Package ‘mice’

July 10, 2019

Type Package
Version 3.6.0
Title Multivariate Imputation by Chained Equations
Date 2019-07-09
Maintainer Stef van Buuren <stef.vanbuuren@tno.nl>
Depends methods, R (>= 2.10.0), lattice
Imports broom, dplyr, grDevices, graphics, MASS, mitml, nnet,
parallel, Rcpp, rlang, rpart, splines, stats, survival, utils
Suggests AGD, CALIBERrfimpute, gamlss, lme4, mitools, nlme, pan,
randomForest, Zelig. BSDA, knitr, rmarkdown, testthat, HSAUR3,
micemd, miceadds, tidyr
LinkingTo Rcpp
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
License GPL-2 | GPL-3
LazyLoad yes
LazyData yes
URL http://stefvanbuuren.github.io/mice/,
http://www.stefvanbuuren.name, 
http://www.stefvanbuuren.name/fimd/
BugReports  https://github.com/stefvanbuuren/mice/issues
RoxygenNote  6.1.1
NeedsCompilation  yes
Author  Stef van Buuren [aut, cre],
        Karin Groothuis-Oudshoorn [aut],
        Alexander Robitzsch [ctb],
        Gerko Vink [ctb],
        Lisa Doove [ctb],
        Shahab Jolani [ctb],
        Rianne Schouten [ctb],
        Philipp Gaffert [ctb],
        Florian Meinfeldt [ctb],
        Bernie Gray [ctb]
Repository  CRAN
Date/Publication  2019-07-10 08:00:03 UTC

R topics documented:

- pmm.match ................................................. 5
- ampute .................................................... 6
- anova.mira ............................................... 9
- appendbreak ............................................. 10
- as.mids ................................................. 11
- as.mira .................................................. 12
- as.mitml.result ........................................ 13
- boys ....................................................... 14
- brandsma ................................................ 15
- bwplot.mads ........................................... 17
- bwplot.mids ............................................ 18
- cbind.mids .............................................. 21
- cc ......................................................... 23
- cci ......................................................... 24
- complete .................................................. 25
- construct.blocks ....................................... 26
- D1 ......................................................... 27
- D2 ......................................................... 28
- D3 ......................................................... 29
- densityplot.mids ....................................... 29
- employee ............................................... 32
- estimice ............................................... 33
- extractBS ............................................... 34
- fdd ......................................................... 35
- fdgs ....................................................... 37
- fico ....................................................... 38
- fix.coef ............................................... 39
- flux ....................................................... 40
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fluxplot</td>
<td>42</td>
</tr>
<tr>
<td>getfit</td>
<td>43</td>
</tr>
<tr>
<td>getqbar</td>
<td>44</td>
</tr>
<tr>
<td>glm.mids</td>
<td>45</td>
</tr>
<tr>
<td>ibind</td>
<td>46</td>
</tr>
<tr>
<td>ic</td>
<td>47</td>
</tr>
<tr>
<td>ici</td>
<td>48</td>
</tr>
<tr>
<td>is.mads</td>
<td>48</td>
</tr>
<tr>
<td>is.mids</td>
<td>49</td>
</tr>
<tr>
<td>is.mipo</td>
<td>49</td>
</tr>
<tr>
<td>is.mira</td>
<td>50</td>
</tr>
<tr>
<td>is.mitml.result</td>
<td>50</td>
</tr>
<tr>
<td>leiden85</td>
<td>51</td>
</tr>
<tr>
<td>lm.mids</td>
<td>51</td>
</tr>
<tr>
<td>mads-class</td>
<td>53</td>
</tr>
<tr>
<td>make.blocks</td>
<td>54</td>
</tr>
<tr>
<td>make.blots</td>
<td>55</td>
</tr>
<tr>
<td>make.formulas</td>
<td>56</td>
</tr>
<tr>
<td>make.method</td>
<td>57</td>
</tr>
<tr>
<td>make.post</td>
<td>58</td>
</tr>
<tr>
<td>make.predictorMatrix</td>
<td>59</td>
</tr>
<tr>
<td>make.visitSequence</td>
<td>59</td>
</tr>
<tr>
<td>make.where</td>
<td>60</td>
</tr>
<tr>
<td>mammalsleep</td>
<td>61</td>
</tr>
<tr>
<td>md.pairs</td>
<td>62</td>
</tr>
<tr>
<td>md.pattern</td>
<td>63</td>
</tr>
<tr>
<td>mdc</td>
<td>64</td>
</tr>
<tr>
<td>mice</td>
<td>66</td>
</tr>
<tr>
<td>mice.impute.2l.bin</td>
<td>72</td>
</tr>
<tr>
<td>mice.impute.2l.lmer</td>
<td>73</td>
</tr>
<tr>
<td>mice.impute.2l.norm</td>
<td>75</td>
</tr>
<tr>
<td>mice.impute.2l.pan</td>
<td>76</td>
</tr>
<tr>
<td>mice.impute.2lonly.mean</td>
<td>78</td>
</tr>
<tr>
<td>mice.impute.2lonly.norm</td>
<td>80</td>
</tr>
<tr>
<td>mice.impute.2lonly.pmm</td>
<td>82</td>
</tr>
<tr>
<td>mice.impute.cart</td>
<td>84</td>
</tr>
<tr>
<td>mice.impute.jomoImpute</td>
<td>85</td>
</tr>
<tr>
<td>mice.impute.lda</td>
<td>87</td>
</tr>
<tr>
<td>mice.impute.logreg</td>
<td>88</td>
</tr>
<tr>
<td>mice.impute.logreg.boot</td>
<td>90</td>
</tr>
<tr>
<td>mice.impute.mean</td>
<td>91</td>
</tr>
<tr>
<td>mice.impute.midastouch</td>
<td>92</td>
</tr>
<tr>
<td>mice.impute.norm</td>
<td>94</td>
</tr>
<tr>
<td>mice.impute.norm.boot</td>
<td>96</td>
</tr>
<tr>
<td>mice.impute.norm.nob</td>
<td>97</td>
</tr>
<tr>
<td>mice.impute.norm.predict</td>
<td>98</td>
</tr>
<tr>
<td>mice.impute.panImpute</td>
<td>99</td>
</tr>
<tr>
<td>mice.impute.passive</td>
<td>101</td>
</tr>
</tbody>
</table>
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


ampute

Generate Missing Data for Simulation Purposes

Description

This function generates multivariate missing data in a MCAR, MAR or MNAR manner. 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 dataframe. Values should be numeric. Categorical variables should have been transformed into 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 a variable should have missing values and 1 indicates a variable should remain complete. The user may specify as many patterns as desired. One pattern (a vector) or double patterns are 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, weights of the variables that will be made incomplete, should be zero. For a MNAR mechanism, these weights might 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. If it is MNAR, variables that
ampute will be observed will be weighted with $\theta$. If 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 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 discrete distribution. If TRUE, the probabilities of being missing are based on a continuous logistic distribution function. \texttt{ampute\_continuous} will be used to calculate and assign the probabilities. These will be based on argument type. If FALSE, the probabilities of being missing are based on a discrete distribution \texttt{(ampute\_discrete)} based on the odds argument. Default is TRUE.

type A vector of strings containing the type of missingness for each pattern. Either "LEFT", "MID", "TAIL" or "RIGHT". If a single missingness type is entered, all patterns will be created by the same type. If missingness types should differ over patterns, a vector of missingness types should be entered. Default is RIGHT for all patterns and is the result of \texttt{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 amount of odds values defines in how many quantiles the sum scores will be 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 the patterns, specify NA for cells remaining empty. Default is 4 quantiles with odds values 1, 2, 3 and 4, the result of \texttt{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 but the amputed data set.

Details

When new multiple imputation techniques are tested, missing values need to be generated in simulated data sets. The generation of missing values is what we call: amputation. The function \texttt{ampute} is developed to perform any kind of amputation desired by the researcher. An extensive example and more explanation of the function can be found in the vignette \texttt{Generate missing values with ampute}, available in \texttt{mice} as well. For imputation, the function \texttt{mice} is advised.

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 several variables need to be amputed, the procedure is repeated multiple times.

With this 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, \texttt{ampute} is developed to perform the amputation according the researcher’s desires.

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 0, 0, 1. 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 divided over the patterns with these probabilities. 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). For a more thorough explanation of these definitions, I refer to Van Buuren (2012).

When the user sets the missingness mechanism to "MCAR", the candidates have an equal probability of having missing values. No other specifications have to be made. 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. That is the 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 might 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 might 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 might 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 logit function, as described by Van Buuren (2012). The user can specify the type of missingness, which, again, may differ between patterns.

For an extensive example of the working of the function, I gladly refer to the vignette Generate missing values with ampute.

