Package ‘miceadds’

January 9, 2024

Type Package

Title Some Additional Multiple Imputation Functions, Especially for 'mice'

Version 3.17-44

Date 2024-01-08 19:08:24

Author Alexander Robitzsch [aut,cre] (<`https://orcid.org/0000-0002-8226-3132`>),
Simon Grund [aut] (<`https://orcid.org/0000-0002-1290-8986`>),
Thorsten Henke [ctb]

Maintainer Alexander Robitzsch <`robitzsch@ipn.uni-kiel.de`>

Description Contains functions for multiple imputation which complements existing functionality in R.
In particular, several imputation methods for the mice package (van Buuren & Groothuis-Oudshoorn, 2011, <doi:10.18637/jss.v045.i03>) are implemented.
Main features of the miceadds package include plausible value imputation (Mislevy, 1991, <doi:10.1007/BF02294457>), multilevel imputation for variables at any level or with any number of hierarchical and non-hierarchical levels (Grund, Luedtke & Robitzsch, 2018, <doi:10.1177/1094428117703686>; van Buuren, 2018, Ch.7, <doi:10.1201/9780429492259>), imputation using partial least squares (PLS) for high dimensional predictors (Robitzsch, Pham & Yanagida, 2016), nested multiple imputation (Rubin, 2003, <doi:10.1111/1467-9574.00217>), substantive model compatible imputation (Bartlett et al., 2015, <doi:10.1177/0962280214521348>), and features for the generation of synthetic datasets (Reiter, 2005, <doi:10.1111/j.1467-985X.2004.00343.x>; Nowok, Raab, & Dibben, 2016, <doi:10.18637/jss.v074.i11>).

Depends R (>= 3.5-0), mice (>= 3.0.0)

Imports graphics, methods, mitools, Rcpp, stats, utils

LinkingTo Rcpp, RcppArmadillo

Suggests BIFIEsurvey, blme, car, CDM, coda, foreign, inline, lme4, MASS, Matrix, MBESS, MCMCglmm, mdmb, pls, numDeriv, readxl, sandwich, sirt, sjlabelled, synthpop, TAM

Enhances Amelia, imputeR, jomo, micemd, mitml, pan, simputation

URL `https://github.com/alexanderrobitzsch/miceadds`,
`https://sites.google.com/view/alexander-robitzsch/software`

License GPL (>= 2)

BugReports `https://github.com/alexanderrobitzsch/miceadds/issues?state=open`

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-01-09 10:10:02 UTC

R topics documented:

`miceadds-package` *Some Additional Multiple Imputation Functions, Especially for 'mice'*

Description

Contains functions for multiple imputation which complements existing functionality in R. In particular, several imputation methods for the mice package (van Buuren & Groothuis-Oudshoorn, 2011, <doi:10.18637/jss.v045.i03>) are implemented. Main features of the miceadds package include plausible value imputation (Mislevy, 1991, <doi:10.1007/BF02294457>), multilevel imputation for variables at any level or with any number of hierarchical and non-hierarchical levels (Grund, Luedtke & Robitzsch, 2018, <doi:10.1177/1094428117703686>; van Buuren, 2018, Ch.7, <doi:10.1201/9780429492259>), imputation using partial least squares (PLS) for high dimensional predictors (Robitzsch, Pham & Yanagida, 2016), nested multiple imputation (Rubin, 2003, <doi:10.1111/1467-9574.00217>), substantive model compatible imputation (Bartlett et al., 2015, <doi:10.1177/0962280214521348>), and features for the generation of synthetic datasets (Reiter, 2005, <doi:10.1111/j.1467-985X.2004.00343.x>; Nowok, Raab, & Dibben, 2016, <doi:10.18637/jss.v074.i11>).

Details

- The **miceadds** package contains some functionality for imputation of multilevel data. The function `mice.impute.ml.lmer` is a general function for imputing multilevel data with hierarchical or cross-classified structures for variables at an arbitrary level. This imputation method uses the `lme4::lmer` function in the **lme4** package. The imputation method `mice.impute.2lonly.function` conducts an imputation for a variable at a higher level for already defined imputation methods in the **mice** package. Two-level imputation is available in several functions in the **mice** package (`mice::mice.impute.2l.pan`, `mice::mice.impute.2l.norm`) as well in **micemd** and **hmi** packages. The **miceadds** package contains additional imputation methods for two-level datasets: `mice.impute.2l.continuous` for normally distributed data, `mice.impute.2l.pmm` for predictive mean matching in multilevel models and `mice.impute.2l.binary` for binary data.
- In addition to the usual `mice` imputation function which employs parallel chains, the function `mice.1chain` does multiple imputation from a single chain.
- Nested multiple imputation can be conducted with `mice.nmi`. The function `NMIcombine` conducts statistical inference for nested multiply imputed datasets.
- Imputation based on partial least squares regression is implemented in `mice.impute.pls`.
- Unidimensional plausible value imputation for latent variables (or variables with measurement error) in the **mice** sequential imputation framework can be applied by using the method `mice.impute.plausible.values`.
- Substantive model compatible multiple imputation using fully conditional specification can be conducted with `mice.impute.smcfcs`.
- The function `syn_mice` allows the generation of synthetic datasets with imputation methods for **mice**. It has similar functionality as the **synthpop** package (Nowok, Raab, & Dibben, 2016). The function `mice.impute.synthpop` allows the usage of **synthpop** synthesization methods in **mice**, while `syn.mice` allows the usage of **mice** imputation methods in **synthpop**.
- The method `mice.impute.simputation` is a wrapper function to imputation methods in the **simputation** package. The methods `mice.impute.imputeR.lmFun` and `mice.impute.imputeR.cFun` are wrapper functions to imputation methods in the **imputeR** package.
- The **miceadds** package also includes some functions R utility functions (e.g. `write.pspp`, `ma.scale2`).
- Imputations for questionnaire items can be accomplished by two-way imputation (`tw.imputation`).

Author(s)

Alexander Robitzsch [aut,cre] (<https://orcid.org/0000-0002-8226-3132>), Simon Grund [aut] (<https://orcid.org/0000-0002-1290-8986>), Thorsten Henke [ctb]

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

References

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

Grund, S., Luedtke, O., & Robitzsch, A. (2018). Multiple imputation of multilevel data in organizational research. *Organizational Research Methods, 21*(1), 111-149. doi:10.1177/1094428117703686

Mislevy, R. J. (1991). Randomization-based inference about latent variables from complex samples. *Psychometrika, 56*(2), 177-196. doi:10.1007/BF02294457

Nowok, B., Raab, G., & Dibben, C. (2016). **synthpop**: Bespoke creation of synthetic data in R. *Journal of Statistical Software, 74*(11), 1-26. doi:10.18637/jss.v074.i11

Reiter, J. P. (2005) Releasing multiply-imputed, synthetic public use microdata: An illustration and empirical study. *Journal of the Royal Statistical Society, Series A, 168*(1), 185-205. doi:10.1111/j.1467985X.2004.00343.x

Robitzsch, A., Pham, G., & Yanagida, T. (2016). Fehlende Daten und Plausible Values. In S. Breit & C. Schreiner (Hrsg.). *Large-Scale Assessment mit R: Methodische Grundlagen der oesterreichischen Bildungsstandardueberpruefung* (S. 259-293). Wien: facultas.

Rubin, D. B. (2003). Nested multiple imputation of NMES via partially incompatible MCMC. *Statistica Neerlandica, 57*(1), 3-18. doi:10.1111/14679574.00217

van Buuren, S. (2018). *Flexible imputation of missing data*. Boca Raton: CRC Press. doi:10.1201/9780429492259

van Buuren, S., & Groothuis-Oudshoorn, K. (2011). **mice**: Multivariate imputation by chained equations in R. *Journal of Statistical Software, 45*(3), 1-67. doi:10.18637/jss.v045.i03

See Also

See also the CRAN task view *Missing Data*:
`https://CRAN.R-project.org/view=MissingData`

See other R packages for conducting multiple imputation: **mice**, **Amelia**, **pan**, **mi**, **norm**, **norm2**, **BaBooN**, **VIM**, ...

Some links to internet sites related to missing data:

http://missingdata.lshtm.ac.uk/
http://www.stefvanbuuren.nl/mi/
http://www.bristol.ac.uk/cmm/software/realcom/
https://rmisstastic.netlify.com/

Examples

```
##
##   ::'''''''''''''''''''''''''''''''''::
##   :: miceadds 0.11-69 (2013-12-01)   ::
##   ::'''''''''''''''''''''''''''''''''::
##
##  ----------------------- mice at work ---------------------------------
##
##                         (\-.
##                         / _`> .---------.
##                 _)     / _)=  |'-------'|
##                (      / _/    |O   O   o|
```

```
##                  `-.__(___)_   | o 0 . o |
##                                `---------'
##
##                                           oo__
##                                          <;___)------
##                                     oo__   " "
##                                    <;___)------      oo__
##                                      " "            <;___)------
##                                                       " "
```

`complete.miceadds` *Creates Imputed Dataset from a* `mids.nmi` *or* `mids.1chain` *Object*

Description

Creates imputed dataset from a `mids.nmi` or `mids.1chain` object.

Usage

```
## S3 method for class 'mids.nmi'
complete(data, action=c(1,1), ...)

## S3 method for class 'mids.1chain'
complete(data, action=1, ...)
```

Arguments

`data`	Object of class `mids.nmi` (for `complete.mids.nmi`) or `mids.1chain` (for `complete.mids.1chain`)
`action`	A vector of length two indicating to indices of between and within imputed dataset for for `complete.mids.nmi` and an integer for the index of imputed dataset for `complete.mids.1chain`.
`...`	More arguments to be passed

See Also

See also the corresponding `mice::complete` function and `mitml::mitmlComplete`.

Imputation methods: `mice.nmi`, `mice.1chain`

Examples

```
## Not run:
################################################################################
# EXAMPLE 1: Nested multiple imputation and dataset extraction for TIMSS data
################################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
```

```
# remove first four variables
M <- length(datlist)
for (ll in 1:M){
    datlist[[ll]] <- datlist[[ll]][, -c(1:4) ]
}

#***************
# (1) nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist,  m=4, maxit=3 )
summary(imp1)

#***************
# (2) nested multiple imputation using mice.1chain
imp2 <- miceadds::mice.nmi( datlist, Nimp=4, burnin=10,iter=22, type="mice.1chain")
summary(imp2)

#**************
# extract dataset for third orginal dataset the second within imputation
dat32a <- miceadds::complete.mids.nmi( imp1, action=c(3,2) )
dat32b <- miceadds::complete.mids.nmi( imp2, action=c(3,2) )

#############################################################################
# EXAMPLE 2: Imputation from one chain and extracting dataset for nhanes data
#############################################################################

library(mice)
data(nhanes, package="mice")

# nhanes data in one chain
imp1 <- miceadds::mice.1chain( nhanes, burnin=5, iter=40, Nimp=4,
            method=rep("norm", 4 ) )

# extract first imputed dataset
dati1 <- miceadds::complete.mids.1chain( imp1, action=1 )

## End(Not run)
```

`crlrem` R *Utilities: Removing CF Line Endings*

Description

This function removes CF line endings from a text file and writes the processed file in the working directory.

