that cannot be done, the procedure assumes (perhaps incorrectly) a large sample.
`df.com`	Deprecated

Details

The `D3()` function implement the LR-method by Meng and Rubin (1992). The implementation of the method relies on the `broom` package, the standard `update` mechanism for statistical models in `R` and the `offset` function.

The function calculates `m` repetitions of the full (or null) models, calculates the mean of the estimates of the (fixed) parameter coefficients β. For each imputed imputed dataset, it calculates the likelihood for the model with the parameters constrained to β.

The `mitml::testModels()` function offers similar functionality for a subset of statistical models. Results of `mice::D3()` and `mitml::testModels()` differ in multilevel models because the `testModels()` also constrains the variance components parameters. For more details on

Value

An object of class `mice.anova`

References

Meng, X. L., and D. B. Rubin. 1992. Performing Likelihood Ratio Tests with Multiply-Imputed Data Sets. *Biometrika*, 79 (1): 103–11.

https://stefvanbuuren.name/fimd/sec-multiparameter.html#sec:likelihoodratio

http://bbolker.github.io/mixedmodels-misc/glmmFAQ.html#setting-residual-variances-to-a-fixed-value

See Also

`fix.coef`

Examples

```
# Compare two linear models:
imp <- mice(nhanes2, seed = 51009, print = FALSE)
mi1 <- with(data = imp, expr = lm(bmi ~ age + hyp + chl))
mi0 <- with(data = imp, expr = lm(bmi ~ age + hyp))
D3(mi1, mi0)
## Not run:
# Compare two logistic regression models
imp <- mice(boys, maxit = 2, print = FALSE)
fit1 <- with(imp, glm(gen > levels(gen)[1] ~ hgt + hc + reg, family = binomial))
fit0 <- with(imp, glm(gen > levels(gen)[1] ~ hgt + hc, family = binomial))
D3(fit1, fit0)

## End(Not run)
```

`densityplot.mids` *Density plot of observed and imputed data*

Description

Plotting methods for imputed data using **lattice**. `densityplot` produces plots of the densities. The function automatically separates the observed and imputed data. The functions extend the usual features of **lattice**.

Usage

```
## S3 method for class 'mids'
densityplot(
  x,
  data,
  na.groups = NULL,
  groups = NULL,
  as.table = TRUE,
  plot.points = FALSE,
  theme = mice.theme(),
  mayreplicate = TRUE,
  thicker = 2.5,
  allow.multiple = TRUE,
  outer = TRUE,
  drop.unused.levels = lattice::lattice.getOption("drop.unused.levels"),
  panel = lattice::lattice.getOption("panel.densityplot"),
 default.prepanel = lattice::lattice.getOption("prepanel.default.densityplot"),
  ...,
  subscripts = TRUE,
```

```
  subset = TRUE
)
```

Arguments

`x`	A `mids` object, typically created by `mice()` or `mice.mids()`.
`data`	Formula that selects the data to be plotted. This argument follows the **lattice** rules for *formulas*, describing the primary variables (used for the per-panel display) and the optional conditioning variables (which define the subsets plotted in different panels) to be used in the plot. The formula is evaluated on the complete data set in the `long` form. Legal variable names for the formula include `names(x$data)` plus the two administrative factors `.imp` and `.id`. **Extended formula interface:** The primary variable terms (both the LHS `y` and RHS `x`) may consist of multiple terms separated by a '+' sign, e.g., `y1 + y2 ~ x | a * b`. This formula would be taken to mean that the user wants to plot both `y1 ~ x | a * b` and `y2 ~ x | a * b`, but with the `y1 ~ x` and `y2 ~ x` in *separate panels*. This behavior differs from standard **lattice**. *Only combine terms of the same type*, i.e. only factors or only numerical variables. Mixing numerical and categorical data occasionally produces odds labeling of vertical axis. The function `densityplot` does not use the `y` terms in the formula. Density plots for `x1` and `x2` are requested as `~ x1 + x2`.
`na.groups`	An expression evaluating to a logical vector indicating which two groups are distinguished (e.g. using different colors) in the display. The environment in which this expression is evaluated in the response indicator `is.na(x$data)`. The default `na.group = NULL` contrasts the observed and missing data in the LHS `y` variable of the display, i.e. groups created by `is.na(y)`. The expression `y` creates the groups according to `is.na(y)`. The expression `y1 & y2` creates groups by `is.na(y1) & is.na(y2)`, and `y1 | y2` creates groups as `is.na(y1) | is.na(y2)`, and so on.
`groups`	This is the usual `groups` arguments in **lattice**. It differs from `na.groups` because it evaluates in the completed data `data.frame(complete(x, "long", inc=TRUE))` (as usual), whereas `na.groups` evaluates in the response indicator. See `xyplot` for more details. When both `na.groups` and `groups` are specified, `na.groups` takes precedence, and `groups` is ignored.
`as.table`	See `xyplot`.
`plot.points`	A logical used in `densityplot` that signals whether the points should be plotted.
`theme`	A named list containing the graphical parameters. The default function `mice.theme` produces a short list of default colors, line width, and so on. The extensive list may be obtained from `trellis.par.get()`. Global graphical parameters like `col` or `cex` in high-level calls are still honored, so first experiment with the global parameters. Many setting consists of a pair. For example, `mice.theme` defines two symbol colors. The first is for the observed data, the second for the imputed data. The theme settings only exist during the call, and do not affect the trellis graphical parameters.

`mayreplicate`	A logical indicating whether color, line widths, and so on, may be replicated. The graphical functions attempt to choose "intelligent" graphical parameters. For example, the same color can be replicated for different element, e.g. use all reds for the imputed data. Replication may be switched off by setting the flag to `FALSE`, in order to allow the user to gain full control.
`thicker`	Used in `densityplot`. Multiplication factor of the line width of the observed density. `thicker=1` uses the same thickness for the observed and imputed data.
`allow.multiple`	See `xyplot`.
`outer`	See `xyplot`.
`drop.unused.levels`	See `xyplot`.
`panel`	See `xyplot`.
`default.prepanel`	See `xyplot`.
`...`	Further arguments, usually not directly processed by the high-level functions documented here, but instead passed on to other functions.
`subscripts`	See `xyplot`.
`subset`	See `xyplot`.

Details

The argument `na.groups` may be used to specify (combinations of) missingness in any of the variables. The argument `groups` can be used to specify groups based on the variable values themselves. Only one of both may be active at the same time. When both are specified, `na.groups` takes precedence over `groups`.

Use the `subset` and `na.groups` together to plots parts of the data. For example, select the first imputed data set by by `subset=.imp==1`.

Graphical parameters like `col`, `pch` and `cex` can be specified in the arguments list to alter the plotting symbols. If `length(col)==2`, the color specification to define the observed and missing groups. `col[1]` is the color of the 'observed' data, `col[2]` is the color of the missing or imputed data. A convenient color choice is `col=mdc(1:2)`, a transparent blue color for the observed data, and a transparent red color for the imputed data. A good choice is `col=mdc(1:2), pch=20, cex=1.5`. These choices can be set for the duration of the session by running `mice.theme()`.

Value

The high-level functions documented here, as well as other high-level Lattice functions, return an object of class `"trellis"`. The `update.trellis` method can be used to subsequently update components of the object, and the `print.trellis` method (usually called by default) will plot it on an appropriate plotting device.

Note

The first two arguments (`x` and `data`) are reversed compared to the standard Trellis syntax implemented in **lattice**. This reversal was necessary in order to benefit from automatic method dispatch.

In **mice** the argument `x` is always a `mids` object, whereas in **lattice** the argument `x` is always a formula.

In **mice** the argument `data` is always a formula object, whereas in **lattice** the argument `data` is usually a data frame.

All other arguments have identical interpretation.

`densityplot` errs on empty groups, which occurs if all observations in the subgroup contain `NA`. The relevant error message is: `Error in density.default: ... need at least 2 points to select a bandwidth automatically`. There is yet no workaround for this problem. Use the more robust `bwplot` or `stripplot` as a replacement.

Author(s)

Stef van Buuren

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer.

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

Examples

```
imp <- mice(boys, maxit = 1)

### density plot of head circumference per imputation
### blue is observed, red is imputed
densityplot(imp, ~ hc | .imp)

### All combined in one panel.
densityplot(imp, ~hc)
```

`employee` *Employee selection data*

Description

A toy example from Craig Enders.

Usage

```
employee
```

Format

A data frame with 20 rows and 3 variables:

IQ candidate IQ score

wbeing candidate well-being score

jobperf candidate job performance score

Details

Enders describes these data as follows: I designed these data to mimic an employee selection scenario in which prospective employees complete an IQ test and a psychological well-being questionnaire during their interview. The company subsequently hires the applications that score in the upper half of the IQ distribution, and a supervisor rates their job performance following a 6-month probationary period. Note that the job performance scores are missing at random (MAR) (i.e. individuals in the lower half of the IQ distribution were never hired, and thus have no performance rating). In addition, I randomly deleted three of the well-being scores in order to mimic a situation where the applicant's well-being questionnaire is inadvertently lost.

A larger version of this data set in present as `data.enders.employee`.

Source

Enders (2010), Applied Missing Data Analysis, p. 218

`estimice` *Computes least squares parameters*

Description

This function computes least squares estimates, variance/covariance matrices, residuals and degrees of freedom according to ridge regression, QR decomposition or Singular Value Decomposition. This function is internally called by .norm.draw(), but can be called by any user-specified imputation function.

Usage

```
estimice(x, y, ls.meth = "qr", ridge = 1e-05, ...)
```

Arguments

`x`	Matrix (`n` x `p`) of complete covariates.
`y`	Incomplete data vector of length `n`
`ls.meth`	the method to use for obtaining the least squares estimates. By default parameters are drawn by means of QR decomposition.
`ridge`	A small numerical value specifying the size of the ridge used. The default value `ridge = 1e-05` represents a compromise between stability and unbiasedness. Decrease `ridge` if the data contain many junk variables. Increase `ridge` for highly collinear data.
`...`	Other named arguments.

Details

When calculating the inverse of the crossproduct of the predictor matrix, problems may arise. For example, taking the inverse is not possible when the predictor matrix is rank deficient, or when the estimation problem is computationally singular. This function detects such error cases and automatically falls back to adding a ridge penalty to the diagonal of the crossproduct to allow for proper calculation of the inverse.

Value

A `list` containing components `c` (least squares estimate), `r` (residuals), `v` (variance/covariance matrix) and `df` (degrees of freedom).

Note

This functions adds a star to variable names in the mice iteration history to signal that a ridge penalty was added. In that case, it also adds an entry to `loggedEvents`.

Author(s)

Gerko Vink, 2018

`extractBS` *Extract broken stick estimates from a `lmer` object*

Description

Extract broken stick estimates from a `lmer` object

Usage

```
extractBS(fit)
```

Arguments

`fit`	An object of class `lmer`

Value

A matrix containing broken stick estimates

Author(s)

Stef van Buuren, 2012

`fdd` *SE Fireworks disaster data*

Description

Multiple outcomes of a randomized study to reduce post-traumatic stress.

Format

`fdd` is a data frame with 52 rows and 65 columns:

id Client number

trt Treatment (E=EMDR, C=CBT)

pp Per protocol (Y/N)

trtp Number of parental treatments

sex Sex: M/F

etn Ethnicity: NL/OTHER

age Age (years)

trauma Trauma count (1-5)

prop1 PROPS total score T1

prop2 PROPS total score T2

prop3 PROPS total score T3

crop1 CROPS total score T1

crop2 CROPS total score T2

crop3 CROPS total score T3

masc1 MASC score T1

masc2 MASC score T2

masc3 MASC score T3

cbcl1 CBCL T1

cbcl3 CBCL T3

prs1 PRS total score T1

prs2 PRS total score T2

prs3 PRS total score T3

ypa1 PTSD-RI B intrusive recollection parent T1

ypb1 PTSD-RI C avoidant/numbing parent T1

ypc1 PTSD-RI D hyper-arousal parent T1

yp1 PTSD-RI B+C+D parent T1

ypa2 PTSD-RI B intrusive recollection parent T2

ypb2 PTSD-RI C avoidant/numbing parent T2

ypc2 PTSD-RI D hyper-arousal parent T2

yp2 PTSD-RI B+C+D parent T1

ypa3 PTSD-RI B intrusive recollection parent T3

ypb3 PTSD-RI C avoidant/numbing parent T3

ypc3 PTSD-RI D hyper-arousal parent T3

yp3 PTSD-RI B+C+D parent T3

yca1 PTSD-RI B intrusive recollection child T1

ycb1 PTSD-RI C avoidant/numbing child T1

ycc1 PTSD-RI D hyper-arousal child T1

yc1 PTSD-RI B+C+D child T1

yca2 PTSD-RI B intrusive recollection child T2

ycb2 PTSD-RI C avoidant/numbing child T2

ycc2 PTSD-RI D hyper-arousal child T2

yc2 PTSD-RI B+C+D child T2

yca3 PTSD-RI B intrusive recollection child T3

ycb3 PTSD-RI C avoidant/numbing child T3

ycc3 PTSD-RI D hyper-arousal child T3

yc3 PTSD-RI B+C+D child T3

ypf1 PTSD-RI parent full T1

ypf2 PTSD-RI parent full T2

ypf3 PTSD-RI parent full T3

ypp1 PTSD parent partial T1

ypp2 PTSD parent partial T2

ypp3 PTSD parent partial T3

ycf1 PTSD child full T1

ycf2 PTSD child full T2

ycf3 PTSD child full T3

ycp1 PTSD child partial T1

ycp2 PTSD child partial T2

ycp3 PTSD child partial T3

cbin1 CBCL Internalizing T1

cbin3 CBCL Internalizing T3

cbex1 CBCL Externalizing T1

cbex3 CBCL Externalizing T3

bir1 Birlison T1

bir2 Birlison T2

bir3 Birlison T3

`fdd.pred` is the 65 by 65 binary predictor matrix used to impute `fdd`.

Details

Data from a randomized experiment to reduce post-traumatic stress by two treatments: Eye Movement Desensitization and Reprocessing (EMDR) (experimental treatment), and cognitive behavioral therapy (CBT) (control treatment). 52 children were randomized to one of these two treatments. Outcomes were measured at three time points: at baseline (pre-treatment, T1), post-treatment (T2, 4-8 weeks), and at follow-up (T3, 3 months). For more details, see de Roos et al (2011). Some person covariates were reshuffled. The imputation methodology is explained in Chapter 9 of van Buuren (2012).

Source

de Roos, C., Greenwald, R., den Hollander-Gijsman, M., Noorthoorn, E., van Buuren, S., de Jong, A. (2011). A Randomised Comparison of Cognitive Behavioral Therapy (CBT) and Eye Movement Desensitisation and Reprocessing (EMDR) in disaster-exposed children. *European Journal of Psychotraumatology*, *2*, 5694.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL. Boca Raton, FL.: Chapman & Hall/CRC Press.

Examples

```
data <- fdd
md.pattern(fdd)
```

`fdgs` *Fifth Dutch growth study 2009*

Description

Age, height, weight and region of 10030 children measured within the Fifth Dutch Growth Study 2009

Format

`fdgs` is a data frame with 10030 rows and 8 columns:

id Person number

reg Region (factor, 5 levels)

age Age (years)

sex Sex (boy, girl)

hgt Height (cm)

wgt Weight (kg)

hgt.z Height Z-score

wgt.z Weight Z-score

Details

The data set contains data from children of Dutch descent (biological parents are born in the Netherlands). Children with growth-related diseases were excluded. The data were used to construct new growth charts of children of Dutch descent (Schonbeck 2013), and to calculate overweight and obesity prevalence (Schonbeck 2011).

Some groups were underrepresented. Multiple imputation was used to create synthetic cases that were used to correct for the nonresponse. See Van Buuren (2012), chapter 8 for details.

Source

Schonbeck, Y., Talma, H., van Dommelen, P., Bakker, B., Buitendijk, S. E., Hirasing, R. A., van Buuren, S. (2011). Increase in prevalence of overweight in Dutch children and adolescents: A comparison of nationwide growth studies in 1980, 1997 and 2009. *PLoS ONE*, *6*(11), e27608.

Schonbeck, Y., Talma, H., van Dommelen, P., Bakker, B., Buitendijk, S. E., Hirasing, R. A., van Buuren, S. (2013). The world's tallest nation has stopped growing taller: the height of Dutch children from 1955 to 2009. *Pediatric Research*, *73*(3), 371-377.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Boca Raton, FL.: Chapman & Hall/CRC Press.

Examples

```
data <- data(fdgs)
summary(data)
```

`fico` *Fraction of incomplete cases among cases with observed*

Description

FICO is an outbound statistic defined by the fraction of incomplete cases among cases with `Yj` observed (White and Carlin, 2010).

Usage

```
fico(data)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as NA's.

Value

A vector of length `ncol(data)` of FICO statistics.

Author(s)

Stef van Buuren, 2012

References

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

White, I.R., Carlin, J.B. (2010). Bias and efficiency of multiple imputation compared with complete-case analysis for missing covariate values. *Statistics in Medicine*, *29*, 2920-2931.

See Also

`fluxplot`, `flux`, `md.pattern`

`filter.mids` *Subset rows of a* `mids` *object*

Description

This function takes a `mids` object and returns a new `mids` object that pertains to the subset of the data identified by the expression in `...`. The expression may use column values from the incomplete data in `.data$data`.

Usage

```
## S3 method for class 'mids'
filter(.data, ..., .preserve = FALSE)
```

Arguments

`.data`	A `mids` object.
`...`	Expressions that return a logical value, and are defined in terms of the variables in `.data$data`. If multiple expressions are specified, they are combined with the `&` operator. Only rows for which all conditions evaluate to `TRUE` are kept.
`.preserve`	Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Value

An S3 object of class `mids`

Note

The function calculates a logical vector `include` of length `nrow(.data$data)`. The function constructs the elements of the filtered `mids` object as follows:

`data`	Select rows in `.data$data` for which `include == TRUE`
`imp`	Select rows each imputation `data.frame` in `.data$imp` for which `include == TRUE`
`m`	Equals `.data$m`
`where`	Select rows in `.data$where` for which `include == TRUE`
`blocks`	Equals `.data$blocks`
`call`	Equals `.data$call`
`nmis`	Recalculate `nmis` based on the selected `data` rows
`method`	Equals `.data$method`
`predictorMatrix`	Equals `.data$predictorMatrix`
`visitSequence`	Equals `.data$visitSequence`
`formulas`	Equals `.data$formulas`
`post`	Equals `.data$post`
`blots`	Equals `.data$blots`
`ignore`	Select positions in `.data$ignore` for which `include == TRUE`
`seed`	Equals `.data$seed`
`iteration`	Equals `.data$iteration`
`lastSeedValue`	Equals `.data$lastSeedValue`
`chainMean`	Set to `NULL`
`chainVar`	Set to `NULL`
`loggedEvents`	Equals `.data$loggedEvents`
`version`	Replaced with current version
`date`	Replaced with current date

Author(s)

Patrick Rockenschaub

See Also

`filter`

Examples

```
imp <- mice(nhanes, m = 2, maxit = 1, print = FALSE)

# example with external logical vector
imp_f <- filter(imp, c(rep(TRUE, 13), rep(FALSE, 12)))

nrow(complete(imp))
nrow(complete(imp_f))

# example with calculated include vector
imp_f2 <- filter(imp, age >= 2 & hyp == 1)
nrow(complete(imp_f2)) # should be 5
```

`fix.coef` *Fix coefficients and update model*

Description

Refits a model with a specified set of coefficients.

Usage

```
fix.coef(model, beta = NULL)
```

Arguments

`model`	An R model, e.g., produced by `lm` or `glm`
`beta`	A numeric vector with `length(coef)` model coefficients. If the vector is not named, the coefficients should be given in the same order as in `coef(model)`. If the vector is named, the procedure attempts to match on names.

Details

The function calculates the linear predictor using the new coefficients, and reformulates the model using the `offset` argument. The linear predictor is called `offset`, and its coefficient will be `1` by definition. The new model only fits the intercept, which should be `0` if we set `beta = coef(model)`.

Value

An updated R model object

Author(s)

Stef van Buuren, 2018

Examples

```
model0 <- lm(Volume ~ Girth + Height, data = trees)
formula(model0)
coef(model0)
deviance(model0)

# refit same model
model1 <- fix.coef(model0)
formula(model1)
coef(model1)
deviance(model1)

# change the beta's
model2 <- fix.coef(model0, beta = c(-50, 5, 1))
coef(model2)
deviance(model2)
```

```
# compare predictions
plot(predict(model0), predict(model1))
abline(0, 1)
plot(predict(model0), predict(model2))
abline(0, 1)

# compare proportion explained variance
cor(predict(model0), predict(model0) + residuals(model0))^2
cor(predict(model1), predict(model1) + residuals(model1))^2
cor(predict(model2), predict(model2) + residuals(model2))^2

# extract offset from constrained model
summary(model2$offset)

# it also works with factors and missing data
model0 <- lm(bmi ~ age + hyp + chl, data = nhanes2)
model1 <- fix.coef(model0)
model2 <- fix.coef(model0, beta = c(15, -8, -8, 2, 0.2))
```

`flux` *Influx and outflux of multivariate missing data patterns*

Description

Influx and outflux are statistics of the missing data pattern. These statistics are useful in selecting predictors that should go into the imputation model.

Usage

```
flux(data, local = names(data))
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as NA's.
`local`	A vector of names of columns of `data`. The default is to include all columns in the calculations.

Details

Infux and outflux have been proposed by Van Buuren (2018), chapter 4.

Influx is equal to the number of variable pairs `(Yj , Yk)` with `Yj` missing and `Yk` observed, divided by the total number of observed data cells. Influx depends on the proportion of missing data of the variable. Influx of a completely observed variable is equal to 0, whereas for completely missing variables we have influx = 1. For two variables with the same proportion of missing data, the variable with higher influx is better connected to the observed data, and might thus be easier to impute.

Outflux is equal to the number of variable pairs with `Yj` observed and `Yk` missing, divided by the total number of incomplete data cells. Outflux is an indicator of the potential usefulness of `Yj` for imputing other variables. Outflux depends on the proportion of missing data of the variable. Outflux of a completely observed variable is equal to 1, whereas outflux of a completely missing variable is equal to 0. For two variables having the same proportion of missing data, the variable with higher outflux is better connected to the missing data, and thus potentially more useful for imputing other variables.

FICO is an outbound statistic defined by the fraction of incomplete cases among cases with `Yj` observed (White and Carlin, 2010).

Value

A data frame with `ncol(data)` rows and six columns: pobs = Proportion observed, influx = Influx outflux = Outflux ainb = Average inbound statistic aout = Average outbound statistic fico = Fraction of incomplete cases among cases with `Yj` observed

Author(s)

Stef van Buuren, 2012

References

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

White, I.R., Carlin, J.B. (2010). Bias and efficiency of multiple imputation compared with complete-case analysis for missing covariate values. *Statistics in Medicine*, *29*, 2920-2931.

See Also

`fluxplot`, `md.pattern`, `fico`

`fluxplot` *Fluxplot of the missing data pattern*

Description

Influx and outflux are statistics of the missing data pattern. These statistics are useful in selecting predictors that should go into the imputation model.

Usage

```
fluxplot(
  data,
  local = names(data),
  plot = TRUE,
  labels = TRUE,
  xlim = c(0, 1),
```

```
  ylim = c(0, 1),
  las = 1,
  xlab = "Influx",
  ylab = "Outflux",
  main = paste("Influx-outflux pattern for", deparse(substitute(data))),
  eqscplot = TRUE,
  pty = "s",
  lwd = 1,
  ...
)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as NA's.
`local`	A vector of names of columns of `data`. The default is to include all columns in the calculations.
`plot`	Should a graph be produced?
`labels`	Should the points be labeled?
`xlim`	See `par`.
`ylim`	See `par`.
`las`	See `par`.
`xlab`	See `par`.
`ylab`	See `par`.
`main`	See `par`.
`eqscplot`	Should a square plot be produced?
`pty`	See `par`.
`lwd`	See `par`. Controls axis line thickness and diagonal
`...`	Further arguments passed to `plot()` or `eqscplot()`.

Details

Infux and outflux have been proposed by Van Buuren (2012), chapter 4.

Influx is equal to the number of variable pairs `(Yj , Yk)` with `Yj` missing and `Yk` observed, divided by the total number of observed data cells. Influx depends on the proportion of missing data of the variable. Influx of a completely observed variable is equal to 0, whereas for completely missing variables we have influx = 1. For two variables with the same proportion of missing data, the variable with higher influx is better connected to the observed data, and might thus be easier to impute.

Outflux is equal to the number of variable pairs with `Yj` observed and `Yk` missing, divided by the total number of incomplete data cells. Outflux is an indicator of the potential usefulness of `Yj` for imputing other variables. Outflux depends on the proportion of missing data of the variable. Outflux of a completely observed variable is equal to 1, whereas outflux of a completely missing variable is equal to 0. For two variables having the same proportion of missing data, the variable with higher outflux is better connected to the missing data, and thus potentially more useful for imputing other variables.

Value

An invisible data frame with `ncol(data)` rows and six columns: pobs = Proportion observed, influx = Influx outflux = Outflux ainb = Average inbound statistic aout = Average outbound statistic fico = Fraction of incomplete cases among cases with `Yj` observed

Author(s)

Stef van Buuren, 2012

References

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

White, I.R., Carlin, J.B. (2010). Bias and efficiency of multiple imputation compared with complete-case analysis for missing covariate values. *Statistics in Medicine*, *29*, 2920-2931.

See Also

`flux`, `md.pattern`, `fico`

`futuremice` *Wrapper function that runs MICE in parallel*

Description

This is a wrapper function for `mice`, using multiple cores to execute `mice` in parallel. As a result, the imputation procedure can be sped up, which may be useful in general. By default, `futuremice` distributes the number of imputations `m` about equally over the cores.

Usage

```
futuremice(
  data,
  m = 5,
  parallelseed = NA,
  n.core = NULL,
  seed = NA,
  use.logical = TRUE,
  future.plan = "multisession",
  packages = NULL,
  globals = NULL,
  ...
)
```

Arguments

`data`	A data frame or matrix containing the incomplete data. Similar to the first argument of `mice`.
`m`	The number of desired imputated datasets. By default $m=5$ as with `mice`
`parallelseed`	A scalar to be used to obtain reproducible results over the futures. The default `parallelseed = NA` will result in a seed value that is randomly drawn between -999999999 and 999999999.
`n.core`	A scalar indicating the number of cores that should be used.
`seed`	A scalar to be used as the seed value for the mice algorithm within each parallel stream. Please note that the imputations will be the same for all streams and, hence, this should be used if and only if `n.core = 1` and if it is desired to obtain the same output as under `mice`.
`use.logical`	A logical indicating whether logical (`TRUE`) or physical (`FALSE`) CPU's on machine should be used.
`future.plan`	A character indicating how `futures` are resolved. The default `multisession` resolves futures asynchronously (in parallel) in separate `R` sessions running in the background. See `plan` for more information on future plans.
`packages`	A character vector with additional packages to be used in `mice` (e.g., for using external imputation functions).
`globals`	A character string with additional functions to be exported to each future (e.g., user-written imputation functions).
`...`	Named arguments that are passed down to function `mice`.

Details

This function relies on package `furrr`, which is a package for R versions 3.2.0 and later. We have chosen to use furrr function `future_map` to allow the use of `futuremice` on Mac, Linux and Windows systems.

This wrapper function combines the output of `future_map` with function `ibind` from the `mice` package. A `mids` object is returned and can be used for further analyses.

A seed value can be specified in the global environment, which will yield reproducible results. A seed value can also be specified within the `futuremice` call, through specifying the argument `parallelseed`. If `parallelseed` is not specified, a seed value is drawn randomly by default, and accessible through `$parallelseed` in the output object. Hence, results will always be reproducible, regardless of whether the seed is specified in the global environment, or by setting the same seed within the function (potentially by extracting the seed from the `futuremice` output object.

Value

A mids object as defined by `mids-class`

Author(s)

Thom Benjamin Volker, Gerko Vink

References

Volker, T.B. and Vink, G. (2022). futuremice: The future starts today. https://www.gerkovink.com/miceVignettes/futuremice/Vignette_futuremice.html

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`future`, `furrr`, `future_map`, `plan`, `mice`, `mids-class`

Examples