Value

Returns an S3 object of class mads-class (multivariate amputed data set)
Author(s)
Rianne Schouten [aut, cre], Gerko Vink [aut], Peter Lugtig [ctb], 2016

References


See Also
mads-class, bwplot, xyplot, mice

Examples
# Simulate data set with \code{mvrnorm} from package \code{\pkg{MASS}}.
require(MASS)
sigma <- matrix(data = c(1, 0.2, 0.2, 0.2, 1, 0.2, 0.2, 0.2, 1), nrow = 3)
complete.data <- mvrnorm(n = 100, mu = c(5, 5, 5), Sigma = sigma)
# Perform quick amputation
result1 <- ampute(data = complete.data)
# Change default matrices as desired
patterns[1:3, 2] <- 0
odds <- result1$odds
odds[2,3:4] <- c(2, 4)
odds[3,] <- c(3, 1, NA, NA)
# Rerun amputation
result2 <- ampute(data = complete.data, patterns = patterns, freq = c(0.3, 0.3, 0.4), cont = FALSE, odds = odds)
# Run an amputation procedure with continuous probabilities
result3 <- ampute(data = complete.data, type = c("RIGHT", "TAIL", "LEFT"))
Usage

```r
## 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

```r
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 multiply 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 logistics 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.
.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

```r
# 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 = 2)
is.mids(test4)
identical(complete(test4, action = "long", include = TRUE), X)

# example without an .id variable
# variable .id not preserved
X3 <- X[, -2]
test5 <- as.mids(X3)
is.mids(test5)
identical(complete(test5, action = "long", include = TRUE)[, -2], X[, -2])

# as() syntax has fewer options
test7 <- as(X, "mids")
test8 <- as(X2, "mids")
test9 <- as(X2[, -2], "mids")
rev <- ncol(X):1
test10 <- as(X[, rev], "mids")

# 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)
```

Create a mira object from repeated analyses
as.mitml.result

Description
The as.mita() 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

\emph{Growth of Dutch boys}

\begin{itemize}
  \item \textbf{age} Decimal age (0-21 years)
  \item \textbf{hgt} Height (cm)
  \item \textbf{wgt} Weight (kg)
  \item \textbf{bmi} Body mass index
  \item \textbf{hc} Head circumference (cm)
  \item \textbf{gen} Genital Tanner stage (G1-G5)
  \item \textbf{phb} Pubic hair (Tanner P1-P6)
  \item \textbf{tv} Testicular volume (ml)
  \item \textbf{reg} Region (north, east, west, south, city)
\end{itemize}

\textbf{Description}

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

\textbf{Format}

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

Details

Random sample of 10\% from the cross-sectional data used to construct the Dutch growth references 1997. Variables \texttt{gen} and \texttt{phb} are ordered factors. \texttt{reg} is a factor.

\textbf{Source}


Examples

```r
# 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
require(MASS)
oldpar <- par(mfrow=c(1,2))
truehist(z$bmi[!is.na(boys$bmi)],h=1,xlim=c(10,30),ymax=0.25,
         col=mdc(1),xlab="BMI observed")
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))
truehist(z2$bmi[!is.na(boys$bmi)],h=1,xlim=c(10,30),ymax=0.25,col=mdc(1),
         xlab="BMI observed")
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 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](https://www.stats.ox.ac.uk/~snijders/mlbook.htm) by function data = raw/R/brandsma.R
References


bwplot.mads

Box-and-whisker plot of amputed and non-amputed data

Description

Plotting method to investigate the result of function `ampute`, the relation between the data variables and the amputed data. The function does not show which data is amputed. It does show how the amputed values are related to the variable values.

Usage

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

Arguments

- `x`: A mads (`mads-class`) 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-class` or the vignette *Multivariate Amputation using Ampute* to understand the contents of class object mads.

Author(s)
Rianne Schouten, 2016

See Also
`ampute, bwplot, Lattice` for an overview of the package, `mads-class`

---

**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
```r
## 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 (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 is 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.group` takes precedence over `groups`.

Use the `subset` and `na.group` together to plots parts of the data. For example, select the first imputed data set 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=mc(1:2)`, a transparent blue color for the observed data, and a transparent red color for the imputed data. A good choice is `col=mc(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` method can be used to subsequently update components of the object, and the `print` 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**


**See Also**

`mice`, `xyplot`, `densityplot`, `stripplot`, `lattice` for an overview of the package, as well as `bwplot`, `panel.bwplot`, `print.trellis`, `trellis.par.set`
**cbind.mids**

**Description**

This 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.

**Usage**

```r
cbind.mids(x, y = NULL, ...)
```

**Arguments**

- `x` A `mids` object.
- `y` A `mids` object, or a data.frame, matrix, factor or vector.
- `...` Additional data.frame, matrix, vector or factor. These can be given as named arguments.

**Details**

*Pre-requisites:* If `y` is a `mids`-object, the rows of `x$data` and `y$data` should match, as well as the number of imputations (`m`). Other `y` are transformed into a data.frame whose rows should match with `x$data`.

The function renames any duplicated variable or block names by appending "\1", "\2" to duplicated names.

**Value**

An S3 object of class `mids`
Note

The 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$blocks.
- **nmis**: Equals c(x$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$blots).
- **seed**: Taken from x$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.

Author(s)

Karin Groothuis-Oudshoorn, Stef van Buuren

See Also

cbind, rbind.mids, ibind, mids

Examples

```r
# 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
imp2$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, ch1 = nhanes$ch1)

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

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

```r
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`
cci

**Examples**

```r
# 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**

```r
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**

```r
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**

*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**

```r
complete(data, action = 1L, include = FALSE, mild = FALSE, ...)```

**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$nm` 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.
- `...`: 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.
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 "21" in their names) other codes (e.g, 2 or -2) are also allowed.

Value

A `blocks` object.

See Also

`make.blocks`, `name.blocks`

Examples

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

---

**D1**

*Compare two nested models using D1-statistic*

Description

Compare two nested models using D1-statistic

Usage

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

fit1  An object of class \textit{mira}, produced by with().

fit0  An object of class \textit{mira}, produced by with(). The model in fit0 is a nested within fit1. The default null model \texttt{fit0} = \texttt{NULL} compares fit1 to the intercept-only model.

df.com  A single number or a numeric vector denoting the complete-data degrees of freedom for the hypothesis test. If not specified, it is set equal to \texttt{df.residual} of model fit1. Not used for method D2().

...  Not used.

---

\textbf{D2}  \textit{Compare two nested models using D2-statistic}

Description

Compare two nested models using D2-statistic

Usage

\texttt{D2(fit1, fit0 = NULL, df.com = NULL, use = \texttt{"wald"}, ...)}

Arguments

fit1  An object of class \textit{mira}, produced by with().

fit0  An object of class \textit{mira}, produced by with(). The model in fit0 is a nested within fit1. The default null model \texttt{fit0} = \texttt{NULL} compares fit1 to the intercept-only model.

df.com  A single number or a numeric vector denoting the complete-data degrees of freedom for the hypothesis test. If not specified, it is set equal to \texttt{df.residual} of model fit1. Not used for method D2().

use  A character string denoting Wald- or likelihood-based based tests. Can be either \texttt{"wald"} or \texttt{"likelihood"}. Only used if \texttt{method=\texttt{"D2"}}.

...  Not used.

Details

The D2 method does not use the \texttt{df.com} parameter, so it does not pass it down to \texttt{testModels}. This prevents the following warning thrown by \texttt{testModels}: Complete-data degrees of freedom are not available for use with \texttt{D2}, and thus were ignored.
**D3**

*Compare two nested models using D3-statistic*

**Description**

Compare two nested models using D3-statistic

**Usage**

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

**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.  
- `df.com`: A single number or a numeric vector denoting the complete-data degrees of freedom for the hypothesis test. If not specified, it is set equal to `df.residual` of model `fit1`. Not used for method `D2()`.  
- `...`: Not used.

---

**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**

```r
## 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 `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.
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 method can be used to subsequently update components of the object, and the print 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 err 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


See Also

mice, xyplot, stripplot, bwplot, lattice for an overview of the package, as well as densityplot, panel.densityplot, print.trellis, trellis.par.set

Examples

```r
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)
```

densityplot

employee

Employee selection data

Description

A toy example from Craig Enders.

Usage

employee
estimice

Format

A data frame with 20 rows and 3 variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQ</td>
<td>candidate IQ score</td>
</tr>
<tr>
<td>wbeing</td>
<td>candidate well-being score</td>
</tr>
<tr>
<td>jobperf</td>
<td>candidate job performance score</td>
</tr>
</tbody>
</table>

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 is 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).

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
### 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
ycl  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
yf1  PTSD-RI parent full T1
yf2  PTSD-RI parent full T2
yf3  PTSD-RI parent full T3
yp1  PTSD parent partial T1
yp2  PTSD parent partial T2
yp3  PTSD parent partial T3
yf1  PTSD child full T1
yf2  PTSD child full T2
yf3  PTSD child full T3
ycp1  PTSD child partial T1
ycp2  PTSD child partial T2
ycp3  PTSD child partial T3
cbi1  CBCL Internalizing T1
cbi3  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


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


Examples

```r
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 \(y_j\) observed (White and Carlin, 2010).

Usage

```r
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.
**fix.coef**

**Author(s)**
Stef van Buuren, 2012

**References**


**See Also**
`fluxplot, flux, md.pattern`

---

**fix.coef**

*Fix coefficients and update model*

**Description**
Refits a model with a specified set of coefficients.

**Usage**

```r
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))` 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

```r
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
1 - cor(predict(model0), predict(model0) + residuals(model0))^2
1 - cor(predict(model1), predict(model1) + residuals(model1))^2
1 - cor(predict(model2), predict(model2) + residuals(model2))^2

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

---

**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**

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

**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**

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

Influx is equal to the number of variable pairs \((Y_j, Y_k)\) with \(Y_j\) missing and \(Y_k\) 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 \(Y_j\) observed and \(Y_k\) missing, divided by the total number of incomplete data cells. Outflux is an indicator of the potential usefulness of \(Y_j\) 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 \(Y_j\) observed (White and Carlin, 2010).

**Value**

A data frame with `nrow(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 \(Y_j\) observed.

**Author(s)**

Stef van Buuren, 2012

**References**


**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**

```r
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**

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

Influx is equal to the number of variable pairs \((Y_j, Y_k)\) with \(Y_j\) missing and \(Y_k\) 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 \( Y_j \) observed and \( Y_k \) missing, divided by the total number of incomplete data cells. Outflux is an indicator of the potential usefulness of \( Y_j \) 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: \( \text{pobs} = \text{Proportion observed} \), \( \text{influx} = \text{Influx} \), \( \text{outflux} = \text{Outflux} \), \( \text{ainb} = \text{Average inbound statistic} \), \( \text{aout} = \text{Average outbound statistic} \), \( \text{fico} = \text{Fraction of incomplete cases among cases with} \ Y_j \text{ observed} \)

Author(s)

Stef van Buuren, 2012

References


See Also

flux, md.pattern, fico

---

**getfit**

*Extract list of fitted model*

**Description**

getfit returns the list of objects containing the repeated analysis results, or optionally, one of these fit objects.