Usage

```
crlrem( filename1, filename2 )
```

Arguments

`filename1`	Name of the original file (possibly with CF line endings)
`filename2`	Name of the processed file (without CF line endings)

Author(s)

This is code by Dirk Eddelbuettel copied from `https://stat.ethz.ch/pipermail/r-devel/2010-September/058480.html`

Examples

```
## Not run:
filename1 <- "rm.arraymult__0.02.cpp"
filename2 <- "rm.arraymult__0.03.cpp"
crlrem( filename1, filename2 )
## End(Not run)
```

`cxxfunction.copy` R *Utilities: Copy of an* **Rcpp** *File*

Description

Copies the **Rcpp** function into the working directory.

Usage

```
cxxfunction.copy(cppfct, name)
```

Arguments

`cppfct`	Rcpp function
`name`	Name of the output Rcpp function to be generated

References

Eddelbuettel, D. & Francois, R. (2011). **Rcpp**: Seamless R and **C++** integration. *Journal of Statistical Software, 40*(8), 1-18. doi:10.18637/jss.v040.i08

See Also

`inline::cxxfunction`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Rcpp code logistic distribution
#############################################################################

library(Rcpp)
library(inline)

# define Rcpp file
code1 <- "
    // input array A
    Rcpp::NumericMatrix AA(A);
    // Rcpp::IntegerVector dimAA(dimA);
    int nrows=AA.nrow();
    int ncolumns=AA.ncol();
    Rcpp::NumericMatrix Alogis(nrows,ncolumns)  ;
    // compute logistic distribution
    for (int ii=0; ii<nrows; ii++){
        Rcpp::NumericVector h1=AA.row(ii) ;
        Rcpp::NumericVector res=plogis( h1 ) ;
        for (int jj=0;jj<ncolumns;jj++){
            Alogis(ii,jj)=res[jj] ;
                        }
                    }
    return( wrap(Alogis) );
    "
# compile Rcpp code
fct_rcpp <- inline::cxxfunction( signature( A="matrix"), code1,
              plugin="Rcpp", verbose=TRUE )
# copy function and save it as object 'calclogis'
name <- "calclogis"  # name of the function
cxxfunction.copy( cppfct=fct_rcpp, name=name )
# function is available as object named as name
Reval( paste0( name, " <- fct_rcpp " ) )
# test function
m1 <- outer( seq( -2, 2, len=10 ), seq(-1.5,1.5,len=4) )
calclogis(m1)

## End(Not run)
```

`data.allison` *Datasets from Allison's* Missing Data *Book*

Description

Datasets from Allison's missing data book (Allison 2002).

Usage

```
data(data.allison.gssexp)
data(data.allison.hip)
data(data.allison.usnews)
```

Format

- Data `data.allison.gssexp`:

  ```
  'data.frame': 2991 obs. of 14 variables:
  $ AGE : num 33 59 NA 59 21 22 40 25 41 45 ...
  $ EDUC : num 12 12 12 8 13 15 9 12 12 12 ...
  $ FEMALE : num 1 0 1 0 1 1 1 0 1 1 ...
  $ SPANKING: num 1 1 2 2 NA 1 3 1 1 NA ...
  $ INCOM : num 11.2 NA 16.2 18.8 13.8 ...
  $ NOCHILD : num 0 0 0 0 1 1 0 0 0 0 ...
  $ NODOUBT : num NA NA NA 1 NA NA 1 NA NA 1 ...
  $ NEVMAR : num 0 0 0 0 1 1 0 1 0 0 ...
  $ DIVSEP : num 1 0 0 0 0 0 0 0 0 1 ...
  $ WIDOW : num 0 0 0 0 0 0 1 0 1 0 ...
  $ BLACK : num 1 1 1 0 1 1 0 1 1 1 ...
  $ EAST : num 1 1 1 1 1 1 1 1 1 1 ...
  $ MIDWEST : num 0 0 0 0 0 0 0 0 0 0 ...
  $ SOUTH : num 0 0 0 0 0 0 0 0 0 0 ...
  ```

- Data `data.allison.hip`:

  ```
  'data.frame': 880 obs. of 7 variables:
  $ SID : num 1 1 1 1 2 2 2 2 9 9 ...
  $ WAVE: num 1 2 3 4 1 2 3 4 1 2 ...
  $ ADL : num 3 2 3 3 3 1 2 1 3 3 ...
  $ PAIN: num 0 5 0 0 0 1 5 NA 0 NA ...
  $ SRH : num 2 4 2 2 4 1 1 2 2 3 ...
  $ WALK: num 1 0 0 0 0 0 0 0 1 NA ...
  $ CESD: num 9 28 31 11.6 NA ...
  ```

- Data `data.allison.usnews`:

  ```
  'data.frame': 1302 obs. of 7 variables:
  $ CSAT : num 972 961 NA 881 NA ...
  $ ACT : num 20 22 NA 20 17 20 21 NA 24 26 ...
  $ STUFAC : num 11.9 10 9.5 13.7 14.3 32.8 18.9 18.7 16.7 14 ...
  $ GRADRAT: num 15 NA 39 NA 40 55 51 15 69 72 ...
  $ RMBRD : num 4.12 3.59 4.76 5.12 2.55 ...
  $ PRIVATE: num 1 0 0 0 0 1 0 0 0 1 ...
  $ LENROLL: num 4.01 6.83 4.49 7.06 6.89 ...
  ```

Source

The datasets were downloaded from *http://www.ats.ucla.edu/stat/examples/md/*.

References

Allison, P. D. (2002). *Missing data*. Newbury Park, CA: Sage.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Hip dataset | Imputation using a wide format
#############################################################################

# at first, the hip dataset is 'melted' for imputation

data(data.allison.hip)
  ##   head(data.allison.hip)
  ##     SID WAVE ADL PAIN SRH WALK   CESD
  ##   1   1    1   3    0   2    1  9.000
  ##   2   1    2   2    5   4    0 28.000
  ##   3   1    3   3    0   2    0 31.000
  ##   4   1    4   3    0   2    0 11.579
  ##   5   2    1   3    0   4    0     NA
  ##   6   2    2   1    1   1    0  2.222

library(reshape)
hip.wide <- reshape::reshape(data.allison.hip, idvar=”SID”, timevar=”WAVE”,
                direction=”wide”)
  ##   > head(hip.wide, 2)
  ##     SID ADL.1 PAIN.1 SRH.1 WALK.1 CESD.1 ADL.2 PAIN.2 SRH.2 WALK.2 CESD.2 ADL.3
  ##   1   1     3      0     2      1      9     2      5     4      0 28.000     3
  ##   5   2     3      0     4      0     NA     1      1     1      0  2.222     2
  ##     PAIN.3 SRH.3 WALK.3 CESD.3 ADL.4 PAIN.4 SRH.4 WALK.4 CESD.4
  ##   1      0     2      0     31     3      0     2      0 11.579
  ##   5      5     1      0     12     1     NA     2      0     NA

# imputation of the hip wide dataset
imp <- mice::mice( as.matrix( hip.wide[,-1] ), m=5, maxit=3 )
summary(imp)

## End(Not run)
```

`data.enders` *Datasets from Enders'* Missing Data *Book*

Description

Datasets from Enders' missing data book (2010).

Usage

```
data(data.enders.depression)
data(data.enders.eatingattitudes)
data(data.enders.employee)
```

Format

- Dataset `data.enders.depression`:

  ```
  'data.frame': 280 obs. of 8 variables:
  $ txgroup: int 0 0 0 0 0 0 0 0 0 0 ...
  $ dep1 : int 46 49 40 47 33 44 45 53 40 55 ...
  $ dep2 : int 44 42 28 47 33 41 43 35 43 45 ...
  $ dep3 : int 26 29 31 NA 34 34 34 35 35 36 ...
  $ r2 : int 0 0 0 0 0 0 0 0 0 0 ...
  $ r3 : int 0 0 0 1 0 0 0 0 0 0 ...
  $ pattern: int 3 3 3 2 3 3 3 3 3 3 ...
  $ dropout: int 0 0 0 1 0 0 0 0 0 0 ...
  ```

- Dataset `data.enders.eatingattitudes`:

  ```
  'data.frame': 400 obs. of 14 variables:
  $ id : num 1 2 3 4 5 6 7 8 9 10 ...
  $ eat1 : num 4 6 3 3 3 4 5 4 4 6 ...
  $ eat2 : num 4 5 3 3 2 5 4 3 7 5 ...
  $ eat10: num 4 6 2 4 3 4 4 4 6 5 ...
  $ eat11: num 4 6 2 3 3 5 4 4 5 5 ...
  $ eat12: num 4 6 3 4 3 4 4 4 4 6 ...
  $ eat14: num 4 7 2 4 3 4 4 4 6 6 ...
  $ eat24: num 3 6 3 3 3 4 4 4 4 5 ...
  $ eat3 : num 4 5 3 3 4 4 3 6 4 5 ...
  $ eat18: num 5 6 3 5 4 5 3 6 4 6 ...
  $ eat21: num 4 5 2 4 4 4 3 5 4 5 ...
  $ bmi : num 18.9 26 18.3 18.2 24.4 ...
  $ wsb : num 9 13 6 5 10 7 11 8 10 12 ...
  $ anx : num 11 19 8 14 7 11 12 12 14 12 ..
  ```

- Dataset `data.enders.employee`:

  ```
  'data.frame': 480 obs. of 9 variables:
  $ id : num 1 2 3 4 5 6 7 8 9 10 ...
  $ age : num 40 53 46 37 44 39 33 43 35 37 ...
  $ tenure : num 10 14 10 8 9 10 7 9 9 10 ...
  $ female : num 1 1 1 1 1 1 1 1 1 1 ...
  $ wbeing : num 8 6 NA 7 NA 7 NA 7 7 5 ...
  $ jobsat : num 8 5 7 NA 5 NA 5 NA 7 6 ...
  $ jobperf : num 6 5 7 5 5 7 7 7 7 6 ...
  $ turnover: num 0 0 0 0 0 0 0 0 1 0 ...
  $ iq : num 106 93 107 94 107 118 103 106 108 97 ...
  ```

Source

The datasets were downloaded from `https://www.appliedmissingdata.com/`.

References

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

`data.graham` *Datasets from Grahams* Missing Data *Book*

Description

Datasets from Grahams missing data book (2012).