```
# 150 imputations in dataset nhanes, performed by 3 cores
## Not run:
imp1 <- futuremice(data = nhanes, m = 150, n.core = 3)
# Making use of arguments in mice.
imp2 <- futuremice(data = nhanes, m = 100, method = "norm.nob")
imp2$method
fit <- with(imp2, lm(bmi ~ hyp))
pool(fit)

## End(Not run)
```

`getfit` *Extract list of fitted models*

Description

Function `getfit()` returns the list of objects containing the repeated analysis results, or optionally, one of these fitted objects. The function looks for a list element called `analyses`, and return this component as a list with `mira` class. If element `analyses` is not found in `x`, then it returns `x` as a `mira` object.

Usage

```
getfit(x, i = -1L, simplify = FALSE)
```

Arguments

`x`	An object of class `mira`, typically produced by a call to `with()`.
`i`	An integer between 1 and `x$m` signalling the index of the repeated analysis. The default `i= -1` return a list with all analyses.
`simplify`	Should the return value be unlisted?

Details

No checking is done for validity of objects. The function also processes objects of class `mitml.result` from the `mitml` package.

Value

If `i = -1` an object of class `mira` containing all analyses. If `i` selects one of the analyses, then it return an object whose with class inherited from that element.

Author(s)

Stef van Buuren, 2012, 2020

See Also

`mira`, `with.mids`

Examples

```
imp <- mice(nhanes, print = FALSE, seed = 21443)
fit <- with(imp, lm(bmi ~ chl + hyp))
f1 <- getfit(fit)
class(f1)
f2 <- getfit(fit, 2)
class(f2)
```

`getqbar` *Extract estimate from* `mipo` *object*

Description

`getqbar` returns a named vector of pooled estimates.

Usage

```
getqbar(x)
```

Arguments

`x`	An object of class `mipo`

`glm.mids` *Generalized linear model for* `mids` *object*

Description

Applies `glm()` to a multiply imputed data set

Usage

```
glm.mids(formula, family = gaussian, data, ...)
```

Arguments

`formula`	a formula expression as for other regression models, of the form response ~ predictors. See the documentation of `lm` and `formula` for details.
`family`	The family of the glm model
`data`	An object of type `mids`, which stands for 'multiply imputed data set', typically created by function `mice()`.
`...`	Additional parameters passed to `glm`.

Details

This function is included for backward compatibility with V1.0. The function is superseded by `with.mids`.

Value

An objects of class `mira`, which stands for 'multiply imputed repeated analysis'. This object contains `data$m` distinct `glm.objects`, plus some descriptive information.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

Van Buuren, S., Groothuis-Oudshoorn, C.G.M. (2000) *Multivariate Imputation by Chained Equations: MICE V1.0 User's manual.* Leiden: TNO Quality of Life.

See Also

`with.mids`, `glm`, `mids`, `mira`

Examples

```
imp <- mice(nhanes)

# logistic regression on the imputed data
fit <- glm.mids((hyp == 2) ~ bmi + chl, data = imp, family = binomial)
fit
```

`ibind` *Enlarge number of imputations by combining* `mids` *objects*

Description

This function combines two `mids` objects `x` and `y` into a single `mids` object, with the objective of increasing the number of imputed data sets. If the number of imputations in `x` and `y` are `m(x)` and `m(y)`, then the combined object will have `m(x)+m(y)` imputations.

Usage

```
ibind(x, y)
```

Arguments

`x`	A `mids` object.
`y`	A `mids` object.

Details

The two `mids` objects are required to have the same underlying multiple imputation model and should be fitted on the same data.

Value

An S3 object of class `mids`

Author(s)

Karin Groothuis-Oudshoorn, Stef van Buuren

See Also

`mids`

Examples

```
data(nhanes)
imp1 <- mice(nhanes, m = 1, maxit = 2, print = FALSE)
imp1$m

imp2 <- mice(nhanes, m = 3, maxit = 3, print = FALSE)
imp2$m

imp12 <- ibind(imp1, imp2)
imp12$m
plot(imp12)
```

`ic` *Select incomplete cases*

Description

Extracts incomplete cases from a data set. The companion function for selecting the complete cases is `cc`.

Usage

```
ic(x)
```

Arguments

`x`	An `R` object. Methods are available for classes `mids`, `data.frame` and `matrix`. Also, `x` could be a vector.

Value

A `vector`, `matrix` or `data.frame` containing the data of the complete cases.

Author(s)

Stef van Buuren, 2017.

See Also

`cc`, `ici`

Examples

```
ic(nhanes) # get the 12 rows with incomplete cases
ic(nhanes[1:10, ]) # incomplete cases within the first ten rows
ic(nhanes[, c("bmi", "hyp")]) # restrict extraction to variables bmi and hyp
```

`ici` *Incomplete case indicator*

Description

This array is useful for extracting the subset of incomplete cases. The companion function `cci()` selects the complete cases.

Usage

```
ici(x)
```

Arguments

`x`	An `R` object. Currently supported are methods for the following classes: `mids`.

Value

Logical vector indicating the incomplete cases,

Author(s)

Stef van Buuren, 2017.

See Also

`cci`, `ic`

Examples

```
ici(nhanes) # indicator for 12 rows with incomplete cases
```

`is.mads` *Check for* `mads` *object*

Description

Check for `mads` object

Usage

```
is.mads(x)
```

Arguments

`x`	An object

Value

A logical indicating whether `x` is an object of class `mads`

`is.mids` *Check for* `mids` *object*

Description

Check for `mids` object

Usage

```
is.mids(x)
```

Arguments

`x`	An object

Value

A logical indicating whether `x` is an object of class `mids`

`is.mipo` *Check for* `mipo` *object*

Description

Check for `mipo` object

Usage

```
is.mipo(x)
```

Arguments

`x`	An object

Value

A logical indicating whether `x` is an object of class `mipo`

`is.mira` *Check for* `mira` *object*

Description

Check for `mira` object

Usage

```
is.mira(x)
```

Arguments

`x`	An object

Value

A logical indicating whether `x` is an object of class `mira`

`is.mitml.result` *Check for* `mitml.result` *object*

Description

Check for `mitml.result` object

Usage

```
is.mitml.result(x)
```

Arguments

`x`	An object

Value

A logical indicating whether `x` is an object of class `mitml.result`

`leiden85` *Leiden 85+ study*

Description

Subset of data from the Leiden 85+ study

Format

`leiden85` is a data frame with 956 rows and 336 columns.

Details

The data set concerns of subset of 956 members of a very old (85+) cohort in Leiden.

Multiple imputation of this data set has been described in Boshuizen et al (1998), Van Buuren et al (1999) and Van Buuren (2012), chapter 7.

The data set is not available as part of `mice`.

Source

Lagaay, A. M., van der Meij, J. C., Hijmans, W. (1992). Validation of medical history taking as part of a population based survey in subjects aged 85 and over. *Brit. Med. J.*, *304*(6834), 1091-1092.

Izaks, G. J., van Houwelingen, H. C., Schreuder, G. M., Ligthart, G. J. (1997). The association between human leucocyte antigens (HLA) and mortality in community residents aged 85 and older. *Journal of the American Geriatrics Society*, *45*(1), 56-60.

Boshuizen, H. C., Izaks, G. J., van Buuren, S., Ligthart, G. J. (1998). Blood pressure and mortality in elderly people aged 85 and older: Community based study. *Brit. Med. J.*, *316*(7147), 1780-1784.

Van Buuren, S., Boshuizen, H.C., Knook, D.L. (1999) Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine*, **18**, 681–694.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

`lm.mids` *Linear regression for* `mids` *object*

Description

Applies `lm()` to multiply imputed data set

Usage

```
lm.mids(formula, data, ...)
```

Arguments

`formula`	a formula object, with the response on the left of a ~ operator, and the terms, separated by + operators, on the right. See the documentation of `lm` and `formula` for details.
`data`	An object of type 'mids', which stands for 'multiply imputed data set', typically created by a call to function `mice()`.
`...`	Additional parameters passed to `lm`

Details

This function is included for backward compatibility with V1.0. The function is superseded by `with.mids`.

Value

An objects of class `mira`, which stands for 'multiply imputed repeated analysis'. This object contains `data$m` distinct `lm.objects`, plus some descriptive information.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

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

See Also

`lm`, `mids`, `mira`

Examples

```
imp <- mice(nhanes)
fit <- lm.mids(bmi ~ hyp + chl, data = imp)
fit
```

`mads` *Multivariate amputed data set* (`mads`)

Description

The `mads` object is an S3 class that contains an amputed dataset, i.e., a dataset with simulated missing values. The `ampute()` function produces a `mads` object. The `mads()` function is the S3 constructor. The S3 class `mads` has the following methods: `bwplot.mads()`,`print()`, `summary()`, and `xyplot.mads()`.

Usage

```
mads(
  call,
  prop,
  patterns,
  freq,
  mech,
  weights,
  cont,
  type,
  odds,
  amp,
  cand,
  scores,
  data
)

## S3 method for class 'mads'
print(x, ...)

## S3 method for class 'mads'
summary(object, ...)
```

Arguments

`call`	The function call that created the object.
`prop`	A numeric vector of proportions.
`patterns`	A matrix of missing data patterns.
`freq`	A numeric vector of frequencies for each pattern.
`mech`	A character string describing the missing data mechanism.
`weights`	A matrix of weights.
`cont`	A logical vector indicating continuous variables.
`type`	A character vector describing variable types.
`odds`	A matrix of odds ratios.
`amp`	A data frame for amplitude information.
`cand`	An integer vector of candidate variables.
`scores`	A list of scores.
`data`	A data frame representing the original data.
`x`	Object of class `mads`
`...`	Other parameters
`object`	Object of class `mads`

Value

An object of class `"mads"`.

`print()` returns the input object invisibly.

`summary()` returns the input object invisibly.

Contents

- **`call`:** The function call.
- **`prop`:** Proportion of cases with missing values. Note: even when the proportion is entered as the proportion of missing cells (when `bycases == TRUE`), this object contains the proportion of missing cases.
- **`patterns`:** A data frame of size #patterns by #variables where `0` indicates a variable has missing values and `1` indicates a variable remains complete.
- **`freq`:** A vector of length #patterns containing the relative frequency with which the patterns occur. For example, if the vector is `c(0.4, 0.4, 0.2)`, this means that of all cases with missing values, 40 percent is candidate for pattern 1, 40 percent for pattern 2 and 20 percent for pattern 3. The vector sums to 1.
- **`mech`:** A string specifying the missingness mechanism, either `"MCAR"`, `"MAR"` or `"MNAR"`.
- **`weights`:** A data frame of size #patterns by #variables. It contains the weights that were used to calculate the weighted sum scores. The weights may differ between patterns and between variables.
- **`cont`:** Logical, whether probabilities are based on continuous logit functions or on discrete odds distributions.
- **`type`:** A vector of strings containing the type of missingness for each pattern. Either `"LEFT"`, `"MID"`, `"TAIL"` or `"RIGHT"`. The first type refers to the first pattern, the second type to the second pattern, etc.
- **`odds`:** A matrix where #patterns defines the #rows. Each row contains the odds of being missing for the corresponding pattern. The amount of odds values defines in how many quantiles the sum scores were divided. The values are relative probabilities: a quantile with odds value 4 will have a probability of being missing that is four times higher than a quantile with odds 1. The #quantiles may differ between patterns, NA is used for cells remaining empty.
- **`amp`:** A data frame containing the input data with NAs for the amputed values.
- **`cand`:** A vector that contains the pattern number for each case. A value between 1 and #patterns is given. For example, a case with value 2 is candidate for missing data pattern 2.
- **`scores`:** A list containing vectors with weighted sum scores of the candidates. The first vector refers to the candidates of the first pattern, the second vector refers to the candidates of the second pattern, etc. The length of the vectors differ because the number of candidates is different for each pattern.
- **`data`:** The complete data set that was entered in `ampute`.

Author(s)

Rianne Schouten, 2016

See Also

`ampute`, Vignette titled "Multivariate Amputation using Ampute".

`make.blocks` *Creates a* `blocks` *argument*

Description

The `make.blocks()` helper function generates a suitable `blocks` argument for the `[=mice]{mice}` function.

Usage

```
make.blocks(x, partition = c("scatter", "collect", "void"))
```

Arguments

`x`	A `data.frame`, character vector with variable names, or `list` with variable names.
`partition`	A character vector of length 1 used to assign variables to blocks when `data` is a `data.frame`. Value `"scatter"` (default) assigns each variable to it own block. Value `"collect"` assigns all variables to a single block.

Details

Choices `"scatter"` and `"collect"` represent to two extreme scenarios for assigning variables to imputation blocks. Use `"scatter"` to create an imputation model based on *fully conditionally specification* (FCS). Use `"collect"` to gather all variables to be imputed by a *joint model* (JM). Scenario's in-between these two extremes represent *hybrid* imputation models that combine FCS and JM.

Any variable not listed in will not be imputed. Specification `"void"` represents the extreme scenario that skips imputation of all variables.

A variable may be a member of multiple blocks. The variable will be re-imputed in each block, so the final imputations for variable will come from the last block that was executed. This scenario may be useful where the same complete background factors appear in multiple imputation blocks.

A variable may appear multiple times within a given block. If a univariate imputation model is applied to such a block, then the variable is re-imputed each time as it appears in the block.

Value

A named list of character vectors with variables names.

Examples

```
make.blocks(nhanes)
make.blocks(c("age", "sex", "edu"))
```

---

make.blots *Creates a* `blots` *argument*

Description

This helper function creates a valid `blots` object. The `blots` object is an argument to the `mice` function. The name `blots` is a contraction of blocks-dots. Through `blots`, the user can specify any additional arguments that are specifically passed down to the lowest level imputation function.

Usage

```
make.blots(data, blocks = make.blocks(data))
```

Arguments

`data`	A `data.frame` with the source data
`blocks`	An optional specification for blocks of variables in the rows. The default assigns each variable in its own block.

Value

A matrix

See Also

`make.blocks`

Examples

```
make.predictorMatrix(nhanes)
make.blots(nhanes, blocks = name.blocks(c("age", "hyp"), "xxx"))
```

---

make.calltype *Create calltype of the imputation model*

Description

The helper `make.calltype()` creates a vector that identifies per block if the imputation model is taken from `predictorMatrix` or `formulas`. The function is used internally by `mice()`.

Usage

```
make.calltype(calltype, predictorMatrix, formulas, priority = "pred")
```

Arguments

`calltype`	A character vector of length equal to the number of blocks in `predictorMatrix`. Each element specifies how the imputation model for the corresponding block is defined. Valid values are `"pred"` and `"formula"`. If `NULL`, the calltype will be `"pred"` for all blocks, unless `priority` is `"formula"`.
`predictorMatrix`	A matrix specifying the predictors for each block. Each row corresponds to a block, and each column corresponds to a variable. Non-zero entries indicate that the variable is used as a predictor for the block.
`formulas`	A list of formulas, where each element corresponds to a block in `predictorMatrix`. If a formula is provided for a block, the corresponding `calltype` entry is set to `"formula"`. If `NULL`, formulas are not used to modify `calltype`.
`priority`	A character string specifying the default value for `calltype` when it is `NULL`. Defaults to `"pred"`. If `priority == "formula"`, the calltype will be `"formula"` for blocks found in `formulas` with a matching name.

Value

A character vector of length equal to the number of rows in `predictorMatrix`. Each element is either `"pred"` or `"formula"`, indicating how the imputation model is specified for the corresponding block.

Examples

```
# Example predictorMatrix
predictorMatrix <- matrix(1, nrow = 3, ncol = 3,
 dimnames = list(c("block1", "block2", "block3"), c("x1", "x2", "y")))
predictorMatrix[1, 3] <- 0

# Case 1: No calltype or formulas specified
make.calltype(NULL, predictorMatrix, NULL)

# Case 2: Formulas provided
formulas <- list(
  NULL,
  y ~ x1 + x2,
  NULL
)
make.calltype(NULL, predictorMatrix, formulas)

# Case 3: Custom calltype
calltype <- c("pred", "formula", "pred")
make.calltype(calltype, predictorMatrix, NULL)
```

`make.formulas` *Creates a* `formulas` *argument*

Description

This helper function creates a valid `formulas` object. The `formulas` object is an argument to the `mice` function. It is a list of formula's that specifies the target variables and the predictors by means of the standard ~ operator.

Usage

```
make.formulas(data, blocks = make.blocks(data), predictorMatrix = NULL)
```

Arguments

`data`	A `data.frame` with the source data
`blocks`	An optional specification for blocks of variables in the rows. The default assigns each variable in its own block.
`predictorMatrix`	A `predictorMatrix` specified by the user.

Value

A list of formula's.

See Also

`make.blocks`, `make.predictorMatrix`

Examples

```
f1 <- make.formulas(nhanes)
f1
f2 <- make.formulas(nhanes, blocks = make.blocks(nhanes, "collect"))
f2

# for editing, it may be easier to work with the character vector
c1 <- as.character(f1)
c1

# fold it back into a formula list
f3 <- name.formulas(lapply(c1, as.formula))
f3
```

`make.method` *Creates a* `method` *argument*

Description

This helper function creates a valid `method` vector. The `method` vector is an argument to the `mice` function that specifies the method for each block.

Usage

```
make.method(
  data,
  where = make.where(data),
  blocks = make.blocks(data),
  defaultMethod = c("pmm", "logreg", "polyreg", "polr")
)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as `NA`.
`where`	A data frame or matrix with logicals of the same dimensions as `data` indicating where in the data the imputations should be created. The default, `where = is.na(data)`, specifies that the missing data should be imputed. The `where` argument may be used to overimpute observed data, or to skip imputations for selected missing values. Note: Imputation methods that generate imptutations outside of `mice`, like `mice.impute.panImpute()` may depend on a complete predictor space. In that case, a custom `where` matrix can not be specified.
`blocks`	List of vectors with variable names per block. List elements may be named to identify blocks. Variables within a block are imputed by a multivariate imputation method (see `method` argument). By default each variable is placed into its own block, which is effectively fully conditional specification (FCS) by univariate models (variable-by-variable imputation). Only variables whose names appear in `blocks` are imputed. The relevant columns in the `where` matrix are set to `FALSE` of variables that are not block members. A variable may appear in multiple blocks. In that case, it is effectively re-imputed each time that it is visited.
`defaultMethod`	A vector of length 4 containing the default imputation methods for 1) numeric data, 2) factor data with 2 levels, 3) factor data with > 2 unordered levels, and 4) factor data with > 2 ordered levels. By default, the method uses `pmm`, predictive mean matching (numeric data) `logreg`, logistic regression imputation (binary data, factor with 2 levels) `polyreg`, polytomous regression imputation for unordered categorical data (factor > 2 levels) `polr`, proportional odds model for (ordered, > 2 levels).

Value

Vector of `length(blocks)` element with method names

See Also

`mice`

Examples

```
make.method(nhanes2)
```

`make.post` *Creates a* `post` *argument*

Description

This helper function creates a valid `post` vector. The `post` vector is an argument to the `mice` function that specifies post-processing for a variable after each iteration of imputation.

Usage

```
make.post(data)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as `NA`.

Value

Character vector of `ncol(data)` element

See Also

`mice`

Examples

```
make.post(nhanes2)
```

`make.predictorMatrix` *Creates a* `predictorMatrix` *argument*

Description

This helper function creates a valid `predictMatrix`. The `predictorMatrix` is an argument to the `mice` function. It specifies the target variable or block in the rows, and the predictor variables on the columns. An entry of `0` means that the column variable is NOT used to impute the row variable or block. A nonzero value indicates that it is used.

Usage

```
make.predictorMatrix(data, blocks = make.blocks(data), predictorMatrix = NULL)
```

Arguments

`data`	A `data.frame` with the source data
`blocks`	An optional specification for blocks of variables in the rows. The default assigns each variable in its own block.
`predictorMatrix`	A predictor matrix from which rows with the same names are copied into the output predictor matrix.

Value

A matrix

See Also

`make.blocks`

Examples

```
make.predictorMatrix(nhanes)
make.predictorMatrix(nhanes, blocks = make.blocks(nhanes, "collect"))
```

`make.visitSequence` *Creates a* `visitSequence` *argument*

Description

This helper function creates a valid `visitSequence`. The `visitSequence` is an argument to the `mice` function that specifies the sequence in which blocks are imputed.

Usage

```
make.visitSequence(data = NULL, blocks = NULL)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as `NA`.
`blocks`	List of vectors with variable names per block. List elements may be named to identify blocks. Variables within a block are imputed by a multivariate imputation method (see `method` argument). By default each variable is placed into its own block, which is effectively fully conditional specification (FCS) by univariate models (variable-by-variable imputation). Only variables whose names appear in `blocks` are imputed. The relevant columns in the `where` matrix are set to `FALSE` of variables that are not block members. A variable may appear in multiple blocks. In that case, it is effectively re-imputed each time that it is visited.

Value

Vector containing block names

See Also

`mice`

Examples

```
make.visitSequence(nhanes)
```

`make.where` *Creates a* `where` *argument*

Description

This helper function creates a valid `where` matrix. The `where` matrix is an argument to the `mice` function. It has the same size as `data` and specifies which values are to be imputed (`TRUE`) or nor (`FALSE`).

Usage

```
make.where(data, keyword = c("missing", "all", "none", "observed"))
```

Arguments

`data`	A `data.frame` with the source data
`keyword`	An optional keyword, one of `"missing"` (missing values are imputed), `"observed"` (observed values are imputed), `"all"` and `"none"`. The default is `keyword = "missing"`

Value

A matrix with logical

See Also

`make.blocks`, `make.predictorMatrix`

Examples

```
head(make.where(nhanes), 3)

# create & analyse synthetic data
where <- make.where(nhanes2, "all")
imp <- mice(nhanes2,
  m = 10, where = where,
  print = FALSE, seed = 123
)
fit <- with(imp, lm(chl ~ bmi + age + hyp))
summary(pool.syn(fit))
```

`mammalsleep` *Mammal sleep data*

Description

Dataset from Allison and Cicchetti (1976) of 62 mammal species on the interrelationship between sleep, ecological, and constitutional variables. The dataset contains missing values on five variables.

Format

`mammalsleep` is a data frame with 62 rows and 11 columns:

species Species of animal

bw Body weight (kg)

brw Brain weight (g)

sws Slow wave ("nondreaming") sleep (hrs/day)

ps Paradoxical ("dreaming") sleep (hrs/day)

ts Total sleep (hrs/day) (sum of slow wave and paradoxical sleep)

mls Maximum life span (years)

gt Gestation time (days)

pi Predation index (1-5), 1 = least likely to be preyed upon

sei Sleep exposure index (1-5), 1 = least exposed (e.g. animal sleeps in a well-protected den), 5 = most exposed

odi Overall danger index (1-5) based on the above two indices and other information, 1 = least danger (from other animals), 5 = most danger (from other animals)

Details

Allison and Cicchetti (1976) investigated the interrelationship between sleep, ecological, and constitutional variables. They assessed these variables for 39 mammalian species. The authors concluded that slow-wave sleep is negatively associated with a factor related to body size. This suggests that large amounts of this sleep phase are disadvantageous in large species. Also, paradoxical sleep (REM sleep) was associated with a factor related to predatory danger, suggesting that large amounts of this sleep phase are disadvantageous in prey species.

Source

Allison, T., Cicchetti, D.V. (1976). Sleep in Mammals: Ecological and Constitutional Correlates. Science, 194(4266), 732-734.

Examples

```
sleep <- data(mammalsleep)
```

`matchindex` *Find index of matched donor units*

Description

Find index of matched donor units

Usage

```
matchindex(d, t, k = 5L)
```

Arguments

`d`	Numeric vector with values from donor cases.
`t`	Numeric vector with values from target cases.
`k`	Integer, number of unique donors from which a random draw is made. For `k = 1` the function returns the index in `d` corresponding to the closest unit. For multiple imputation, the advice is to set values in the range of `k = 5` to `k = 10`.

Details

For each element in `t`, the method finds the `k` nearest neighbours in `d`, randomly draws one of these neighbours, and returns its position in vector `d`.

Fast predictive mean matching algorithm in seven steps:

1. Shuffle records to remove effects of ties
2. Obtain sorting order on shuffled data
3. Calculate index on input data and sort it
4. Pre-sample vector `h` with values between 1 and `k`

For each of the `n0` elements in `t`:

1. find the two adjacent neighbours
2. find the `h_i`'th nearest neighbour
3. store the index of that neighbour

Return vector of `n0` positions in `d`.

We may use the function to perform predictive mean matching under a given predictive model. To do so, specify both `d` and `t` as predictions from the same model. Suppose that `y` contains the observed outcomes of the donor cases (in the same sequence as `d`), then `y[matchindex(d, t)]` returns one matched outcome for every target case.

See `https://github.com/amices/mice/issues/236`. This function is a replacement for the `matcher()` function that has been in default in `mice` since version `2.22` (June 2014).

Value

An integer vector with `length(t)` elements. Each element is an index in the array `d`.

Author(s)

Stef van Buuren, Nasinski Maciej, Alexander Robitzsch

Examples

```
set.seed(1)

# Inputs need not be sorted
d <- c(-5, 5, 0, 10, 12)
t <- c(-6, -4, 0, 2, 4, -2, 6)

# Index (in vector a) of closest match
idx <- matchindex(d, t, 1)
idx

# To check: show values of closest match

# Random draw among indices of the 5 closest predictors
matchindex(d, t)

# An example
train <- mtcars[1:20, ]
test <- mtcars[21:32, ]
fit <- lm(mpg ~ disp + cyl, data = train)
d <- fitted.values(fit)
t <- predict(fit, newdata = test)  # note: not using mpg
idx <- matchindex(d, t)

# Borrow values from train to produce 12 synthetic values for mpg in test.
# Synthetic values are plausible values that could have been observed if
# they had been measured.
train$mpg[idx]
```

```
# Exercise: Create a distribution of 1000 plausible values for each of the
# twelve mpg entries in test, and count how many times the true value
# (which we know here) is located within the inter-quartile range of each
# distribution. Is your count anywhere close to 500? Why? Why not?
```

`md.pairs` *Missing data pattern by variable pairs*

Description

Number of observations per variable pair.