**Usage**

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

x  An object of class mira or mitml.result, typically produced by a call to with().
i  An integer between 1 and \( x \) signaling the number of the repeated analysis. The default \( i = -1 \) return a list with all analyses.
simplify  Should the return value be unlisted?

Value

If \( i = -1 \) an object of class mitml.result containing all analyses, otherwise it returns the fitted object of the \( i \)'th repeated analysis.

Author(s)

Stef van Buuren, March 2012.

See Also

mira, with.mids

Examples

```r
imp <- mice(nhanes)
fit <- with(imp, lm(bmi~chl+hyp))
getfit(fit)
getfit(fit, 2)
```

---

**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


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, rbind.mids, cbind.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)`
is.mids

Arguments

x An object

Value

A logical indicating whether x is an object of class mads

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


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


See Also

- `lm`, `mids`, `mira`

Examples

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

The mads object contains an amputed data set. The mads object is generated by the ampute function. The mads class of objects has methods for the following generic functions: print, summary, bwplot and xyplot.

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.
**Note**

Many of the functions of the mice package do not use the S4 class definitions, and instead rely on the S3 list equivalent `oldClass(obj) <- "mads"`.

**Author(s)**

Rianne Schouten, 2016

**See Also**

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

---

**make.blocks**

*Creates a blocks argument*

**Description**

This helper function generates a list of the type needed for `blocks` argument in the `{mice}` function.

**Usage**

```r
make.blocks(data, partition = c("scatter", "collect", "void"),
            calltype = "type")
```

**Arguments**

- `data` 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) will assign each column to its own block. Value "collect" assigns all variables to one block, whereas "void" produces an empty list.
- `calltype` A character vector of length(=block) elements that indicates how the imputation model is specified. If `calltype = "type"` (the default), the underlying imputation model is called by means of the type argument. The type argument for block h is equivalent to row h in the `predictorMatrix`. The alternative is `calltype = "formula"`. This will pass `formulas[[h]]` to the underlying imputation function for block h, together with the current data. The calltype of a block is set automatically during initialization. Where a choice is possible, `calltype = "formula"` is preferred over "type" since this is more flexible and extendable. However, what precisely happens depends also on the capabilities of the imputation function that is called.
Details

Choices "scatter" and "collect" represent 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

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

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

```r
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
make.formulas

See Also

make.blocks

Examples

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


desc

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

## 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

```r
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.
- **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 pmr, 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  

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 just after 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 `predictorMatrix`. 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

```r
make.predictorMatrix(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

```r
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

```r
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

```r
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 not (FALSE).

Usage

```r
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**

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

---

**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.
### md.pairs

**Source**


**Examples**

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

---

**Description**

Number of observations per variable pair.

**Usage**

```r
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**


**Examples**

```r
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**

```r
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


Examples

```r
mdc(pattern(nhanes))
# age hyp bmi chl
# 13 1 1 1 0
# 1 1 1 0 1
# 3 1 1 1 0 1
# 1 1 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

```r
mdc(r = "observed", s = "symbol", transparent = TRUE, 
cso = hcl(240, 100, 40, 0.7), 
csi = hcl(0, 100, 40, 0.7), 
csc = "gray50", 
clo = hcl(240, 100, 40, 0.8), 
cli = 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 calculated from the length of $r$ and $s$. Elements of the input vectors are repeated if needed.

Author(s)

Stef van Buuren, sept 2012.

References


See Also

hcl, rgb, xyplot.mids, xyplot, trellis.par.set
Examples

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

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

---

mice: Multivariate Imputation by Chained Equations

Description

The mice package implements a method to deal with missing data. The package creates multiple imputations (replacement values) for multivariate missing data. The method is based on Fully Conditional Specification, where each incomplete variable is imputed by a separate model. The MICE algorithm can impute mixes of continuous, binary, unordered categorical and ordered categorical data. In addition, MICE can impute continuous two-level data, and maintain consistency between imputations by means of passive imputation. Many diagnostic plots are implemented to inspect the quality of the imputations.

Generates Multivariate Imputations by Chained Equations (MICE)

Usage

```r
mice(data, m = 5, method = NULL, predictorMatrix, where = NULL,
    blocks, visitSequence = NULL, formulas, blots = NULL, post = NULL,
    defaultMethod = c("pmm", "logreg", "polyreg", "polr"), maxit = 5,
    printFlag = TRUE, seed = NA, data.init = NULL, ...)```

Arguments

data A data frame or a matrix containing the incomplete data. Missing values are coded as NA.
m Number of multiple imputations. The default is m=5.
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 \texttt{predictorMatrix} is a square matrix of 
\texttt{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 \texttt{-2}) are also allowed.

\textbf{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, \texttt{where = is.na(data)},
specifies that the missing data should be imputed. The \texttt{where} argument may be
used to overimpute observed data, or to skip imputations for selected missing
values.

\textbf{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 impu-
tation method (see \texttt{method} argument). By default each variable is placed into
its own block, which is effectively fully conditional specification (FCS) by uni-
variate models (variable-by-variable imputation). Only variables whose names
appear in \texttt{blocks} are imputed. The relevant columns in the \texttt{where} matrix are
set to \texttt{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.

\textbf{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 collec-
tion 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 \texttt{visitSequence = "roman" vis-
its the blocks (left to right) in the order in which they appear in \texttt{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).

\textbf{formulas} \\
A named list of formula's, or expressions that can be converted into formula's
by \texttt{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 \texttt{formulas} argument is an alternative to the \texttt{predictorMatrix}
argument that allows for more flexibility in specifying imputation models, e.g.,
for specifying interaction terms.

\textbf{blots} \\
A named list of a list's that can be used to pass down arguments to lower level
imputation function. The entries of element \texttt{blots[\texttt{[blockname]}]} are passed
down to the function called for block \texttt{blockname}.

\textbf{post} \\
A vector of strings with length \texttt{ncol(data)} specifying expressions as strings.
Each string is parsed and executed within the \texttt{sampler()} function to post-process
imputed values during the iterations. The default is a vector of empty strings,
indicating no post-processing.

\textbf{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 \texttt{pmm}, predictive
mean matching (numeric data) \texttt{logreg}, logistic regression imputation (binary
data, factor with 2 levels) \texttt{polyreg}, polytomous regression imputation for un-
ordered categorical data (factor > 2 levels) \texttt{polr}, proportional odds model for  
(ordered, > 2 levels).
maxit  A scalar giving the number of iterations. The default is 5.
printFlag  If TRUE, mice will print history on console. Use print=FALSE for silent computation.
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.
data.init  A data frame of the same size and type as data, without missing data, used to initialize imputations before the start of the iterative process. The default NULL implies that starting imputation are created by a simple random draw from the data. Note that specification of data.init will start all m Gibbs sampling streams from the same imputation.
...  Named arguments that are passed down to the univariate imputation functions.

Details

The mice package contains functions to

- Inspect the missing data pattern
- Impute the missing data m times, resulting in m completed data sets
- Diagnose the quality of the imputed values
- Analyze each completed data set
- Pool the results of the repeated analyses
- Store and export the imputed data in various formats
- Generate simulated incomplete data
- Incorporate custom imputation methods

Generates multiple imputations for incomplete multivariate data by Gibbs sampling. Missing data can occur anywhere in the data. The algorithm imputes an incomplete column (the target column) by generating 'plausible' synthetic values given other columns in the data. Each incomplete column must act as a target column, and has its own specific set of predictors. The default set of predictors for a given target consists of all other columns in the data. For predictors that are incomplete themselves, the most recently generated imputations are used to complete the predictors prior to imputation of the target column.

A separate univariate imputation model can be specified for each column. The default imputation method depends on the measurement level of the target column. In addition to these, several other methods are provided. You can also write their own imputation functions, and call these from within the algorithm.

The data may contain categorical variables that are used in a regressions on other variables. The algorithm creates dummy variables for the categories of these variables, and imputes these from the corresponding categorical variable.

Built-in univariate imputation methods are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Target Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmm</td>
<td>any</td>
<td>Predictive mean matching</td>
</tr>
<tr>
<td>midastouch</td>
<td>any</td>
<td>Weighted predictive mean matching</td>
</tr>
<tr>
<td>sample</td>
<td>any</td>
<td>Random sample from observed values</td>
</tr>
<tr>
<td>cart</td>
<td>any</td>
<td>Classification and regression trees</td>
</tr>
</tbody>
</table>
These corresponding functions are coded in the mice library under names mice.impute.method, where method is a string with the name of the univariate imputation method name, for example norm. The method argument specifies the methods to be used. For the j’th column, mice() calls the first occurrence of paste('mice.impute.', method[j], sep = '') in the search path. The mechanism allows users to write customized imputation function, mice.impute.myfunc. To call it for all columns specify method=’myfunc’. To call it only for, say, column 2 specify method=c(‘norm’,’myfunc’,’logreg’,...). Passive imputation: mice() supports a special built-in method, called passive imputation. This method can be used to ensure that a data transform always depends on the most recently generated imputations. In some cases, an imputation model may need transformed data in addition to the original data (e.g. log, quadratic, recodes, interaction, sum scores, and so on).
Passive imputation maintains consistency among different transformations of the same data. Passive imputation is invoked if ~ is specified as the first character of the string that specifies the univariate method. mice() interprets the entire string, including the ~ character, as the formula argument in a call to model.frame(formula,data[!r[,j]],). This provides a simple mechanism for specifying deterministic dependencies among the columns. For example, suppose that the missing entries in variables data$height and data$weight are imputed. The body mass index (BMI) can be calculated within mice by specifying the string ‘~I(weight/height^2)’ as the univariate imputation method for the target column data$bmi. Note that the ~ mechanism works only on those entries which have missing values in the target column. You should make sure that the combined observed and imputed parts of the target column make sense. An easy way to create consistency is by coding all entries in the target as NA, but for large data sets, this could be inefficient. Note that you may also need to adapt the default predictorMatrix to evade linear dependencies among the predictors that could cause errors like Error in solve.default() or Error: system is exactly singular. Though not strictly needed, it is often useful to specify visitSequence such that the column that is imputed by the ~ mechanism is visited each time after one of its predictors was visited. In that way, deterministic relation between columns will always be synchronized.
A new argument `ls.meth` can be parsed to the lower level .norm.draw to specify the method for generating the least squares estimates and any subsequently derived estimates. Argument `ls.meth` takes one of three inputs: "qr" for QR-decomposition, "svd" for singular value decomposition and "ridge" for ridge regression. `ls.meth` defaults to `ls.meth = "qr"`.

**Auxiliary predictors in formulas specification:** For a given block, the formulas specification takes precedence over the corresponding row in the predictMatrix argument. This precedence is, however, restricted to the subset of variables specified in the terms of the block formula. Any variables not specified by formulas are imputed according to the predictMatrix specification. Variables with non-zero type values in the predictMatrix will be added as main effects to the formulas, which will act as supplementary covariates in the imputation model. It is possible to turn off this behavior by specifying the argument `auxiliary = FALSE`.

**Value**

Returns an S3 object of class `mids` (multiply imputed data set)

**Functions**

The main functions are:

- `mice()`: Impute the missing data *m* times
- `with()`: Analyze completed data sets
- `pool()`: Combine parameter estimates
- `complete()`: Export imputed data
- `ampute()`: Generate missing data

**Vignettes**

There is a detailed series of six online vignettes that walk you through solving realistic inference problems with mice.

We suggest going through these vignettes in the following order

1. Ad hoc methods and the MICE algorithm
2. Convergence and pooling
3. Inspecting how the observed data and missingness are related
4. Passive imputation and post-processing
5. Imputing multilevel data
6. Sensitivity analysis with `mice`


**Methodology**

The `mice` software was published in the *Journal of Statistical Software* (Van Buuren and Groothuis-Oudshoorn, 2011). The first application of the method concerned missing blood pressure data (Van Buuren et. al., 1999). The term *Fully Conditional Specification* was introduced in 2006 to describe a
general class of methods that specify imputations model for multivariate data as a set of conditional distributions (Van Buuren et al., 2006). Further details on mixes of variables and applications can be found in the book *Flexible Imputation of Missing Data. Second Edition*. Chapman & Hall/CRC. Boca Raton, FL.

Author(s)
Stef van Buuren &lt;stef.vanbuuren@tno.nl&gt;, Karin Groothuis-Oudshoorn &lt;c.g.m.oudshoorn@utwente.nl&gt;, 2000-2010, with contributions of Alexander Robitzsch, Gerko Vink, Shahab Jolani, Roel de Jong, Jason Turner, Lisa Doove, John Fox, Frank E. Harrell, and Peter Malewski.

References


See Also

*mice, with.mids, pool, complete, ampute*

*mids, with.mids, set.seed, complete*

Examples