Usage

```
data(data.graham.ex3)
data(data.graham.ex6)
data(data.graham.ex8a)
data(data.graham.ex8b)
data(data.graham.ex8c)
```

Format

- Dataset `data.graham.ex3`:

  ```
  'data.frame': 2756 obs. of 20 variables:
  $ school : int 1 1 1 1 1 1 1 1 1 1 ...
  $ alc7 : int 1 1 1 7 3 6 1 5 4 3 ...
  $ rskreb71: int 1 3 1 2 1 NA 1 2 1 2 ...
  $ rskreb72: int NA NA NA NA NA NA NA 3 2 3 ...
  $ rskreb73: int NA NA NA NA NA NA NA 2 1 2 ...
  $ rskreb74: int NA NA NA NA NA NA NA 3 2 4 ...
  $ likepa71: int 4 2 3 3 2 NA 1 4 3 3 ...
  $ likepa72: int 5 2 4 2 2 NA 5 3 3 2 ...
  $ likepa73: int 4 1 3 3 2 NA 1 3 2 3 ...
  $ likepa74: int 5 3 1 5 4 4 3 4 3 2 ...
  $ likepa75: int 4 4 4 4 3 3 4 4 3 3 ...
  $ posatt71: int 1 1 1 1 1 2 1 NA NA NA ...
  $ posatt72: int 1 2 1 1 1 2 4 NA NA NA ...
  $ posatt73: int 1 1 1 1 1 2 1 NA NA NA ...
  $ alc8 : int 1 8 4 8 5 7 1 3 5 3 ...
  $ rskreb81: int 1 4 1 2 2 3 2 3 1 4 ...
  $ rskreb82: int NA NA NA NA NA NA NA 3 1 4 ...
  $ rskreb83: int NA NA NA NA NA NA NA 2 1 2 ...
  $ rskreb84: int NA NA NA NA NA NA NA 3 2 4 ...
  $ alc9 : int 3 NA 7 NA 5 7 NA 6 6 7 ...
  ```

- Dataset `data.graham.ex6`:

  ```
  'data.frame': 2756 obs. of 9 variables:
  $ school : int 1 1 1 1 1 1 1 1 1 1 ...
  ```

```
$ program : int 0 0 0 0 0 0 0 0 0 0 ...
$ alc7 : int 1 1 1 7 3 6 1 5 4 3 ...
$ riskreb7: int 1 3 1 2 1 NA 1 2 1 2 ...
$ likepar7: int 4 2 3 3 2 NA 1 4 3 3 ...
$ posatt7 : int 1 1 1 1 1 2 1 NA NA NA ...
$ alc8 : int 1 8 4 8 5 7 1 3 5 3 ...
$ riskreb8: int 1 4 1 2 2 3 2 3 1 4 ...
$ alc9 : int 3 NA 7 NA 5 7 NA 6 6 7 ...
```

- Dataset `data.graham.ex8a`:

```
'data.frame': 1023 obs. of 20 variables:
$ skill1 : int 28 29 27 29 29 NA NA NA 29 NA ...
$ skill2 : int NA NA 29 29 NA NA NA NA NA 21 ...
$ skill3 : int NA NA 29 29 29 NA 28 10 29 25 ...
$ skill4 : int NA 29 25 29 29 28 29 NA NA NA ...
$ skill5 : int 29 29 28 28 29 NA 29 10 NA 25 ...
$ iplanV1: int 14 18 15 17 16 NA NA NA 18 NA ...
$ iplanV2: int NA NA 17 16 NA NA NA NA NA 16 ...
$ iplanV3: int NA NA 16 18 18 NA 17 1 18 16 ...
$ iplanV4: int NA 18 14 18 14 6 18 NA NA NA ...
$ iplanV5: int 13 18 12 18 18 NA 18 3 NA 5 ...
$ planA1 : int 1 0 2 8 3 NA NA NA 7 NA ...
$ planA2 : int NA NA 0 4 NA NA NA NA NA 6 ...
$ planA3 : int NA NA 1 4 7 NA 2 0 1 7 ...
$ planA4 : int NA 8 0 4 6 0 0 NA NA NA ...
$ planA5 : int 0 7 1 5 7 NA 2 0 NA 6 ...
$ planV1 : int NA NA NA NA NA NA NA NA NA NA ...
$ planV2 : int NA NA NA NA NA NA NA NA NA 1 ...
$ planV3 : int NA NA 1 NA NA NA NA 0 NA 1 ...
$ planV4 : int NA NA NA NA 2 NA NA NA NA NA ...
$ planV5 : int 2 NA 2 NA NA NA NA 0 NA NA ...
```

- Dataset `data.graham.ex8b`:

```
'data.frame': 2570 obs. of 6 variables:
$ rskreb71: int 1 3 1 2 1 NA 1 2 1 2 ...
$ rskreb72: int NA NA NA NA NA NA NA 3 2 3 ...
$ posatt71: int 1 1 1 1 1 2 1 NA NA NA ...
$ posatt72: int 1 2 1 1 1 2 4 NA NA NA ...
$ posatt73: int 1 1 1 1 1 2 1 NA NA NA ...
$ posatt : int 3 4 3 3 3 6 6 NA NA NA ...
```

- Dataset `data.graham.ex8c`:

```
'data.frame': 2756 obs. of 16 variables:
$ s1 : int 1 1 1 1 1 1 1 1 1 1 ...
$ s2 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s3 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s4 : int 0 0 0 0 0 0 0 0 0 0 ...
```

```
$ s5 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s6 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s7 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s8 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s9 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s10 : int 0 0 0 0 0 0 0 0 0 0 ...
$ s11 : int 0 0 0 0 0 0 0 0 0 0 ...
$ xalc7 : int 1 1 1 7 3 6 1 5 4 3 ...
$ rskreb72: int NA NA NA NA NA NA NA 3 2 3 ...
$ likepa71: int 4 2 3 3 2 NA 1 4 3 3 ...
$ posatt71: int 1 1 1 1 1 2 1 NA NA NA ...
$ alc8 : int 1 8 4 8 5 7 1 3 5 3 ...
```

Source

The datasets were downloaded from http://methodology.psu.edu/pubs/books/missing.

References

Graham, J. W. (2012). *Missing data*. New York: Springer. doi:10.1007/9781461440185

Examples

```
## Not run:
library(mitools)
library(mice)
library(Amelia)
library(jomo)

#############################################################################
# EXAMPLE 1: data.graham.8a | Imputation under multivariate normal model
#############################################################################

data(data.graham.ex8a)
dat <- data.graham.ex8a
dat <- dat[,1:10]
vars <- colnames(dat)
V <- length(vars)
# remove persons with completely missing data
dat <- dat[ rowMeans( is.na(dat) ) < 1, ]
summary(dat)

# some descriptive statistics
psych::describe(dat)

#**************
# imputation under a multivariate normal model
M <- 7  # number of imputations

#--------- mice package
```

```
# define imputation method
impM <- rep("norm", V)
names(impM) <- vars
# mice imputation
imp1a <- mice::mice( dat, method=impM, m=M, maxit=4 )
summary(imp1a)
# convert into a list of datasets
datlist1a <- miceadds::mids2datlist(imp1a)

#--------- Amelia package
imp1b <- Amelia::amelia( dat, m=M )
summary(imp1b)
datlist1b <- imp1b$imputations

#--------- jomo package
imp1c <- jomo::jomo1con(Y=dat, nburn=100, nbetween=10, nimp=M)
str(imp1c)
# convert into a list of datasets
datlist1c <- miceadds::jomo2datlist(imp1c)

# alternatively one can use the jomo wrapper function
imp1c1 <- jomo::jomo(Y=dat, nburn=100, nbetween=10, nimp=M)

#############################################################################
# EXAMPLE 2: data.graham.8b | Imputation with categorical variables
#############################################################################

data(data.graham.ex8b)
dat <- data.graham.ex8b
vars <- colnames(dat)
V <- length(vars)

# descriptive statistics
psych::describe(dat)

#******************************
# imputation in mice using predictive mean matching
imp1a <- mice::mice( dat, m=5, maxit=10)
datlist1a <- mitools::imputationList( miceadds::mids2datlist(imp1a) )
print(datlist1a)

#******************************
# imputation in jomo treating all variables as categorical

# Note that variables must have values from 1 to N
# use categorize function from sirt package here
dat.categ <- sirt::categorize( dat, categorical=colnames(dat), lowest=1 )
dat0 <- dat.categ$data

# imputation in jomo treating all variables as categorical
Y_numcat <- apply( dat0, 2, max, na.rm=TRUE )
imp1b <- jomo::jomo1cat(Y.cat=dat0, Y.numcat=Y_numcat, nburn=100,
                 nbetween=10, nimp=5)
```

```
# recode original categories
datlist1b <- sirt::decategorize( imp1b, categ_design=dat.categ$categ_design )
# convert into a list of datasets
datlist1b <- miceadds::jomo2datlist(datlist1b)
datlist1b <- mitools::imputationList( datlist1b )

# Alternatively, jomo can be used but categorical variables must be
# declared as factors
dat <- dat0
# define two variables as factors
vars <- miceadds::scan.vec(" rskreb71 rskreb72")
for (vv in vars){
    dat[, vv] <- as.factor( dat[,vv] )
          }
# use jomo
imp1b1 <- jomo::jomo(Y=dat, nburn=30, nbetween=10, nimp=5)

#*****************************
# compare frequency tables for both imputation packages
fun_prop <- function( variable ){
            t1 <- table(variable)
            t1 / sum(t1)
                }

# variable rskreb71
res1a <-  with( datlist1a, fun_prop(rskreb71) )
res1b <-  with( datlist1b, fun_prop(rskreb71) )
summary( miceadds::NMIcombine(qhat=res1a, NMI=FALSE ) )
summary( miceadds::NMIcombine(qhat=res1b, NMI=FALSE ) )

# variable posatt
res2a <-  with( datlist1a, fun_prop(posatt) )
res2b <-  with( datlist1b, fun_prop(posatt) )
summary( miceadds::NMIcombine(qhat=res2a, NMI=FALSE ) )
summary( miceadds::NMIcombine(qhat=res2b, NMI=FALSE ) )

## End(Not run)
```

`data.internet` *Dataset Internet*

Description

Dataset with items corresponding to internet attitudes.

Usage

```
data(data.internet)
```

Format

A data frame with 281 observations on the following 22 variables.