Usage

```
md.pairs(data)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as `NA`.

Details

The four components in the output value is have the following interpretation:

list('rr') response-response, both variables are observed

list('rm') response-missing, row observed, column missing

list('mr') missing -response, row missing, column observed

list('mm') missing -missing, both variables are missing

Value

A list of four components named `rr`, `rm`, `mr` and `mm`. Each component is square numerical matrix containing the number observations within four missing data pattern.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2009

References

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

Examples

```
pat <- md.pairs(nhanes)
pat

# show that these four matrices decompose the total sample size
# for each pair
pat$rr + pat$rm + pat$mr + pat$mm

# percentage of usable cases to impute row variable from column variable
round(100 * pat$mr / (pat$mr + pat$mm))
```

`md.pattern` *Missing data pattern*

Description

Display missing-data patterns.

Usage

```
md.pattern(x, plot = TRUE, rotate.names = FALSE)
```

Arguments

`x`	A data frame or a matrix containing the incomplete data. Missing values are coded as NA's.
`plot`	Should the missing data pattern be made into a plot. Default is `plot = TRUE`.
`rotate.names`	Whether the variable names in the plot should be placed horizontally or vertically. Default is `rotate.names = FALSE`.

Details

This function is useful for investigating any structure of missing observations in the data. In specific case, the missing data pattern could be (nearly) monotone. Monotonicity can be used to simplify the imputation model. See Schafer (1997) for details. Also, the missing pattern could suggest which variables could potentially be useful for imputation of missing entries.

Value

A matrix with `ncol(x)+1` columns, in which each row corresponds to a missing data pattern (1=observed, 0=missing). Rows and columns are sorted in increasing amounts of missing information. The last column and row contain row and column counts, respectively.

Author(s)

Gerko Vink, 2018, based on an earlier version of the same function by Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

Schafer, J.L. (1997), Analysis of multivariate incomplete data. London: Chapman&Hall.

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

Examples

```
md.pattern(nhanes)
#     age hyp bmi chl
#  13   1   1   1   1  0
#   1   1   1   0   1  1
#   3   1   1   1   0  1
#   1   1   0   0   1  2
#   7   1   0   0   0  3
#   0   8   9  10 27
```

`mdc` *Graphical parameter for missing data plots*

Description

`mdc` returns colors used to distinguish observed, missing and combined data in plotting. `mice.theme` return a partial list of named objects that can be used as a theme in `stripplot`, `bwplot`, `densityplot` and `xyplot`.

Usage

```
mdc(
  r = "observed",
  s = "symbol",
  transparent = TRUE,
  cso = grDevices::hcl(240, 100, 40, 0.7),
  csi = grDevices::hcl(0, 100, 40, 0.7),
  csc = "gray50",
  clo = grDevices::hcl(240, 100, 40, 0.8),
  cli = grDevices::hcl(0, 100, 40, 0.8),
  clc = "gray50"
)
```

Arguments

`r`	A numerical or character vector. The numbers 1-6 request colors as follows: 1=`cso`, 2=`csi`, 3=`csc`, 4=`clo`, 5=`cli` and 6=`clc`. Alternatively, `r` may contain the strings ' `observed`', '`missing`', or '`both`', or abbreviations thereof.
`s`	A character vector containing the strings '`symbol`' or ' `line`', or abbreviations thereof.
`transparent`	A logical indicating whether alpha-transparency is allowed. The default is `TRUE`.

`cso`	The symbol color for the observed data. The default is a transparent blue.
`csi`	The symbol color for the missing or imputed data. The default is a transparent red.
`csc`	The symbol color for the combined observed and imputed data. The default is a grey color.
`clo`	The line color for the observed data. The default is a slightly darker transparent blue.
`cli`	The line color for the missing or imputed data. The default is a slightly darker transparent red.
`clc`	The line color for the combined observed and imputed data. The default is a grey color.

Details

This function eases consistent use of colors in plots. The default follows the Abayomi convention, which uses blue for observed data, red for missing or imputed data, and black for combined data.

Value

`mdc()` returns a vector containing color definitions. The length of the output vector is calculate from the length of `r` and `s`. Elements of the input vectors are repeated if needed.

Author(s)

Stef van Buuren, sept 2012.

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer.

See Also

`hcl`, `rgb`, `xyplot`, `trellis.par.set`

Examples

```
# all six colors
mdc(1:6)

# lines color for observed and missing data
mdc(c("obs", "mis"), "lin")
```

`mice.impute.2l.bin` *Imputation by a two-level logistic model using* `glmer`

Description

Imputes univariate systematically and sporadically missing data using a two-level logistic model using `lme4::glmer()`

Usage

```
mice.impute.2l.bin(y, ry, x, type, wy = NULL, intercept = TRUE, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`type`	Vector of length `ncol(x)` identifying random and class variables. Random variables are identified by a '2'. The class variable (only one is allowed) is coded as '-2'. Fixed effects are indicated by a '1'.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`intercept`	Logical determining whether the intercept is automatically added.
`...`	Arguments passed down to `glmer`

Details

Data are missing systematically if they have not been measured, e.g., in the case where we combine data from different sources. Data are missing sporadically if they have been partially observed.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Shahab Jolani, 2015; adapted to mice, SvB, 2018

References

Jolani S., Debray T.P.A., Koffijberg H., van Buuren S., Moons K.G.M. (2015). Imputation of systematically missing predictors in an individual participant data meta-analysis: a generalized approach using MICE. *Statistics in Medicine*, 34:1841-1863.

See Also

Other univariate-2l: `mice.impute.2l.lmer()`, `mice.impute.2l.norm()`, `mice.impute.2l.pan()`

Examples

```
library(tidyr)
library(dplyr)
data("toenail2")
data <- tidyr::complete(toenail2, patientID, visit) %>%
  tidyr::fill(treatment) %>%
  dplyr::select(-time) %>%
  dplyr::mutate(patientID = as.integer(patientID))
## Not run:
pred <- mice(data, print = FALSE, maxit = 0, seed = 1)$pred
pred["outcome", "patientID"] <- -2
imp <- mice(data, method = "2l.bin", pred = pred, maxit = 1, m = 1, seed = 1)

## End(Not run)
```

`mice.impute.2l.lmer` *Imputation by a two-level normal model using* `lmer`

Description

Imputes univariate systematically and sporadically missing data using a two-level normal model using `lme4::lmer()`.

Usage

```
mice.impute.2l.lmer(y, ry, x, type, wy = NULL, intercept = TRUE, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`type`	Vector of length `ncol(x)` identifying random and class variables. Random variables are identified by a '2'. The class variable (only one is allowed) is coded as '-2'. Fixed effects are indicated by a '1'.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`intercept`	Logical determining whether the intercept is automatically added.
`...`	Arguments passed down to `lmer`

Details

Data are missing systematically if they have not been measured, e.g., in the case where we combine data from different sources. Data are missing sporadically if they have been partially observed.

While the method is fully Bayesian, it may fix parameters of the variance-covariance matrix or the random effects to their estimated value in cases where creating draws from the posterior is not possible. The procedure throws a warning when this happens.

If `lme4::lmer()` fails, the procedure prints the warning `"lmer does not run. Simplify imputation model"` and returns the current imputation. If that happens we see flat lines in the trace line plots. Thus, the appearance of flat trace lines should be taken as an additional alert to a problem with imputation model fitting.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Shahab Jolani, 2017

References

Jolani S. (2017) Hierarchical imputation of systematically and sporadically missing data: An approximate Bayesian approach using chained equations. Forthcoming.

Jolani S., Debray T.P.A., Koffijberg H., van Buuren S., Moons K.G.M. (2015). Imputation of systematically missing predictors in an individual participant data meta-analysis: a generalized approach using MICE. *Statistics in Medicine*, 34:1841-1863.

Van Buuren, S. (2011) Multiple imputation of multilevel data. In Hox, J.J. and and Roberts, J.K. (Eds.), *The Handbook of Advanced Multilevel Analysis*, Chapter 10, pp. 173–196. Milton Park, UK: Routledge.

See Also

Other univariate-2l: `mice.impute.2l.bin()`, `mice.impute.2l.norm()`, `mice.impute.2l.pan()`

`mice.impute.2l.norm` *Imputation by a two-level normal model*

Description

Imputes univariate missing data using a two-level normal model

Usage

```
mice.impute.2l.norm(y, ry, x, type, wy = NULL, intercept = TRUE, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`type`	Vector of length `ncol(x)` identifying random and class variables. Random variables are identified by a '2'. The class variable (only one is allowed) is coded as '-2'. Random variables also include the fixed effect.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`intercept`	Logical determining whether the intercept is automatically added.
`...`	Other named arguments.

Details

Implements the Gibbs sampler for the linear multilevel model with heterogeneous with-class variance (Kasim and Raudenbush, 1998). Imputations are drawn as an extra step to the algorithm. For simulation work see Van Buuren (2011).

The random intercept is automatically added in `mice.impute.2L.norm()`. A model within a random intercept can be specified by `mice(...,intercept = FALSE)`.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Note

Added June 25, 2012: The currently implemented algorithm does not handle predictors that are specified as fixed effects (type=1). When using `mice.impute.2l.norm()`, the current advice is to specify all predictors as random effects (type=2).

Warning: The assumption of heterogeneous variances requires that in every class at least one observation has a response in `y`.

Author(s)

Roel de Jong, 2008

References

Kasim RM, Raudenbush SW. (1998). Application of Gibbs sampling to nested variance components models with heterogeneous within-group variance. Journal of Educational and Behavioral Statistics, 23(2), 93–116.

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

Van Buuren, S. (2011) Multiple imputation of multilevel data. In Hox, J.J. and and Roberts, J.K. (Eds.), *The Handbook of Advanced Multilevel Analysis*, Chapter 10, pp. 173–196. Milton Park, UK: Routledge.

See Also

Other univariate-2l: `mice.impute.2l.bin()`, `mice.impute.2l.lmer()`, `mice.impute.2l.pan()`

`mice.impute.2l.pan` *Imputation by a two-level normal model using* `pan`

Description

Imputes univariate missing data using a two-level normal model with homogeneous within group variances. Aggregated group effects (i.e. group means) can be automatically created and included as predictors in the two-level regression (see argument `type`). This function needs the `pan` package.

Usage

```
mice.impute.2l.pan(
  y,
  ry,
  x,
  type,
  intercept = TRUE,
  paniter = 500,
  groupcenter.slope = FALSE,
  ...
)
```

Arguments

`y`	Incomplete data vector of length `n`
`ry`	Vector of missing data pattern (`FALSE`=missing, `TRUE`=observed)
`x`	Matrix (`n` x `p`) of complete covariates.
`type`	Vector of length `ncol(x)` identifying random and class variables. Random effects are identified by a '2'. The group variable (only one is allowed) is coded as '-2'. Random effects also include the fixed effect. If for a covariates X1 group means shall be calculated and included as further fixed effects choose '3'. In addition to the effects in '3', specification '4' also includes random effects of X1.
`intercept`	Logical determining whether the intercept is automatically added.
`paniter`	Number of iterations in `pan`. Default is 500.
`groupcenter.slope`	If `TRUE`, in case of group means (`type` is '3' or'4') group mean centering for these predictors are conducted before doing imputations. Default is `FALSE`.
`...`	Other named arguments.

Details

Implements the Gibbs sampler for the linear two-level model with homogeneous within group variances which is a special case of a multivariate linear mixed effects model (Schafer & Yucel, 2002). For a two-level imputation with heterogeneous within-group variances see `mice.impute.2l.norm`. % The random intercept is automatically added in % `mice.impute.2l.norm()`.

Value

A vector of length `nmis` with imputations.

Note

This function does not implement the `where` functionality. It always produces `nmis` imputation, irrespective of the `where` argument of the `mice` function.

Author(s)

Alexander Robitzsch (IPN - Leibniz Institute for Science and Mathematics Education, Kiel, Germany), `<robitzsch@ipn.uni-kiel.de>`

Alexander Robitzsch (IPN - Leibniz Institute for Science and Mathematics Education, Kiel, Germany), `<robitzsch@ipn.uni-kiel.de>`.

References

Schafer J L, Yucel RM (2002). Computational strategies for multivariate linear mixed-effects models with missing values. *Journal of Computational and Graphical Statistics*. **11**, 437-457.

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

See Also

Other univariate-2l: `mice.impute.2l.bin()`, `mice.impute.2l.lmer()`, `mice.impute.2l.norm()`

Examples

```
# simulate some data
# two-level regression model with fixed slope

# number of groups
G <- 250
# number of persons
n <- 20
# regression parameter
beta <- .3
# intraclass correlation
rho <- .30
# correlation with missing response
rho.miss <- .10
# missing proportion
missrate <- .50
```

```
y1 <- rep(rnorm(G, sd = sqrt(rho)), each = n) + rnorm(G * n, sd = sqrt(1 - rho))
x <- rnorm(G * n)
y <- y1 + beta * x
dfr0 <- dfr <- data.frame("group" = rep(1:G, each = n), "x" = x, "y" = y)
dfr[rho.miss * x + rnorm(G * n, sd = sqrt(1 - rho.miss)) < qnorm(missrate), "y"] <- NA

# empty imputation in mice
imp0 <- mice(as.matrix(dfr), maxit = 0)
predM <- imp0$predictorMatrix
impM <- imp0$method

# specify predictor matrix and method
predM1 <- predM
predM1["y", "group"] <- -2
predM1["y", "x"] <- 1 # fixed x effects imputation
impM1 <- impM
impM1["y"] <- "2l.pan"

# multilevel imputation
imp1 <- mice(as.matrix(dfr),
  m = 1, predictorMatrix = predM1,
  method = impM1, maxit = 1
)

# multilevel analysis
library(lme4)
mod <- lmer(y ~ (1 + x | group) + x, data = complete(imp1))
summary(mod)

# Examples of predictorMatrix specification

# random x effects
# predM1["y","x"] <- 2

# fixed x effects and group mean of x
# predM1["y","x"] <- 3

# random x effects and group mean of x
# predM1["y","x"] <- 4
```

`mice.impute.2lonly.mean` *Imputation of most likely value within the class*

Description

Method `2lonly.mean` replicates the most likely value within a class of a second-level variable. It works for numeric and factor data. The function is primarily useful as a quick fixup for data in which the second-level variable is inconsistent.

Usage

```
mice.impute.2lonly.mean(y, ry, x, type, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`type`	Vector of length `ncol(x)` identifying random and class variables. The class variable (only one is allowed) is coded as `-2`.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

Observed values in `y` are averaged within the class, and replicated to the missing `y` within that class. This function is primarily useful for repairing incomplete data that are constant within the class, but vary over classes.

For numeric variables, `mice.impute.2lonly.mean()` imputes the class mean of `y`. If `y` is a second-level variable, then conventionally all observed `y` will be identical within the class, and the function just provides a quick fix for any missing `y` by filling in the class mean.

For factor variables, `mice.impute.2lonly.mean()` imputes the most frequently occuring category within the class.

If there are no observed `y` in the class, all entries of the class are set to `NA`. Note that this may produce problems later on in `mice` if imputation routines are called that expects predictor data to be complete. Methods designed for imputing this type of second-level variables include `mice.impute.2lonly.norm` and `mice.impute.2lonly.pmm`.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Gerko Vink, Stef van Buuren, 2019

References

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Boca Raton, FL.: Chapman & Hall/CRC Press.

See Also

Other univariate-2lonly: `mice.impute.2lonly.norm()`, `mice.impute.2lonly.pmm()`

`mice.impute.2lonly.norm`

Imputation at level 2 by Bayesian linear regression

Description

Imputes univariate missing data at level 2 using Bayesian linear regression analysis. Variables are level 1 are aggregated at level 2. The group identifier at level 2 must be indicated by `type = -2` in the `predictorMatrix`.

Usage

```
mice.impute.2lonly.norm(y, ry, x, type, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`type`	Group identifier must be specified by '-2'. Predictors must be specified by '1'.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

This function allows in combination with `mice.impute.2l.pan` switching regression imputation between level 1 and level 2 as described in Yucel (2008) or Gelman and Hill (2007, p. 541).

The function checks for partial missing level-2 data. Level-2 data are assumed to be constant within the same cluster. If one or more entries are missing, then the procedure aborts with an error message that identifies the cluster with incomplete level-2 data. In such cases, one may first fill in the cluster mean (or mode) by the `2lonly.mean` method to remove inconsistencies.

Value

A vector of length `nmis` with imputations.

Note

For a more general approach, see `miceadds::mice.impute.2lonly.function()`.

Author(s)

Alexander Robitzsch (IPN - Leibniz Institute for Science and Mathematics Education, Kiel, Germany), `<robitzsch@ipn.uni-kiel.de>`

References

Gelman, A. and Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge, Cambridge University Press.

Yucel, RM (2008). Multiple imputation inference for multivariate multilevel continuous data with ignorable non-response. *Philosophical Transactions of the Royal Society A*, **366**, 2389-2404.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`mice.impute.norm`, `mice.impute.2lonly.pmm`, `mice.impute.2l.pan`, `mice.impute.2lonly.mean`

Other univariate-2lonly: `mice.impute.2lonly.mean()`, `mice.impute.2lonly.pmm()`

Examples

```
# simulate some data
# x,y ... level 1 variables
# v,w ... level 2 variables

G <- 250 # number of groups
n <- 20 # number of persons
beta <- .3 # regression coefficient
rho <- .30 # residual intraclass correlation
rho.miss <- .10 # correlation with missing response
missrate <- .50 # missing proportion
y1 <- rep(rnorm(G, sd = sqrt(rho)), each = n) + rnorm(G * n, sd = sqrt(1 - rho))
w <- rep(round(rnorm(G), 2), each = n)
v <- rep(round(runif(G, 0, 3)), each = n)
x <- rnorm(G * n)
y <- y1 + beta * x + .2 * w + .1 * v
dfr0 <- dfr <- data.frame("group" = rep(1:G, each = n), "x" = x, "y" = y, "w" = w, "v" = v)
dfr[rho.miss * x + rnorm(G * n, sd = sqrt(1 - rho.miss)) < qnorm(missrate), "y"] <- NA
dfr[rep(rnorm(G), each = n) < qnorm(missrate), "w"] <- NA
dfr[rep(rnorm(G), each = n) < qnorm(missrate), "v"] <- NA

# empty mice imputation
imp0 <- mice(as.matrix(dfr), maxit = 0)
predM <- imp0$predictorMatrix
impM <- imp0$method

# multilevel imputation
predM1 <- predM
predM1[c("w", "y", "v"), "group"] <- -2
predM1["y", "x"] <- 1 # fixed x effects imputation
impM1 <- impM
```

```
impM1[c("y", "w", "v")] <- c("2l.pan", "2lonly.norm", "2lonly.pmm")

# y ... imputation using pan
# w ... imputation at level 2 using norm
# v ... imputation at level 2 using pmm

imp1 <- mice(as.matrix(dfr),
  m = 1, predictorMatrix = predM1,
  method = impM1, maxit = 1, paniter = 500
)

# Demonstration that 2lonly.norm aborts for partial missing data.
# Better use 2lonly.mean for repair.
data <- data.frame(
  patid = rep(1:4, each = 5),
  sex = rep(c(1, 2, 1, 2), each = 5),
  crp = c(
    68, 78, 93, NA, 143,
    5, 7, 9, 13, NA,
    97, NA, 56, 52, 34,
    22, 30, NA, NA, 45
  )
)
pred <- make.predictorMatrix(data)
pred[, "patid"] <- -2
# only missing value (out of five) for patid == 1
data[3, "sex"] <- NA
## Not run:
# The following fails because 2lonly.norm found partially missing
# level-2 data
# imp <- mice(data, method = c("", "2lonly.norm", "2l.pan"),
#             predictorMatrix = pred, maxit = 1, m = 2)
# > iter imp variable
# > 1   1  sex  crpError in .imputation.level2(y = y, ... :
# >   Method 2lonly.norm found the following clusters with partially missing
# >    level-2 data: 1
# > Method 2lonly.mean can fix such inconsistencies.

## End(Not run)

# In contrast, if all sex values are missing for patid == 1, it runs fine,
# except on r-patched-solaris-x86. I used dontrun to evade CRAN errors.
## Not run:
data[1:5, "sex"] <- NA
imp <- mice(data,
  method = c("", "2lonly.norm", "2l.pan"),
  predictorMatrix = pred, maxit = 1, m = 2
)

## End(Not run)
```

`mice.impute.2lonly.pmm`

Imputation at level 2 by predictive mean matching

Description

Imputes univariate missing data at level 2 using predictive mean matching. Variables are level 1 are aggregated at level 2. The group identifier at level 2 must be indicated by `type = -2` in the `predictorMatrix`.

Usage

```
mice.impute.2lonly.pmm(y, ry, x, type, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`type`	Group identifier must be specified by '-2'. Predictors must be specified by '1'.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

This function allows in combination with `mice.impute.2l.pan` switching regression imputation between level 1 and level 2 as described in Yucel (2008) or Gelman and Hill (2007, p. 541).

The function checks for partial missing level-2 data. Level-2 data are assumed to be constant within the same cluster. If one or more entries are missing, then the procedure aborts with an error message that identifies the cluster with incomplete level-2 data. In such cases, one may first fill in the cluster mean (or mode) by the `2lonly.mean` method to remove inconsistencies.

Value

A vector of length `nmis` with imputations.

Note

The extension to categorical variables transforms a dependent factor variable by means of the `as.integer()` function. This may make sense for categories that are approximately ordered, but less so for pure nominal measures.

For a more general approach, see `miceadds::mice.impute.2lonly.function()`.

Author(s)

Alexander Robitzsch (IPN - Leibniz Institute for Science and Mathematics Education, Kiel, Germany), `<robitzsch@ipn.uni-kiel.de>`

References

Gelman, A. and Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge, Cambridge University Press.

Yucel, RM (2008). Multiple imputation inference for multivariate multilevel continuous data with ignorable non-response. *Philosophical Transactions of the Royal Society A*, **366**, 2389-2404.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`mice.impute.pmm`, `mice.impute.2lonly.norm`, `mice.impute.2l.pan`, `mice.impute.2lonly.mean`

Other univariate-2lonly: `mice.impute.2lonly.mean()`, `mice.impute.2lonly.norm()`

Examples

```
# simulate some data
# x,y ... level 1 variables
# v,w ... level 2 variables

G <- 250 # number of groups
n <- 20 # number of persons
beta <- .3 # regression coefficient
rho <- .30 # residual intraclass correlation
rho.miss <- .10 # correlation with missing response
missrate <- .50 # missing proportion
y1 <- rep(rnorm(G, sd = sqrt(rho)), each = n) + rnorm(G * n, sd = sqrt(1 - rho))
w <- rep(round(rnorm(G), 2), each = n)
v <- rep(round(runif(G, 0, 3)), each = n)
x <- rnorm(G * n)
y <- y1 + beta * x + .2 * w + .1 * v
dfr0 <- dfr <- data.frame("group" = rep(1:G, each = n), "x" = x, "y" = y, "w" = w, "v" = v)
dfr[rho.miss * x + rnorm(G * n, sd = sqrt(1 - rho.miss)) < qnorm(missrate), "y"] <- NA
dfr[rep(rnorm(G), each = n) < qnorm(missrate), "w"] <- NA
dfr[rep(rnorm(G), each = n) < qnorm(missrate), "v"] <- NA

# empty mice imputation
imp0 <- mice(as.matrix(dfr), maxit = 0)
predM <- imp0$predictorMatrix
impM <- imp0$method

# multilevel imputation
predM1 <- predM
predM1[c("w", "y", "v"), "group"] <- -2
predM1["y", "x"] <- 1 # fixed x effects imputation
impM1 <- impM
```

```
impM1[c("y", "w", "v")] <- c("2l.pan", "2lonly.norm", "2lonly.pmm")

# turn v into a categorical variable
dfr$v <- as.factor(dfr$v)
levels(dfr$v) <- LETTERS[1:4]

# y ... imputation using pan
# w ... imputation at level 2 using norm
# v ... imputation at level 2 using pmm

# skip imputation on solaris
is.solaris <- function() grepl("SunOS", Sys.info()["sysname"])
if (!is.solaris()) {
  imp <- mice(dfr,
    m = 1, predictorMatrix = predM1,
    method = impM1, maxit = 1, paniter = 500
  )
}
```

`mice.impute.cart` *Imputation by classification and regression trees*

Description

Imputes univariate missing data using classification and regression trees.

Usage

```
mice.impute.cart(y, ry, x, wy = NULL, minbucket = 5, cp = 1e-04, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`minbucket`	The minimum number of observations in any terminal node used. See `rpart.control` for details.
`cp`	Complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. See `rpart.control` for details.
`...`	Other named arguments passed down to `rpart()`.

Details

Imputation of `y` by classification and regression trees. The procedure is as follows:

1. Fit a classification or regression tree by recursive partitioning;
2. For each `ymis`, find the terminal node they end up according to the fitted tree;
3. Make a random draw among the member in the node, and take the observed value from that draw as the imputation.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Numeric vector of length `sum(!ry)` with imputations

Author(s)

Lisa Doove, Stef van Buuren, Elise Dusseldorp, 2012

References

Doove, L.L., van Buuren, S., Dusseldorp, E. (2014), Recursive partitioning for missing data imputation in the presence of interaction Effects. Computational Statistics & Data Analysis, 72, 92-104.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984), Classification and regression trees, Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`mice`, `mice.impute.rf`, `rpart`, `rpart.control`

Other univariate imputation functions: `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

Examples

```
imp <- mice(nhanes2, meth = "cart", minbucket = 4)
plot(imp)
```

`mice.impute.jomoImpute`

Multivariate multilevel imputation using `jomo`

Description

This function is a wrapper around the `jomoImpute` function from the `mitml` package so that it can be called to impute blocks of variables in `mice`. The `mitml::jomoImpute` function provides an interface to the `jomo` package for multiple imputation of multilevel data `https://CRAN.R-project.org/package=jomo`. Imputations can be generated using `type` or `formula`, which offer different options for model specification.

Usage

```
mice.impute.jomoImpute(
  data,
  formula,
  type,
  m = 1,
  silent = TRUE,
  format = "imputes",
  ...
)
```

Arguments

`data`	A data frame containing incomplete and auxiliary variables, the cluster indicator variable, and any other variables that should be present in the imputed datasets.
`formula`	A formula specifying the role of each variable in the imputation model. The basic model is constructed by `model.matrix`, thus allowing to include derived variables in the imputation model using `I()`. See `jomoImpute`.
`type`	An integer vector specifying the role of each variable in the imputation model (see `jomoImpute`)
`m`	The number of imputed data sets to generate. Default is 10.
`silent`	(optional) Logical flag indicating if console output should be suppressed. Default is `FALSE`.
`format`	A character vector specifying the type of object that should be returned. The default is `format = "list"`. No other formats are currently supported.
`...`	Other named arguments: `n.burn`, `n.iter`, `group`, `prior`, `silent` and others.

Value

A list of imputations for all incomplete variables in the model, that can be stored in the the `imp` component of the `mids` object.

Note

The number of imputations `m` is set to 1, and the function is called `m` times so that it fits within the `mice` iteration scheme.

This is a multivariate imputation function using a joint model.

Author(s)

Stef van Buuren, 2018, building on work of Simon Grund, Alexander Robitzsch and Oliver Luedtke (authors of `mitml` package) and Quartagno and Carpenter (authors of `jomo` package).

References

Grund S, Luedtke O, Robitzsch A (2016). Multiple Imputation of Multilevel Missing Data: An Introduction to the R Package pan. SAGE Open.