```r
# do default multiple imputation on a numeric matrix
```
mice.impute.2l.bin

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 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 \( \text{sum}(\text{wy}) \)

Author(s)

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

References


See Also

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

Examples

```r
library(tidyrr)
library(dplyr)
data("toenail", package = "HSAUR3")
data <- tidyrr::complete(toenail, patientID, visit) %>%
tidyrr::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

```r
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 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.

Value

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

Author(s)

Shahab Jolani, 2017

References


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

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 \texttt{mice.impute.2l.norm}.
  The random intercept is automatically added in \texttt{mice.impute.2l.norm}().

Value
  A vector of length \( \text{nmis} \) with imputations.

Note
  This function does not implement the \texttt{where} functionality. It always produces \( \text{nmis} \) imputation, irrespective of the \texttt{where} argument of the \texttt{mice} function.

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

References

See Also
  Other univariate-2l: \texttt{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 imputationMethod
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 ,
    imputationMethod = 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 the mean within the class
mice.impute.2lonly.mean

Description

Imputes the mean of within the class

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. If there are no observed data in the class, all entries of the class are set to NaN. This function is primarily useful for repairing incomplete data that are constant within the class, but that vary over the classes.

Value

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

Author(s)

Gerko Vink, Stef van Buuren, 2013

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**

```r
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.21.pan` switching regression imputation between level 1 and level 2 as described in Yucel (2008) or Gelman and Hill (2007, p. 541).

**Value**

A vector of length `nmis` with imputations.

**Author(s)**

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

**References**


See Also

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

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
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, imputationMethod = impM1, maxit=1, paniter=500)
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).

Value

A vector of length nmis with imputations.

Note

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

Author(s)

Alexander Robitzsch (IPN - Leibniz Institute for Science and Mathematics Education, Kiel, Ger-
many), <robitzsch@ipn.uni-kiel.de>.
mice.impute.2lonly.pmm

References


See Also

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

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

Examples

```r
# 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(1 - 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

imp <- mice(dfr, m = 1, predictorMatrix = predM1, 
imputationMethod = 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 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 ymiss, 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 \( \text{sum}(\text{wy}) \).
Numeric vector of length \( \text{sum}(\text{!y}) \) 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.


See Also

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

Other univariate imputation functions: mice.impute.lda, mice.impute.logreg.boot, mice.impute.logreg, mice.impute.mean, mice.impute.midastouch, mice.impute.norm.boot, mice.impute.norm.nob, mice.impute.norm.predict, mice.impute.norm, mice.impute.pmm, mice.impute.polr, mice.impute.polyreg, mice.impute.quadratic, mice.impute.rf, mice.impute.rf

Examples

```r
require(rpart)
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](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 to 10.

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 Quartagno and Carpenter (authors of jomo package).

References


See Also

jomoImpute

Other multivariate-2l: mice.impute.panImpute
mice.impute.lda

Examples

# 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)

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 to include bootstrap - disabled since

Author(s)
Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References
Van Buuren, S., Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equa-
gies for the Statistical Analysis of Incomplete Data Sets. Ph.D. Thesis, TNO Prevention and
Health/Erasmus University Rotterdam. ISBN 90-74479-08-1.
Berlin.

See Also
mice, link{mice.impute.polyreg}, lda
Other univariate imputation functions: mice.impute.cart, mice.impute.logreg.boot, mice.impute.logreg,
mice.impute.mean, mice.impute.midastouch, mice.impute.norm.boot, mice.impute.norm.nob,
mice.impute.norm.predict, mice.impute.norm, mice.impute.pmm, mice.impute.polr, mice.impute.polyreg,
mice.impute.quadratic, mice.impute.rf, mice.impute.rf

---

**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.
**mice.impute.logreg**

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**


**See Also**

`mice`, `glm`, `glm.fit`  
Other univariate imputation functions: `mice.impute.cart`, `mice.impute.lda`, `mice.impute.logreg.boot`, `mice.impute.mean`, `mice.impute.midastouch`, `mice.impute.norm.boot`, `mice.impute.norm.nob`, `mice.impute.norm.predict`, `mice.impute.norm`, `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 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


mice.impute.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 \( \text{length}(y) \) indicating the the subset \( y[\text{ry}] \) of elements in \( y \) to which the imputation model is fitted. The \( \text{ry} \) generally distinguishes the observed (TRUE) and missing values (FALSE) in \( y \).
- **x**: Numeric design matrix with \( \text{length}(y) \) rows with predictors for \( y \). Matrix \( x \) may have no missing values.
- **wy**: Logical vector of length \( \text{length}(y) \). 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 \( \text{sum}(wy) \)

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


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.

See Also

mice, mean

Other univariate imputation functions: mice.impute.cart, mice.impute.lda, mice.impute.logreg.boot, mice.impute.logreg, mice.impute.midastouch, mice.impute.norm.boot, mice.impute.norm.nob, mice.impute.norm.predict, mice.impute.norm, mice.impute.pmm, mice.impute.pmm.boot, mice.impute.polr, mice.impute.polyreg, mice.impute.quadratic, mice.impute.rf, mice.impute.ri
**mice.impute.midastouch**

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 object name 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 object name 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 \( y_{obs} \) (nobs x 1) and \( y_{mis} \) (nmis x nobs).
4. Calculate the distance between all \( y_{obs} \) and the corresponding \( y_{mis} \).
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 \( \sum(w) \)

**Author(s)**

Philipp Gaffert, Florian Meinfelder, Volker Bosch 2015

**References**


See Also

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

Examples

```r
# 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.norm**  
*Imputation by Bayesian linear regression*

Description

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

Usage

```r
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.
**mice.impute.norm**

`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 `\kappa`.
3. Calculate regression weights \( \hat{\beta} = V X'_{obs} y_{obs} \).
4. Draw a random variable \( \hat{g} \sim \chi^2_\nu \) with \( \nu = n_1 - q \).
5. Calculate \( \hat{\sigma}^2 = (y_{obs} - X_{obs}\hat{\beta})'(y_{obs} - X_{obs}\hat{\beta})/\hat{g} \).
6. Draw \( q \) independent \( N(0, 1) \) variates in vector \( \hat{z}_1 \).
7. Calculate `V1/2` by Cholesky decomposition.
8. Calculate \( \hat{\beta} = \hat{\beta} + \hat{\sigma}\hat{z}_1V^{1/2} \).
9. Draw \( n_0 \) independent \( N(0, 1) \) variates in vector \( \hat{z}_2 \).
10. Calculate the \( n_0 \) values \( y_{imp} = X_{mis}\hat{\beta} + \hat{z}_2\hat{\sigma} \).