The format of the dataset is

```
'data.frame': 281 obs. of 22 variables:
$ IN1 : num 1 5 2 3 1 3 2 3 2 1 ...
$ IN2 : num 4 3 2 7 7 4 4 7 4 3 ...
$ IN3 : num 4 5 4 2 1 2 5 2 2 4 ...
[...]
$ IN20: num 3 2 2 3 3 4 2 7 2 2 ...
$ IN21: num 3 3 6 5 4 4 5 5 6 5 ...
$ IN22: num 3 4 2 5 3 5 3 7 3 5 ...
```

Details

The following text is copied from *http://people.few.eur.nl/groenen/Data/index.htm*

The data set is based on a questionnaire on attitudes towards the Internet. It consists of evaluations of 22 statements about the Internet by 281 students at Erasmus University Rotterdam. These data were gathered around 2002 before the wide availability of broadband Internet access in the Netherlands. The statements were evaluated using a seven-point Likert scale, ranging from 1 (completely disagree) to 7 (completely agree).

We would like to thank Peter Verhoef for making these data available.

Each variable (statement) is coded as follows:

1. Completely disagree
2. Disagree
3. Slightly disagree
4. Neutral
5. Slightly agree
6. Agree
7. Completely agree

Internet items:

1. Paying using Internet is safe
2. Surfing the Internet is easy
3. Internet is unreliable
4. Internet is slow
5. Internet is user-friendly
6. Internet is the future's means of communication
7. Internet is addictive
8. Internet is fast
9. Sending personal data using the Internet is unsafe
10. The prices of Internet subscriptions are high
11. Internet offers many possibilities for abuse
12. The costs of surfing are high
13. Internet offers unbounded opportunities
14. Internet phone costs are high

15. The content of web sites should be regulated
16. Internet is easy to use
17. I like surfing
18. I often speak with friends about the Internet
19. I like to be informed of important new things
20. I always attempt new things on the Internet first
21. I regularly visit websites recommended by others
22. I know much about the Internet

Source

Peter Verhoef

http://people.few.eur.nl/groenen/Data/index.htm

Examples

```
data(data.internet)
# missing proportions
colMeans( is.na(data.internet) )
```

`data.largescale` *Large-scale Dataset for Testing Purposes (Many Cases, Few Variables)*

Description

Large-scale dataset with many cases and few variables included for testing purposes.

Usage

```
data(data.largescale)
```

Format

A data frame with 14000 observations on the following 13 variables. The format is

```
'data.frame': 14000 obs. of 13 variables:
$ id: num 1e+07 1e+07 1e+07 1e+07 1e+07 ...
$ D1: num 0 0 0 0 1 0 0 0 0 0 ...
$ D2: num 0 0 0 1 0 1 0 1 0 0 ...
$ D3: num 0 0 0 0 0 0 0 0 0 0 ...
$ D4: num 0 0 0 1 0 0 0 1 0 0 ...
$ D5: num 0 0 0 0 0 1 0 0 0 0 ...
$ v1: num 118 117 94 106 86 117 96 96 82 95 ...
$ v2: num 101 101 86 101 65 94 72 75 70 99 ...
$ v3: num 0 0 0 0 0 1 0 0 0 0 ...
$ v4: num 3 NA 3 5 2 5 5 5 4 2 ...
```

```
$ v5: num 0 NA 0 0 0 1 0 0 0 0 ...
$ v6: num 3 3 3 4 NA 1 3 3 2 3 ...
$ v7: num 51 36 14 47 22 17 13 37 47 38 ...
```

`data.ma` *Example Datasets for* **miceadds** *Package*

Description

Example datasets for **miceadds** package.

Usage

```
data(data.ma01)
data(data.ma02)
data(data.ma03)
data(data.ma04)
data(data.ma05)
data(data.ma06)
data(data.ma07)
data(data.ma08)
```

Format

- Dataset `data.ma01`:

  Dataset with students nested within school and student weights (`studwgt`). The format is

  ```
  'data.frame': 4073 obs. of 11 variables:
  $ idstud : num 1e+07 1e+07 1e+07 1e+07 1e+07 ...
  $ idschool: num 1001 1001 1001 1001 1001 ...
  $ studwgt : num 6.05 6.05 5.27 5.27 6.05 ...
  $ math : int 594 605 616 524 685 387 536 594 387 562 ...
  $ read : int 647 651 539 551 689 502 503 597 580 576 ...
  $ migrant : int 0 0 0 1 0 0 1 0 0 0 ...
  $ books : int 6 6 5 2 6 3 4 6 6 5 ...
  $ hisei : int NA 77 69 45 66 53 43 NA 64 50 ...
  $ paredu : int 3 7 7 2 7 3 4 NA 7 3 ...
  $ female : int 1 1 0 0 1 1 0 0 1 1 ...
  $ urban : num 1 1 1 1 1 1 1 1 1 1 ...
  ```

- Dataset `data.ma02`:

  10 multiply imputed datasets of incomplete data `data.ma01`. The format is

  ```
  List of 10
  $ :'data.frame': 4073 obs. of 11 variables:
  $ :'data.frame': 4073 obs. of 11 variables:
  $ :'data.frame': 4073 obs. of 11 variables:
  ```

```
$ :'data.frame': 4073 obs. of 11 variables:
$ :'data.frame': 4073 obs. of 11 variables:
$ :'data.frame': 4073 obs. of 11 variables:
$ :'data.frame': 4073 obs. of 11 variables:
$ :'data.frame': 4073 obs. of 11 variables:
$ :'data.frame': 4073 obs. of 11 variables:
$ :'data.frame': 4073 obs. of 11 variables:
```

- Dataset `data.ma03`:

  This dataset contains one variable `math_EAP` for which a conditional posterior distribution with EAP and its associated standard deviation is available.

  ```
  'data.frame': 120 obs. of 8 variables:
  $ idstud : int 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 ...
  $ female : int 0 1 1 1 1 0 1 1 1 1 ...
  $ migrant : int 1 1 0 1 1 0 0 0 1 0 ...
  $ hisei : int 44 NA 26 NA 32 60 31 NA 34 26 ...
  $ educ : int NA 2 NA 1 4 NA 2 NA 2 NA ...
  $ read_wle : num 74.8 78.1 103.2 81.2 119.2 ...
  $ math_EAP : num 337 342 264 285 420 ...
  $ math_SEEAP: num 28 29.5 28.6 28.5 27.5 ...
  ```

- Dataset `data.ma04`:

  This dataset contains two hypothetical scales `A` and `B` and single variables `V5`, `V6` and `V7`.

  ```
  'data.frame': 281 obs. of 13 variables:
  $ group: int 1 1 1 1 1 1 1 1 1 1 ...
  $ A1 : int 2 2 2 1 1 3 3 NA 2 1 ...
  $ A2 : int 2 2 2 3 1 2 4 4 4 4 ...
  $ A3 : int 2 3 3 4 1 3 2 2 2 4 ...
  $ A4 : int 3 4 6 4 7 5 3 5 5 1 ...
  $ V5 : int 2 2 5 5 4 3 4 1 3 4 ...
  $ V6 : int 2 5 5 1 1 3 2 2 2 4 ...
  $ V7 : int 6 NA 4 5 6 2 5 5 6 7 ...
  $ B1 : int 7 NA 6 4 5 2 5 7 3 7 ...
  $ B2 : int 6 NA NA 6 3 3 4 6 6 7 ...
  $ B3 : int 7 NA 7 4 3 4 3 7 5 NA ...
  $ B4 : int 4 5 6 5 4 3 4 5 2 1 ...
  $ B5 : int 7 NA 7 4 4 3 5 7 5 4 ...
  ```

- Dataset `data.ma05`:

  This is a two-level dataset with students nested within classes. Variables at the student level are `Dscore`, `Mscore`, `denote`, `manote`, `misei` and `migrant`. Variables at the class level are `sprengel` and `groesse`.

  ```
  'data.frame': 1673 obs. of 10 variables:
  $ idstud : int 100110001 100110002 100110003 100110004 100110005 ...
  $ idclass : int 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 ...
  $ Dscore : int NA 558 643 611 518 552 NA 534 409 543 ...
  $ Mscore : int 404 563 569 621 653 651 510 NA 517 566 ...
  ```

```
$ denote : int NA 1 1 1 3 2 3 2 3 2 ...
$ manote : int NA 1 1 1 1 1 2 2 2 1 ...
$ misei : int NA 51 NA 38 NA 50 53 53 38 NA ...
$ migrant : int NA 0 0 NA 0 0 0 0 0 NA ...
$ sprengel: int 0 0 0 0 0 0 0 0 0 0 ...
$ groesse : int 25 25 25 25 25 25 25 25 25 25 ...
```

- Dataset `data.ma06`:

  This is a dataset in which the variable `FC` is only available with grouped values (coarse data or interval data).

  ```
  'data.frame': 198 obs. of 7 variables:
  $ id : num 1001 1002 1003 1004 1005 ...
  $ A1 : int 14 7 10 15 0 5 9 6 8 0 ...
  $ A2 : int 5 6 4 8 2 5 4 0 7 0 ...
  $ Edu : int 4 3 1 5 5 1 NA 1 5 3 ...
  $ FC : int 3 2 2 2 2 NA NA 2 2 NA ...
  $ FC_low: num 10 5 5 5 5 0 0 5 5 0 ...
  $ FC_upp: num 15 10 10 10 10 100 100 10 10 100 ...
  ```

- Dataset `data.ma07`:

  This is a three-level dataset in which the variable `FC` is only available with grouped values (coarse data or interval data).

  ```
  'data.frame': 1600 obs. of 9 variables:
  $ id3: num 1001 1001 1001 1001 1001 ...
  $ id2: num 101 101 101 101 101 101 101 101 101 101 ...
  $ id1: int 1 2 3 4 5 6 7 8 9 10 ...
  $ x1 : num 0.91 1.88 NA 1.52 0.93 0.51 2.11 0.99 2.42 NA ...
  $ x2 : num -0.58 1.12 0.87 -0.01 -0.14 0.48 1.85 -0.9 0.93 0.63 ...
  $ y1 : num 1.66 1.66 1.66 1.66 1.66 1.66 1.66 1.66 1.66 1.66 ...
  $ y2 : num 0.96 0.96 0.96 0.96 0.96 0.96 0.96 0.96 0.96 0.96 ...
  $ z1 : num -0.53 -0.53 -0.53 -0.53 -0.53 -0.53 -0.53 -0.53 -0.53 -0.53 ...
  $ z2 : num 0.42 0.42 0.42 0.42 0.42 0.42 0.42 0.42 0.42 0.42 ...
  ```

- Dataset `data.ma08`:

  List with several vector of strings containing descriptive data from published articles. See `string_to_matrix` for converting these strings into matrices.

  ```
  List of 4
  $ mat1: chr [1:6] "1. T1_mental_health" ...
  $ mat2: chr [1:16] "1. Exp voc-T1 -" ...
  $ mat3: chr [1:12] "1. TOWRE age 7\t-\t\t\t\t\t\t" ...
  $ mat4: chr [1:18] "1. Vocab. age 7\t-\t\t\t\t\t" ...
  ```

- Dataset `data.ma09`:

  This is a subset of a PISA dataset that is used for generating synthetic data.

  ```
  'data.frame': 342 obs. of 41 variables:
  $ SEX : int 1 2 1 2 1 2 2 2 2 1 ...
  ```

```
$ AGE : num 16 15.9 16.3 15.5 15.9 ...
$ HISEI : int 37 46 66 51 25 NA 54 52 51 69 ...
$ FISCED : int 3 3 6 3 3 NA 3 3 2 2 ...
$ MISCED : int 3 4 4 4 3 NA 4 3 4 4 ...
$ PV1MATH: num 643 556 510 604 462 ...
$ M474Q01: int 1 1 1 1 0 1 1 1 1 0 ...
$ M155Q02: int 2 2 2 2 2 0 0 2 2 2 ...
$ M155Q01: int 1 1 0 1 1 1 1 1 1 1 ...
[...]
```

`data.smallscale` *Small-Scale Dataset for Testing Purposes (Moderate Number of Cases, Many Variables)*

Description

Small-scale dataset for testing purposes (moderate number of cases, many variables)

Usage

```
data(data.smallscale)
```

Format

A data frame with 675 observations on the following 164 variables. The format is

```
'data.frame': 675 obs. of 164 variables:
$ v1 : num 3 3 2 3 3 0 1 0 3 NA ...
$ v2 : num 3 0 1 3 0 0 0 3 2 NA ...
$ v3 : num 0 0 2 3 2 0 1 0 0 NA ...
$ v4 : num 1 3 3 3 NA 0 0 0 3 NA ...
$ v5 : num 0 0 3 3 0 0 3 1 3 3 ...
$ v6 : num 8 8 9 8 9 9 9 8 9 9 ...
[...]
```

`datlist2Amelia` *Converting an Object of class* `amelia`

Description

This function converts a list of multiply imputed data sets to an object of class `amelia`.

Usage

```
datlist2Amelia(datlist)
```

Arguments

`datlist`	List of multiply imputed data sets or an object of class `mids`

Value

An object of class `amelia`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation of NHANES data using mice package
#############################################################################

library(mice)
library(Amelia)

data(nhanes,package="mice")
set.seed(566)  # fix random seed

# imputation with mice
imp <- mice::mice(nhanes, m=7)

# conversion to amelia object
amp <- miceadds::datlist2Amelia(datlist=imp)
str(amp)
plot(amp)
print(amp)
summary(amp)

## End(Not run)
```

`datlist2mids` *Converting a List of Multiply Imputed Data Sets into a* `mids` *Object*

Description

This function converts a list of multiply imputed data sets to a `mice::mids` object.

Usage

```
datlist2mids(dat.list, progress=FALSE)
datalist2mids(dat.list, progress=FALSE)
```

Arguments

`dat.list`	List of multiply imputed data sets or an object of class `imputationList` (see `mitools::imputationList`)
`progress`	An optional logical indicating whether conversion process be displayed

Value

An object of class `mids`

See Also

See `mice::as.mids` for converting a multiply imputed dataset in long format into a `mids` object.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation of NHANES data using Amelia package
#############################################################################

library(mice)
library(Amelia)

data(nhanes,package="mice")
set.seed(566)  # fix random seed

# impute 10 datasets using Amelia
a.out <- Amelia::amelia(x=nhanes, m=10)
# plot of observed and imputed data
plot(a.out)

# convert list of multiply imputed datasets into a mids object
a.mids <- miceadds::datlist2mids( a.out$imputations )

# linear regression: apply mice functionality lm.mids
mod <- with( a.mids, stats::lm( bmi ~ age ) )
summary( mice::pool( mod ) )
  ##                    est       se         t       df     Pr(>|t|)     lo 95
  ##  (Intercept) 30.624652 2.626886 11.658158 8.406608 1.767631e-06 24.617664
  ##  age         -2.280607 1.323355 -1.723352 8.917910 1.192288e-01 -5.278451
  ##                   hi 95 nmis       fmi    lambda
  ##  (Intercept) 36.6316392   NA 0.5791956 0.4897257
  ##  age          0.7172368    0 0.5549945 0.4652567

# fit linear regression model in Zelig
library(Zelig)
mod2 <- Zelig::zelig( bmi ~ age, model="ls", data=a.out, cite=FALSE)
summary(mod2)
  ##  Model: Combined Imputations
  ##              Estimate Std.Error z value Pr(>|z|)
  ##  (Intercept)   30.625     2.627  11.658  0.00000 ***
  ##  age           -2.281     1.323  -1.723  0.08482
  ##  ---
  ##  Signif. codes:  '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# fit linear regression using mitools package
library(mitools)
datimp <- mitools::imputationList(a.out$imputations)
```

```
mod3 <- with( datimp, stats::lm( bmi ~ age ) )
summary( mitools::MIcombine( mod3 ) )
  ##  Multiple imputation results:
  ##        with(datimp, stats::lm(bmi ~ age))
  ##        MIcombine.default(mod3)
  ##                results       se    (lower     upper) missInfo
  ##  (Intercept) 30.624652 2.626886 25.304594 35.9447092     51
  ##  age         -2.280607 1.323355 -4.952051  0.3908368     49

## End(Not run)
```

`datlist_create` *Creates Objects of Class* `datlist` *or* `nested.datlist`

Description

Creates objects of class `datlist` or `nested.datlist`.

The functions `nested.datlist2datlist` and `datlist2nested.datlist` provide list conversions for imputed datasets.

Usage

```
datlist_create(datasets)

nested.datlist_create(datasets)

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

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

nested.datlist2datlist(datlist)

datlist2nested.datlist(datlist, Nimp)
```

Arguments

`datasets`	For `datlist_create`: List of datasets, objects of class `imputationList`, `mids`, `mids.1chain`, For `nested.datlist_create`: nested list of datasets, `NestedImputationList`, `mids.nmi`
`x`	Object of classes `datlist` or `nested.datlist`
`datlist`	Object of classes `datlist`, `imputationList`, `nested.datlist` or `NestedImputationList`.
`Nimp`	Vector of length 2 containing numbers of between and within imputations.
`...`	Further arguments to be passed

Value

Object of class `datlist` or `nested.datlist`

Examples

```
## Not run:
## The function datlist_create is currently defined as
function (datasets)
{
    class(datasets) <- "datlist"
    return(datasets)
  }

#############################################################################
# EXAMPLE 1: Create object of class datlist
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2

# class datlist
obj1 <- miceadds::datlist_create(data.timss2)

#############################################################################
# EXAMPLE 2: Multiply imputed datasets: Different object classes
#############################################################################

library(mice)
data(nhanes2, package="mice")
set.seed(990)

# nhanes2 data imputation
imp1 <- miceadds::mice.1chain( nhanes2, burnin=5, iter=25, Nimp=5 )
# object of class datlist
imp2 <- miceadds::mids2datlist(imp1)
# alternatively, one can use datlist_create
imp2b <- miceadds::datlist_create(imp1)
# object of class imputationList
imp3 <- mitools::imputationList(imp2)
# retransform object in class datlist
imp2c <- miceadds::datlist_create(imp3)
str(imp2c)

#############################################################################
# EXAMPLE 3: Nested multiply imputed datasets
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
   # list of 5 datasets containing 5 plausible values
```

```
#** define imputation method and predictor matrix
data <- datlist[[1]]
V <- ncol(data)
# variables
vars <- colnames(data)
# variables not used for imputation
vars_unused <- miceadds::scan.vec("IDSTUD TOTWGT  JKZONE  JKREP" )

#- define imputation method
impMethod <- rep("norm", V )
names(impMethod) <- vars
impMethod[ vars_unused ] <- ""

#- define predictor matrix
predM <- matrix( 1, V, V )
colnames(predM) <- rownames(predM) <- vars
diag(predM) <- 0
predM[, vars_unused ] <- 0

# object of class nmi
imp1 <- miceadds::mice.nmi( datlist, method=impMethod, predictorMatrix=predM,
                m=4, maxit=3 )
# object of class nested.datlist
imp2 <- miceadds::mids2datlist(imp1)
# object of class NestedImputationList
imp3 <- miceadds::NestedImputationList(imp2)
# redefine class nested.datlist
imp4 <- miceadds::nested.datlist_create(imp3)

#############################################################################
# EXAMPLE 4: Conversions between nested lists of datasets and lists of datasets
#############################################################################

library(BIFIEsurvey)
data(data.timss4, package="BIFIEsurvey" )
datlist <- data.timss4

# object of class nested.datlist
datlist1a <- miceadds::nested.datlist_create(datlist)
# object of class NestedImputationList
datlist1b <- miceadds::NestedImputationList(datlist)

# conversion to datlist
datlist2a <- miceadds::nested.datlist2datlist(datlist1a)  # class datlist
datlist2b <- miceadds::nested.datlist2datlist(datlist1b)  # class imputationList

# convert into a nested list with 2 between nests and 10 within nests
datlist3a <- miceadds::datlist2nested.datlist(datlist2a, Nimp=c(2,10) )
datlist3b <- miceadds::datlist2nested.datlist(datlist2b, Nimp=c(4,5) )

## End(Not run)
```

`draw.pv.ctt` *Plausible Value Imputation Using a Known Measurement Error Variance (Based on Classical Test Theory)*

Description

This function provides unidimensional plausible value imputation with a known measurement error variance or classical test theory (Mislevy, 1991). The reliability of the scale is estimated by Cronbach's Alpha or can be provided by the user.

Usage

```
draw.pv.ctt(y, dat.scale=NULL, x=NULL, samp.pars=TRUE,
      alpha=NULL, sig.e=NULL, var.e=NULL, true.var=NULL)
```

Arguments

`y`	Vector of scale scores if `y` should not be used.
`dat.scale`	Matrix of item responses
`x`	Matrix of covariates
`samp.pars`	An optional logical indicating whether scale parameters (reliability or measurement error standard deviation) should be sampled
`alpha`	Reliability estimate of the scale. The default of `NULL` means that Cronbach's alpha will be used as a reliability estimate.
`sig.e`	Optional vector of the standard deviation of the error. Note that it is *not* the error variance.
`var.e`	Optional vector of the variance of the error.
`true.var`	True score variance

Details

The linear model is assumed for drawing plausible values of a variable Y contaminated by measurement error. Assuming $Y = \theta + e$ and a linear regression model for θ

$$\theta = \boldsymbol{X}\beta + \epsilon$$

(plausible value) imputations from the posterior distribution $P(\theta|Y, \boldsymbol{X})$ are drawn. See Mislevy (1991) for details.

Value

A vector with plausible values

Note

Plausible value imputation is also labeled as multiple overimputation (Blackwell, Honaker & King, 2011).

References

Blackwell, M., Honaker, J., & King, G. (2011). *Multiple overimputation: A unified approach to measurement error and missing data*. Technical Report.