Quartagno M and Carpenter JR (2015). Multiple imputation for IPD meta-analysis: allowing for heterogeneity and studies with missing covariates. Statistics in Medicine, 35:2938-2954, 2015.

See Also

`jomoImpute`

Other multivariate-2l: `mice.impute.panImpute()`

Examples

```
## Not run:
# Note: Requires mitml 0.3-5.7
blocks <- list(c("bmi", "chl", "hyp"), "age")
method <- c("jomoImpute", "pmm")
ini <- mice(nhanes, blocks = blocks, method = method, maxit = 0)
pred <- ini$pred
pred["B1", "hyp"] <- -2
imp <- mice(nhanes, blocks = blocks, method = method, pred = pred, maxit = 1)

## End(Not run)
```

`mice.impute.lasso.logreg`

Imputation by direct use of lasso logistic regression

Description

Imputes univariate missing binary data using lasso logistic regression with bootstrap.

Usage

```
mice.impute.lasso.logreg(y, ry, x, wy = NULL, nfolds = 10, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`nfolds`	The number of folds for the cross-validation of the lasso penalty. The default is 10.
`...`	Other named arguments.

Details

The method consists of the following steps:

1. For a given `y` variable under imputation, draw a bootstrap version y* with replacement from the observed cases `y[ry]`, and stores in x* the corresponding values from `x[ry, ]`.
2. Fit a regularised (lasso) logistic regression with y* as the outcome, and x* as predictors. A vector of regression coefficients bhat is obtained. All of these coefficients are considered random draws from the imputation model parameters posterior distribution. Same of these coefficients will be shrunken to 0.
3. Compute predicted scores for m.d., i.e. logit-1(X bhat)
4. Compare the score to a random (0,1) deviate, and impute.

The method is based on the Direct Use of Regularized Regression (DURR) proposed by Zhao & Long (2016) and Deng et al (2016).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Edoardo Costantini, 2021

References

Deng, Y., Chang, C., Ido, M. S., & Long, Q. (2016). Multiple imputation for general missing data patterns in the presence of high-dimensional data. Scientific reports, 6(1), 1-10.

Zhao, Y., & Long, Q. (2016). Multiple imputation in the presence of high-dimensional data. Statistical Methods in Medical Research, 25(5), 2021-2035.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.`
`mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.lasso.norm`

Imputation by direct use of lasso linear regression

Description

Imputes univariate missing normal data using lasso linear regression with bootstrap.

Usage

```
mice.impute.lasso.norm(y, ry, x, wy = NULL, nfolds = 10, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`nfolds`	The number of folds for the cross-validation of the lasso penalty. The default is 10.
`...`	Other named arguments.

Details

The method consists of the following steps:

1. For a given y variable under imputation, draw a bootstrap version y* with replacement from the observed cases `y[ry]`, and stores in x* the corresponding values from `x[ry, ]`.
2. Fit a regularised (lasso) linear regression with y* as the outcome, and x* as predictors. A vector of regression coefficients bhat is obtained. All of these coefficients are considered random draws from the imputation model parameters posterior distribution. Same of these coefficients will be shrunken to 0.

3. Draw the imputed values from the predictive distribution defined by the original (non-bootstrap) data, bhat, and estimated error variance.

The method is based on the Direct Use of Regularized Regression (DURR) proposed by Zhao & Long (2016) and Deng et al (2016).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Edoardo Costantini, 2021

References

Deng, Y., Chang, C., Ido, M. S., & Long, Q. (2016). Multiple imputation for general missing data patterns in the presence of high-dimensional data. Scientific reports, 6(1), 1-10.

Zhao, Y., & Long, Q. (2016). Multiple imputation in the presence of high-dimensional data. Statistical Methods in Medical Research, 25(5), 2021-2035.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.lasso.select.logreg`

Imputation by indirect use of lasso logistic regression

Description

Imputes univariate missing data using logistic regression following a preprocessing lasso variable selection step.

Usage

```
mice.impute.lasso.select.logreg(y, ry, x, wy = NULL, nfolds = 10, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`nfolds`	The number of folds for the cross-validation of the lasso penalty. The default is 10.
`...`	Other named arguments.

Details

The method consists of the following steps:

1. For a given `y` variable under imputation, fit a linear regression with lasso penalty using `y[ry]` as dependent variable and `x[ry, ]` as predictors. The coefficients that are not shrunk to 0 define the active set of predictors that will be used for imputation.
2. Fit a logit with the active set of predictors, and find (bhat, V(bhat))
3. Draw BETA from N(bhat, V(bhat))
4. Compute predicted scores for m.d., i.e. logit-1(X BETA)
5. Compare the score to a random (0,1) deviate, and impute.

The user can specify a `predictorMatrix` in the `mice` call to define which predictors are provided to this univariate imputation method. The lasso regularization will select, among the variables indicated by the user, the ones that are important for imputation at any given iteration. Therefore, users may force the exclusion of a predictor from a given imputation model by speficing a `0` entry. However, a non-zero entry does not guarantee the variable will be used, as this decision is ultimately made by the lasso variable selection procedure.

The method is based on the Indirect Use of Regularized Regression (IURR) proposed by Zhao & Long (2016) and Deng et al (2016).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Edoardo Costantini, 2021

References

Deng, Y., Chang, C., Ido, M. S., & Long, Q. (2016). Multiple imputation for general missing data patterns in the presence of high-dimensional data. Scientific reports, 6(1), 1-10.

Zhao, Y., & Long, Q. (2016). Multiple imputation in the presence of high-dimensional data. Statistical Methods in Medical Research, 25(5), 2021-2035.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.lasso.select.norm`
Imputation by indirect use of lasso linear regression

Description

Imputes univariate missing data using Bayesian linear regression following a preprocessing lasso variable selection step.

Usage

```
mice.impute.lasso.select.norm(y, ry, x, wy = NULL, nfolds = 10, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`nfolds`	The number of folds for the cross-validation of the lasso penalty. The default is 10.
`...`	Other named arguments.

Details

The method consists of the following steps:

1. For a given `y` variable under imputation, fit a linear regression with lasso penalty using `y[ry]` as dependent variable and `x[ry, ]` as predictors. Coefficients that are not shrunk to 0 define an active set of predictors that will be used for imputation
2. Define a Bayesian linear model using `y[ry]` as the dependent variable, the active set of `x[ry, ]` as predictors, and standard non-informative priors

3. Draw parameter values for the intercept, regression weights, and error variance from their posterior distribution
4. Draw imputations from the posterior predictive distribution

The user can specify a `predictorMatrix` in the `mice` call to define which predictors are provided to this univariate imputation method. The lasso regularization will select, among the variables indicated by the user, the ones that are important for imputation at any given iteration. Therefore, users may force the exclusion of a predictor from a given imputation model by specifying a `0` entry. However, a non-zero entry does not guarantee the variable will be used, as this decision is ultimately made by the lasso variable selection procedure.

The method is based on the Indirect Use of Regularized Regression (IURR) proposed by Zhao & Long (2016) and Deng et al (2016).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Edoardo Costantini, 2021

References

Deng, Y., Chang, C., Ido, M. S., & Long, Q. (2016). Multiple imputation for general missing data patterns in the presence of high-dimensional data. Scientific reports, 6(1), 1-10.

Zhao, Y., & Long, Q. (2016). Multiple imputation in the presence of high-dimensional data. Statistical Methods in Medical Research, 25(5), 2021-2035.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.lda` *Imputation by linear discriminant analysis*

Description

Imputes univariate missing data using linear discriminant analysis

Usage

```
mice.impute.lda(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments. Not used.

Details

Imputation of categorical response variables by linear discriminant analysis. This function uses the Venables/Ripley functions `lda()` and `predict.lda()` to compute posterior probabilities for each incomplete case, and draws the imputations from this posterior.

This function can be called from within the Gibbs sampler by specifying `"lda"` in the `method` argument of `mice()`. This method is usually faster and uses fewer resources than calling the function, but the statistical properties may not be as good (Brand, 1999). `mice.impute.polyreg`.

Value

Vector with imputed data, of type factor, and of length `sum(wy)`

Warning

The function does not incorporate the variability of the discriminant weight, so it is not 'proper' in the sense of Rubin. For small samples and rare categories in the `y`, variability of the imputed data could therefore be underestimated.

Added: SvB June 2009 Tried to include bootstrap, but disabled since bootstrapping may easily lead to constant variables within groups.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

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

Brand, J.P.L. (1999). Development, Implementation and Evaluation of Multiple Imputation Strategies for the Statistical Analysis of Incomplete Data Sets. Ph.D. Thesis, TNO Prevention and Health/Erasmus University Rotterdam. ISBN 90-74479-08-1.

Venables, W.N. & Ripley, B.D. (1997). Modern applied statistics with S-PLUS (2nd ed). Springer, Berlin.

See Also

`mice`, `mice.impute.polyreg`, `lda`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.logreg` *Imputation by logistic regression*

Description

Imputes univariate missing data using logistic regression.

Usage

```
mice.impute.logreg(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

Imputation for binary response variables by the Bayesian logistic regression model (Rubin 1987, p. 169-170). The Bayesian method consists of the following steps:

1. Fit a logit, and find (bhat, V(bhat))
2. Draw BETA from N(bhat, V(bhat))
3. Compute predicted scores for m.d., i.e. logit-1(X BETA)
4. Compare the score to a random (0,1) deviate, and impute.

The method relies on the standard `glm.fit` function. Warnings from `glm.fit` are suppressed. Perfect prediction is handled by the data augmentation method.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn

References

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

Brand, J.P.L. (1999). Development, Implementation and Evaluation of Multiple Imputation Strategies for the Statistical Analysis of Incomplete Data Sets. Ph.D. Thesis, TNO Prevention and Health/Erasmus University Rotterdam. ISBN 90-74479-08-1.

Venables, W.N. & Ripley, B.D. (1997). Modern applied statistics with S-Plus (2nd ed). Springer, Berlin.

White, I., Daniel, R. and Royston, P (2010). Avoiding bias due to perfect prediction in multiple imputation of incomplete categorical variables. Computational Statistics and Data Analysis, 54:22672275.

See Also

`mice`, `glm`, `glm.fit`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.logreg.boot` *Imputation by logistic regression using the bootstrap*

Description

Imputes univariate missing data using logistic regression by a bootstrapped logistic regression model. The bootstrap method draws a simple bootstrap sample with replacement from the observed data `y[ry]` and `x[ry, ]`.

Usage

```
mice.impute.logreg.boot(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000, 2011

References

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

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`mice`, `glm`, `glm.fit`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.mean` *Imputation by the mean*

Description

Imputes the arithmetic mean of the observed data

Usage

```
mice.impute.mean(y, ry, x = NULL, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Warning

Imputing the mean of a variable is almost never appropriate. See Little and Rubin (2002, p. 61-62) or Van Buuren (2012, p. 10-11)

References

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

Little, R.J.A. and Rubin, D.B. (2002). Statistical Analysis with Missing Data. New York: John Wiley and Sons.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`mice`, `mean`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.midastouch`

Imputation by predictive mean matching with distance aided donor selection

---

Description

Imputes univariate missing data using predictive mean matching.

Usage

```
mice.impute.midastouch(
  y,
  ry,
  x,
  wy = NULL,
  ridge = 1e-05,
  midas.kappa = NULL,
  outout = TRUE,
  neff = NULL,
  debug = NULL,
  ...
)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`ridge`	The ridge penalty used in `.norm.draw()` to prevent problems with multicollinearity. The default is `ridge = 1e-05`, which means that 0.01 percent of the diagonal is added to the cross-product. Larger ridges may result in more biased estimates. For highly noisy data (e.g. many junk variables), set `ridge = 1e-06` or even lower to reduce bias. For highly collinear data, set `ridge = 1e-04` or higher.
`midas.kappa`	Scalar. If `NULL` (default) then the optimal `kappa` gets selected automatically. Alternatively, the user may specify a scalar. Siddique and Belin 2008 find `midas.kappa = 3` to be sensible.
`outout`	Logical. If `TRUE` (default) one model is estimated for each donor (leave-one-out principle). For speedup choose `outout = FALSE`, which estimates one model for all observations leading to in-sample predictions for the donors and out-of-sample predictions for the recipients. Mind the inappropriateness, though.

`neff`	FOR EXPERTS. Null or character string. The name of an existing environment in which the effective sample size of the donors for each loop (CE iterations times multiple imputations) is supposed to be written. The effective sample size is necessary to compute the correction for the total variance as originally suggested by Parzen, Lipsitz and Fitzmaurice 2005. The objectname is `midastouch.neff`.
`debug`	FOR EXPERTS. Null or character string. The name of an existing environment in which the input is supposed to be written. The objectname is `midastouch.inputlist`.
`...`	Other named arguments.

Details

Imputation of `y` by predictive mean matching, based on Rubin (1987, p. 168, formulas a and b) and Siddique and Belin 2008. The procedure is as follows:

1. Draw a bootstrap sample from the donor pool.
2. Estimate a beta matrix on the bootstrap sample by the leave one out principle.
3. Compute type II predicted values for `yobs` (nobs x 1) and `ymis` (nmis x nobs).
4. Calculate the distance between all `yobs` and the corresponding `ymis`.
5. Convert the distances in drawing probabilities.
6. For each recipient draw a donor from the entire pool while considering the probabilities from the model.
7. Take its observed value in `y` as the imputation.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Philipp Gaffert, Florian Meinfelder, Volker Bosch 2015

References

Gaffert, P., Meinfelder, F., Bosch V. (2018) Towards an MI-proper Predictive Mean Matching, JSM 2018. Discussion Paper.

Little, R.J.A. (1988), Missing data adjustments in large surveys (with discussion), Journal of Business Economics and Statistics, 6, 287–301.

Parzen, M., Lipsitz, S. R., Fitzmaurice, G. M. (2005), A note on reducing the bias of the approximate Bayesian bootstrap imputation variance estimator. Biometrika **92**, 4, 971–974.

Rubin, D.B. (1987), Multiple imputation for nonresponse in surveys. New York: Wiley.

Siddique, J., Belin, T.R. (2008), Multiple imputation using an iterative hot-deck with distance-based donor selection. Statistics in medicine, **27**, 1, 83–102

Van Buuren, S., Brand, J.P.L., Groothuis-Oudshoorn C.G.M., Rubin, D.B. (2006), Fully conditional specification in multivariate imputation. *Journal of Statistical Computation and Simulation*, **76**, 12, 1049–1064.

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

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

Examples

```
# do default multiple imputation on a numeric matrix
imp <- mice(nhanes, method = "midastouch")
imp

# list the actual imputations for BMI
imp$imp$bmi

# first completed data matrix
complete(imp)

# imputation on mixed data with a different method per column
mice(nhanes2, method = c("sample", "midastouch", "logreg", "norm"))
```

`mice.impute.mnar.logreg` *Imputation under MNAR mechanism by NARFCS*

Description

Imputes univariate data under a user-specified MNAR mechanism by linear or logistic regression and NARFCS. Sensitivity analysis under different model specifications may shed light on the impact of different MNAR assumptions on the conclusions.

Usage

```
mice.impute.mnar.logreg(y, ry, x, wy = NULL, ums = NULL, umx = NULL, ...)

mice.impute.mnar.norm(y, ry, x, wy = NULL, ums = NULL, umx = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.

`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`ums`	A string containing the specification of the unidentifiable part of the imputation model (the *unidentifiable model specification"), that is, the desired δ-adjustment (offset) as a function of other variables and values for the corresponding deltas (sensitivity parameters). See details.
`umx`	An auxiliary data matrix containing variables that do not appear in the identifiable part of the imputation procedure but that have been specified via `ums` as being predictors in the unidentifiable part of the imputation model. See details.
`...`	Other named arguments.

Details

This function imputes data that are thought to be Missing Not at Random (MNAR) by the NARFCS method. The NARFCS procedure (Tompsett et al, 2018) generalises the so-called δ-adjustment sensitivity analysis method of Van Buuren, Boshuizen & Knook (1999) to the case with multiple incomplete variables within the FCS framework. In practical terms, the NARFCS procedure shifts the imputations drawn at each iteration of `mice` by a user-specified quantity that can vary across subjects, to reflect systematic departures of the missing data from the data distribution imputed under MAR.

Specification of the NARFCS model is done by the `blots` argument of `mice()`. The `blots` parameter is a named list. For each variable to be imputed by `mice.impute.mnar.norm()` or `mice.impute.mnar.logreg()` the corresponding element in `blots` is a list with at least one argument `ums` and, optionally, a second argument `umx`. For example, the high-level call might like something like `mice(nhanes[, c(2, 4)], method = c("pmm", "mnar.norm"),blots = list(chl = list(ums = "-3+2*bmi")))`.

The `ums` parameter is required, and might look like this: `"-4+1*Y"`. The `ums` specifcation must have the following characteristics:

1. A single term corresponding to the intercept (constant) term, not multiplied by any variable name, must be included in the expression;
2. Each term in the expression (corresponding to the intercept or a predictor variable) must be separated by either a `"+"` or `"-"` sign, depending on the sign of the sensitivity parameter;
3. Within each non-intercept term, the sensitivity parameter value comes first and the predictor variable comes second, and these must be separated by a `"*"` sign;
4. For categorical predictors, for example a variable `Z` with K + 1 categories `("Cat0","Cat1", ...,"CatK")`, K category-specific terms are needed, and those not in `umx` (see below) must be specified by concatenating the variable name with the name of the category (e.g. `ZCat1`) as this is how they are named in the design matrix (argument `x`) passed to the univariate imputation function. An example is `"2+1*ZCat1-3*ZCat2"`.

If given, the `umx` specification must have the following characteristics:

1. It contains only complete variables, with no missing values;
2. It is a numeric matrix. In particular, categorical variables must be represented as dummy indicators with names corresponding to what is used in `ums` to refer to the category-specific terms (see above);

3. It has the same number of rows as the `data` argument passed on to the main `mice` function;
4. It does not contain variables that were already predictors in the identifiable part of the model for the variable under imputation.

Limitation: The present implementation can only condition on variables that appear in the identifiable part of the imputation model (`x`) or in complete auxiliary variables passed on via the `umx` argument. It is not possible to specify models where the offset depends on incomplete auxiliary variables.

For an MNAR alternative see also `mice.impute.ri`.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Margarita Moreno-Betancur, Stef van Buuren, Ian R. White, 2020.

References

Tompsett, D. M., Leacy, F., Moreno-Betancur, M., Heron, J., & White, I. R. (2018). On the use of the not-at-random fully conditional specification (NARFCS) procedure in practice. *Statistics in Medicine*, **37**(15), 2338-2353. doi:10.1002/sim.7643.

Van Buuren, S., Boshuizen, H.C., Knook, D.L. (1999) Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine*, **18**, 681–694.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

Examples

```
# 1: Example with no auxiliary data: only pass unidentifiable model specification (ums)

# Specify argument to pass on to mnar imputation functions via "blots" argument
mnar.blot <- list(X = list(ums = "-4"), Y = list(ums = "2+1*ZCat1-3*ZCat2"))

# Run NARFCS by using mnar imputation methods and passing argument via blots
impNARFCS <- mice(mnar_demo_data,
  method = c("mnar.logreg", "mnar.norm", ""),
  blots = mnar.blot, seed = 234235, print = FALSE
)

# Obtain MI results: Note they coincide with those from old version at
# https://github.com/moreno-betancur/NARFCS
```

```
pool(with(impNARFCS, lm(Y ~ X + Z)))$pooled$estimate

# 2: Example passing also auxiliary data to MNAR procedure (umx)
# Assumptions:
# - Auxiliary data are complete, no missing values
# - Auxiliary data are a numeric matrix
# - Auxiliary data have same number of rows as x
# - Auxiliary data have no overlapping variable names with x

# Specify argument to pass on to mnar imputation functions via "blots" argument
aux <- matrix(0:1, nrow = nrow(mnar_demo_data))
dimnames(aux) <- list(NULL, "even")
mnar.blot <- list(
  X = list(ums = "-4"),
  Y = list(ums = "2+1*ZCat1-3*ZCat2+0.5*even", umx = aux)
)

# Run NARFCS by using mnar imputation methods and passing argument via blots
impNARFCS <- mice(mnar_demo_data,
  method = c("mnar.logreg", "mnar.norm", ""),
  blots = mnar.blot, seed = 234235, print = FALSE
)

# Obtain MI results: As expected they differ (slightly) from those
# from old version at https://github.com/moreno-betancur/NARFCS
pool(with(impNARFCS, lm(Y ~ X + Z)))$pooled$estimate
```

`mice.impute.mpmm` *Imputation by multivariate predictive mean matching*

Description

Imputes multivariate incomplete data among which there are specific relations, for instance, polynomials, interactions, range restrictions and sum scores.

Usage

```
mice.impute.mpmm(data, format = "imputes", ...)
```

Arguments

`data`	matrix with exactly two missing data patterns
`format`	A character vector specifying the type of object that should be returned. The default is `format = "imputes"`.
`...`	Other named arguments.

Details

This function implements the predictive mean matching and applies canonical regression analysis to select donors fora set of missing variables. In general, canonical regressionanalysis looks for a linear combination of covariates that predicts a linear combination of outcomes (a set of missing variables) optimally in a least-square sense (Israels, 1987). The predicted value of the linear combination of the set of missing variables would be applied to perform predictive mean matching.

Value

A matrix with imputed data, which has `ncol(y)` columns and `sum(wy)` rows.

Note

The function requires variables in the block have the same missingness pattern. If there are more than one missingness pattern, the function will return a warning.

Author(s)

Mingyang Cai and Gerko Vink

See Also

`mice.impute.pmm` Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

Examples

```
# simulate data
beta2 <- beta1 <- .5
x <- rnorm(1000)
e <- rnorm(1000, 0, 1)
y <- beta1 * x + beta2 * x^2 + e
dat <- data.frame(y = y, x = x, x2 = x^2)
m <- as.logical(rbinom(1000, 1, 0.25))
dat[m, c("x", "x2")] <- NA

# impute
blk <- list("y", c("x", "x2"))
meth <- c("", "mpmm")
imp <- mice(dat, blocks = blk, method = meth, print = FALSE,
    m = 2, maxit = 2)

# analyse and check
summary(pool(with(imp, lm(y ~ x + x2))))
```

```
with(dat, plot(x, x2, col = mdc(1)))
with(complete(imp), points(x[m], x2[m], col = mdc(2)))
```

`mice.impute.norm` *Imputation by Bayesian linear regression*

Description

Calculates imputations for univariate missing data by Bayesian linear regression, also known as the normal model.

Usage

```
mice.impute.norm(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

Imputation of `y` by the normal model by the method defined by Rubin (1987, p. 167). The procedure is as follows:

1. Calculate the cross-product matrix $S = X'_{obs}X_{obs}$.
2. Calculate $V = (S + diag(S)\kappa)^{-1}$, with some small ridge parameter κ.
3. Calculate regression weights $\hat{\beta} = VX'_{obs}y_{obs}$.
4. Draw a random variable $\dot{g} \sim \chi^2_\nu$ with $\nu = n_1 - q$.
5. Calculate $\dot{\sigma}^2 = (y_{obs} - X_{obs}\hat{\beta})'(y_{obs} - X_{obs}\hat{\beta})/\dot{g}$.
6. Draw q independent $N(0,1)$ variates in vector $\dot{z}_1$.
7. Calculate $V^{1/2}$ by Cholesky decomposition.
8. Calculate $\dot{\beta} = \hat{\beta} + \dot{\sigma}\dot{z}_1 V^{1/2}$.
9. Draw n_0 independent $N(0,1)$ variates in vector $\dot{z}_2$.
10. Calculate the n_0 values $y_{imp} = X_{mis}\dot{\beta} + \dot{z}_2\dot{\sigma}$.

Using `mice.impute.norm` for all columns emulates Schafer's NORM method (Schafer, 1997).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn

References

Rubin, D.B (1987). Multiple Imputation for Nonresponse in Surveys. New York: John Wiley & Sons.

Schafer, J.L. (1997). Analysis of incomplete multivariate data. London: Chapman & Hall.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.norm.boot` *Imputation by linear regression, bootstrap method*

Description

Imputes univariate missing data using linear regression with bootstrap

Usage

```
mice.impute.norm.boot(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

Draws a bootstrap sample from `x[ry,]` and `y[ry]`, calculates regression weights and imputes with normal residuals.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Gerko Vink, Stef van Buuren, 2018

References

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

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.norm.nob` *Imputation by linear regression without parameter uncertainty*

Description

Imputes univariate missing data using linear regression analysis without accounting for the uncertainty of the model parameters.

Usage

```
mice.impute.norm.nob(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.

`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

This function creates imputations using the spread around the fitted linear regression line of `y` given `x`, as fitted on the observed data.

This function is provided mainly to allow comparison between proper (e.g., as implemented in `mice.impute.norm` and improper (this function) normal imputation methods.

For large data, having many rows, differences between proper and improper methods are small, and in those cases one may opt for speed by using `mice.impute.norm.nob`.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Warning

The function does not incorporate the variability of the regression weights, so it is not 'proper' in the sense of Rubin. For small samples, variability of the imputed data is therefore underestimated.

Author(s)

Gerko Vink, Stef van Buuren, Karin Groothuis-Oudshoorn, 2018

References

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

Brand, J.P.L. (1999). Development, Implementation and Evaluation of Multiple Imputation Strategies for the Statistical Analysis of Incomplete Data Sets. Ph.D. Thesis, TNO Prevention and Health/Erasmus University Rotterdam.

See Also

`mice`, `mice.impute.norm`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.norm.predict` *Imputation by linear regression through prediction*

Description

Imputes the "best value" according to the linear regression model, also known as *regression imputation*.

Usage

```
mice.impute.norm.predict(y, ry, x, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

Calculates regression weights from the observed data and returns predicted values to as imputations. This method is known as *regression imputation*.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Warning

THIS METHOD SHOULD NOT BE USED FOR DATA ANALYSIS. This method is seductive because it imputes the most likely value according to the model. However, it ignores the uncertainty of the missing values and artificially amplifies the relations between the columns of the data. Application of richer models having more parameters does not help to evade these issues. Stochastic regression methods, like `mice.impute.pmm` or `mice.impute.norm`, are generally preferred.

At best, prediction can give reasonable estimates of the mean, especially if normality assumptions are plausible. See Little and Rubin (2002, p. 62-64) or Van Buuren (2012, p. 11-13, p. 45-46) for a discussion of this method.

Author(s)

Gerko Vink, Stef van Buuren, 2018

References

Little, R.J.A. and Rubin, D.B. (2002). Statistical Analysis with Missing Data. New York: John Wiley and Sons.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.panImpute` *Impute multilevel missing data using* `pan`

Description

This function is a wrapper around the `panImpute` function from the `mitml` package so that it can be called to impute blocks of variables in `mice`. The `mitml::panImpute` function provides an interface to the `pan` package for multiple imputation of multilevel data (Schafer & Yucel, 2002). Imputations can be generated using `type` or `formula`, which offer different options for model specification.

Usage

```
mice.impute.panImpute(
  data,
  formula,
  type,
  m = 1,
  silent = TRUE,
  format = "imputes",
  ...
)
```

Arguments

`data`	A data frame containing incomplete and auxiliary variables, the cluster indicator variable, and any other variables that should be present in the imputed datasets.