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**


**See Also**

Other univariate imputation functions: `mice.impute.cart`, `mice.impute.lda`, `mice.impute.logreg.boot`, `mice.impute.logreg`, `mice.impute.mean`, `mice.impute.midastouch`, `mice.impute.norm.boot`, `mice.impute.norm.nob`, `mice.impute.norm.predict`, `mice.impute.pmm`, `mice.impute.polar`, `mice.impute.polyreg`, `mice.impute.quadratic`, `mice.impute.rf`, `mice.impute.rj`
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 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


See Also

Other univariate imputation functions: mice.impute.cart, mice.impute.lda, mice.impute.logreg.boot, mice.impute.logreg, mice.impute.mean, mice.impute.midastouch, mice.impute.norm.nob, mice.impute.norm.predict, mice.impute.norm, mice.impute.pmm, mice.impute.polr, mice.impute.polyreg, mice.impute.quadratic, mice.impute.rf, mice.impute.rj
mice.impute.norm.nob

**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 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
mice.impute.norm.predict

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 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 \( \text{sum}(\text{wy}) \)

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


See Also

Other univariate imputation functions: mice.impute.cart, mice.impute.lda, mice.impute.logreg.boot, mice.impute.logreg, mice.impute.mean, mice.impute.midastouch, mice.impute.norm.boot, mice.impute.norm.nob, mice.impute.norm, mice.impute.pmm, 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


See Also

- `panImpute`
- Other multivariate-2l: `mice.impute.jomoImpute`
**Examples**

```r
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)
```

**Description**

Calculate new variable during imputation

**Usage**

```r
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 \( Q = \frac{W}{H^2} \) based on two variables, or a mean variable like \( \frac{x_1 + x_2 + x_3}{3} \). 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**

mice.impute.pmm

Imputation by predictive mean matching

Description

Calculates imputations for univariate missing data by predictive mean matching.

Usage

mice.impute.pmm(y, ry, x, wy = NULL, donors = 5L, matchtype = 1L,
    ridge = 1e-05, ...)

Arguments

y          Vector to be imputed
ry         Logical vector of length \( \text{length}(y) \) indicating the the subset \( y[\text{ry}] \) of elements in \( y \) to which the imputation model is fitted. The \( \text{ry} \) generally distinguishes the observed (TRUE) and missing values (FALSE) in \( y \).
x          Numeric design matrix with \( \text{length}(y) \) rows with predictors for \( y \). Matrix \( x \) may have no missing values.
wy         Logical vector of length \( \text{length}(y) \). 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 \( \text{donors} = 5L \). Setting \( \text{donors} = 1L \) 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 \( y_{obs} \) and the drawn values of \( y_{mis} \) (called type-1 matching). Other choices are matchtype = 0L (distance between predicted values) and matchtype = 2L (distance between drawn values).
ridge      The ridge penalty used in \text{norm.draw()}\) to prevent problems with multicollinearity. The default is \( \text{ridge} = 1e-05 \), 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 \( \text{ridge} = 1e-06 \) or even lower to reduce bias. For highly collinear data, set \( \text{ridge} = 1e-04 \) or higher.
...

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 $V = (S + \text{diag}(S)\kappa)^{-1}$, with some small ridge parameter $\kappa$.
3. Calculate regression weights $\hat{\beta} = VX_{obs}'y_{obs}$.
4. Draw $q$ independent $N(0, 1)$ variates in vector $\hat{z}_1$.
5. Calculate $V^{1/2}$ by Cholesky decomposition.
6. Calculate $\hat{\beta} = \hat{\beta} + \hat{\sigma}\hat{z}_1V^{1/2}$.
7. Calculate $\hat{\eta}(i, j) = |X_{obs,[i]}|\hat{\beta} - X_{mis,[j]}\hat{\beta}$ 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 $\sum_d \hat{\eta}(i, j)$ 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 $\hat{y}_j = y_{i_j}$ 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 $\text{sum}(wy)$

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn

References


See Also

Other univariate imputation functions: mice.impute.cart, mice.impute.lda, mice.impute.logreg.boot, mice.impute.logreg, mice.impute.mean, mice.impute.midastouch, mice.impute.norm.boot, mice.impute.norm.nob, mice.impute.norm.predict, mice.impute.norm, mice.impute.pirl, mice.impute.polyreg, mice.impute.quadratic, mice.impute.rf, mice.impute.rl
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')

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, ...)

Arguments

y  Vector to be imputed
ry Logical vector of length length(y) indicating 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.
**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, and a record is written to the `loggedInEvents` component of the `mids` object.

**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**


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 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

See Also
mice.multinom.polr
Other univariate imputation functions: mice.impute.cart, mice.impute.lda, mice.impute.logreg.boot,
mice.impute.logreg, mice.impute.mean, mice.impute.midastouch, mice.impute.norm.boot,
mice.impute.norm.nob, mice.impute.norm.predict, mice.impute.norm, mice.impute.pmm,
mice.impute.polyreg, mice.impute.quadratic, mice.impute.rf, mice.impute.ri
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**


**See Also**

`mice`, `multinom`, `polr`

Other univariate imputation functions: `mice.impute.cart`, `mice.impute.lda`, `mice.impute.logreg.boot`, `mice.impute.logreg`, `mice.impute.mean`, `mice.impute.midastouch`, `mice.impute.norm.boot`, `mice.impute.norm.nob`, `mice.impute.norm.predict`, `mice.impute.norm`, `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, ...)`
Arguments

- **y**: Vector to be imputed
- **ry**: Logical vector of length `length(y)` indicating 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 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)

Gerko Vink (University of Utrecht), <g.vink@uu.nl>

See Also

- `mice.impute.pmm`

Other univariate imputation functions: `mice.impute.cart`, `mice.impute.lda`, `mice.impute.logreg.boot`, `mice.impute.logreg`, `mice.impute.mean`, `mice.impute.midastouch`, `mice.impute.norm.boot`, `mice.impute.norm.nob`, `mice.impute.norm.predict`, `mice.impute.norm`, `mice.impute.pmm`, `mice.impute.polr`, `mice.impute.polyreg`, `mice.impute.rf`, `mice.impute.ri`
Examples

```r
require(lattice)

# 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)

# 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

```r
mice.impute.rf(y, ry, x, wy = NULL, ntree = 10, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>y</code></td>
<td>Vector to be imputed</td>
</tr>
<tr>
<td><code>ry</code></td>
<td>Logical vector of length <code>length(y)</code> indicating the subset <code>y[ry]</code> of elements in <code>y</code> to which the imputation model is fitted. The <code>ry</code> generally distinguishes the observed (TRUE) and missing values (FALSE) in <code>y</code>.</td>
</tr>
</tbody>
</table>
mice.impute.rf

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.

... Other named arguments passed down to randomForest() and randomForest:::randomForest.default

Details

Imputation of y by random forests. The method calls randomForest() 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), and is available in the package CALIBER.rfimpute. 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

References


See Also

mice, mice.impute.cart, randomForest, mice.impute.rfcat, mice.impute.rfcont

Other univariate imputation functions: mice.impute.cart, mice.impute.lda, mice.impute.logreg.boot, mice.impute.logreg, mice.impute.mean, mice.impute.midastouch, mice.impute.norm.boot, mice.impute.norm, mice.impute.norm.predict, mice.impute.norm, mice.impute.pmm, mice.impute.polr, mice.impute.polyreg, mice.impute.quadratic, mice.impute.ri
mice.impute.ri

Examples

library("lattice")
imp <- mice(nhanes2, meth = "rf", ntree = 3)
plot(imp)

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

Value

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

Author(s)

Shahab Jolani (University of Utrecht) <s.jolani@uu.nl>
References


See Also

Other univariate imputation functions: `mice.impute.cart`, `mice.impute.lda`, `mice.impute.logreg.boot`, `mice.impute.logreg`, `mice.impute.mean`, `mice.impute.midastouch`, `mice.impute.norm.boot`, `mice.impute.norm.nob`, `mice.impute.norm.predict`, `mice.impute.norm`, `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**

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

**Arguments**

- `y` Vector to be imputed
- `ry` Logical vector of length length(y) indicating 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
mice.mids

References

<table>
<thead>
<tr>
<th>mice.mids</th>
<th>Multivariate Imputation by Chained Equations (Iteration Step)</th>
</tr>
</thead>
</table>

Description
Takes a mids object, and produces a new object of class mids.

Usage
mice.mids(obj, maxit = 1, printFlag = TRUE, ...)

Arguments
- obj: An object of class mids, typically produced by a previous call to mice() or mice.mids()
- maxit: The number of additional Gibbs sampling iterations.
- 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:

- RAM memory may become easily exhausted if the number of iterations is large. Returning to prompt/session level may alleviate these problems.
- The user can compute customized convergence statistics at specific points, e.g. after each iteration, for monitoring convergence. - For computing ‘few extra iterations’.

Note: The imputation model itself is specified in the mice() function and cannot be changed with mice.mids. The state of the random generator is saved with the mids object.

Author(s)
Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References
mice.theme

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
Description

The mids object contains a multiply imputed data set. The mids object is generated by functions mice(), mice.mids(), cbind.mids(), rbind.mids() and ibind.mids().

Details

The mids class of objects has methods for the following generic functions: print, summary, plot. The loggedEvents entry is a matrix with five columns containing a record of automatic removal actions. It is NULL if no action was made. At initialization the program does the following three actions:

1. A variable that contains missing values, that is not imputed and that is used as a predictor is removed.
2. A constant variable is removed.
3. A collinear variable is removed.

During iteration, the program does the following actions:

1. One or more variables that are linearly dependent are removed (for categorical data, a 'variable' corresponds to a dummy variable).
2. 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.

Slots

- .Data: Object of class "list" containing the following slots:
  - 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.
  - 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.
mids-class

mids: An array containing the number of missing observations per column.
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: The sequence in which columns are visited.
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.
seed: The seed value of the solution.
iteration: Last Gibbs sampling iteration number.
lastSeedValue: The most recent seed value.
chainMean: A list of m components. Each component is a length(visitSequence) by maxit matrix 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: A list with similar structure of chainMean, containing the covariances 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.

Note

The mice package does not use the S4 class definitions, and instead relies on the S3 list equivalent oldClass(obj) <- "mids".

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References


See Also

mice, mira, mipo
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, filedat = "midsdata.txt", filesps = "readmids.sps",
    path = getwd(), sep = "\t", dec = ",", silent = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imp</td>
<td>The imp argument is an object of class mids, typically produced by the mice() function.</td>
</tr>
<tr>
<td>filedat</td>
<td>A character string describing the name of the output data file.</td>
</tr>
<tr>
<td>filesps</td>
<td>A character string describing the name of the output syntax file.</td>
</tr>
<tr>
<td>path</td>
<td>A character string containing the path of the output file. The value in path is appended to filedat and filesps. By default, files are written to the current R working directory. If path=NULL then no file path appending is done.</td>
</tr>
<tr>
<td>sep</td>
<td>The separator between the data fields.</td>
</tr>
<tr>
<td>dec</td>
<td>The decimal separator for numerical data.</td>
</tr>
<tr>
<td>silent</td>
<td>A logical flag stating whether the names of the files should be printed.</td>
</tr>
</tbody>
</table>

Details

This function automates most of the work needed to export a mids object to SPSS. It uses a modified version of writeForeignSPSS() from the foreign package. The modified version allows for a choice of the field and decimal separators, and makes some improvements to the formatting, so that the generated syntax file is amenable to the INCLUDE statement in SPSS.

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. The read.spss() in package foreign for reading .sav uses its own internal numbering scheme 1, 2, 3, ... for the levels of a factor. Consequently, changes in factor code can cause discrepancies in factor level when re-imported to SPSS. The solution is to manually recode the factor level in SPSS.

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 not do any pooling.
Value

The return value is NULL.

Author(s)

Stef van Buuren, dec 2010.

See Also

mids

mira-class

Multiply imputed repeated analyses (mira)

Description

The mira object is generated by the with.mids() function. 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. Pooling requires that coef() and vcov() methods are available for fitted object. The mira class of objects has methods for the following generic functions: print, summary.

Slots

#'

Object of class "list" containing the following slots:

data: 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.

Note

Many of the functions of the mice package do not use the S4 class definitions, and instead rely on the S3 list equivalent oldClass(obj) <- "mira".

Author(s)

Stef van Buuren, Karin Groothuis-Oudshoorn, 2000

References

See Also

with.mids, mids, mipo

Arguments

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.

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 `formulas`. 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