Mislevy, R. J. (1991). Randomization-based inference about latent variables from complex samples. *Psychometrika, 56*(2), 177-196. doi:10.1007/BF02294457

See Also

See also `sirt::plausible.value.imputation.raschtype` for plausible value imputation.

Plausible value imputations can be conducted in **mice** using the imputation method `mice.impute.plausible.values`.

Plausible values can be drawn in **Amelia** by specifying observation-level priors, see `Amelia::moPrep` and `Amelia::amelia`.

Examples

```
## Not run:

#############################################################################
# SIMULATED EXAMPLE 1: Scale scores
#############################################################################

set.seed(899)
n <- 5000       # number of students
x <- round( stats::runif( n, 0,1 ) )
y <- stats::rnorm(n)
# simulate true score theta
theta <- .6 + .4*x + .5 * y + stats::rnorm(n)
# simulate observed score by adding measurement error
sig.e <- rep( sqrt(.40), n )
theta_obs <- theta + stats::rnorm( n, sd=sig.e)

# calculate alpha
( alpha <- stats::var( theta ) / stats::var( theta_obs ) )
# [1] 0.7424108
#=> Ordinarily, sig.e or alpha will be known, assumed or estimated by using items,
#    replications or an appropriate measurement model.

# create matrix of predictors
X <- as.matrix( cbind(x, y ) )

# plausible value imputation with scale score
imp1 <- miceadds::draw.pv.ctt( y=theta_obs, x=X, sig.e=sig.e )
# check results
stats::lm( imp1 ~ x + y )

# imputation with alpha as an input
imp2 <- miceadds::draw.pv.ctt( y=theta_obs, x=X, alpha=.74 )
stats::lm( imp2 ~ x + y )

#--- plausible value imputation in Amelia package
```

```
library(Amelia)

# define data frame
dat <- data.frame( "x"=x, "y"=y, "theta"=theta_obs )
# generate observation-level priors for theta
priors <- cbind( 1:n, 3, theta_obs, sig.e )
             # 3 indicates column index for theta
overimp <- priors[,1:2]
# run Amelia
imp <- Amelia::amelia( dat, priors=priors, overimp=overimp, m=10)
# create object of class datlist and evaluate results
datlist <- miceadds::datlist_create( imp$imputations )
withPool_MI( with( datlist, stats::var(theta) ) )
stats::var(theta)       # compare with true variance
mod <- with( datlist,  stats::lm( theta ~ x + y ) )
mitools::MIcombine(mod)

## End(Not run)
```

`filename_split` *Some Functionality for Strings and File Names*

Description

The function `filename_split` splits a file name into parts.

The function `string_extract_part` extracts a part of a string.

The function `string_to_matrix` converts a string into a matrix.

Usage

```
filename_split(file_name, file_sep="__", file_ext=".")
filename_split_vec( file_names, file_sep="__", file_ext=".")

string_extract_part( vec, part=1, sep="__", remove_empty=TRUE )

string_to_matrix(x, rownames=NULL, col_elim=NULL, as_numeric=FALSE,
                 diag_val=NULL, extend=FALSE, col1_numeric=FALSE, split=" ")
```

Arguments

`file_name`	File name
`file_names`	File names
`file_sep`	Separator within file name
`file_ext`	Separator for file extension
`vec`	Vector with strings
`part`	Integer indicating the part of the string to be selected

`sep`	String separator
`remove_empty`	Logical indicating whether empty entries (" "") should be removed.
`x`	String vector
`rownames`	Column index for row names
`col_elim`	Indices for elimination of columns
`as_numeric`	Logical indicating whether numeric conversion is requested
`diag_val`	Optional values for inclusion in diagonal of matrix
`extend`	Optional indicating whether numeric matrix should be extended to become a symmetric matrix
`col1_numeric`	Logical indicating whether second column is selected in such a way that it has to be always a numeric (see Example 5)
`split`	String used for splitting

Value

List with components of the file name (see Examples).

See Also

`files_move`

Examples

```
#############################################################################
# EXAMPLE 1: Demonstration example for filename_split
#############################################################################

# file name
file_name <- "pisa_all_waves_invariant_items_DATA_ITEMS_RENAMED__DESCRIPTIVES__2016-10-12_1000.csv"

# apply function
miceadds::filename_split( file_name )
  ##  $file_name
 ## [1] "pisa_all_waves_invariant_items_DATA_ITEMS_RENAMED__DESCRIPTIVES__2016-10-12_1000.csv"
  ##  $stem
  ##  [1] "pisa_all_waves_invariant_items_DATA_ITEMS_RENAMED__DESCRIPTIVES"
  ##  $suffix
  ##  [1] "2016-10-12_1000"
  ##  $ext
  ##  [1] "csv"
  ##  $main
  ##  [1] "pisa_all_waves_invariant_items_DATA_ITEMS_RENAMED__DESCRIPTIVES.csv"

#############################################################################
# EXAMPLE 2: Example string_extract_part
#############################################################################

vec <- c("ertu__DES", "ztu__DATA", "guzeuue745_ghshgk34__INFO", "zzu78347834_ghghwuz")
```

```
miceadds::string_extract_part( vec=vec, part=1, sep="__" )
miceadds::string_extract_part( vec=vec, part=2, sep="__" )
  ##  > miceadds::string_extract_part( vec=vec, part=1, sep="__" )
  ##  [1] "ertu"                "ztu"                 "guzeuue745_ghshgk34"
  ##  [4] "zzu78347834_ghghwuz"
  ##  > miceadds::string_extract_part( vec=vec, part=2, sep="__" )
  ##  [1] "DES"  "DATA" "INFO" NA

## Not run:
#############################################################################
# EXAMPLE 3: Reading descriptive information from published articles
#############################################################################
data(data.ma08)
library(stringr)

#**** reading correlations (I)
dat <- data.ma08$mat1
miceadds::string_to_matrix(dat, rownames=2, col_elim=c(1,2))

#**** reading correlations including some processing (II)
dat0 <- data.ma08$mat2
dat <- dat0[1:14]

# substitute "*"
dat <- gsub("*", "", dat, fixed=TRUE )

# replace blanks in variable names
s1 <- stringr::str_locate(dat, "[A-z] [A-z]")
start <- s1[,"start"] + 1
for (ss in 1:length(start) ){
    if ( ! is.na( start[ss] ) ){
        substring( dat[ss], start[ss], start[ss] ) <- "_"
    }
}

# character matrix
miceadds::string_to_matrix(dat)
# numeric matrix containing correlations
miceadds::string_to_matrix(dat, rownames=2, col_elim=c(1,2), as_numeric=TRUE, diag_val=1,
           extend=TRUE )
#** reading means and SDs
miceadds::string_to_matrix(dat0[ c(15,16)], rownames=1, col_elim=c(1), as_numeric=TRUE )

#**** reading correlations (III)
dat <- data.ma08$mat3
dat <- gsub(" age ", "_age_", dat, fixed=TRUE )
miceadds::string_to_matrix(dat, rownames=2, col_elim=c(1,2), as_numeric=TRUE, diag_val=1,
       extend=TRUE )

#**** reading correlations (IV)
dat <- data.ma08$mat4 <- dat0

# remove spaces in variable names
```

```
dat <- gsub(" age ", "_age_", dat, fixed=TRUE )
s1 <- stringr::str_locate_all(dat, "[A-z,.] [A-z]")
NL <- length(dat)
for (ss in 1:NL ){
    NR <- nrow(s1[[ss]])
    if (NR>1){
        start <- s1[[ss]][2,1]+1
        if ( ! is.na( start ) ){
            substring( dat[ss], start, start ) <- "_"
        }
    }
}

miceadds::string_to_matrix(dat, rownames=2, col_elim=c(1,2), as_numeric=TRUE, diag_val=1,
     extend=TRUE )

#############################################################################
# EXAMPLE 4: Input string of length one
#############################################################################


pm0 <- "
0.828
0.567 0.658
0.664 0.560 0.772
0.532 0.428 0.501 0.606
0.718 0.567 0.672 0.526 0.843"

miceadds::string_to_matrix(x=pm0, as_numeric=TRUE, extend=TRUE)

#############################################################################
# EXAMPLE 5: String with variable names and blanks
#############################################################################

tab1 <- "
Geometric Shapes .629 .021 (.483) -.049 (.472)
Plates .473 .017 (.370) .105 (.405)
Two Characteristics .601 .013 (.452) -.033 (.444)
Crossing Out Boxes .597 -.062 (.425) -.036 (.445)
Numbers/Letters .731 .004 (.564) .003 (.513)
Numbers/Letters mixed .682 .085 (.555) .082 (.514)"

miceadds::string_to_matrix(x=tab1, col1_numeric=TRUE)

## End(Not run)
```

`files_move` *Moves Files from One Directory to Another Directory*

Description

Moves older (defined in alphanumeric order) files from one directory to another directory. If directories do not exist, they will be automatically created.

Usage

```
files_move(path1, path2, file_sep="__", pattern=NULL, path2_name="__ARCH")
```

Arguments

`path1`	Original directory
`path2`	Target directory in which the files should be moved
`file_sep`	Separator for files
`pattern`	Pattern in file names to be searched for
`path2_name`	Part of the name of `path2` if argument `path2` is missing. If `path2` is not provided, it has to be a subdirectory of `path1`.

See Also

`filename_split`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Move older files in '__ARCHIVE' directory
#############################################################################

# specify path
path1 <- "p:/IPN/Projects/PISA/Trend_2015/2__Data/All_Waves/"
# specify target directory which is an archive
path2 <- file.path( path1, "__ARCHIVE" )
# move files
files_move( path1, path2 )

## End(Not run)
```

`fleishman_sim` *Simulating Univariate Data from Fleishman Power Normal Transformations*

Description

Simulates univariate non-normal data by using Fleishman power transformations (Fleishman, 1978; Demirtas & Hedeker, 2007).

Usage

```
fleishman_sim(N=1, coef=NULL, mean=0, sd=1, skew=0, kurt=0)

fleishman_coef(mean=0, sd=1, skew=0, kurt=0)
```

Arguments

`N`	Number of simulated values
`coef`	Optional list containing coefficients of Fleishman polynomial estimated by `fleishman_coef`.
`mean`	Mean
`sd`	Standard deviation
`skew`	Skewness
`kurt`	(Excess) kurtosis

Details

For $Z \sim N(0,1)$, the Fleishman power normal variable X is defined as $X = a + bZ + cZ^2 + dZ^3$.

Value

Vector of simulated values (`fleishman_sim`) or list of coefficients (`fleishman_coef`).

References

Demirtas, H., & Hedeker, D. (2008). Imputing continuous data under some non-Gaussian distributions. *Statistica Neerlandica, 62*(2), 193-205. doi:10.1111/j.14679574.2007.00377.x

Fleishman, A. I. (1978). A method for simulating non-normal distributions. *Psychometrika, 43*(4), 521-532. doi:10.1007/BF02293811

See Also

See also the `BinOrdNonNor::Fleishman.coef.NN` function in the **BinOrdNonNor** package.

See the `nnig_sim` function for simulating a non-normally distributed multivariate variables.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Simulating values with Fleishman polynomial
#############################################################################

#* define mean, standard deviation, skewness and kurtosis
mean <- .75
sd <- 2
skew <- 1
kurt <- 3

#* compute coefficients of Fleishman polynomial
```

```
coeff <- miceadds::fleishman_coef(mean=mean, sd=sd, skew=skew, kurt=kurt)
print(coeff)

# sample size
N <- 1000
set.seed(2018)
#* simulate values based on estimated coefficients
X <- miceadds::fleishman_sim(N=N, coef=coeff)
#* simulate values based on input of moments
X <- miceadds::fleishman_sim(N=N, mean=mean, sd=sd, skew=skew, kurt=kurt)

## End(Not run)
```

`grep.vec` R *Utilities: Vector Based Versions of* `grep`

Description

These functions slightly extend the usage of `grep` but it is extended to a vector argument.