`formula`	A formula specifying the role of each variable in the imputation model. The basic model is constructed by `model.matrix`, thus allowing to include derived variables in the imputation model using `I()`. See `panImpute`.
`type`	An integer vector specifying the role of each variable in the imputation model (see `panImpute`)
`m`	The number of imputed data sets to generate.
`silent`	(optional) Logical flag indicating if console output should be suppressed. Default is to `FALSE`.
`format`	A character vector specifying the type of object that should be returned. The default is `format = "list"`. No other formats are currently supported.
`...`	Other named arguments: `n.burn`, `n.iter`, `group`, `prior`, `silent` and others.

Value

A list of imputations for all incomplete variables in the model, that can be stored in the the `imp` component of the `mids` object.

Note

The number of imputations `m` is set to 1, and the function is called `m` times so that it fits within the `mice` iteration scheme.

This is a multivariate imputation function using a joint model.

Author(s)

Stef van Buuren, 2018, building on work of Simon Grund, Alexander Robitzsch and Oliver Luedtke (authors of `mitml` package) and Joe Schafer (author of `pan` package).

References

Grund S, Luedtke O, Robitzsch A (2016). Multiple Imputation of Multilevel Missing Data: An Introduction to the R Package `pan`. SAGE Open.

Schafer JL (1997). Analysis of Incomplete Multivariate Data. London: Chapman & Hall.

Schafer JL, and Yucel RM (2002). Computational strategies for multivariate linear mixed-effects models with missing values. Journal of Computational and Graphical Statistics, 11, 437-457.

See Also

`panImpute`

Other multivariate-2l: `mice.impute.jomoImpute()`

Examples

```
blocks <- list(c("bmi", "chl", "hyp"), "age")
method <- c("panImpute", "pmm")
ini <- mice(nhanes, blocks = blocks, method = method, maxit = 0)
pred <- ini$pred
```

```
pred["B1", "hyp"] <- -2
imp <- mice(nhanes, blocks = blocks, method = method, pred = pred, maxit = 1)
```

`mice.impute.passive` *Passive imputation*

Description

Calculate new variable during imputation

Usage

```
mice.impute.passive(data, func)
```

Arguments

`data`	A data frame
`func`	A `formula` specifying the transformations on data

Details

Passive imputation is a special internal imputation function. Using this facility, the user can specify, at any point in the `mice` Gibbs sampling algorithm, a function on the imputed data. This is useful, for example, to compute a cubic version of a variable, a transformation like `Q = W/H^2` based on two variables, or a mean variable like `(x_1+x_2+x_3)/3`. The so derived variables might be used in other places in the imputation model. The function allows to dynamically derive virtually any function of the imputed data at virtually any time.

Value

The result of applying `formula`

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

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

See Also

`mice`

`mice.impute.pmm` *Imputation by predictive mean matching*

Description

Imputation by predictive mean matching

Usage

```
mice.impute.pmm(
  y,
  ry,
  x,
  wy = NULL,
  donors = 5L,
  matchtype = 1L,
  exclude = NULL,
  quantify = TRUE,
  trim = 1L,
  ridge = 1e-05,
  use.matcher = FALSE,
  ...
)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`donors`	The size of the donor pool among which a draw is made. The default is `donors = 5L`. Setting `donors = 1L` always selects the closest match, but is not recommended. Values between 3L and 10L provide the best results in most cases (Morris et al, 2015).
`matchtype`	Type of matching distance. The default choice (`matchtype = 1L`) calculates the distance between the *predicted* value of `yobs` and the *drawn* values of `ymis` (called type-1 matching). Other choices are `matchtype = 0L` (distance between predicted values) and `matchtype = 2L` (distance between drawn values).
`exclude`	Dependent values to exclude from the imputation model and the collection of donor values

`quantify`	Logical. If `TRUE`, factor levels are replaced by the first canonical variate before fitting the imputation model. If false, the procedure reverts to the old behaviour and takes the integer codes (which may lack a sensible interpretation). Relevant only of `y` is a factor.
`trim`	Scalar integer. Minimum number of observations required in a category in order to be considered as a potential donor value. Relevant only of `y` is a factor.
`ridge`	The ridge penalty used in `.norm.draw()` to prevent problems with multicollinearity. The default is `ridge = 1e-05`, which means that 0.01 percent of the diagonal is added to the cross-product. Larger ridges may result in more biased estimates. For highly noisy data (e.g. many junk variables), set `ridge = 1e-06` or even lower to reduce bias. For highly collinear data, set `ridge = 1e-04` or higher.
`use.matcher`	Logical. Set `use.matcher = TRUE` to specify the C function `matcher()`, the now deprecated matching function that was default in versions `2.22` (June 2014) to `3.11.7` (Oct 2020). Since version `3.12.0` `mice()` uses the much faster `matchindex` C function. Use the deprecated `matcher` function only for exact reproduction.
`...`	Other named arguments.

Details

Imputation of `y` by predictive mean matching, based on van Buuren (2012, p. 73). The procedure is as follows:

1. Calculate the cross-product matrix $S = X'_{obs}X_{obs}$.
2. Calculate $V = (S + diag(S)\kappa)^{-1}$, with some small ridge parameter κ.
3. Calculate regression weights $\hat\beta = VX'_{obs}y_{obs}$.
4. Draw q independent $N(0,1)$ variates in vector $\dot z_1$.
5. Calculate $V^{1/2}$ by Cholesky decomposition.
6. Calculate $\dot\beta = \hat\beta + \dot\sigma\dot z_1 V^{1/2}$.
7. Calculate $\dot\eta(i,j) = |X_{obs,[i]|}\hat\beta - X_{mis,[j]}\dot\beta$ with $i = 1,\ldots,n_1$ and $j = 1,\ldots,n_0$.
8. Construct n_0 sets Z_j, each containing d candidate donors, from y_{obs} such that $\sum_d \dot\eta(i,j)$ is minimum for all $j = 1,\ldots,n_0$. Break ties randomly.
9. Draw one donor i_j from Z_j randomly for $j = 1,\ldots,n_0$.
10. Calculate imputations $\dot y_j = y_{i_j}$ for $j = 1,\ldots,n_0$.

The name *predictive mean matching* was proposed by Little (1988).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Gerko Vink, Stef van Buuren, Karin Groothuis-Oudshoorn

References

Little, R.J.A. (1988), Missing data adjustments in large surveys (with discussion), Journal of Business Economics and Statistics, 6, 287–301.

Morris TP, White IR, Royston P (2015). Tuning multiple imputation by predictive mean matching and local residual draws. BMC Med Res Methodol. ;14:75.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

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

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

Examples

```
# We normally call mice.impute.pmm() from within mice()
# But we may call it directly as follows (not recommended)

set.seed(53177)
xname <- c("age", "hgt", "wgt")
r <- stats::complete.cases(boys[, xname])
x <- boys[r, xname]
y <- boys[r, "tv"]
ry <- !is.na(y)
table(ry)

# percentage of missing data in tv
sum(!ry) / length(ry)

# Impute missing tv data
yimp <- mice.impute.pmm(y, ry, x)
length(yimp)
hist(yimp, xlab = "Imputed missing tv")

# Impute all tv data
yimp <- mice.impute.pmm(y, ry, x, wy = rep(TRUE, length(y)))
length(yimp)
hist(yimp, xlab = "Imputed missing and observed tv")
plot(jitter(y), jitter(yimp),
  main = "Predictive mean matching on age, height and weight",
  xlab = "Observed tv (n = 224)",
  ylab = "Imputed tv (n = 224)"
)
abline(0, 1)
```

```
cor(y, yimp, use = "pair")

# Use blots to exclude different values per column
# Create blots object
blots <- make.blots(boys)
# Exclude ml 1 through 5 from tv donor pool
blots$tv$exclude <- c(1:5)
# Exclude 100 random observed heights from tv donor pool
blots$hgt$exclude <- sample(unique(boys$hgt), 100)
imp <- mice(boys, method = "pmm", print = FALSE, blots = blots, seed=123)
blots$hgt$exclude %in% unlist(c(imp$imp$hgt)) # MUST be all FALSE
blots$tv$exclude %in% unlist(c(imp$imp$tv)) # MUST be all FALSE

# Factor quantification
xname <- c("age", "hgt", "wgt")
br <- boys[c(1:10, 101:110, 501:510, 601:620, 701:710), ]
r <- stats::complete.cases(br[, xname])
x <- br[r, xname]
y <- factor(br[r, "tv"])
ry <- !is.na(y)
table(y)

# impute factor by optimizing canonical correlation y, x
mice.impute.pmm(y, ry, x)

# only categories with at least 2 cases can be donor
mice.impute.pmm(y, ry, x, trim = 2L)

# in addition, eliminate category 20
mice.impute.pmm(y, ry, x, trim = 2L, exclude = 20)

# to get old behavior: as.integer(y))
mice.impute.pmm(y, ry, x, quantify = FALSE)
```

`mice.impute.polr` *Imputation of ordered data by polytomous regression*

Description

Imputes missing data in a categorical variable using polytomous regression

Usage

```
mice.impute.polr(
  y,
  ry,
  x,
  wy = NULL,
  nnet.maxit = 100,
  nnet.trace = FALSE,
```

```
  nnet.MaxNWts = 1500,
  polr.to.loggedEvents = FALSE,
  ...
)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`nnet.maxit`	Tuning parameter for `nnet()`.
`nnet.trace`	Tuning parameter for `nnet()`.
`nnet.MaxNWts`	Tuning parameter for `nnet()`.
`polr.to.loggedEvents`	A logical indicating whether each fallback to the `multinom()` function should be written to `loggedEvents`. The default is `FALSE`.
`...`	Other named arguments.

Details

The function `mice.impute.polr()` imputes for ordered categorical response variables by the proportional odds logistic regression (polr) model. The function repeatedly applies logistic regression on the successive splits. The model is also known as the cumulative link model.

By default, ordered factors with more than two levels are imputed by `mice.impute.polr`.

The algorithm of `mice.impute.polr` uses the function `polr()` from the `MASS` package.

In order to avoid bias due to perfect prediction, the algorithm augment the data according to the method of White, Daniel and Royston (2010).

The call to `polr` might fail, usually because the data are very sparse. In that case, `multinom` is tried as a fallback. If the local flag `polr.to.loggedEvents` is set to TRUE, a record is written to the `loggedEvents` component of the `mids` object. Use `mice(data, polr.to.loggedEvents = TRUE)` to set the flag.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Note

In December 2019 Simon White alerted that the `polr` could always fail silently. I can confirm this behaviour for versions `mice 3.0.0 - mice 3.6.6`, so any method requests for `polr` in these versions were in fact handled by `multinom`. See `https://github.com/amices/mice/issues/206` for details.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000-2010

References

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

Brand, J.P.L. (1999) *Development, implementation and evaluation of multiple imputation strategies for the statistical analysis of incomplete data sets.* Dissertation. Rotterdam: Erasmus University.

White, I.R., Daniel, R. Royston, P. (2010). Avoiding bias due to perfect prediction in multiple imputation of incomplete categorical variables. *Computational Statistics and Data Analysis*, 54, 2267-2275.

Venables, W.N. & Ripley, B.D. (2002). *Modern applied statistics with S-Plus (4th ed)*. Springer, Berlin.

See Also

`mice`, `multinom`, `polr`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.polyreg` *Imputation of unordered data by polytomous regression*

Description

Imputes missing data in a categorical variable using polytomous regression

Usage

```
mice.impute.polyreg(
  y,
  ry,
  x,
  wy = NULL,
  nnet.maxit = 100,
  nnet.trace = FALSE,
  nnet.MaxNWts = 1500,
  ...
)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`nnet.maxit`	Tuning parameter for `nnet()`.
`nnet.trace`	Tuning parameter for `nnet()`.
`nnet.MaxNWts`	Tuning parameter for `nnet()`.
`...`	Other named arguments.

Details

The function `mice.impute.polyreg()` imputes categorical response variables by the Bayesian polytomous regression model. See J.P.L. Brand (1999), Chapter 4, Appendix B.

By default, unordered factors with more than two levels are imputed by `mice.impute.polyreg()`.

The method consists of the following steps:

1. Fit categorical response as a multinomial model
2. Compute predicted categories
3. Add appropriate noise to predictions

The algorithm of `mice.impute.polyreg` uses the function `multinom()` from the `nnet` package.

In order to avoid bias due to perfect prediction, the algorithm augment the data according to the method of White, Daniel and Royston (2010).

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000-2010

References

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

Brand, J.P.L. (1999) *Development, implementation and evaluation of multiple imputation strategies for the statistical analysis of incomplete data sets.* Dissertation. Rotterdam: Erasmus University.

White, I.R., Daniel, R. Royston, P. (2010). Avoiding bias due to perfect prediction in multiple imputation of incomplete categorical variables. *Computational Statistics and Data Analysis*, 54, 2267-2275.

Venables, W.N. & Ripley, B.D. (2002). *Modern applied statistics with S-Plus (4th ed)*. Springer, Berlin.

See Also

`mice`, `multinom`, `polr`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.quadratic()`, `mice.impute.rf()`, `mice.impute.ri()`

`mice.impute.quadratic` *Imputation of quadratic terms*

Description

Imputes incomplete variable that appears as both main effect and quadratic effect in the complete-data model.

Usage

```
mice.impute.quadratic(y, ry, x, wy = NULL, quad.outcome = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`quad.outcome`	The name of the outcome in the quadratic analysis as a character string. For example, if the substantive model of interest is `y ~ x + xx`, then `"y"` would be the `quad.outcome`
`...`	Other named arguments.

Details

This function implements the "polynomial combination" method. First, the polynomial combination $Z = Y\beta_1 + Y^2\beta_2$ is formed. Z is imputed by predictive mean matching, followed by a decomposition of the imputed data Z into components Y and Y^2. See Van Buuren (2012, pp. 139-141) and Vink et al (2012) for more details. The method ensures that 1) the imputed data for Y and Y^2 are mutually consistent, and 2) that provides unbiased estimates of the regression weights in a complete-data linear regression that use both Y and Y^2.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Note

There are two situations to consider. If only the linear term `Y` is present in the data, calculate the quadratic term `YY` after imputation. If both the linear term `Y` and the the quadratic term `YY` are variables in the data, then first impute `Y` by calling `mice.impute.quadratic()` on `Y`, and then impute `YY` by passive imputation as `meth["YY"] <- "~I(Y^2)"`. See example section for details. Generally, we would like `YY` to be present in the data if we need to preserve quadratic relations between `YY` and any third variables in the multivariate incomplete data that we might wish to impute.

Author(s)

Mingyang Cai and Gerko Vink

See Also

`mice.impute.pmm` Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Vink, G., van Buuren, S. (2013). Multiple Imputation of Squared Terms. *Sociological Methods & Research*, 42:598-607.

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.rf()`, `mice.impute.ri()`

Examples

```
# Create Data
B1 <- .5
B2 <- .5
X <- rnorm(1000)
XX <- X^2
e <- rnorm(1000, 0, 1)
Y <- B1 * X + B2 * XX + e
dat <- data.frame(x = X, xx = XX, y = Y)

# Impose 25 percent MCAR Missingness
dat[0 == rbinom(1000, 1, 1 - .25), 1:2] <- NA

# Prepare data for imputation
ini <- mice(dat, maxit = 0)
meth <- c("quadratic", "~I(x^2)", "")
pred <- ini$pred
pred[, "xx"] <- 0

# Impute data
```

```
imp <- mice(dat, meth = meth, pred = pred, quad.outcome = "y")

# Pool results
pool(with(imp, lm(y ~ x + xx)))

# Plot results
stripplot(imp)
plot(dat$x, dat$xx, col = mdc(1), xlab = "x", ylab = "xx")
cmp <- complete(imp)
points(cmp$x[is.na(dat$x)], cmp$xx[is.na(dat$x)], col = mdc(2))
```

`mice.impute.rf` *Imputation by random forests*

Description

Imputes univariate missing data using random forests.

Usage

```
mice.impute.rf(
  y,
  ry,
  x,
  wy = NULL,
  ntree = 10,
  rfPackage = c("ranger", "randomForest", "literanger"),
  ...
)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`ntree`	The number of trees to grow. The default is 10.
`rfPackage`	A single string specifying the backend for estimating the random forest. The default backend is the `ranger` package. An alternative is `literanger` which predicts faster but does not support all forest types and split rules from `ranger`. Also implemented as an alternative is the `randomForest` package, which used to be the default in mice 3.13.10 and earlier.
`...`	Other named arguments passed down to `mice:::install.on.demand()`, `randomForest::randomFores` `randomForest:::randomForest.default()`, `ranger::ranger()`, and `literanger::train()`.

Details

Imputation of `y` by random forests. The method calls `randomForrest()` which implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. See Appendix A.1 of Doove et al. (2014) for the definition of the algorithm used.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Note

An alternative implementation was independently developed by Shah et al (2014). This were available as functions `CALIBERrfimpute::mice.impute.rfcat` and `CALIBERrfimpute::mice.impute.rfcont` (now archived). Simulations by Shah (Feb 13, 2014) suggested that the quality of the imputation for 10 and 100 trees was identical, so mice 2.22 changed the default number of trees from `ntree = 100` to `ntree = 10`.

Author(s)

Lisa Doove, Stef van Buuren, Elise Dusseldorp, 2012; Patrick Rockenschaub, 2021

References

Doove, L.L., van Buuren, S., Dusseldorp, E. (2014), Recursive partitioning for missing data imputation in the presence of interaction Effects. Computational Statistics & Data Analysis, 72, 92-104.

Shah, A.D., Bartlett, J.W., Carpenter, J., Nicholas, O., Hemingway, H. (2014), Comparison of random forest and parametric imputation models for imputing missing data using MICE: A CALIBER study. American Journal of Epidemiology, doi:10.1093/aje/kwt312.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`mice`, `mice.impute.cart`, `randomForest`, `ranger`, `train`

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.ri()`

Examples

```
## Not run:
imp <- mice(nhanes2, meth = "rf", ntree = 3)
plot(imp)

## End(Not run)
```

`mice.impute.ri` *Imputation by the random indicator method for nonignorable data*

Description

Imputes nonignorable missing data by the random indicator method.

Usage

```
mice.impute.ri(y, ry, x, wy = NULL, ri.maxit = 10, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`ri.maxit`	Number of inner iterations
`...`	Other named arguments.

Details

The random indicator method estimates an offset between the distribution of the observed and missing data using an algorithm that iterates over the response and imputation models.

This routine assumes that the response model and imputation model have same predictors.

For an MNAR alternative see also `mice.impute.mnar.logreg`.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Shahab Jolani (University of Utrecht)

References

Jolani, S. (2012). *Dual Imputation Strategies for Analyzing Incomplete Data*. Dissertation. University of Utrecht, Dec 7 2012.

See Also

Other univariate imputation functions: `mice.impute.cart()`, `mice.impute.lasso.logreg()`, `mice.impute.lasso.norm()`, `mice.impute.lasso.select.logreg()`, `mice.impute.lasso.select.norm()`, `mice.impute.lda()`, `mice.impute.logreg()`, `mice.impute.logreg.boot()`, `mice.impute.mean()`, `mice.impute.midastouch()`, `mice.impute.mnar.logreg()`, `mice.impute.mpmm()`, `mice.impute.norm()`, `mice.impute.norm.boot()`, `mice.impute.norm.nob()`, `mice.impute.norm.predict()`, `mice.impute.pmm()`, `mice.impute.polr()`, `mice.impute.polyreg()`, `mice.impute.quadratic()`, `mice.impute.rf()`

`mice.impute.sample` *Imputation by simple random sampling*

Description

Imputes a random sample from the observed `y` data

Usage

```
mice.impute.sample(y, ry, x = NULL, wy = NULL, ...)
```

Arguments

`y`	Vector to be imputed
`ry`	Logical vector of length `length(y)` indicating the the subset `y[ry]` of elements in `y` to which the imputation model is fitted. The `ry` generally distinguishes the observed (`TRUE`) and missing values (`FALSE`) in `y`.
`x`	Numeric design matrix with `length(y)` rows with predictors for `y`. Matrix `x` may have no missing values.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`...`	Other named arguments.

Details

This function takes a simple random sample from the observed values in `y`, and returns these as imputations.

Value

Vector with imputed data, same type as `y`, and of length `sum(wy)`

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000, 2017

References

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

`mice.mids` *Multivariate Imputation by Chained Equations (Iteration Step)*

Description

Takes a `mids` object, performs `maxit` iterations and produces a new object of class `"mids"`.

Usage

```
mice.mids(obj, newdata = NULL, maxit = 1, printFlag = TRUE, ...)
```

Arguments

`obj`	An object of class `mids`, typically produces by a previous call to `mice()` or `mice.mids()`
`newdata`	An optional `data.frame` for which multiple imputations are generated according to the model in `obj`.
`maxit`	The number of additional Gibbs sampling iterations. The default is 1.
`printFlag`	A Boolean flag. If `TRUE`, diagnostic information during the Gibbs sampling iterations will be written to the command window. The default is `TRUE`.
`...`	Named arguments that are passed down to the univariate imputation functions.

Details

This function enables the user to split up the computations of the Gibbs sampler into smaller parts. This is useful for the following reasons:

- To add a few extra iteration to an existing solution.
- If RAM memory is exhausted. Returning to prompt/session level may alleviate such problems.
- To customize convergence statistics at specific points, e.g., after every `maxit` iterations to monitor convergence.

The imputation model itself is specified in the `mice()` function and cannot be changed in `mice.mids()`. The state of the random generator is saved with the `mids` object. This ensures that the imputations are reproducible.

Value

`mice.mids` returns an object of class `"mids"`.

See Also

`complete`, `mice`, `set.seed`, `mids`

Examples

```
imp1 <- mice(nhanes, maxit = 1, seed = 123)
imp2 <- mice.mids(imp1)

# yields the same result as
imp <- mice(nhanes, maxit = 2, seed = 123)

# verification
identical(imp$imp, imp2$imp)
#
```

`mice.theme` *Set the theme for the plotting Trellis functions*

Description

The `mice.theme()` function sets default choices for Trellis plots that are built into **mice**.

Usage

```
mice.theme(transparent = TRUE, alpha.fill = 0.3)
```

Arguments

`transparent`	A logical indicating whether alpha-transparency is allowed. The default is `TRUE`.
`alpha.fill`	A numerical values between 0 and 1 that indicates the default alpha value for fills.

Value

`mice.theme()` returns a named list that can be used as a theme in the functions in **lattice**. By default, the `mice.theme()` function sets `transparent <- TRUE` if the current device `.Device` supports semi-transparent colors.

Author(s)

Stef van Buuren 2011

mids *Multiply imputed data set (`mids`)*

Description

The `mids` object is an S3 class that represents a multiply imputed data set. The `mids()` function is the S3 constructor. The following functions produce a `mids` object: `mids()`, `mice()`, `mice.mids()`, `cbind()`, `rbind()`, `ibind()`, `as.mids()` and `filter()`.

Usage

```
mids(
  data = data.frame(),
  imp = list(),
  m = integer(),
  where = matrix,
  blocks = list(),
  call = match.call(),
  nmis = integer(),
  method = character(),
  predictorMatrix = matrix(),
  visitSequence = character(),
  formulas = list(),
  calltype = character(),
  post = character(),
  blots = list(),
  ignore = logical(),
  seed = integer(),
  iteration = integer(),
 lastSeedValue = tryCatch(get(".Random.seed", envir = globalenv(), mode = "integer",
    inherits = FALSE), error = function(e) NULL),
  chainMean = list(),
  chainVar = list(),
  loggedEvents = data.frame(),
  version = packageVersion("mice"),
  date = Sys.Date()
)

## S3 method for class 'mids'
plot(
  x,
  y = NULL,
  theme = mice.theme(),
  layout = c(2, 3),
  type = "l",
  col = 1:10,
  lty = 1,
```

```
  ...
)

## S3 method for class 'mids'
print(x, ...)

## S3 method for class 'mids'
summary(object, ...)
```

Arguments

`data`	A data frame or a matrix containing the incomplete data. Missing values are coded as `NA`.
`imp`	Calculated field
`m`	Number of multiple imputations. The default is `m=5`.
`where`	A data frame or matrix with logicals of the same dimensions as `data` indicating where in the data the imputations should be created. The default, `where = is.na(data)`, specifies that the missing data should be imputed. The `where` argument may be used to overimpute observed data, or to skip imputations for selected missing values. Note: Imputation methods that generate imptutations outside of `mice`, like `mice.impute.panImpute()` may depend on a complete predictor space. In that case, a custom `where` matrix can not be specified.
`blocks`	List of vectors with variable names per block. List elements may be named to identify blocks. Variables within a block are imputed by a multivariate imputation method (see `method` argument). By default each variable is placed into its own block, which is effectively fully conditional specification (FCS) by univariate models (variable-by-variable imputation). Only variables whose names appear in `blocks` are imputed. The relevant columns in the `where` matrix are set to `FALSE` of variables that are not block members. A variable may appear in multiple blocks. In that case, it is effectively re-imputed each time that it is visited.
`call`	Calculated field
`nmis`	Calculated field
`method`	Can be either a single string, or a vector of strings with length `length(blocks)`, specifying the imputation method to be used for each column in data. If specified as a single string, the same method will be used for all blocks. The default imputation method (when no argument is specified) depends on the measurement level of the target column, as regulated by the `defaultMethod` argument. Columns that need not be imputed have the empty method `""`. See details.
`predictorMatrix`	A numeric matrix of `length(blocks)` rows and `ncol(data)` columns, containing 0/1 data specifying the set of predictors to be used for each target column. Each row corresponds to a variable block, i.e., a set of variables to be imputed. A value of `1` means that the column variable is used as a predictor for the target block (in the rows). By default, the `predictorMatrix` is a square matrix of `ncol(data)` rows and columns with all 1's, except for the diagonal. Note: For

	two-level imputation models (which have `"2l"` in their names) other codes (e.g, `2` or `-2`) are also allowed.
`visitSequence`	A vector of block names of arbitrary length, specifying the sequence of blocks that are imputed during one iteration of the Gibbs sampler. A block is a collection of variables. All variables that are members of the same block are imputed when the block is visited. A variable that is a member of multiple blocks is re-imputed within the same iteration. The default `visitSequence = "roman"` visits the blocks (left to right) in the order in which they appear in `blocks`. One may also use one of the following keywords: `"arabic"` (right to left), `"monotone"` (ordered low to high proportion of missing data) and `"revmonotone"` (reverse of monotone). *Special case*: If you specify both `visitSequence = "monotone"` and `maxit = 1`, then the procedure will edit the `predictorMatrix` to conform to the monotone pattern. Realize that convergence in one iteration is only guaranteed if the missing data pattern is actually monotone. The procedure does not check this.
`formulas`	A named list of formula's, or expressions that can be converted into formula's by `as.formula`. List elements correspond to blocks. The block to which the list element applies is identified by its name, so list names must correspond to block names. The `formulas` argument is an alternative to the `predictorMatrix` argument that allows for more flexibility in specifying imputation models, e.g., for specifying interaction terms.
`calltype`	A character vector of `length(block)` elements that indicates how the imputation model is specified. Entries can one of two values: `"pred"` or `"formula"`. If `calltype = "pred"`, the predictors of the imputation model for the block are specified by the corresponding row of the `predictorMatrix`. If `calltype = "formula"` the imputation model is specified by relevant entry in `formulas`. The default depends on the presence of the `formulas` argument. If `formulas` is present, then `mice()` sets `calltype = "formula"` for any block for which a formula is specified. Otherwise, `calltype = "pred"`.
`post`	A vector of strings with length `ncol(data)` specifying expressions as strings. Each string is parsed and executed within the `sampler()` function to post-process imputed values during the iterations. The default is a vector of empty strings, indicating no post-processing. Multivariate (block) imputation methods ignore the `post` parameter.
`blots`	A named `list` of `alist`'s that can be used to pass down arguments to lower level imputation function. The entries of element `blots[[blockname]]` are passed down to the function called for block `blockname`.
`ignore`	A logical vector of `nrow(data)` elements indicating which rows are ignored when creating the imputation model. The default `NULL` includes all rows that have an observed value of the variable to imputed. Rows with `ignore` set to `TRUE` do not influence the parameters of the imputation model, but are still imputed. We may use the `ignore` argument to split `data` into a training set (on which the imputation model is built) and a test set (that does not influence the imputation model estimates). Note: Multivariate imputation methods, like `mice.impute.jomoImpute()` or `mice.impute.panImpute()`, do not honour the `ignore` argument.