```r
# 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**

```r
ccc(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
```

---

**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 $H_0(T)$ 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**


Examples

```r
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, 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**  

_HANES 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**


**See Also**

nhanes2
Examples

```r
imp <- mice(nhanes2)  # create 5 imputed data sets
complete(imp)          # print the first imputed data set
```

---

**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/m²)
- **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**


**See Also**

- `nhanes`

**Examples**

```r
imp <- mice(nhanes2)  # create 5 imputed data sets
complete(imp)          # print the first imputed data set
```
nic  

**Number of incomplete cases**

**Description**
Calculates the number of incomplete cases.

**Usage**
```r
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**
```r
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**
```r
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.

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 impu-
tation method (see method argument). By default each variable is placed into
its own block, which is effectively fully conditional specification (FCS) by uni-
variate 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 \((n \times p)\) 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


parlmice

*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.

Usage

```r
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 imputed 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, 2018, based on an earlier version by Rianne Schouten and Gerko Vink, 2017.

References


See Also

parallel, parLapply, makeCluster, mice, mids-class
Examples

```r
# 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)
```

<table>
<thead>
<tr>
<th>pattern</th>
<th>Datasets with various missing data patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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


Details

Van Buuren (2012) uses these four artificial datasets to illustrate various missing data patterns.

Examples

```r
require(lattice)
require(MASS)

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 <- levelplot(r-var+rec|as.factor(pat), data=mdpat,
as.table=TRUE, aspect="iso",
shrink=c(0.9),
col.regions = mdc(1:2),
```
plot.mids

Plot the trace lines of the MICE algorithm

Description

Trace line plots portray the value of an estimate against the iteration number. The estimate can be anything that you can calculate, but typically are chosen as parameter of scientific interest. The plot method for a mids object plots the mean and standard deviation of the imputed (not observed) values against the iteration number for each of the $m$ replications. By default, the function plot the development of the mean and standard deviation for each incomplete variable. On convergence, the streams should intermingle and be free of any trend.

Usage

```r
## S3 method for class 'mids'
plot(x, y = NULL, theme = mice.theme(), layout = c(2, 3), type = "l", col = 1:10, lty = 1, ...)
```

Arguments

- `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().
pool

A vector of length 2 given the number of columns and rows in the plot. The default is c(2, 3).

Parameter type of `panel.xyplot`.

Parameter col of `panel.xyplot`.

Parameter lty of `panel.xyplot`.

Extra arguments for `xyplot`.

An object of class "trellis".

Stef van Buuren 2011

`mice, mids, xyplot`

The `pool()` function combines the estimates from \( m \) repeated complete data analyses. The typical sequence of steps to do 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 is an object of class `mipo`;
4. Optionally, compare pooled estimates from different scientific models by the `pool.compare()` function.

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.

`pool(object, dfcom = NULL)`
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. The default (dfcom = NULL) is to extract this information from the first fitted model. When that fails the warning "Large sample assumed" is printed, and the parameter is set dfcom = 999999. Use the dfcom parameter to specify the correct degrees of freedom.

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 function requires the following input from each fitted model:

1. the estimates of the model, usually obtainable by coef()
2. the standard error of each estimate;
3. the residual degrees of freedom of the model.

The pool() function relies on the broom::tidy and broom::glance function for extracting this information from a list of fitted models.

The degrees of freedom calculation uses the Barnard-Rubin adjustment for small samples (Barnard and Rubin, 1999).

Value

An object of class mipo, which stands for ‘multiple imputation pooled outcome’.

References


See Also

with.mids, as.mira, glance, tidy
Examples

# 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))

pool.compare

Compare two nested models fitted to imputed data

Description

Compares two nested models after m repeated complete data analysis

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

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 call1 is the call that created fit1. Component call12 is the call that created the
imputations. Component call1 is the call that created fit0. Component call0 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 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

See Also
lm.mids, glm.mids

Examples

### To compare two linear models:
imp <- mice(nhanes2, seed = 10009, print = FALSE)
mi1 <- with(data = imp, expr = lm(bmi ~ age + hyp + chl))
mi0 <- with(data = imp, expr = lm(bmi ~ age + hyp))
pc <- pool.compare(mi1, mi0)
pc$pvalue

### Comparison of two general linear models (logistic regression).
## Not run:
imp <- mice(boys, maxit = 2, print = FALSE)
fit1 <- with(imp, glm(as.factor(gen) ~ levels(gen)[1] ~ hgt + hc + reg, family = binomial))
fit0 <- with(imp, glm(as.factor(gen) ~ levels(gen)[1] ~ hgt + hc, family = binomial))
pool.compare(fit1, fit0, method = 'likelihood')$pvalue

# using factors
fit1 <- with(imp, glm(as.factor(gen) ~ levels(gen)[1] ~ hgt + hc + reg, family = binomial))
fit0 <- with(imp, glm(as.factor(gen) ~ levels(gen)[1] ~ hgt + hc, family = binomial))
pool.compare(fit1, fit0, method = 'likelihood')$pvalue

## End(Not run)
Description

Pools R^2 of m repeated complete data models.

Usage

pool.r.squared(object, adjusted = FALSE)

Arguments

object An object of class 'mira', produced by lm.mids or with.mids with lm as modeling function.

adjusted A logical value. If adjusted=TRUE then the adjusted R^2 is calculated. The default value is FALSE.

Details

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.

Value

Returns a 1x4 table with components. Component est is the pooled R^2 estimate. Component lo95 is the 95 % lower bound of the pooled R^2. Component hi95 is the 95 % upper bound of the pooled R^2. Component fmi is the fraction of missing information due to nonresponse.

Author(s)

Karin Groothuis-Oudshoorn and Stef van Buuren, 2009

References


See Also

pool.pool.scalar
Examples

```r
imp<-mice(nhanes)

fit<-lm.mids(chl~age+hyp+bmi,imp)
pool.r.squared(fit)
pool.r.squared(fit,adjusted=TRUE)

#fit<-lm.mids(chl~age+hyp+bmi,imp)
#
#> pool.r.squared(fit)
#  est  lo 95  hi 95  fmi
#  R^2 0.5108041 0.1479687 0.7791927 0.3024413
#
#> pool.r.squared(fit,adjusted=TRUE)
#  est  lo 95  hi 95  fmi
#  adj R^2 0.4398066 0.08251427 0.743172 0.3404165
#
```

Description

Pools univariate estimates of m repeated complete data analysis

Usage