Usage

```
grep.vec(pattern.vec, x, operator="AND", ...)

grepvec( pattern.vec, x, operator="AND", value=FALSE, ...)

grep_leading( pattern, x, value=FALSE )

grepvec_leading( patternvec, x, value=FALSE )
```

Arguments

`pattern.vec`	String which should be looked for in vector `x`
`x`	A character vector
`operator`	An optional string. The default argument `"AND"` searches all entries in `x` which contain all elements of `pattern.vec`. If `operator` is different from the default, then the `"OR"` logic applies, i.e. the functions searches for vector entries which contain at least one of the strings in `pattern.vec`.
`pattern`	String
`patternvec`	Vector of strings
`value`	Logical indicating whether indices or values are requested
`...`	Arguments to be passed to `base::grep` (e.g., `fixed=TRUE`)

Examples

```
#############################################################################
# EXAMPLE 1: Toy example
#############################################################################

vec <- c("abcd", "bcde", "aedf", "cdf" )
# search for entries in vec with contain 'a' and 'f'
#  -> operator="AND"
grep.vec( pattern.vec=c("a","f"), x=vec )
  ##   $x
  ##   [1] "aedf"
  ##   $index.x
  ##   [1] 3

grepvec( pattern.vec=c("a","f"), x=vec, value=TRUE)
grepvec( pattern.vec=c("a","f"), x=vec, value=FALSE)

# search for entries in vec which contain 'a' or 'f'
grep.vec( pattern.vec=c("a","f"), x=vec, operator="OR")
  ##   $x
  ##   [1] "abcd" "aedf" "cdf"
  ##   $index.x
  ##   [1] 1 3 4
```

`GroupMean` *Calculation of Groupwise Descriptive Statistics for Matrices*

Description

Calculates some groupwise descriptive statistics.

Usage

```
GroupMean(data, group, weights=NULL, extend=FALSE, elim=FALSE)

GroupSum(data, group, weights=NULL, extend=FALSE)

GroupSD(data, group, weights=NULL, extend=FALSE)

# group mean of a variable
gm(y, cluster, elim=FALSE)

# centering within clusters
cwc(y, cluster)
```

Arguments

`data`	A numeric data frame

`group`	A vector of group identifiers
`weights`	An optional vector of sample weights
`extend`	Optional logical indicating whether the group means (or sums) should be extended to the original dimensions of the dataset.
`elim`	Logical indicating whether a case in a row should be removed from the calculation of the mean in a cluster
`y`	Variable
`cluster`	Cluster identifier

Value

A data frame or a vector with groupwise calculated statistics

See Also

`mitml::clusterMeans`

`base::rowsum`, `stats::aggregate`, `stats::ave`

Examples

```
## Not run:

#############################################################################
# EXAMPLE 1: Group means and standard deviations for data.ma02
#############################################################################

data(data.ma02, package="miceadds" )
dat <- data.ma02[[1]] # select first dataset

#--- group means for read and math
GroupMean( dat[, c("read","math") ], group=dat$idschool )
# using rowsum
a1 <- base::rowsum( dat[, c("read","math") ], dat$idschool )
a2 <- base::rowsum( 1+0*dat[, c("read","math") ], dat$idschool )
(a1/a2)[1:10,]
# using aggregate
stats::aggregate(  dat[, c("read","math") ], list(dat$idschool), mean )[1:10,]

#--- extend group means to original dataset
GroupMean( dat[, c("read","math") ], group=dat$idschool, extend=TRUE )
# using ave
stats::ave( dat[, "read" ], dat$idschool  )
stats::ave( dat[, "read" ], dat$idschool, FUN=mean )

#--- group standard deviations
GroupSD( dat[, c("read","math") ], group=dat$idschool)[1:10,]
# using aggregate
stats::aggregate(  dat[, c("read","math") ], list(dat$idschool), sd )[1:10,]

#############################################################################
```

```
# EXAMPLE 2: Calculating group means and group mean centering
#############################################################################

data(data.ma07, package="miceadds")
dat <- data.ma07

# compute group means
miceadds::gm( dat$x1, dat$id2 )
# centering within clusters
miceadds::cwc( dat$x1, dat$id2 )

# evaluate formula with model.matrix
X <- model.matrix( ~ I( miceadds::cwc(x1, id2) ) + I( miceadds::gm(x1,id2) ), data=dat )
head(X)

## End(Not run)
```

`index.dataframe` R *Utilities: Include an Index to a Data Frame*

Description

This function includes an index variable to a data frame in the first column.

Usage

```
index.dataframe(data,systime=FALSE)
```

Arguments

`data`	Data frame
`systime`	Should system time be included in the second column of the data frame?

Examples

```
dfr <- matrix( 2*1:12-3, 4,3 )
colnames(dfr) <- paste0("X",1:ncol(dfr))
index.dataframe( dfr)
  ##     index X1 X2 X3
  ##   1     1 -1  7 15
  ##   2     2  1  9 17
  ##   3     3  3 11 19
  ##   4     4  5 13 21
index.dataframe( dfr, systime=TRUE)
  ##     index         file_created X1 X2 X3
  ##   1     1  2013-08-22 10:26:28 -1  7 15
  ##   2     2  2013-08-22 10:26:28  1  9 17
  ##   3     3  2013-08-22 10:26:28  3 11 19
  ##   4     4  2013-08-22 10:26:28  5 13 21
```

`in_CI` *Indicator Function for Analyzing Coverage*

Description

Indicator function for analyzing coverage. The output indicates whether a value lies within a computed confidence interval.

Usage

```
in_CI(est, se, true, level=0.95, df=Inf)
```

Arguments

`est`	Vector of estimates
`se`	Vector of standard errors
`true`	Vector of true parameters
`level`	Confidence level
`df`	Degrees of freedom for t distribution. The default corresponds to the normal distribution.

Value

Logical vector

Examples

```
#############################################################################
# EXAMPLE 1: Toy example
#############################################################################

#-- simulate estimates and standard errors
set.seed(987)
n <- 10
est <- stats::rnorm( n, sd=1)
se <- stats::runif( n, 0, .7 )
level <- .95
true <- 0

#-- apply coverage function
in_ci <- miceadds::in_CI( est, se, true)
#-- check correctness
cbind( est, se, true, in_ci )
```

`jomo2datlist` *Converts a **jomo** Data Frame in Long Format into a List of Datasets or an Object of Class* `mids`

Description

Converts a **jomo** data frame in long format into a list of datasets or an object of class `mids`.

Usage

```
jomo2datlist(jomo.dataframe, variable="Imputation")

jomo2mids(jomo.dataframe, variable="Imputation")
```

Arguments

`jomo.dataframe`	Data frame generated in **jomo** package
`variable`	Variable name for imputation index

Value

List of multiply imputed datasets

See Also

See the **jomo** package.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Dataset nhanes | jomo imputation and conversion into a data list
#############################################################################

data(nhanes, package="mice")
dat <- nhanes

# impute under multivariate normal model in jomo
imp1 <- jomo::jomo1con(Y=dat, nburn=100, nbetween=10, nimp=5)
# convert into a list of datasets
datlist1 <- miceadds::jomo2datlist(imp1)
# convert into mids object
datlist2 <- miceadds::jomo2datlist(imp1)

## End(Not run)
```

`kernelpls.fit2` *Kernel PLS Regression*

Description

Fits a PLS regression model with the kernel algorithm (Dayal & Macgregor, 1997).

Usage

```
kernelpls.fit2(X, Y, ncomp)

## S3 method for class 'kernelpls.fit2'
predict(object,X, ...)
```

Arguments

`X`	Matrix of regressors
`Y`	Vector of a univariate outcome
`ncomp`	Number of components to be extracted
`object`	Object of class `kernelpls.fit2`
`...`	Further arguments to be passed

Value

The same list as in `{pls::kernelpls.fit}` is produced.

In addition, R^2 measures are contained in `R2`.

Author(s)

This code is a **Rcpp** translation of the original `pls::kernelpls.fit` function from the **pls** package (see Mevik & Wehrens, 2007).

References

Dayal, B., & Macgregor, J. F. (1997). Improved PLS algorithms. *Journal of Chemometrics, 11*(1), 73-85.

Mevik, B. H., & Wehrens, R. (2007). The **pls** package: Principal component and partial least squares regression in R. *Journal of Statistical Software, 18*, 1-24. doi:10.18637/jss.v018.i02

See Also

See the **pls** package for further estimation algorithms.

Examples

```
## Not run:
#############################################################################
# SIMULATED EXAMPLE 1: 300 cases on 100 variables
#############################################################################
set.seed(789)
library(mvtnorm)

N <- 300        # number of cases
p <- 100        # number of predictors
rho1 <- .6      # correlations between predictors

# simulate data
Sigma <- base::diag(1-rho1,p) + rho1
X <- mvtnorm::rmvnorm( N, sigma=Sigma )
beta <- base::seq( 0, 1, len=p )
y <- ( X %*% beta )[,1] + stats::rnorm( N, sd=.6 )
Y <- base::matrix(y,nrow=N, ncol=1 )

# PLS regression
res <- miceadds::kernelpls.fit2( X=X, Y=Y, ncomp=20 )

# predict new scores
Xpred <- predict( res, X=X[1:10,] )

#############################################################################
# EXAMPLE 2: Dataset yarn from pls package
#############################################################################

# use kernelpls.fit from pls package
library(pls)
data(yarn,package="pls")
mod1 <- pls::kernelpls.fit( X=yarn$NIR, Y=yarn$density, ncomp=10 )
# use kernelpls.fit2 from miceadds package
Y <- base::matrix( yarn$density, ncol=1 )
mod2 <- miceadds::kernelpls.fit2( X=yarn$NIR, Y=Y, ncomp=10 )