`seed`	An integer that is used as argument by the `set.seed()` for offsetting the random number generator. Default is to leave the random number generator alone.
`iteration`	Calculated field
`lastSeedValue`	Calculated field
`chainMean`	Calculated field
`chainVar`	Calculated field
`loggedEvents`	Calculated field
`version`	Calculated field
`date`	Calculated field
`x`	An object of class `mids`
`y`	A formula that specifies which variables, stream and iterations are plotted. If omitted, all streams, variables and iterations are plotted.
`theme`	The trellis theme to applied to the graphs. The default is `mice.theme()`.
`layout`	A vector of length 2 given the number of columns and rows in the plot. The default is `c(2, 3)`.
`type`	Parameter `type` of `panel.xyplot`.
`col`	Parameter `col` of `panel.xyplot`.
`lty`	Parameter `lty` of `panel.xyplot`.
`...`	Others arguments
`object`	Object of class `mids`

Details

The S3 class `mids` has the following methods: `bwplot()`, `complete()`, `densityplot()`, `plot()`, `print()`, `stripplot()`, `summary()`, `with()` and `xyplot()`.

Value

`mids()` returns a `mids` object.

`plot()` returns a `xyplot` object.

`print()` returns the input object invisibly.

`summary()` returns the input object invisibly.

Structure

Objects of class `"mids"` are lists with the following elements:

`data`: Original (incomplete) data set.

`imp`: A list of `ncol(data)` components with the generated multiple imputations. Each list component is a `data.frame` (`nmis[j]` by `m`) of imputed values for variable `j`. A `NULL` component is used for variables for which not imputations are generated.

`m`: Number of imputations.

`where`: The `where` argument of the `mice()` function.

`blocks`**:** The `blocks` argument of the `mice()` function.

`call`**:** Call that created the object.

`nmis`**:** An Named vector with counts of missing values per variable

`method`**:** A vector of strings of `length(blocks` specifying the imputation method per block.

`predictorMatrix`**:** A numerical matrix of containing integers specifying the predictor set.

`visitSequence`**:** A vector of variable and block names that specifies how variables and blocks are visited in one iteration throuh the data.

`formulas`**:** A named list of formula's, or expressions that can be converted into formula's by `as.formula`. List elements correspond to blocks. The block to which the list element applies is identified by its name, so list names must correspond to block names.

`post`**:** A vector of strings of length `length(blocks)` with commands for post-processing.

`blots`**:** "Block dots". The `blots` argument to the `mice()` function.

`ignore`**:** A logical vector of length `nrow(data)` indicating the rows in `data` used to build the imputation model. (new in `mice 3.12.0`)

`seed`**:** The seed value of the solution.

`iteration`**:** Last Gibbs sampling iteration number.

`lastSeedValue`**:** Random number generator state.

`chainMean`**:** An array of dimensions `ncol` by `maxit` by `m` elements containing the mean of the generated multiple imputations. The array can be used for monitoring convergence. Note that observed data are not present in this mean.

`chainVar`**:** An array with similar structure as `chainMean`, containing the variance of the imputed values.

`loggedEvents`**:** A `data.frame` with five columns containing warnings, corrective actions, and other inside info.

`version`**:** Version number of `mice` package that created the object.

`date`**:** Date at which the object was created.

LoggedEvents

The `loggedEvents` entry is a matrix with five columns containing a record of automatic removal actions. It is `NULL` is no action was made. At initialization the program removes constant variables, and removes variables to cause collinearity. During iteration, the program does the following actions:

- One or more variables that are linearly dependent are removed (for categorical data, a 'variable' corresponds to a dummy variable)
- Proportional odds regression imputation that does not converge and is replaced by `polyreg`.

Explanation of elements in `loggedEvents`:

`it` iteration number at which the record was added,

`im` imputation number,

`dep` name of the dependent variable,

`meth` imputation method used,

`out` a (possibly long) character vector with the names of the altered or removed predictors.

Methods

The `mids` class of objects has methods for the following generic functions: `print`, `summary`, `plot`.

Plot

The `plot()` metho plots the trace lines of the MICE algorithm. The `plot` method for a `mids` object plots the mean imputed value per imputation and the mean standard deviation of the imputed values against the iteration number for each of the m replications. By default, the function creates a plot for each incomplete variable. On convergence, the streams should intermingle and be free of any trend.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn

References

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

See Also

`mice`, `mira`, `mipo`, `xyplot`

Examples

```
data <- data.frame(a = c(1, NA, 3), b = c(NA, 2, 3))
q <- list(
  a = structure(
    list(`1` = 3, `2` = 3, `3` = 3, `4` = 3, `5` = 3),
         row.names = "2", class = "data.frame"),
  b = structure(
    list(`1` = 3, `2` = 3, `3` = 2, `4` = 2, `5` = 3),
         row.names = "1", class = "data.frame"))

imp <- mids(
  data = data,
  imp = q,
  m = 5,
  where = is.na(data),
  blocks = list(a = "a", b = "b"),
  nmis = colSums(is.na(data)),
  method = c(a = "mean", b = "norm"),
 predictorMatrix = matrix(1, nrow = 2, ncol = 2, dimnames = list(c("a", "b"), c("a", "b"))),
  visitSequence = c("a", "b"),
  formulas = list(a = a ~ b, b = b ~ a),
  post = NULL,
  blots = NULL,
  ignore = logical(nrow(data)),
  seed = 123,
  iteration = 1,
```

```
  chainMean = list(a = c(1, 2, 3), b = c(3, 2, 1)),
  chainVar = list(a = c(1.1, 1.2, 1.3), b = c(0.9, 1.0, 1.1)),
  loggedEvents = NULL)

print(imp)
imp <- mice(nhanes, print = FALSE)
plot(imp, bmi + chl ~ .it | .ms, layout = c(2, 1))
```

`mids2mplus` *Export* `mids` *object to Mplus*

Description

Converts a `mids` object into a format recognized by Mplus, and writes the data and the Mplus input files

Usage

```
mids2mplus(
  imp,
  file.prefix = "imp",
  path = getwd(),
  sep = "\t",
  dec = ".",
  silent = FALSE
)
```

Arguments

`imp`	The `imp` argument is an object of class `mids`, typically produced by the `mice()` function.
`file.prefix`	A character string describing the prefix of the output data files.
`path`	A character string containing the path of the output file. By default, files are written to the current `R` working directory.
`sep`	The separator between the data fields.
`dec`	The decimal separator for numerical data.
`silent`	A logical flag stating whether the names of the files should be printed.

Details

This function automates most of the work needed to export a `mids` object to `Mplus`. The function writes the multiple imputation datasets, the file that contains the names of the multiple imputation data sets and an `Mplus` input file. The `Mplus` input file has the proper file names, so in principle it should run and read the data without alteration. `Mplus` will recognize the data set as a multiply imputed data set, and do automatic pooling in procedures where that is supported.

Value

The return value is `NULL`.

Author(s)

Gerko Vink, 2011.

See Also

`mids`, `mids2spss`

`mids2spss` *Export* `mids` *object to SPSS*

Description

Converts a `mids` object into a format recognized by SPSS, and writes the data and the SPSS syntax files.

Usage

```
mids2spss(
  imp,
  filename = "midsdata",
  path = getwd(),
  compress = FALSE,
  silent = FALSE
)
```

Arguments

`imp`	The `imp` argument is an object of class `mids`, typically produced by the `mice()` function.
`filename`	A character string describing the name of the output data file and its extension.
`path`	A character string containing the path of the output file. The value in `path` is appended to `filedat`. By default, files are written to the current `R` working directory. If `path=NULL` then no file path appending is done.
`compress`	A logical flag stating whether the resulting SPSS set should be a compressed `.zsav` file.
`silent`	A logical flag stating whether the location of the saved file should be printed.

Details

This function automates most of the work needed to export a `mids` object to SPSS. It uses `haven::write_sav()` to facilitate the export to an SPSS `.sav` or `.zsav` file.

Below are some things to pay attention to.

The `SPSS` syntax file has the proper file names and separators set, so in principle it should run and read the data without alteration. `SPSS` is more strict than `R` with respect to the paths. Always use the full path, otherwise `SPSS` may not be able to find the data file.

Factors in `R` translate into categorical variables in `SPSS`. The internal coding of factor levels used in `R` is exported. This is generally acceptable for `SPSS`. However, when the data are to be combined with existing `SPSS` data, watch out for any changes in the factor levels codes.

`SPSS` will recognize the data set as a multiply imputed data set, and do automatic pooling in procedures where that is supported. Note however that pooling is an extra option only available to those who license the `MISSING VALUES` module. Without this license, `SPSS` will still recognize the structure of the data, but it will not pool the multiply imputed estimates into a single inference.

Value

The return value is `NULL`.

Author(s)

Gerko Vink, dec 2020.

See Also

`mids`

`mira` *Create an object of class "mira"*

Description

The mira() functions constructs an S3 object representing a set of multiply imputed repeated analyses (mira). The default workflow generates the `mira` object using the `with()` function.

Usage

```
mira(
  call = match.call(),
  call1 = match.call(),
  nmis = integer(),
  analyses = list()
)
```

Arguments

`call`	The function call that created the object.
`call1`	A secondary function call, typically from the first imputation.
`nmis`	An integer vector representing the number of missing values.
`analyses`	A list of analyses performed on the imputed datasets.

Details

The `as.mira()` function takes the results of repeated complete-data analysis stored as a list, and turns it into a `mira` object that can be pooled.

In versions prior to `mice 3.0` pooling required only that `coef()` and `vcov()` methods were available for fitted objects. *This feature is no longer supported*. The reason is that `vcov()` methods are inconsistent across packages, leading to buggy behaviour of the `pool()` function. Since `mice 3.0+`, the `broom` package takes care of filtering out the relevant parts of the complete-data analysis. It may happen that you'll see the messages like `No method for tidying an S3 object of class ...` or `Error: No glance method for objects of class ....` The royal way to solve this problem is to write your own `glance()` and `tidy()` methods and add these to `broom` according to the specifications given in `https://broom.tidymodels.org`.

The `mira` class of objects has methods for the following generic functions: `print`, `summary`.

Value

An object of class `"mira"`. The `mira` class contains the following elements:

`.Data`: Object of class `"list"` containing the following slots:

`call`: The call that created the object.

`call1`: The call that created the `mids` object that was used in `call`.

`nmis`: An array containing the number of missing observations per column.

`analyses`: A list of `m` components containing the individual fit objects from each of the `m` complete data analyses.

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

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

See Also

`with.mids`, `mids`, `mipo`

`mnar_demo_data` *MNAR demo data*

Description

A toy example from Margarita Moreno-Betancur for checking NARFCS.

Usage

```
mnar_demo_data
```

Format

An object of class `data.frame` with 500 rows and 3 columns.

Details

A small dataset with just three columns.

Source

https://github.com/moreno-betancur/NARFCS/blob/master/datmis.csv

---

`name.blocks` *Name imputation blocks*

Description

This helper function names any unnamed elements in the `blocks` specification. This is a convenience function.

Usage

```
name.blocks(x, prefix = "B")
```

Arguments

`x`	A `data.frame`, character vector with variable names, or `list` with variable names.
`prefix`	A character vector of length 1 with the prefix to be using for naming any unnamed blocks with two or more variables.

Details

This function will name any unnamed list elements specified in the optional argument `blocks`. Unnamed blocks consisting of just one variable will be named after this variable. Unnamed blocks containing more than one variables will be named by the `prefix` argument, padded by an integer sequence stating at 1.

Value

A named list of character vectors with variables names.

See Also

`mice`

Examples

```
blocks <- list(c("hyp", "chl"), AGE = "age", c("bmi", "hyp"), "edu")
name.blocks(blocks)
```

`name.formulas` *Name formula list elements*

Description

This helper function names any unnamed elements in the `formula` list. This is a convenience function.

Usage

```
name.formulas(formulas, prefix = "F")
```

Arguments

`formulas`	A named list of formula's, or expressions that can be converted into formula's by `as.formula`. List elements correspond to blocks. The block to which the list element applies is identified by its name, so list names must correspond to block names. The `formulas` argument is an alternative to the `predictorMatrix` argument that allows for more flexibility in specifying imputation models, e.g., for specifying interaction terms.
`prefix`	A character vector of length 1 with the prefix to be using for naming any unnamed blocks with two or more variables.

Details

This function will name any unnamed list elements specified in the optional argument `formula`. Unnamed formula's consisting with just one response variable will be named after this variable. Unnamed formula's containing more than one variable will be named by the `prefix` argument, padded by an integer sequence stating at 1.

Value

Named list of formulas

See Also

`mice`

Examples

```
# fully conditionally specified main effects model
form1 <- list(
  bmi ~ age + chl + hyp,
  hyp ~ age + bmi + chl,
  chl ~ age + bmi + hyp
)
form1 <- name.formulas(form1)
imp1 <- mice(nhanes, formulas = form1, print = FALSE, m = 1, seed = 12199)

# same model using dot notation
form2 <- list(bmi ~ ., hyp ~ ., chl ~ .)
form2 <- name.formulas(form2)
imp2 <- mice(nhanes, formulas = form2, print = FALSE, m = 1, seed = 12199)
identical(complete(imp1), complete(imp2))

# same model using repeated multivariate imputation
form3 <- name.blocks(list(all = bmi + hyp + chl ~ .))
imp3 <- mice(nhanes, formulas = form3, print = FALSE, m = 1, seed = 12199)
cmp3 <- complete(imp3)
identical(complete(imp1), complete(imp3))

# same model using predictorMatrix
imp4 <- mice(nhanes, print = FALSE, m = 1, seed = 12199, auxiliary = TRUE)
identical(complete(imp1), complete(imp4))

# different model: multivariate imputation for chl and bmi
form5 <- list(chl + bmi ~ ., hyp ~ bmi + age)
form5 <- name.formulas(form5)
imp5 <- mice(nhanes, formulas = form5, print = FALSE, m = 1, seed = 71712)
```

`ncc` *Number of complete cases*

Description

Calculates the number of complete cases.

Usage

```
ncc(x)
```

Arguments

`x`	An `R` object. Currently supported are methods for the following classes: `mids`, `data.frame` and `matrix`. Also, `x` can be a vector.

Value

Number of elements in `x` with complete data.

Author(s)

Stef van Buuren, 2017

See Also

`nic`, `cci`

Examples

```
ncc(nhanes) # 13 complete cases
```

`nelsonaalen` *Cumulative hazard rate or Nelson-Aalen estimator*

Description

Calculates the cumulative hazard rate (Nelson-Aalen estimator)

Usage

```
nelsonaalen(data, timevar, statusvar)
```

Arguments

`data`	A data frame containing the data.
`timevar`	The name of the time variable in `data`.
`statusvar`	The name of the event variable, e.g. death in `data`.

Details

This function is useful for imputing variables that depend on survival time. White and Royston (2009) suggested using the cumulative hazard to the survival time H0(T) rather than T or log(T) as a predictor in imputation models. See section 7.1 of Van Buuren (2012) for an example.

Value

A vector with `nrow(data)` elements containing the Nelson-Aalen estimates of the cumulative hazard function.

Author(s)

Stef van Buuren, 2012

References

White, I. R., Royston, P. (2009). Imputing missing covariate values for the Cox model. *Statistics in Medicine*, *28*(15), 1982-1998.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Examples

```
require(MASS)

leuk$status <- 1 ## no censoring occurs in leuk data (MASS)
ch <- nelsonaalen(leuk, time, status)
plot(x = leuk$time, y = ch, ylab = "Cumulative hazard", xlab = "Time")

### See example on http://www.engineeredsoftware.com/lmar/pe_cum_hazard_function.htm
time <- c(43, 67, 92, 94, 149, rep(149, 7))
status <- c(rep(1, 5), rep(0, 7))
eng <- data.frame(time, status)
ch <- nelsonaalen(eng, time, status)
plot(x = time, y = ch, ylab = "Cumulative hazard", xlab = "Time")
```

`nhanes` *NHANES example - all variables numerical*

Description

A small data set with non-monotone missing values.

Format

A data frame with 25 observations on the following 4 variables.

age Age group (1=20-39, 2=40-59, 3=60+)

bmi Body mass index (kg/m**2)

hyp Hypertensive (1=no,2=yes)

chl Total serum cholesterol (mg/dL)

Details

A small data set with all numerical variables. The data set `nhanes2` is the same data set, but with `age` and `hyp` treated as factors.

Source

Schafer, J.L. (1997). *Analysis of Incomplete Multivariate Data.* London: Chapman & Hall. Table 6.14.

See Also

`nhanes2`

Examples

```
# create 5 imputed data sets
imp <- mice(nhanes)

# print the first imputed data set
complete(imp)
```

`nhanes2` *NHANES example - mixed numerical and discrete variables*

Description

A small data set with non-monotone missing values.

Format

A data frame with 25 observations on the following 4 variables.

age Age group (1=20-39, 2=40-59, 3=60+)

bmi Body mass index (kg/m**2)

hyp Hypertensive (1=no,2=yes)

chl Total serum cholesterol (mg/dL)

Details

A small data set with missing data and mixed numerical and discrete variables. The data set `nhanes` is the same data set, but with all data treated as numerical.

Source

Schafer, J.L. (1997). *Analysis of Incomplete Multivariate Data.* London: Chapman & Hall. Table 6.14.

See Also

`nhanes`

Examples

```
# create 5 imputed data sets
imp <- mice(nhanes2)

# print the first imputed data set
complete(imp)
```

---

nic *Number of incomplete cases*

Description

Calculates the number of incomplete cases.

Usage

```
nic(x)
```

Arguments

`x`	An `R` object. Currently supported are methods for the following classes: `mids`, `data.frame` and `matrix`. Also, `x` can be a vector.

Value

Number of elements in `x` with incomplete data.

Author(s)

Stef van Buuren, 2017

See Also

`ncc`, `cci`

Examples

```
nic(nhanes) # the remaining 12 rows
nic(nhanes[, c("bmi", "hyp")]) # number of cases with incomplete bmi and hyp
```

`nimp` *Number of imputations per block*

Description

Calculates the number of cells within a block for which imputation is requested.

Usage

```
nimp(where, blocks = make.blocks(where))
```

Arguments

`where`	A data frame or matrix with logicals of the same dimensions as `data` indicating where in the data the imputations should be created. The default, `where = is.na(data)`, specifies that the missing data should be imputed. The `where` argument may be used to overimpute observed data, or to skip imputations for selected missing values. Note: Imputation methods that generate imptutations outside of `mice`, like `mice.impute.panImpute()` may depend on a complete predictor space. In that case, a custom `where` matrix can not be specified.
`blocks`	List of vectors with variable names per block. List elements may be named to identify blocks. Variables within a block are imputed by a multivariate imputation method (see `method` argument). By default each variable is placed into its own block, which is effectively fully conditional specification (FCS) by univariate models (variable-by-variable imputation). Only variables whose names appear in `blocks` are imputed. The relevant columns in the `where` matrix are set to `FALSE` of variables that are not block members. A variable may appear in multiple blocks. In that case, it is effectively re-imputed each time that it is visited.

Value

A numeric vector of length `length(blocks)` containing the number of cells that need to be imputed within a block.

See Also

`mice`

Examples

```
where <- is.na(nhanes)

# standard FCS
nimp(where)

# user-defined blocks
nimp(where, blocks = name.blocks(list(c("bmi", "hyp"), "age", "chl")))
```

`norm.draw` *Draws values of beta and sigma by Bayesian linear regression*

Description

This function draws random values of beta and sigma under the Bayesian linear regression model as described in Rubin (1987, p. 167). This function can be called by user-specified imputation functions.

Usage

```
norm.draw(y, ry, x, rank.adjust = TRUE, ...)

.norm.draw(y, ry, x, rank.adjust = TRUE, ...)
```

Arguments

`y`	Incomplete data vector of length `n`
`ry`	Vector of missing data pattern (`FALSE`=missing, `TRUE`=observed)
`x`	Matrix (`n` x `p`) of complete covariates.
`rank.adjust`	Argument that specifies whether `NA`'s in the coefficients need to be set to zero. Only relevant when `ls.meth` = `"qr"` AND the predictor matrix is rank-deficient.
`...`	Other named arguments.

Value

A `list` containing components `coef` (least squares estimate), `beta` (drawn regression weights) and `sigma` (drawn value of the residual standard deviation).

Author(s)

Gerko Vink, 2018, for this version, based on earlier versions written by Stef van Buuren, Karin Groothuis-Oudshoorn, 2017

References

Rubin, D.B. (1987). *Multiple imputation for nonresponse in surveys*. New York: Wiley.

`parlmice` *Wrapper function that runs MICE in parallel*

Description

This function is included for backward compatibility. The function is superseded by `futuremice`.

Usage

```
parlmice(
  data,
  m = 5,
  seed = NA,
  cluster.seed = NA,
  n.core = NULL,
  n.imp.core = NULL,
  cl.type = "PSOCK",
  ...
)
```

Arguments

`data`	A data frame or matrix containing the incomplete data. Similar to the first argument of `mice`.
`m`	The number of desired imputated datasets. By default $m=5$ as with `mice`
`seed`	A scalar to be used as the seed value for the mice algorithm within each parallel stream. Please note that the imputations will be the same for all streams and, hence, this should be used if and only if `n.core = 1` and if it is desired to obtain the same output as under `mice`.
`cluster.seed`	A scalar to be used as the seed value. It is recommended to put the seed value here and not outside this function, as otherwise the parallel processes will be performed with separate, random seeds.
`n.core`	A scalar indicating the number of cores that should be used.
`n.imp.core`	A scalar indicating the number of imputations per core.
`cl.type`	The cluster type. Default value is `"PSOCK"`. Posix machines (linux, Mac) generally benefit from much faster cluster computation if `type` is set to `type = "FORK"`.
`...`	Named arguments that are passed down to function `mice` or `makeCluster`.

Details

This function relies on package `parallel`, which is a base package for R versions 2.14.0 and later. We have chosen to use parallel function `parLapply` to allow the use of `parlmice` on Mac, Linux and Windows systems. For the same reason, we use the Parallel Socket Cluster (PSOCK) type by default.

On systems other than Windows, it can be hugely beneficial to change the cluster type to `FORK`, as it generally results in improved memory handling. When memory issues arise on a Windows system, we advise to store the multiply imputed datasets, clean the memory by using `rm` and `gc` and make another run using the same settings.

This wrapper function combines the output of `parLapply` with function `ibind` in `mice`. A `mids` object is returned and can be used for further analyses.

Note that if a seed value is desired, the seed should be entered to this function with argument `seed`. Seed values outside the wrapper function (in an R-script or passed to `mice`) will not result to reproducible results. We refer to the manual of `parallel` for an explanation on this matter.

Value

A mids object as defined by `mids-class`

Author(s)

Gerko Vink, Rianne Schouten

References

Schouten, R. and Vink, G. (2017). parlmice: faster, paraleller, micer. https://www.gerkovink.com/parlMICE/Vignette_parlMICE.html

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`parallel`, `parLapply`, `makeCluster`, `mice`, `mids-class`

Examples

```
# 150 imputations in dataset nhanes, performed by 3 cores
## Not run:
imp1 <- parlmice(data = nhanes, n.core = 3, n.imp.core = 50)
# Making use of arguments in mice.
imp2 <- parlmice(data = nhanes, method = "norm.nob", m = 100)
imp2$method
fit <- with(imp2, lm(bmi ~ hyp))
pool(fit)

## End(Not run)
```

`pattern` *Datasets with various missing data patterns*

Description

Four simple datasets with various missing data patterns

Format

list("pattern1") Data with a univariate missing data pattern

list("pattern2") Data with a monotone missing data pattern

list("pattern3") Data with a file matching missing data pattern

list("pattern4") Data with a general missing data pattern

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Details

Van Buuren (2012) uses these four artificial datasets to illustrate various missing data patterns.

Examples

```
pattern4

data <- rbind(pattern1, pattern2, pattern3, pattern4)
mdpat <- cbind(expand.grid(rec = 8:1, pat = 1:4, var = 1:3), r = as.numeric(as.vector(is.na(data))))

types <- c("Univariate", "Monotone", "File matching", "General")
tp41 <- lattice::levelplot(r ~ var + rec | as.factor(pat),
  data = mdpat,
  as.table = TRUE, aspect = "iso",
  shrink = c(0.9),
  col.regions = mdc(1:2),
  colorkey = FALSE,
  scales = list(draw = FALSE),
  xlab = "", ylab = "",
  between = list(x = 1, y = 0),
  strip = lattice::strip.custom(
    bg = "grey95", style = 1,
    factor.levels = types
  )
)
print(tp41)

md.pattern(pattern4)
p <- md.pairs(pattern4)
p
```

```
### proportion of usable cases
p$mr / (p$mr + p$mm)

### outbound statistics
p$rm / (p$rm + p$rr)


fluxplot(pattern2)
```

`pool` *Combine estimates by pooling rules*

Description

The `pool()` function combines the estimates from `m` repeated complete data analyses. The typical sequence of steps to perform a multiple imputation analysis is:

1. Impute the missing data by the `mice()` function, resulting in a multiple imputed data set (class `mids`);
2. Fit the model of interest (scientific model) on each imputed data set by the `with()` function, resulting an object of class `mira`;
3. Pool the estimates from each model into a single set of estimates and standard errors, resulting in an object of class `mipo`;
4. Optionally, compare pooled estimates from different scientific models by the `D1()` or `D3()` functions.

A common error is to reverse steps 2 and 3, i.e., to pool the multiply-imputed data instead of the estimates. Doing so may severely bias the estimates of scientific interest and yield incorrect statistical intervals and p-values. The `pool()` function will detect this case.

Usage

```
pool(object, dfcom = NULL, rule = NULL, custom.t = NULL)

pool.syn(object, dfcom = NULL, rule = "reiter2003")
```

Arguments

`object`	An object of class `mira` (produced by `with.mids()` or `as.mira()`), or a `list` with model fits.
`dfcom`	A positive number representing the degrees of freedom in the complete-data analysis. Normally, this would be the number of independent observation minus the number of fitted parameters. The default (`dfcom = NULL`) extract this information in the following order: 1) the component `residual.df` returned by `glance()` if a `glance()` function is found, 2) the result of `df.residual(` applied to the first fitted model, and 3) as `999999`. In the last case, the warning `"Large sample assumed"` is printed. If the degrees of freedom is incorrect, specify the appropriate value manually.