```r
pool.scalar(Q, U, n = Inf, k = 1)
```

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.

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 nonresponse and the fraction of missing information.
Value

Returns a list with components. Component \(m\) is the number of imputations. Component \(\text{qhat}\) contains the \(m\) univariate estimates of repeated complete data analyses. Component \(u\) contains the corresponding \(m\) variances of the univariate estimates. Component \(\text{qbar}\) is the pooled univariate estimate, formula (3.1.2) Rubin (1987). Component \(\text{ubar}\) is the mean of the variances (i.e. the pooled within-imputation variance), formula (3.1.3) Rubin (1987). Component \(b\) is the between-imputation variance, formula (3.1.4) Rubin (1987). Component \(t\) is the total variance of the pooled estimated, formula (3.1.5) Rubin (1987). Component \(r\) is the relative increase in variance due to nonresponse, formula (3.1.7) Rubin (1987). Component \(df\) is the degrees of freedom for \(t\) reference distribution, formula (3.1.6) Rubin (1987) or method of Barnard-Rubin (1999) (if method = "smallsample"). Component \(\text{fmi}\) is the fraction missing information due to nonresponse, formula (3.1.10) Rubin (1987).

Author(s)

Karin Groothuis-Oudshoorn and Stef van Buuren, 2009

References


See Also

pool

Examples

```r
imp <- mice(hanhes)
m <- imp$m
Q <- rep(NA, m)
U <- rep(NA, m)
for (i in 1:m) {
  Q[i] <- mean(complete(imp, i)$bmi)
  U[i] <- var(complete(imp, i)$bmi) / nrow(hanhes) # (standard error of estimate)^2
}
pool.scalar(Q, U, n = nrow(hanhes), k = 1) # Barnard-Rubin 1999
```

Description

Hox pupil popularity data with some missing popularity scores
**pops**  

**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**


**Examples**

```r
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 a 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.

Source


Examples

```r
pops <- data(pops)
```

<table>
<thead>
<tr>
<th>potthoffroy</th>
<th>Potthoff-Roy data</th>
</tr>
</thead>
</table>

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
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-pteromaxillary 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


Examples

```r
### 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.mads**

*Print a mads object*

**Description**

Print a mads object
print.mids

Usage

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

Arguments

- `x`: Object of class `mads`
- `...`: Other parameters passed down to `print.default()`

Value

`NULL`

See Also

- `mads`
- `mira`
- `mice.anova`
- `summary.mice.anova`

print.mids  Print a mids object

Description

Print a mids object
- Print a mira object
- Print a mice.anova object
- Print a summary.mice.anova object

Usage

```r
## S3 method for class 'mids'
print(x, ...)
```

```r
## S3 method for class 'mira'
print(x, ...)
```

```r
## S3 method for class 'mice.anova'
print(x, ...)
```

```r
## S3 method for class 'mice.anova.summary'
print(x, ...)
```

Arguments

- `x`: Object of class `mids`, `mira` or `mipo`
- `...`: Other parameters passed down to `print.default()`
**quickpred**  

**Value**

NULL

NULL

NULL

NULL

NULL

**See Also**

mids

mira

mipo

mipo

---

**quickpred**  

*Quick selection of predictors from the data*

**Description**

Selects predictors according to simple statistics

**Usage**

```r
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 value for `mincor` can be useful for auxiliary, less important, variables. The set of predictors 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)`.

Author(s)

Stef van Buuren, Aug 2009

References


See Also

*mice*, *mids*

Examples

```r
# default: include all predictors with absolute correlation over 0.1
```
quickpred(hanhes)

# all predictors with absolute correlation over 0.4
quickpred(hanhes, mincor=0.4)

# include age and bmi, exclude chl
quickpred(hanhes, mincor=0.4, inc=c('age','bmi'), exc='chl')

# only include predictors with at least 30% usable cases
quickpred(hanhes, minpuc=0.3)

# use low threshold for bmi, and high thresholds for hyp and chl
pred <- quickpred(hanhes, mincor=c(0,0.1,0.5,0.5))
pred

# use it directly from mice
imp <- mice(hanhes, pred=quickpred(hanhes, minpuc=0.25, include='age'))

---

**rbind.mids**

*Combine mids objects by rows*

**Description**

This function combines two mids objects rowwise into a single mids object, or combines a mids object with a vector, matrix, factor or dataframe rowwise into a mids object.

**Usage**

rbind.mids(x, y = NULL, ...)

**Arguments**

- **x**
  - A mids object.

- **y**
  - A mids object, or a data.frame, matrix, factor or vector.

- **...**
  - Additional data.frame, matrix, vector or factor. These can be given as named arguments.

**Details**

If y is a mids object, then rbind requires that the number of multiple imputations in x and y is identical. Also, columns of x$data and y$data should match.

If y is not a mids object, the columns of x$data and y should match. The where matrix for y is set to FALSE, signaling that any missing values in y were not imputed.

**Value**

An S3 object of class mids
Note

The 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
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


See Also

cbind.mids, ibind.mids

Examples

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))))
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

Examples

```r
md.pattern(selfreport[, , "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$method
meth[["bmi"]]<- "-bmi(hm,wm)"
pred <- init$prediction
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)
x <- xy.coords(cd$bm, cd$br- cd$bm)
plot(x, 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(lty=1, lwd=0.7)

idx <- cd$src=="krul"
xyc <- x; xyc$x <- xyc$x[idx]; xyc$y <- xyc$y[idx]
ys <- x; ys$x <- ys$x[idx]; y$y <- y$y[idx]
points(xyc, col=mdc(1), cex=0.7)
points(ys, col=mdc(2), cex=0.7)
lines(lowess(xyc), col=mdc(4), lwd=2)
lines(lowess(ys), col=mdc(5), lwd=2)
text(lab=1, 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 \(\text{bounds}[1]\) by \(\text{bounds}[1]\), and replaces any values higher than \(\text{bounds}[2]\) by \(\text{bounds}[2]\).

**Usage**

\[
\text{squeeze}(x, \text{bounds} = c(\text{min}(x[r]), \text{max}(x[r])), r = \text{rep.int(TRUE, length}(x)))
\]

**Arguments**

- \(x\)  
  A numerical vector with values
- \(\text{bounds}\)  
  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 \(\text{length}(x)\) that is used to select a subset in \(x\) before calculating automatic bounds.

**Value**

A vector of length \(\text{length}(x)\).

**Author(s)**

Stef van Buuren, 2011.

---

**stripplot.mids**  
*Stripplot of observed and imputed data*

**Description**

Plotting methods for imputed data using **lattice**. \texttt{stripplot} produces one-dimensional scatterplots. The function automatically separates the observed and imputed data. The functions extend the usual features of **lattice**.
Usage

```r
## 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 method can be used to subsequently update components of the object, and the print 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


See Also

`mice`, `xyplot`, `densityplot`, `bwplot`, `lattice` for an overview of the package, as well as `stripplot`, `panel.stripplot`, `print.trellis`, `trellis.par.set`

Examples

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

### stripplot, all numerical variables
## Not run: stripplot(imp)

### same, but with improved display
## Not run: stripplot(imp, col=c("grey",mdc(2)),pch=c(1,20))

### 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(hcl(0,0,40,0.2), mdc(2)),pch=c(1,20))

### 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)

### 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

Summary of a mids object

Summary of a mads object

Print a mice.anova object

**Usage**

## S3 method for class 'mira'
summary(object, type = c("tidy", "glance", "summary"),
         ...)    

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

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

## 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".
- **...**: Other parameters passed down to print() and summary()
Value

NULL
NULL
NULL
NULL
NULL

See Also

mira
mids
mads
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()
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)

_tbcNtarget_ 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 _tbcNtarget_ 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


**Examples**

```r
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 separation, 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**

References


---

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 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 (2012) chapter 7 for more details on the imputation methodology.

References


Examples

```r
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"))
  }
}
```
```r
tau <- rbind(tau, getfit(cors, s=TRUE))  # 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 7.8 van Buuren (2012) 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.
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 with 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.

...  

Additional parameters passed to expr

Value

A list object of S3 class mira

Author(s)

Karin Oudshoorn, Stef van Buuren 2009-2012

References

See Also

mids, mira, pool, pool.compare, pool.r.squared

Examples

imp <- mice(nhanes2)
fit1 <- with(data=imp, exp=lm(bmi~age+hyp+chl))
fit2 <- with(data=imp, exp=glm(hyp~age+bmi+chl, family=binomial))
anova.imp <- with(data=imp, exp=anova(lm(bmi~age+hyp+chl)))

```r
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

```r
## 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-class or the vignette Multivariate Amputation using Ampute to understand the contents of class object mads.