`rule`	A string indicating the pooling rule. Currently supported are `"rubin1987"` (default, for missing data) and `"reiter2003"` (for synthetic data created from a complete data set).
`custom.t`	A custom character string to be parsed as a calculation rule for the total variance `t`. The custom rule can use the other calculated pooling statistics where the dimensions must come from `.data$`. The default `t` calculation would have the form `".data$ubar + (1 + 1 / .data$m) * .data$b"`. See examples for an example.

Details

The `pool()` function averages the estimates of the complete data model, computes the total variance over the repeated analyses by Rubin's rules (Rubin, 1987, p. 76), and computes the following diagnostic statistics per estimate:

1. Relative increase in variance due to nonresponse `r`;
2. Residual degrees of freedom for hypothesis testing `df`;
3. Proportion of total variance due to missingness `lambda`;
4. Fraction of missing information `fmi`.

The degrees of freedom calculation for the pooled estimates uses the Barnard-Rubin adjustment for small samples (Barnard and Rubin, 1999).

The `pool.syn()` function combines estimates by Reiter's partially synthetic data pooling rules (Reiter, 2003). This combination rule assumes that the data that is synthesised is completely observed. Pooling differs from Rubin's method in the calculation of the total variance and the degrees of freedom.

Pooling requires the following input from each fitted model:

1. the estimates of the model;
2. the standard error of each estimate;
3. the residual degrees of freedom of the model.

The `pool()` and `pool.syn()` functions rely on the `broom::tidy` and `broom::glance` for extracting these parameters.

Since `mice 3.0+`, the `broom` package takes care of filtering out the relevant parts of the complete-data analysis. It may happen that you'll see the messages like `Error: No tidy method for objects of class ...` or `Error: No glance method for objects of class ...`. The message means that your complete-data method used in `with(imp, ...)` has no `tidy` or `glance` method defined in the `broom` package.

The `broom.mixed` package contains `tidy` and `glance` methods for mixed models. If you are using a mixed model, first run `library(broom.mixed)` before calling `pool()`.

If no `tidy` or `glance` methods are defined for your analysis tabulate the `m` parameter estimates and their variance estimates (the square of the standard errors) from the `m` fitted models stored in `fit$analyses`. For each parameter, run `pool.scalar` to obtain the pooled parameters estimate, its variance, the degrees of freedom, the relative increase in variance and the fraction of missing information.

An alternative is to write your own `glance()` and `tidy()` methods and add these to `broom` according to the specifications given in `https://broom.tidymodels.org`. In versions prior to `mice 3.0` pooling required that `coef()` and `vcov()` methods were available for fitted objects. *This feature is no longer supported*. The reason is that `vcov()` methods are inconsistent across packages, leading to buggy behaviour of the `pool()` function.

Since `mice 3.13.2` function `pool()` uses the robust the standard error estimate for pooling when it can extract `robust.se` from the `tidy()` object.

Value

An object of class `mipo`, which stands for 'multiple imputation pooled outcome'. For rule `"reiter2003"` values for `lambda` and `fmi` are set to `NA`, as these statistics do not apply for data synthesised from fully observed data.

References

Barnard, J. and Rubin, D.B. (1999). Small sample degrees of freedom with multiple imputation. *Biometrika*, 86, 948-955.

Rubin, D.B. (1987). *Multiple Imputation for Nonresponse in Surveys*. New York: John Wiley and Sons.

Reiter, J.P. (2003). Inference for Partially Synthetic, Public Use Microdata Sets. *Survey Methodology*, **29**, 181-189.

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

See Also

`with.mids`, `as.mira`, `pool.scalar`, `glance`, `tidy` `https://github.com/amices/mice/issues/142`, `https://github.com/amices/mice/issues/274`

Examples

```
# impute missing data, analyse and pool using the classic MICE workflow
imp <- mice(nhanes, maxit = 2, m = 2)
fit <- with(data = imp, exp = lm(bmi ~ hyp + chl))
summary(pool(fit))

# generate fully synthetic data, analyse and pool
imp <- mice(cars,
  maxit = 2, m = 2,
  where = matrix(TRUE, nrow(cars), ncol(cars))
)
fit <- with(data = imp, exp = lm(speed ~ dist))
summary(pool.syn(fit))

# use a custom pooling rule for the total variance about the estimate
# e.g. use t = b + b/m instead of t = ubar + b + b/m
imp <- mice(nhanes, maxit = 2, m = 2)
fit <- with(data = imp, exp = lm(bmi ~ hyp + chl))
pool(fit, custom.t = ".data$b + .data$b / .data$m")
```

`pool.compare` *Compare two nested models fitted to imputed data*

Description

This function is deprecated in V3. Use `D1` or `D3` instead.

Usage

```
pool.compare(fit1, fit0, method = c("wald", "likelihood"), data = NULL)
```

Arguments

`fit1`	An object of class 'mira', produced by `with.mids()`.
`fit0`	An object of class 'mira', produced by `with.mids()`. The model in `fit0` is a nested fit0 of `fit1`.
`method`	Either `"wald"` or `"likelihood"` specifying the type of comparison. The default is `"wald"`.
`data`	No longer used.

Details

Compares two nested models after m repeated complete data analysis

The function is based on the article of Meng and Rubin (1992). The Wald-method can be found in paragraph 2.2 and the likelihood method can be found in paragraph 3. One could use the Wald method for comparison of linear models obtained with e.g. `lm` (in `with.mids()`). The likelihood method should be used in case of logistic regression models obtained with `glm()` in `with.mids()`.

The function assumes that `fit1` is the larger model, and that model `fit0` is fully contained in `fit1`. In case of `method='wald'`, the null hypothesis is tested that the extra parameters are all zero.

Value

A list containing several components. Component `call` is the call to the `pool.compare` function. Component `call11` is the call that created `fit1`. Component `call12` is the call that created the imputations. Component `call01` is the call that created `fit0`. Component `call02` is the call that created the imputations. Components `method` is the method used to compare two models: 'Wald' or 'likelihood'. Component `nmis` is the number of missing entries for each variable. Component `m` is the number of imputations. Component `qhat1` is a matrix, containing the estimated coefficients of the m repeated complete data analyses from `fit1`. Component `qhat0` is a matrix, containing the estimated coefficients of the m repeated complete data analyses from `fit0`. Component `ubar1` is the mean of the variances of `fit1`, formula (3.1.3), Rubin (1987). Component `ubar0` is the mean of the variances of `fit0`, formula (3.1.3), Rubin (1987). Component `qbar1` is the pooled estimate of `fit1`, formula (3.1.2) Rubin (1987). Component `qbar0` is the pooled estimate of `fit0`, formula (3.1.2) Rubin (1987). Component `Dm` is the test statistic. Component `rm` is the relative increase in variance due to nonresponse, formula (3.1.7), Rubin (1987). Component `df1`: df1 = under the null hypothesis it is assumed that `Dm` has an F distribution with (df1,df2) degrees of freedom. Component `df2`: df2. Component `pvalue` is the P-value of testing whether the model `fit1` is statistically different from the smaller `fit0`.

Author(s)

Karin Groothuis-Oudshoorn and Stef van Buuren, 2009

References

Li, K.H., Meng, X.L., Raghunathan, T.E. and Rubin, D. B. (1991). Significance levels from repeated p-values with multiply-imputed data. Statistica Sinica, 1, 65-92.

Meng, X.L. and Rubin, D.B. (1992). Performing likelihood ratio tests with multiple-imputed data sets. Biometrika, 79, 103-111.

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

See Also

`lm.mids`, `glm.mids`

`pool.r.squared` *Pools R^2 of m models fitted to multiply-imputed data*

Description

The function pools the coefficients of determination R^2 or the adjusted coefficients of determination (R^2_a) obtained with the `lm` modeling function. For pooling it uses the Fisher *z*-transformation.

Usage

```
pool.r.squared(object, adjusted = FALSE)
```

Arguments

`object`	An object of class 'mira' or 'mipo', produced by `lm.mids`, `with.mids`, or `pool` with `lm` as modeling function.
`adjusted`	A logical value. If adjusted=TRUE then the adjusted R^2 is calculated. The default value is FALSE.

Value

Returns a 1x4 table with components. Component `est` is the pooled R^2 estimate. Component `lo95` is the 95 \ Component `hi95` is the 95 \ Component `fmi` is the fraction of missing information due to nonresponse.

Author(s)

Karin Groothuis-Oudshoorn and Stef van Buuren, 2009

References

Harel, O (2009). The estimation of R^2 and adjusted R^2 in incomplete data sets using multiple imputation, Journal of Applied Statistics, 36:1109-1118.

Rubin, D.B. (1987). Multiple Imputation for Nonresponse in Surveys. New York: John Wiley and Sons.

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

See Also

`pool`,`pool.scalar`

Examples

```
imp <- mice(nhanes, print = FALSE, seed = 16117)
fit <- with(imp, lm(chl ~ age + hyp + bmi))

# input: mira object
pool.r.squared(fit)
pool.r.squared(fit, adjusted = TRUE)

# input: mipo object
est <- pool(fit)
pool.r.squared(est)
pool.r.squared(est, adjusted = TRUE)
```

`pool.scalar` *Multiple imputation pooling: univariate version*

Description

Pools univariate estimates of m repeated complete data analysis

Usage

```
pool.scalar(Q, U, n = Inf, k = 1, rule = c("rubin1987", "reiter2003"))

pool.scalar.syn(Q, U, n = Inf, k = 1, rule = "reiter2003")
```

Arguments

`Q`	A vector of univariate estimates of `m` repeated complete data analyses.
`U`	A vector containing the corresponding `m` variances of the univariate estimates.
`n`	A number providing the sample size. If nothing is specified, an infinite sample `n = Inf` is assumed.
`k`	A number indicating the number of parameters to be estimated. By default, `k = 1` is assumed.
`rule`	A string indicating the pooling rule. Currently supported are `"rubin1987"` (default, for missing data) and `"reiter2003"` (for synthetic data created from a complete data set).

Details

The function averages the univariate estimates of the complete data model, computes the total variance over the repeated analyses, and computes the relative increase in variance due to missing data or data synthesisation and the fraction of missing information.

Value

Returns a list with components.

`m`: Number of imputations.

`qhat`: The `m` univariate estimates of repeated complete-data analyses.

`u`: The corresponding `m` variances of the univariate estimates.

`qbar`: The pooled univariate estimate, formula (3.1.2) Rubin (1987).

`ubar`: The mean of the variances (i.e. the pooled within-imputation variance), formula (3.1.3) Rubin (1987).

`b`: The between-imputation variance, formula (3.1.4) Rubin (1987).

`t`: The total variance of the pooled estimated, formula (3.1.5) Rubin (1987).

`r`: The relative increase in variance due to nonresponse, formula (3.1.7) Rubin (1987).

`df`: The degrees of freedom for `t` reference distribution by the method of Barnard-Rubin (1999).

`fmi`: The fraction missing information due to nonresponse, formula (3.1.10) Rubin (1987). (Not defined for synthetic data.)

Author(s)

Karin Groothuis-Oudshoorn and Stef van Buuren, 2009; Thom Volker, 2021

References

Rubin, D.B. (1987). Multiple Imputation for Nonresponse in Surveys. New York: John Wiley and Sons.

Reiter, J.P. (2003). Inference for Partially Synthetic, Public Use Microdata Sets. *Survey Methodology*, **29**, 181-189.

See Also

pool

Examples

```
# missing data imputation with with manual pooling
imp <- mice(nhanes, maxit = 2, m = 2, print = FALSE, seed = 18210)
fit <- with(data = imp, lm(bmi ~ age))

# manual pooling
summary(fit$analyses[[1]])
summary(fit$analyses[[2]])
pool.scalar(Q = c(-1.5457, -1.428), U = c(0.9723^2, 1.041^2), n = 25, k = 2)

# check: automatic pooling using broom
pool(fit)

# manual pooling for synthetic data created from complete data
imp <- mice(cars,
  maxit = 2, m = 2, print = FALSE, seed = 18210,
  where = matrix(TRUE, nrow(cars), ncol(cars))
)
fit <- with(data = imp, lm(speed ~ dist))

# manual pooling: extract Q and U
summary(fit$analyses[[1]])
summary(fit$analyses[[2]])
pool.scalar.syn(Q = c(0.12182, 0.13209), U = c(0.02121^2, 0.02516^2), n = 50, k = 2)

# check: automatic pooling using broom
pool.syn(fit)
```

`pool.table` *Combines estimates from a tidy table*

Description

Combines estimates from a tidy table

Usage

```
pool.table(
  w,
  type = c("all", "minimal", "tests"),
  conf.int = TRUE,
  conf.level = 0.95,
  exponentiate = FALSE,
  dfcom = Inf,
  custom.t = NULL,
```

```
  rule = c("rubin1987", "reiter2003"),
  ...
)
```

Arguments

`w`	A `data.frame` with parameter estimates in tidy format (see details).
`type`	A string, either `"minimal"`, `"tests"` or `"all"`. Use minimal to mimick the output of `summary(pool(fit))`. The default is `"all"`.
`conf.int`	Logical indicating whether to include a confidence interval.
`conf.level`	Confidence level of the interval, used only if `conf.int = TRUE`. Number between 0 and 1.
`exponentiate`	Flag indicating whether to exponentiate the coefficient estimates and confidence intervals (typical for logistic regression).
`dfcom`	A positive number representing the degrees of freedom of the residuals in the complete-data analysis. The `dfcom` argument is used for the Barnard-Rubin adjustment. In a linear regression, `dfcom` would be equivalent to the number of independent observation minus the number of fitted parameters, but the expression becomes more complex for regularized, proportional hazards, or other semi-parametric techniques. Only used if `w` lacks a column named `"df.residual"`.
`custom.t`	A custom character string to be parsed as a calculation rule for the total variance `t`. The custom rule can use the other calculated pooling statistics. The default `t` calculation has the form `".data$ubar + (1 + 1 / .data$m) * .data$b"`.
`rule`	A string indicating the pooling rule. Currently supported are `"rubin1987"` (default, for analyses applied to multiply-imputed incomplete data) and `"reiter2003"` (for analyses applied to synthetic data created from complete data).
`...`	Arguments passed down

Details

The input data `w` is a `data.frame` with columns named:

`term`	a character or factor with the parameter names
`estimate`	a numeric vector with parameter estimates
`std.error`	a numeric vector with standard errors of `estimate`
`residual.df`	a numeric vector with the degrees of freedom

Columns 1-3 are obligatory. Column 4 is optional. Usually, all entries in column 4 are the same. The user can omit column 4, and specify argument `pool.table(..., dfcom = ...)` instead. If both are given, then column `residual.df` takes precedence. If neither are specified, then `mice` tries to calculate the residual degrees of freedom. If that fails (e.g. because there is no information on sample size), `mice` sets `dfcom = Inf`. The value `dfcom = Inf` is acceptable for large samples (n > 1000) and relatively concise parametric models.

Value

`pool.table()` returns a `data.frame` with aggregated estimates, standard errors, confidence intervals and statistical tests.

The meaning of the columns is as follows:

`term`	Parameter name
`m`	Number of multiple imputations
`estimate`	Pooled complete data estimate
`std.error`	Standard error of `estimate`
`statistic`	t-statistic = `estimate` / `std.error`
`df`	Degrees of freedom for `statistic`
`p.value`	One-sided P-value under null hypothesis
`conf.low`	Lower bound of c.i. (default 95 pct)
`conf.high`	Upper bound of c.i. (default 95 pct)
`riv`	Relative increase in variance
`fmi`	Fraction of missing information
`ubar`	Within-imputation variance of `estimate`
`b`	Between-imputation variance of `estimate`
`t`	Total variance, of `estimate`
`dfcom`	Residual degrees of freedom in complete data

Examples

```
# conventional mice workflow
imp <- mice(nhanes2, m = 2, maxit = 2, seed = 1, print = FALSE)
fit <- with(imp, lm(chl ~ age + bmi + hyp))
pld1 <- pool(fit)
pld1$pooled

# using pool.table() on tidy table
tbl <- summary(fit)[, c("term", "estimate", "std.error", "df.residual")]
tbl
pld2 <- pool.table(tbl, type = "minimal")
pld2

identical(pld1$pooled, pld2)

# conventional workflow: all numerical output
all1 <- summary(pld1, type = "all", conf.int = TRUE)
all1

# pool.table workflow: all numerical output
all2 <- pool.table(tbl)
all2

class(all1) <- "data.frame"
identical(all1, all2)
```

`popmis` *Hox pupil popularity data with missing popularity scores*

Description

Hox pupil popularity data with some missing popularity scores

Format

A data frame with 2000 rows and 7 columns:

pupil Pupil number within school

school School number

popular Pupil popularity with 848 missing entries

sex Pupil gender

texp Teacher experience (years)

const Constant intercept term

teachpop Teacher popularity

Details

The original, complete dataset was generated by Joop Hox as an example of well-behaved multilevel data set. The distributed data contains missing data in pupil popularity.

Source

Hox, J. J. (2002) *Multilevel analysis. Techniques and applications.* Mahwah, NJ: Lawrence Erlbaum.

Examples

```
popmis[1:3, ]
```

`pops` *Project on preterm and small for gestational age infants (POPS)*

Description

Subset of data from the POPS study, a national, prospective study on preterm children, including all liveborn infants <32 weeks gestational age and/or <1500 g from 1983 (n = 1338).

Format

`pops` is a data frame with 959 rows and 86 columns. `pops.pred` is the 86 by 86 binary predictor matrix used for specifying the multiple imputation model.

Details

The data set concerns of subset of 959 children that survived up to the age of 19 years.

Hille et al (2005) divided the 959 survivors into three groups: Full responders (examined at an outpatient clinic and completed the questionnaires, n = 596), postal responders (only completed the mailed questionnaires, n = 109), non-responders (did not respond to any of the mailed requests or telephone calls, or could not be traced, n = 254).

Compared to the postal and non-responders, the full response group consists of more girls, contains more Dutch children, has higher educational and social economic levels and has fewer handicaps. The responders form a highly selective subgroup in the total cohort.

Multiple imputation of this data set has been described in Hille et al (2007) and Van Buuren (2012), chapter 8.

Note

This dataset is not part of `mice`.

Source

Hille, E. T. M., Elbertse, L., Bennebroek Gravenhorst, J., Brand, R., Verloove-Vanhorick, S. P. (2005). Nonresponse bias in a follow-up study of 19-year-old adolescents born as preterm infants. Pediatrics, 116(5):662666.

Hille, E. T. M., Weisglas-Kuperus, N., Van Goudoever, J. B., Jacobusse, G. W., Ens-Dokkum, M. H., De Groot, L., Wit, J. M., Geven, W. B., Kok, J. H., De Kleine, M. J. K., Kollee, L. A. A., Mulder, A. L. M., Van Straaten, H. L. M., De Vries, L. S., Van Weissenbruch, M. M., Verloove-Vanhorick, S. P. (2007). Functional outcomes and participation in young adulthood for very preterm and very low birth weight infants: The Dutch project on preterm and small for gestational age infants at 19 years of age. Pediatrics, 120(3):587595.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Examples

```
pops <- data(pops)
```

`potthoffroy` *Potthoff-Roy data*

Description

Data from Potthoff-Roy (1964) with repeated measures on dental fissures.

Format

`tbs` is a data frame with 27 rows and 6 columns:

id Person number

sex Sex M/F

d8 Distance at age 8 years

d10 Distance at age 10 years

d12 Distance at age 12 years

d14 Distance at age 14 years

Details

This data set is the famous Potthoff-Roy data, used to demonstrate MANOVA on repeated measure data. Potthoff and Roy (1964) published classic data on a study in 16 boys and 11 girls, who at ages 8, 10, 12, and 14 had the distance (mm) from the center of the pituitary gland to the pteryomaxillary fissure measured. Changes in pituitary-pteryomaxillary distances during growth is important in orthodontic therapy. The goals of the study were to describe the distance in boys and girls as simple functions of age, and then to compare the functions for boys and girls. The data have been reanalyzed by many authors including Jennrich and Schluchter (1986), Little and Rubin (1987), Pinheiro and Bates (2000), Verbeke and Molenberghs (2000) and Molenberghs and Kenward (2007). See Chapter 9 of Van Buuren (2012) for a challenging exercise using these data.

Source

Potthoff, R. F., Roy, S. N. (1964). A generalized multivariate analysis of variance model usefully especially for growth curve problems. *Biometrika*, *51*(3), 313-326.

Little, R. J. A., Rubin, D. B. (1987). *Statistical Analysis with Missing Data.* New York: John Wiley & Sons.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Examples

```
### create missing values at age 10 as in Little and Rubin (1987)

phr <- potthoffroy
idmis <- c(3, 6, 9, 10, 13, 16, 23, 24, 27)
phr[idmis, 4] <- NA
phr

md.pattern(phr)
```

`print.mira` *Print a* `mira` *object*

Description

Print a `mira` object

Print a `mice.anova` object

Print a `summary.mice.anova` object

Usage

```
## S3 method for class 'mira'
print(x, ...)

## S3 method for class 'mice.anova'
print(x, ...)

## S3 method for class 'mice.anova.summary'
print(x, ...)
```

Arguments

`x`	An object of class `mice.anova`
`...`	Other arguments

Value

`NULL`

`NULL`

`NULL`

See Also

`mira`

`mipo`

`mipo`

`quickpred` *Quick selection of predictors from the data*

Description

Selects predictors according to simple statistics

Usage

```
quickpred(
  data,
  mincor = 0.1,
  minpuc = 0,
  include = "",
  exclude = "",
  method = "pearson"
)
```

Arguments

`data`	Matrix or data frame with incomplete data.
`mincor`	A scalar, numeric vector (of size `ncol(data)`) or numeric matrix (square, of size `ncol(data)` specifying the minimum threshold(s) against which the absolute correlation in the data is compared.
`minpuc`	A scalar, vector (of size `ncol(data)`) or matrix (square, of size `ncol(data)` specifying the minimum threshold(s) for the proportion of usable cases.
`include`	A string or a vector of strings containing one or more variable names from `names(data)`. Variables specified are always included as a predictor.
`exclude`	A string or a vector of strings containing one or more variable names from `names(data)`. Variables specified are always excluded as a predictor.
`method`	A string specifying the type of correlation. Use `'pearson'` (default), `'kendall'` or `'spearman'`. Can be abbreviated.

Details

This function creates a predictor matrix using the variable selection procedure described in Van Buuren et al.~(1999, p.~687–688). The function is designed to aid in setting up a good imputation model for data with many variables.

Basic workings: The procedure calculates for each variable pair (i.e. target-predictor pair) two correlations using all available cases per pair. The first correlation uses the values of the target and the predictor directly. The second correlation uses the (binary) response indicator of the target and the values of the predictor. If the largest (in absolute value) of these correlations exceeds `mincor`, the predictor will be added to the imputation set. The default value for `mincor` is 0.1.

In addition, the procedure eliminates predictors whose proportion of usable cases fails to meet the minimum specified by `minpuc`. The default value is 0, so predictors are retained even if they have no usable case.

Finally, the procedure includes any predictors named in the `include` argument (which is useful for background variables like age and sex) and eliminates any predictor named in the `exclude` argument. If a variable is listed in both `include` and `exclude` arguments, the `include` argument takes precedence.

Advanced topic: `mincor` and `minpuc` are typically specified as scalars, but vectors and squares matrices of appropriate size will also work. Each element of the vector corresponds to a row of the predictor matrix, so the procedure can effectively differentiate between different target variables. Setting a high values for can be useful for auxiliary, less important, variables. The set of predictor for those variables can remain relatively small. Using a square matrix extends the idea to the columns, so that one can also apply cellwise thresholds.

Value

A square binary matrix of size `ncol(data)`.

Note

`quickpred()` uses `data.matrix` to convert factors to numbers through their internal codes. Especially for unordered factors the resulting quantification may not make sense.

Author(s)

Stef van Buuren, Aug 2009

References

van Buuren, S., Boshuizen, H.C., Knook, D.L. (1999) Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine*, **18**, 681–694.

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

See Also

`mice`, `mids`

Examples

```
# default: include all predictors with absolute correlation over 0.1
quickpred(nhanes)

# all predictors with absolute correlation over 0.4
quickpred(nhanes, mincor = 0.4)

# include age and bmi, exclude chl
quickpred(nhanes, mincor = 0.4, inc = c("age", "bmi"), exc = "chl")

# only include predictors with at least 30% usable cases
quickpred(nhanes, minpuc = 0.3)

# use low threshold for bmi, and high thresholds for hyp and chl
```

```
pred <- quickpred(nhanes, mincor = c(0, 0.1, 0.5, 0.5))
pred

# use it directly from mice
imp <- mice(nhanes, pred = quickpred(nhanes, minpuc = 0.25, include = "age"))
```

`selfreport` *Self-reported and measured BMI*

Description

Dataset containing height and weight data (measured, self-reported) from two studies.

Format

A data frame with 2060 rows and 15 variables:

src Study, either `krul` or `mgg` (factor)

id Person identification number

pop Population, all `NL` (factor)

age Age of respondent in years

sex Sex of respondent (factor)

hm Height measured (cm)

wm Weight measured (kg)

hr Height reported (cm)

wr Weight reported (kg)

prg Pregnancy (factor), all `Not pregnant`

edu Educational level (factor)

etn Ethnicity (factor)

web Obtained through web survey (factor)

bm BMI measured (kg/m2)

br BMI reported (kg/m2)

Details

This dataset combines two datasets: `krul` data (Krul, 2010) (1257 persons) and the `mgg` data (Van Keulen 2011; Van der Klauw 2011) (803 persons). The `krul` dataset contains height and weight (both measures and self-reported) from 1257 Dutch adults, whereas the `mgg` dataset contains self-reported height and weight for 803 Dutch adults. Section 7.3 in Van Buuren (2012) shows how the missing measured data can be imputed in the `mgg` data, so corrected prevalence estimates can be calculated.

Source

Krul, A., Daanen, H. A. M., Choi, H. (2010). Self-reported and measured weight, height and body mass index (BMI) in Italy, The Netherlands and North America. *European Journal of Public Health*, *21*(4), 414-419.

Van Keulen, H.M.,, Chorus, A.M.J., Verheijden, M.W. (2011). *Monitor Convenant Gezond Gewicht Nulmeting (determinanten van) beweeg- en eetgedrag van kinderen (4-11 jaar), jongeren (12-17 jaar) en volwassenen (18+ jaar).* TNO/LS 2011.016. Leiden: TNO.