Author(s)
Rianne Schouten, 2016

See Also
ampute, bwplot, Lattice for an overview of the package, mads-class

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'
```r
xyplot(x, data, na.groups = NULL, groups = NULL,
       as.table = TRUE, theme = mice.theme(), allow.multiple = TRUE,
       outer = TRUE,
       drop.unused.levels = lattice::latticegetOption("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 \( y_1 \sim x \mid a * b \) and \( y_2 \sim x \mid a * b \), but with the \( y_1 \sim x \) and \( y_2 \sim x \) in separate panels. This behavior differs from standard \texttt{lattice}. \textit{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.

\textbf{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 is the response indicator \texttt{is.na(x$data)}. The default \texttt{na.group = NULL} contrasts the observed and missing data in the LHS y variable of the display, i.e. groups created by \texttt{is.na(y)}. The expression y creates the groups according to \texttt{is.na(y)}. The expression \( y_1 \& y_2 \) creates groups by \texttt{is.na(y1) \& is.na(y2)}, and \( y_1 \mid y_2 \) creates groups as \texttt{is.na(y1) \mid is.na(y2)}, and so on.

\textbf{groups} 
This is the usual \texttt{groups} arguments in \texttt{lattice}. It differs from \texttt{na.groups} because it evaluates in the completed data \texttt{data.frame(complete(x, "long", inc=TRUE))} (as usual), whereas \texttt{na.groups} evaluates in the response indicator. See \texttt{xyplot} for more details. When both \texttt{na.groups} and \texttt{groups} are specified, \texttt{na.groups} takes precedence, and \texttt{groups} is ignored.

\textbf{as.table} 
See \texttt{xyplot}.

\textbf{theme} 
A named list containing the graphical parameters. The default function \texttt{mice.theme} produces a short list of default colors, line width, and so on. The extensive list may be obtained from \texttt{trellis.par.get()}. Global graphical parameters like \texttt{col} or \texttt{cex} in high-level calls are still honored, so first experiment with the global parameters. Many setting consists of a pair. For example, \texttt{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.

\textbf{allow.multiple} 
See \texttt{xyplot}.

\textbf{outer} 
See \texttt{xyplot}.

\textbf{drop.unused.levels} 
See \texttt{xyplot}.

\textbf{...} 
Further arguments, usually not directly processed by the high-level functions documented here, but instead passed on to other functions.

\textbf{subscripts} 
See \texttt{xyplot}.

\textbf{subset} 
See \texttt{xyplot}.

\textbf{Details} 
The argument \texttt{na.groups} may be used to specify (combinations of) missingness in any of the variables. The argument \texttt{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, \texttt{na.groups} takes precedence over \texttt{groups}.

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

Graphical parameters like \texttt{col}, \texttt{pch} and \texttt{cex} can be specified in the arguments list to alter the plotting symbols. If \texttt{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 method can be used to subsequently update components of the object, and the print 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

See Also
mice, stripplot, densityplot, bwplot, lattice for an overview of the package, as well as xyplot, panel.xyplot, print.trellis, trellis.par.set

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

*Topic classes
  mids-class, 115
  mira-class, 119
*Topic datagen
  mice.impute.21.bin, 72
  mice.impute.21.lmer, 73
  mice.impute.21.norm, 75
  mice.impute.2lonly.mean, 78
  mice.impute.cart, 84
  mice.impute.jomoImpute, 85
  mice.impute.lda, 87
  mice.impute.logreg, 88
  mice.impute.logreg.boot, 90
  mice.impute.mean, 91
  mice.impute.midastouch, 92
  mice.impute.norm, 94
  mice.impute.norm.boot, 96
  mice.impute.norm.nob, 97
  mice.impute.norm.predict, 98
  mice.impute.panImpute, 99
  mice.impute.passive, 101
  mice.impute.pmm, 102
  mice.impute.polr, 104
  mice.impute.polyreg, 106
  mice.impute.quadratic, 107
  mice.impute.rf, 109
  mice.impute.ri, 111
  mice.impute.sample, 112
*Topic datasets
  boys, 14
  brandsma, 15
  employee, 32
  fdd, 35
  fdgs, 37
  leiden85, 51
  mammalsleep, 61
  nhanes, 124
  nhanes2, 125
  pattern, 130
  popmis, 138
  pops, 139
  potthoffroy, 140
  selfreport, 147
  tbc, 155
  toenail, 156
  walking, 158
  windspeed, 159
*Topic hplot
  bwplot.mids, 18
  densityplot.mids, 29
  mdc, 64
  stripplot.mids, 149
  supports.transparent, 154
  xyplot.mids, 162
*Topic htest
  pool, 132
  pool.compare, 134
  pool.r.squared, 136
  pool.scalar, 137
*Topic iteration
  mice, 66
  mice.mids, 113
*Topic manip
  cbind.mids, 21
  complete, 25
  getfit, 43
  ibind, 46
  mids2mplus, 117
  mids2spss, 118
  rbind.mids, 145
*Topic mids
  as.mids, 11
*Topic misc
  fico, 38
  flux, 40
  fluxplot, 42
  nelsonaalen, 123
  quickpred, 143
INDEX

version, 157

*Topic **multivariate**
glm.mids, 45
lm.mids, 51
with.mids, 160

*Topic **univar**
cc, 23
cci, 24
ici, 48
md.pairs, 62
md.pattern, 63
.norm.draw (norm.draw), 127
.pmm.match, 5
21.pan (mice.impute.21.pan), 76
2lonly.norm (mice.impute.2lonly.norm), 80
2lonly.pmm (mice.impute.2lonly.pmm), 82
ampute, 6, 17, 18, 54, 71, 161, 162
ampute.continuous, 7
ampute.default.freq, 6
ampute.default.odds, 7
ampute.default.patterns, 6
ampute.default.type, 7
ampute.default.weights, 6
ampute.discrete, 7
anova.mira, 9
appendbreak, 10
as.mids, 11
as.mira, 12, 133
as.mitml.result, 13

boys, 14
brandsma, 15
bwplot, 9, 18, 20, 32, 152, 162, 164
bwplot (bwplot.mids), 18
bwplot.mads, 17
bwplot.mids, 18
cart (mice.impute.cart), 84
cbind, 22
cbind.mids, 21, 46, 146
cc, 23, 24, 47
cci, 23, 24, 48, 122, 126
complete, 25, 71, 114
complete.cases, 24
construct.blocks, 26
D1, 27

D2, 28
D3, 29
data.enders.employee, 33
densityplot, 20, 32, 152, 164
densityplot (densityplot.mids), 29
densityplot.mids, 29
dev.capabilities, 154

employee, 32
estimice, 33
extractBS, 34

fdd, 35
fdgs, 37
fico, 38, 41, 43
fix.coef, 39
flux, 39, 40, 43
fluxplot, 39, 41, 42
formula, 45, 52, 160

gc, 129
gtfit, 43
gtqbar, 44
glance, 133
glm, 45, 89, 91
glm.fit, 89, 91
glm.mids, 45, 135

hazard (nelsonaalen), 123
hcl, 65, 161

ibind, 22, 46, 129, 146
ic, 47, 48
ici, 23, 24, 47, 48
ici, data.frame-method (ici), 48
ici, matrix-method (ici), 48
ici, mids-method (ici), 48
is.mads, 48
is.mids, 49
is.mipo, 49
is.mira, 50
is.mitml.result, 50

jomoImpute, 86

Lattice, 18, 162
lattice, 20, 32, 152, 161, 164
lda, 88
leiden85, 51
lm, 45, 52, 160
lm.mids, 51, 135
mads, 142, 154
mads-class, 53
make.blocks, 27, 54, 56, 59, 61
make.blocks, 55
make.formulas, 56
make.method, 57
make.post, 58
make.predictorMatrix, 56, 59, 61
make.visitsesequence, 59
make.where, 60
makeCluster, 129
mammalsleep, 61
md.pairs, 62
md.pattern, 6, 39, 41, 43, 63
mice, 64, 154
mean, 92
mgg (selfreport), 147
mice, 6, 7, 9, 20, 26, 32, 58, 60, 66, 71, 85, 88, 89, 91, 92, 98, 102, 106, 107, 110, 114, 116, 120, 121, 127–129, 132, 144, 152, 164
mice-package (mice), 66
mice.impute.2l.bin, 72, 74, 76, 77
mice.impute.2l.lmer, 73, 73, 76, 77
mice.impute.2l.norm, 73, 74, 75, 77
mice.impute.2l.pan, 73, 74, 76, 76, 80–83
mice.impute.ZImonly.mean, 78, 81, 83
mice.impute.ZImonly.norm, 79, 80, 83
mice.impute.ZImonly.pmm, 79, 81, 82
mice.impute.jomoImpute, 85, 100
mice.impute.panImpute, 86, 99
mice.impute.passive, 101
mice.impute.quadratic, 85, 88, 89, 91, 92, 94–96, 98, 99, 103, 106, 107, 107, 110, 112
mice.impute.rfcat, 110
mice.impute.rfcont, 110
mice.impute.ri, 85, 88, 89, 91, 92, 94–96, 98, 99, 103, 106–108, 110, 111
mice.impute.sample, 112
mice.mids, 113
mice.theme, 114
mids, 22, 26, 45, 46, 52, 70, 71, 105, 114, 117, 119, 120, 132, 143, 144, 146, 154, 161
mids (mids-class), 115
mids-class, 115
mids2plus, 117
mids2spss, 117, 118
mipo, 116, 120, 143, 154
mira, 13, 44, 45, 52, 116, 143, 154, 161
mira (mira-class), 119
mira-class, 119
multinom, 106, 107
na.omit, 23
name.blocks, 27, 120
name.formulas, 121
ncc, 122, 126
nelsonaalen, 123
nhanes, 124, 125
nhanes2, 124, 125
nic, 122, 126
nim, 126
norm (mice.impute.norm), 94
norm.boot (mice.impute.norm.boot), 96
norm.draw, 127
norm.nob (mice.impute.norm.nob), 97
norm.predict
(mice.impute.norm.predict), 98

panel.bwplot, 20
panel.densityplot, 32
panel.stripplot, 152
panel.xyplot, 132, 151, 164
panImpute, 100
parallel, 129
parLapply, 129
parLMice, 128
pattern, 130
pattern1 (pattern), 130
pattern2 (pattern), 130
pattern3 (pattern), 130
pattern4 (pattern), 130
plot.mids, 131
pmm (mice.impute.pmm), 102
polr, 106, 107
pool, 71, 132, 136, 138, 161
pool.compare, 134, 161
pool.r.squared, 136, 161
pool.scalar, 136, 137
popmis, 138
pops, 139
potthoffroy, 140
print, 20, 31, 151, 164
print.mads, 141
print.mice.anova (print.mads), 142
print.mids, 142
print.mira (print.mids), 142
print.trellis, 20, 32, 152, 164

quadratic (mice.impute.quadratic), 107
quickpred, 143

randomForest, 110
rbind.mids, 22, 46, 145
rgb, 65
ri (mice.impute.ri), 111