Van der Klauw, M., Van Keulen, H.M., Verheijden, M.W. (2011). *Monitor Convenant Gezond Gewicht Beweeg- en eetgedrag van kinderen (4-11 jaar), jongeren (12-17 jaar) en volwassenen (18+ jaar) in 2010 en 2011.* TNO/LS 2011.055. Leiden: TNO. (in Dutch)

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Examples

```
md.pattern(selfreport[, c("age", "sex", "hm", "hr", "wm", "wr")])

### FIMD Section 7.3.5 Application

bmi <- function(h, w) {
  return(w / (h / 100)^2)
}
init <- mice(selfreport, maxit = 0)
meth <- init$meth
meth["bm"] <- "~bmi(hm,wm)"
pred <- init$pred
pred[, c("src", "id", "web", "bm", "br")] <- 0
imp <- mice(selfreport, pred = pred, meth = meth, seed = 66573, maxit = 2, m = 1)
## imp <- mice(selfreport, pred=pred, meth=meth, seed=66573, maxit=20, m=10)

### Like FIMD Figure 7.6

cd <- complete(imp, 1)
xy <- xy.coords(cd$bm, cd$br - cd$bm)
plot(xy,
  col = mdc(2), xlab = "Measured BMI", ylab = "Reported - Measured BMI",
  xlim = c(17, 45), ylim = c(-5, 5), type = "n", lwd = 0.7
)
polygon(x = c(30, 20, 30), y = c(0, 10, 10), col = "grey95", border = NA)
polygon(x = c(30, 40, 30), y = c(0, -10, -10), col = "grey95", border = NA)
abline(0, 0, lty = 2, lwd = 0.7)

idx <- cd$src == "krul"
xyc <- xy
xyc$x <- xy$x[idx]
xyc$y <- xy$y[idx]
xys <- xy
xys$x <- xy$x[!idx]
xys$y <- xy$y[!idx]
points(xyc, col = mdc(1), cex = 0.7)
```

```
points(xys, col = mdc(2), cex = 0.7)
lines(lowess(xyc), col = mdc(4), lwd = 2)
lines(lowess(xys), col = mdc(5), lwd = 2)
text(1:4, x = c(40, 28, 20, 32), y = c(4, 4, -4, -4), cex = 3)
box(lwd = 1)
```

`squeeze` *Squeeze the imputed values to be within specified boundaries.*

Description

This function replaces any values in `x` that are lower than `bounds[1]` by `bounds[1]`, and replaces any values higher than `bounds[2]` by `bounds[2]`.

Usage

```
squeeze(x, bounds = c(min(x[r]), max(x[r])), r = rep.int(TRUE, length(x)))
```

Arguments

`x`	A numerical vector with values
`bounds`	A numerical vector of length 2 containing the lower and upper bounds. By default, the bounds are to the minimum and maximum values in `x`.
`r`	A logical vector of length `length(x)` that is used to select a subset in `x` before calculating automatic bounds.

Value

A vector of length `length(x)`.

Author(s)

Stef van Buuren, 2011.

`stripplot.mids` *Stripplot of observed and imputed data*

Description

Plotting methods for imputed data using **lattice**. `stripplot` produces one-dimensional scatterplots. The function automatically separates the observed and imputed data. The functions extend the usual features of **lattice**.

Usage

```
## S3 method for class 'mids'
stripplot(
  x,
  data,
  na.groups = NULL,
  groups = NULL,
  as.table = TRUE,
  theme = mice.theme(),
  allow.multiple = TRUE,
  outer = TRUE,
  drop.unused.levels = lattice::lattice.getOption("drop.unused.levels"),
  panel = lattice::lattice.getOption("panel.stripplot"),
  default.prepanel = lattice::lattice.getOption("prepanel.default.stripplot"),
  jitter.data = TRUE,
  horizontal = FALSE,
  ...,
  subscripts = TRUE,
  subset = TRUE
)
```

Arguments

`x`	A `mids` object, typically created by `mice()` or `mice.mids()`.
`data`	Formula that selects the data to be plotted. This argument follows the **lattice** rules for *formulas*, describing the primary variables (used for the per-panel display) and the optional conditioning variables (which define the subsets plotted in different panels) to be used in the plot. The formula is evaluated on the complete data set in the `long` form. Legal variable names for the formula include `names(x$data)` plus the two administrative factors `.imp` and `.id`. **Extended formula interface:** The primary variable terms (both the LHS `y` and RHS `x`) may consist of multiple terms separated by a '+' sign, e.g., `y1 + y2 ~ x | a * b`. This formula would be taken to mean that the user wants to plot both `y1 ~ x | a * b` and `y2 ~ x | a * b`, but with the `y1 ~ x` and `y2 ~ x` in *separate panels*. This behavior differs from standard **lattice**. *Only combine terms of the same type*, i.e. only factors or only numerical variables. Mixing numerical and categorical data occasionally produces odds labeling of vertical axis. For convenience, in `stripplot()` and `bwplot` the formula `y~.imp` may be abbreviated as `y`. This applies only to a single `y`, and does not (yet) work for `y1+y2~.imp`.
`na.groups`	An expression evaluating to a logical vector indicating which two groups are distinguished (e.g. using different colors) in the display. The environment in which this expression is evaluated in the response indicator `is.na(x$data)`. The default `na.group = NULL` contrasts the observed and missing data in the LHS `y` variable of the display, i.e. groups created by `is.na(y)`. The expression `y` creates the groups according to `is.na(y)`. The expression `y1 & y2` creates

	groups by `is.na(y1) & is.na(y2)`, and `y1 | y2` creates groups as `is.na(y1) | is.na(y2)`, and so on.
`groups`	This is the usual `groups` arguments in **lattice**. It differs from `na.groups` because it evaluates in the completed data `data.frame(complete(x, "long", inc=TRUE))` (as usual), whereas `na.groups` evaluates in the response indicator. See `xyplot` for more details. When both `na.groups` and `groups` are specified, `na.groups` takes precedence, and `groups` is ignored.
`as.table`	See `xyplot`.
`theme`	A named list containing the graphical parameters. The default function `mice.theme` produces a short list of default colors, line width, and so on. The extensive list may be obtained from `trellis.par.get()`. Global graphical parameters like `col` or `cex` in high-level calls are still honored, so first experiment with the global parameters. Many setting consists of a pair. For example, `mice.theme` defines two symbol colors. The first is for the observed data, the second for the imputed data. The theme settings only exist during the call, and do not affect the trellis graphical parameters.
`allow.multiple`	See `xyplot`.
`outer`	See `xyplot`.
`drop.unused.levels`	See `xyplot`.
`panel`	See `xyplot`.
`default.prepanel`	See `xyplot`.
`jitter.data`	See `panel.xyplot`.
`horizontal`	See `xyplot`.
`...`	Further arguments, usually not directly processed by the high-level functions documented here, but instead passed on to other functions.
`subscripts`	See `xyplot`.
`subset`	See `xyplot`.

Details

The argument `na.groups` may be used to specify (combinations of) missingness in any of the variables. The argument `groups` can be used to specify groups based on the variable values themselves. Only one of both may be active at the same time. When both are specified, `na.groups` takes precedence over `groups`.

Use the `subset` and `na.groups` together to plots parts of the data. For example, select the first imputed data set by by `subset=.imp==1`.

Graphical parameters like `col`, `pch` and `cex` can be specified in the arguments list to alter the plotting symbols. If `length(col)==2`, the color specification to define the observed and missing groups. `col[1]` is the color of the 'observed' data, `col[2]` is the color of the missing or imputed data. A convenient color choice is `col=mdc(1:2)`, a transparent blue color for the observed data, and a transparent red color for the imputed data. A good choice is `col=mdc(1:2), pch=20, cex=1.5`. These choices can be set for the duration of the session by running `mice.theme()`.

Value

The high-level functions documented here, as well as other high-level Lattice functions, return an object of class `"trellis"`. The `update.trellis` method can be used to subsequently update components of the object, and the `print.trellis` method (usually called by default) will plot it on an appropriate plotting device.

Note

The first two arguments (`x` and `data`) are reversed compared to the standard Trellis syntax implemented in **lattice**. This reversal was necessary in order to benefit from automatic method dispatch.

In **mice** the argument `x` is always a `mids` object, whereas in **lattice** the argument `x` is always a formula.

In **mice** the argument `data` is always a formula object, whereas in **lattice** the argument `data` is usually a data frame.

All other arguments have identical interpretation.

Author(s)

Stef van Buuren

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer.

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

Examples

```
imp <- mice(boys, maxit = 1)

### stripplot, all numerical variables
## Not run:
stripplot(imp)

## End(Not run)

### same, but with improved display
## Not run:
stripplot(imp, col = c("grey", mdc(2)), pch = c(1, 20))

## End(Not run)

### distribution per imputation of height, weight and bmi
### labeled by their own missingness
## Not run:
stripplot(imp, hgt + wgt + bmi ~ .imp,
  cex = c(2, 4), pch = c(1, 20), jitter = FALSE,
  layout = c(3, 1)
)
```

```
## End(Not run)

### same, but labeled with the missingness of wgt (just four cases)
## Not run:
stripplot(imp, hgt + wgt + bmi ~ .imp,
  na = wgt, cex = c(2, 4), pch = c(1, 20), jitter = FALSE,
  layout = c(3, 1)
)

## End(Not run)

### distribution of age and height, labeled by missingness in height
### most height values are missing for those around
### the age of two years
### some additional missings occur in region WEST
## Not run:
stripplot(imp, age + hgt ~ .imp | reg, hgt,
  col = c(grDevices::hcl(0, 0, 40, 0.2), mdc(2)), pch = c(1, 20)
)

## End(Not run)

### heavily jitted relation between two categorical variables
### labeled by missingness of gen
### aggregated over all imputed data sets
## Not run:
stripplot(imp, gen ~ phb, factor = 2, cex = c(8, 1), hor = TRUE)

## End(Not run)

### circle fun
stripplot(imp, gen ~ .imp,
  na = wgt, factor = 2, cex = c(8.6),
  hor = FALSE, outer = TRUE, scales = "free", pch = c(1, 19)
)
```

`summary.mira` *Summary of a* `mira` *object*

Description

Summary of a `mira` object

Print a `mice.anova` object

Usage

```
## S3 method for class 'mira'
summary(object, type = c("tidy", "glance", "summary"), dfcom = NULL, ...)
```

```
## S3 method for class 'mice.anova'
summary(object, ...)
```

Arguments

`object`	A `mira` object
`type`	A length-1 character vector indicating the type of summary. There are three choices: `type = "tidy"` return the parameters estimates of each analyses as a data frame. `type = "glance"` return the fit statistics of each analysis as a data frame. `type = "summary"` returns a list of length `m` with the analysis results. The default is `"tidy"`.
`dfcom`	Manually supplied degrees of freedom. For internal use by `pool()`.
`...`	Other parameters passed down to `print()` and `summary()`

Value

`NULL`

`NULL`

See Also

`mira`

`mipo`

`supports.transparent` *Supports semi-transparent foreground colors?*

Description

This function is used by `mdc()` to find out whether the current device supports semi-transparent foreground colors.

Usage

```
supports.transparent()
```

Details

The function calls the function `dev.capabilities()` from the package `grDevices`. The function return `FALSE` if the status of the current device is unknown.

Value

`TRUE` or `FALSE`

See Also

`mdc` `dev.capabilities`

Examples

```
supports.transparent()
```

`tbc` *Terneuzen birth cohort*

Description

Data of subset of the Terneuzen Birth Cohort data on child growth.

Format

`tbs` is a data frame with 3951 rows and 11 columns:

id Person number

occ Occasion number

nocc Number of occasions

first Is this the first record for this person? (TRUE/FALSE)

typ Type of data (all observed)

age Age (years)

sex Sex 1=M, 2=F

hgt.z Height Z-score

wgt.z Weight Z-score

bmi.z BMI Z-score

ao Adult overweight (0=no, 1=yes)

`tbc.target` is a data frame with 2612 rows and 3 columns:

id Person number

ao Adult overweight (0=no, 1=yes)

bmi.z.jv BMI Z-score as young adult (18-29 years)

Details

This `tbc` data set is a random subset of persons from a much larger collection of data from the Terneuzen Birth Cohort. The total cohort comprises of 2604 unique persons, whereas the subset in `tbc` covers 306 persons. The `tbc.target` is an auxiliary data set containing two outcomes at adult age. For more details, see De Kroon et al (2008, 2010, 2011). The imputation methodology is explained in Chapter 9 of Van Buuren (2012).

Source

De Kroon, M. L. A., Renders, C. M., Kuipers, E. C., van Wouwe, J. P., van Buuren, S., de Jonge, G. A., Hirasing, R. A. (2008). Identifying metabolic syndrome without blood tests in young adults - The Terneuzen birth cohort. *European Journal of Public Health*, *18*(6), 656-660.

De Kroon, M. L. A., Renders, C. M., Van Wouwe, J. P., Van Buuren, S., Hirasing, R. A. (2010). The Terneuzen birth cohort: BMI changes between 2 and 6 years correlate strongest with adult overweight. *PLoS ONE*, *5*(2), e9155.

De Kroon, M. L. A. (2011). *The Terneuzen Birth Cohort. Detection and Prevention of Overweight and Cardiometabolic Risk from Infancy Onward.* Dissertation, Vrije Universiteit, Amsterdam. `https://research.vu.nl/en/publications/the-terneuzen-birth-cohort-detection-and-prevention-of-overw`

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Examples

```
data <- tbc
md.pattern(data)
```

`toenail` *Toenail data*

Description

The toenail data come from a Multicenter study comparing two oral treatments for toenail infection. Patients were evaluated for the degree of separation of the nail. Patients were randomized into two treatments and were followed over seven visits - four in the first year and yearly thereafter. The patients have not been treated prior to the first visit so this should be regarded as the baseline.

Format

A data frame with 1908 observations on the following 5 variables:

`ID` a numeric vector giving the ID of patient

`outcome` a numeric vector giving the response (0=none or mild seperation, 1=moderate or severe)

`treatment` a numeric vector giving the treatment group

`month` a numeric vector giving the time of the visit (not exactly monthly intervals hence not round numbers)

`visit` a numeric vector giving the number of the visit

Details

This dataset was copied from the `DPpackage`, which is scheduled to be discontinued from CRAN in August 2019.

Source

De Backer, M., De Vroey, C., Lesaffre, E., Scheys, I., and De Keyser, P. (1998). Twelve weeks of continuous oral therapy for toenail onychomycosis caused by dermatophytes: A double-blind comparative trial of terbinafine 250 mg/day versus itraconazole 200 mg/day. Journal of the American Academy of Dermatology, 38, 57-63.

References

Lesaffre, E. and Spiessens, B. (2001). On the effect of the number of quadrature points in a logistic random-effects model: An example. Journal of the Royal Statistical Society, Series C, 50, 325-335.

G. Fitzmaurice, N. Laird and J. Ware (2004) Applied Longitudinal Analysis, Wiley and Sons, New York, USA.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`toenail2`

`toenail2` *Toenail data*

Description

The toenail data come from a Multicenter study comparing two oral treatments for toenail infection. Patients were evaluated for the degree of separation of the nail. Patients were randomized into two treatments and were followed over seven visits - four in the first year and yearly thereafter. The patients have not been treated prior to the first visit so this should be regarded as the baseline.

Format

A data frame with 1908 observations on the following 5 variables:

`patientID` a numeric vector giving the ID of patient

`outcome` a factor with 2 levels giving the response

`treatment` a factor with 2 levels giving the treatment group

`time` a numeric vector giving the time of the visit (not exactly monthly intervals hence not round numbers)

`visit` an integer giving the number of the visit

Details

Apart from formatting, this dataset is identical to `toenail`. The formatting is taken identical to `data("toenail", package = "HSAUR3")`.

Source

De Backer, M., De Vroey, C., Lesaffre, E., Scheys, I., and De Keyser, P. (1998). Twelve weeks of continuous oral therapy for toenail onychomycosis caused by dermatophytes: A double-blind comparative trial of terbinafine 250 mg/day versus itraconazole 200 mg/day. Journal of the American Academy of Dermatology, 38, 57-63.

References

Lesaffre, E. and Spiessens, B. (2001). On the effect of the number of quadrature points in a logistic random-effects model: An example. Journal of the Royal Statistical Society, Series C, 50, 325-335.

G. Fitzmaurice, N. Laird and J. Ware (2004) Applied Longitudinal Analysis, Wiley and Sons, New York, USA.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

See Also

`toenail`

`version` *Echoes the package version number*

Description

Echoes the package version number

Usage

```
version(pkg = "mice")
```

Arguments

`pkg`	A character vector with the package name.

Value

A character vector containing the package name, version number and installed directory.

Author(s)

Stef van Buuren, Oct 2010

Examples

```
version()
version("base")
```

`walking` *Walking disability data*

Description

Two items YA and YB measuring walking disability in samples A, B and E.

Format

A data frame with 890 rows on the following 5 variables:

sex Sex of respondent (factor)

age Age of respondent

YA Item administered in samples A and E (factor)

YB Item administered in samples B and E (factor)

src Source: Sample A, B or E (factor)

Details

Example dataset to demonstrate imputation of two items (YA and YB). Item YA is administered to sample A and sample E, item YB is administered to sample B and sample E, so sample E acts as a bridge study. Imputation using a bridge study is better than simple equating or than imputation under independence.

Item YA corresponds to the HAQ8 item, and item YB corresponds to the GAR9 items from Van Buuren et al (2005). Sample E (as well as sample B) is the Euridiss study (n=292), sample A is the ERGOPLUS study (n=306).

See Van Buuren (2018) section 9.4 for more details on the imputation methodology.

References

van Buuren, S., Eyres, S., Tennant, A., Hopman-Rock, M. (2005). Improving comparability of existing data by Response Conversion. *Journal of Official Statistics*, **21**(1), 53-72.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data. Second Edition.* Chapman & Hall/CRC. Boca Raton, FL.

Examples

```
md.pattern(walking)

micemill <- function(n) {
  for (i in 1:n) {
    imp <<- mice.mids(imp) # global assignment
    cors <- with(imp, cor(as.numeric(YA),
      as.numeric(YB),
      method = "kendall"
    ))
```

```
    tau <<- rbind(tau, unlist(cors$analyses)) # global assignment
  }
}

plotit <- function() {
  matplot(
    x = 1:nrow(tau), y = tau,
    ylab = expression(paste("Kendall's ", tau)),
    xlab = "Iteration", type = "l", lwd = 1,
    lty = 1:10, col = "black"
  )
}

tau <- NULL
imp <- mice(walking, max = 0, m = 10, seed = 92786)
pred <- imp$pred
pred[, c("src", "age", "sex")] <- 0
imp <- mice(walking, max = 0, m = 3, seed = 92786, pred = pred)
micemill(5)
plotit()

### to get figure 9.8 van Buuren (2018) use m=10 and micemill(20)
```

`windspeed` *Subset of Irish wind speed data*

Description

Subset of Irish wind speed data

Format

A data frame with 433 rows and 6 columns containing the daily average wind speeds within the period 1961-1978 at meteorological stations in the Republic of Ireland. The data are a random sample from a larger data set.

RochePt Roche Point

Rosslare Rosslare

Shannon Shannon

Dublin Dublin

Clones Clones

MalinHead Malin Head

Details

The original data set is much larger and was analyzed in detail by Haslett and Raftery (1989). Van Buuren et al (2006) used this subset to investigate the influence of extreme MAR mechanisms on the quality of imputation.

References

Haslett, J. and Raftery, A. E. (1989). *Space-time Modeling with Long-memory Dependence: Assessing Ireland's Wind Power Resource (with Discussion).* Applied Statistics 38, 1-50. `https://lib.stat.cmu.edu/datasets/wind.desc` and `https://lib.stat.cmu.edu/datasets/wind.data`

van Buuren, S., Brand, J.P.L., Groothuis-Oudshoorn C.G.M., Rubin, D.B. (2006) Fully conditional specification in multivariate imputation. *Journal of Statistical Computation and Simulation*, **76**, 12, 1049–1064.

Examples

```
windspeed[1:3, ]
```

`with.mids` *Evaluate an expression in multiple imputed datasets*

Description

Performs a computation of each of imputed datasets in data.

Usage

```
## S3 method for class 'mids'
with(data, expr, ...)
```

Arguments

`data`	An object of type `mids`, which stands for 'multiply imputed data set', typically created by a call to function `mice()`.
`expr`	An expression to evaluate for each imputed data set. Formula's containing a dot (notation for "all other variables") do not work.
`...`	Not used

Value

An object of S3 class `mira`

Note

Version 3.11.10 changed to tidy evaluation on a quosure. This change should not affect any code that worked on previous versions. It turned out that the latter statement was not true (#292). Version 3.12.2 reverts to the old `with()` function.

Author(s)

Karin Oudshoorn, Stef van Buuren 2009, 2012, 2020

References

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

See Also

`mids`, `mira`, `pool`, `D1`, `D3`, `pool.r.squared`

Examples

```
imp <- mice(nhanes2, m = 2, print = FALSE, seed = 14221)

# descriptive statistics
getfit(with(imp, table(hyp, age)))

# model fitting and testing
fit1 <- with(imp, lm(bmi ~ age + hyp + chl))
fit2 <- with(imp, glm(hyp ~ age + chl, family = binomial))
fit3 <- with(imp, anova(lm(bmi ~ age + chl)))
```

`xyplot.mads` *Scatterplot of amputed and non-amputed data against weighted sum scores*

Description

Plotting method to investigate relation between amputed data and the weighted sum scores. Based on **lattice**, `xyplot` produces scatterplots. The function plots the variables against the weighted sum scores. The function automatically separates the amputed and non-amputed data to see the relation between the amputation and the weighted sum scores.

Usage

```
## S3 method for class 'mads'
xyplot(
  x,
  data,
  which.pat = NULL,
  standardized = TRUE,
  layout = NULL,
  colors = mdc(1:2),
  ...
)
```

Arguments

`x`	A `mads` object, typically created by `ampute`.
`data`	A string or vector of variable names that needs to be plotted. As a default, all variables will be plotted.
`which.pat`	A scalar or vector indicating which patterns need to be plotted. As a default, all patterns are plotted.
`standardized`	Logical. Whether the scatterplots need to be created from standardized data or not. Default is TRUE.
`layout`	A vector of two values indicating how the scatterplots of one pattern should be divided over the plot. For example, `c(2, 3)` indicates that the scatterplots of six variables need to be placed on 3 rows and 2 columns. There are several defaults for different #variables. Note that for more than 9 variables, multiple plots will be created automatically.
`colors`	A vector of two RGB values defining the colors of the non-amputed and amputed data respectively. RGB values can be obtained with `hcl`.
`...`	Not used, but for consistency with generic

Value

A list containing the scatterplots. Note that a new pattern will always be shown in a new plot.

Note

The `mads` object contains all the information you need to make any desired plots. Check `mads` or the vignette *Multivariate Amputation using Ampute* to understand the contents of class object `mads`.

Author(s)

Rianne Schouten, 2016

See Also

`ampute`, `mads`

`xyplot.mids` *Scatterplot of observed and imputed data*

Description

Plotting methods for imputed data using **lattice**. `xyplot()` produces a conditional scatterplots. The function automatically separates the observed (blue) and imputed (red) data. The function extends the usual features of **lattice**.

Usage

```
## S3 method for class 'mids'
xyplot(
  x,
  data,
  na.groups = NULL,
  groups = NULL,
  as.table = TRUE,
  theme = mice.theme(),
  allow.multiple = TRUE,
  outer = TRUE,
  drop.unused.levels = lattice::lattice.getOption("drop.unused.levels"),
  ...,
  subscripts = TRUE,
  subset = TRUE
)
```

Arguments

`x`	A `mids` object, typically created by `mice()` or `mice.mids()`.
`data`	Formula that selects the data to be plotted. This argument follows the **lattice** rules for *formulas*, describing the primary variables (used for the per-panel display) and the optional conditioning variables (which define the subsets plotted in different panels) to be used in the plot. The formula is evaluated on the complete data set in the `long` form. Legal variable names for the formula include `names(x$data)` plus the two administrative factors `.imp` and `.id`. **Extended formula interface:** The primary variable terms (both the LHS `y` and RHS `x`) may consist of multiple terms separated by a '+' sign, e.g., `y1 + y2 ~ x | a * b`. This formula would be taken to mean that the user wants to plot both `y1 ~ x | a * b` and `y2 ~ x | a * b`, but with the `y1 ~ x` and `y2 ~ x` in *separate panels*. This behavior differs from standard **lattice**. *Only combine terms of the same type*, i.e. only factors or only numerical variables. Mixing numerical and categorical data occasionally produces odds labeling of vertical axis.
`na.groups`	An expression evaluating to a logical vector indicating which two groups are distinguished (e.g. using different colors) in the display. The environment in which this expression is evaluated in the response indicator `is.na(x$data)`. The default `na.group = NULL` contrasts the observed and missing data in the LHS `y` variable of the display, i.e. groups created by `is.na(y)`. The expression `y` creates the groups according to `is.na(y)`. The expression `y1 & y2` creates groups by `is.na(y1) & is.na(y2)`, and `y1 | y2` creates groups as `is.na(y1) | is.na(y2)`, and so on.
`groups`	This is the usual `groups` arguments in **lattice**. It differs from `na.groups` because it evaluates in the completed data `data.frame(complete(x, "long", inc=TRUE))` (as usual), whereas `na.groups` evaluates in the response indicator. See `xyplot` for more details. When both `na.groups` and `groups` are specified, `na.groups` takes precedence, and `groups` is ignored.

`as.table`	See `xyplot`.
`theme`	A named list containing the graphical parameters. The default function `mice.theme` produces a short list of default colors, line width, and so on. The extensive list may be obtained from `trellis.par.get()`. Global graphical parameters like `col` or `cex` in high-level calls are still honored, so first experiment with the global parameters. Many setting consists of a pair. For example, `mice.theme` defines two symbol colors. The first is for the observed data, the second for the imputed data. The theme settings only exist during the call, and do not affect the trellis graphical parameters.
`allow.multiple`	See `xyplot`.
`outer`	See `xyplot`.
`drop.unused.levels`	See `xyplot`.
`...`	Further arguments, usually not directly processed by the high-level functions documented here, but instead passed on to other functions.
`subscripts`	See `xyplot`.
`subset`	See `xyplot`.

Details

The argument `na.groups` may be used to specify (combinations of) missingness in any of the variables. The argument `groups` can be used to specify groups based on the variable values themselves. Only one of both may be active at the same time. When both are specified, `na.groups` takes precedence over `groups`.

Use the `subset` and `na.groups` together to plots parts of the data. For example, select the first imputed data set by by `subset=.imp==1`.

Graphical parameters like `col`, `pch` and `cex` can be specified in the arguments list to alter the plotting symbols. If `length(col)==2`, the color specification to define the observed and missing groups. `col[1]` is the color of the 'observed' data, `col[2]` is the color of the missing or imputed data. A convenient color choice is `col=mdc(1:2)`, a transparent blue color for the observed data, and a transparent red color for the imputed data. A good choice is `col=mdc(1:2), pch=20, cex=1.5`. These choices can be set for the duration of the session by running `mice.theme()`.

Value

The high-level functions documented here, as well as other high-level Lattice functions, return an object of class `"trellis"`. The `update.trellis` method can be used to subsequently update components of the object, and the `print.trellis` method (usually called by default) will plot it on an appropriate plotting device.

Note

The first two arguments (`x` and `data`) are reversed compared to the standard Trellis syntax implemented in **lattice**. This reversal was necessary in order to benefit from automatic method dispatch.

In **mice** the argument `x` is always a `mids` object, whereas in **lattice** the argument `x` is always a formula.

In **mice** the argument `data` is always a formula object, whereas in **lattice** the argument `data` is usually a data frame.

All other arguments have identical interpretation.

Author(s)

Stef van Buuren

References

Sarkar, Deepayan (2008) *Lattice: Multivariate Data Visualization with R*, Springer.

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

Examples

```
imp <- mice(boys, maxit = 1)

# xyplot: scatterplot by imputation number
# observe the erroneous outlying imputed values
# (caused by imputing hgt from bmi)
xyplot(imp, hgt ~ age | .imp, pch = c(1, 20), cex = c(1, 1.5))

# same, but label with missingness of wgt (four cases)
xyplot(imp, hgt ~ age | .imp, na.group = wgt, pch = c(1, 20), cex = c(1, 1.5))
```

Index