## End(Not run)
```

`library_install` R *Utilities: Loading a Package or Installation of a Package if Necessary*

Description

Loads packages specified in vector pkg. If some packages are not yet installed, they will be automatically installed by this function using `install.packages`.

Usage

```
library_install( pkg, ... )
```

Arguments

`pkg`	Vector with package names
`...`	Further arguments to be passed to `install.packages`

Examples

```
## Not run:
# try to load packages PP and MCMCglmm
library_install( pkg=c("PP", "MCMCglmm") )

## End(Not run)
```

`lm.cluster` *Cluster Robust Standard Errors for Linear Models and General Linear Models*

Description

Computes cluster robust standard errors for linear models (`stats::lm`) and general linear models (`stats::glm`) using the `multiwayvcov::vcovCL` function in the **sandwich** package.

Usage

```
lm.cluster(data, formula, cluster, weights=NULL, subset=NULL )

glm.cluster(data, formula, cluster, weights=NULL, subset=NULL, family="gaussian" )

## S3 method for class 'lm.cluster'
summary(object,...)
## S3 method for class 'glm.cluster'
summary(object,...)

## S3 method for class 'lm.cluster'
coef(object,...)
## S3 method for class 'glm.cluster'
coef(object,...)

## S3 method for class 'lm.cluster'
vcov(object,...)
## S3 method for class 'glm.cluster'
vcov(object,...)
```

Arguments

`data`	Data frame
`formula`	An R formula

`cluster`	Variable name for cluster variable contained in `data` or a vector with cluster identifiers
`subset`	Optional vector specifying a subset of observations to be used.
`weights`	Optional vector of weights to be used.
`family`	Description of the error distribution and link function to be used in the model, see `stats::glm`.
`...`	Further arguments to be passed to `stats::lm` and `stats::glm`
`object`	Object of class `lm.cluster` or `glm.cluster`

Value

List with following entries

`lm_res`	Value of `stats::lm`
`glm_res`	Value of `stats::glm`
`vcov`	Covariance matrix of parameter estimates

See Also

`stats::lm`, `stats::glm`, `sandwich::vcovCL`

Examples

```
## Not run:

#############################################################################
# EXAMPLE 1: Cluster robust standard errors data.ma01
#############################################################################

data(data.ma01)
dat <- data.ma01

#*** Model 1: Linear regression
mod1 <- miceadds::lm.cluster( data=dat, formula=read ~ hisei + female,
               cluster="idschool" )
coef(mod1)
vcov(mod1)
summary(mod1)

# estimate Model 1, but cluster is provided as a vector
mod1b <- miceadds::lm.cluster( data=dat, formula=read ~ hisei + female,
                 cluster=dat$idschool)
summary(mod1b)

#*** Model 2: Logistic regression
dat$highmath <- 1 * ( dat$math > 600 )   # create dummy variable
mod2 <- miceadds::glm.cluster( data=dat, formula=highmath ~ hisei + female,
                cluster="idschool", family="binomial")
coef(mod2)
vcov(mod2)
```

```
summary(mod2)

#############################################################################
# EXAMPLE 2: Cluster robust standard errors for multiply imputed datasets
#############################################################################

library(mitools)
data(data.ma05)
dat <- data.ma05

# imputation of the dataset: use six imputations
resp <- dat[, - c(1:2) ]
imp <- mice::mice( resp, method="norm", maxit=3, m=6 )
datlist <- miceadds::mids2datlist( imp )

# linear regression with cluster robust standard errors
mod <- lapply(  datlist, FUN=function(data){
            miceadds::lm.cluster( data=data, formula=denote ~ migrant+ misei,
                    cluster=dat$idclass )
            }  )
# extract parameters and covariance matrix
betas <- lapply( mod, FUN=function(rr){ coef(rr) } )
vars <- lapply( mod, FUN=function(rr){ vcov(rr) } )
# conduct statistical inference
summary( miceadds::pool_mi( qhat=betas, u=vars ) )

#------ compute global F-test for hypothesis that all predictors have zero coefficient values
library(mitml)
Nimp <- 6 # number of imputations
np <- length(betas[[1]])   # number of parameters
beta_names <- names(betas[[1]])
# define vector of parameters for which constraints should be tested
constraints <- beta_names[-1]
# create input for mitml::testConstraints function
qhat <- matrix( unlist(betas), ncol=Nimp)
rownames(qhat) <- beta_names
uhat <- array( unlist(vars), dim=c(np,np,Nimp))
dimnames(uhat) <- list( beta_names, beta_names, NULL )
# compute global F-test
Ftest <- mitml::testConstraints( qhat=betas, uhat=vars, constraints=constraints )
print(Ftest)

#############################################################################
# EXAMPLE 3: Comparing miceadds::lm.cluster() and lme4::lmer()
#############################################################################

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

# center hisei variable
dat$hisei <- dat$hisei - mean(dat$hisei)

# define school mean hisei
```

```
dat$hisei_gm <- miceadds::GroupMean(dat$hisei, dat$idschool, extend=TRUE)[,2]
dat$cluster_size <- miceadds::GroupSum(1+0*dat$hisei, dat$idschool, extend=TRUE)[,2]
dat$hisei_wc <- dat$hisei - dat$hisei_gm



#*** Model 1a: lm, hisei with clustering
mod1a <- miceadds::lm.cluster( data=dat, formula=read~hisei, cluster="idschool" )

#*** Model 1b: lmer, hisei
mod1b <- lme4::lmer( data=dat, formula=read~hisei+(1|idschool) )

cbind( coef(mod1a), fixef(mod1b))
 ##  > cbind( coef(mod1a), fixef(mod1b))
 ##                    [,1]        [,2]
 ##  (Intercept) 509.181691 507.8684752
 ##  hisei         1.524776   0.8161745

# variance explanation
vmod1b <- r2mlm::r2mlm(mod1b)
vmod1b$Decompositions

#*** Model 2a: lm, hisei and hisei_gm with clustering
mod2a <- miceadds::lm.cluster( data=dat, formula=read~hisei_wc+hisei_gm,
                                  cluster="idschool" )

#*** Model 2b: lmer, multilevel model
mod2b <- lme4::lmer( data=dat, formula=read~hisei_wc+hisei_gm + (1|idschool) )

# variance explanation
vmod2b <- r2mlm::r2mlm(mod2b)
vmod2b$Decompositions

cbind( coef(mod2a), fixef(mod2b))
 ##  > cbind( coef(mod2a), fixef(mod2b))
 ##                     [,1]        [,2]
 ##  (Intercept) 509.1816911 508.0478629
 ##  hisei_wc      0.7503773   0.7503773
 ##  hisei_gm      5.8424012   5.5681941


## End(Not run)
```

`lmer_vcov` *Statistical Inference for Fixed and Random Structure for Fitted Models in* **lme4**

Description

The function `lmer_vcov` conducts statistical inference for fixed coefficients and standard deviations and correlations of random effects structure of models fitted in the **lme4** package.

The function `lmer_pool` applies the Rubin formula for inference for fitted **lme4** models for multiply imputed datasets.

Usage

```
lmer_vcov(object, level=.95, use_reml=FALSE, ...)

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

lmer_vcov2(object, level=.95, ...)

lmer_pool( models, level=.95, ...)
## S3 method for class 'lmer_pool'
summary(object, digits=4, file=NULL, ...)

lmer_pool2( models, level=.95, ...)
```

Arguments

`object`	Fitted object in **lme4**
`level`	Confidence level
`use_reml`	Logical indicating whether REML estimates should be used for variance components (if provided)
`digits`	Number of digits used for rounding in summary
`file`	Optional file name for sinking output
`models`	List of models fitted in **lme4** for a multiply imputed dataset
`...`	Further arguments to be passed

Value

List with several entries:

`par_summary`	Parameter summary
`coef`	Estimated parameters
`vcov`	Covariance matrix of estimates
`...`	Further values

Author(s)

Function originally from Ben Bolker, http://rpubs.com/bbolker/varwald

See Also

`lme4::lmer`, `mitml::testEstimates`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Single model fitted in lme4
#############################################################################

library(lme4)
data(data.ma01, package="miceadds")
dat <- na.omit(data.ma01)

#* fit multilevel model
formula <- math ~ hisei + miceadds::gm( books, idschool ) + ( 1 + books | idschool )
mod1 <- lme4::lmer( formula, data=dat, REML=FALSE)
summary(mod1)

#* statistical inference
res1 <- miceadds::lmer_vcov( mod1 )
summary(res1)
coef(res1)
vcov(res1)

#############################################################################
# EXAMPLE 2: lme4 model for multiply imputed dataset
#############################################################################

library(lme4)
data(data.ma02, package="miceadds")
datlist <- miceadds::datlist_create(data.ma02)

#** fit lme4 model for all imputed datasets
formula <- math ~ hisei + miceadds::gm( books, idschool ) + ( 1 | idschool )
models <- list()
M <- length(datlist)
for (mm in 1:M){
    models[[mm]] <- lme4::lmer( formula, data=datlist[[mm]], REML=FALSE)
}

#** statistical inference
res1 <- miceadds::lmer_pool(models)
summary(res1)

## End(Not run)
```

`load.data` R *Utilities: Loading/Reading Data Files using* **miceadds**

Description

The function `load.data` is a wrapper function for loading or reading data frames or matrices.

The function `load.files` loads multiple files in a data frame.

Usage

```
load.data( filename, type=NULL, path=getwd(), load_fun=NULL, spss.default=TRUE, ...)

load.files( files, type=NULL, path=getwd(), ...)
```

Arguments

`filename`	Name of the data file (matrix or data frame). This can also be a part of the file name and the most recent file is loaded. `filename` can also be a vector which strings and a file is loaded which contains all the specified strings.
`type`	The type of file in which the data frame or matrix should be loaded. This can be `Rdata` (for R binary format, using `load.Rdata2`), `csv` (using `utils::read.csv2`), `csv2` (using `utils::read.csv`), `table` (using `utils::read.table`; the dataset must have the file extension `dat` or `txt`), `xlsx` (using `readxl::read_excel`; or using the extension `xls`), `sav` (using `foreign::read.spss`), `RDS`. If an alternative data loading function `load_fun` is chosen, `type` must be the file extension.
`path`	Directory from which the dataset should be loaded. It can also be set to `NULL` if the absolute path is already included in `filename`.
`load_fun`	User-specified loading function
`spss.default`	Optional logical which is only applied for `type="sav"` indicating whether the arguments `to.data.frame=TRUE` and `use.value.labels=FALSE` are used.
`...`	Further arguments to be passed to `load.Rdata2`, `utils::read.csv2`, `utils::read.csv`, `utils::read.table`, `readxl::read_excel`, `foreign::read.spss`, or `load_fun`.
`files`	Vector of file names

See Also

See also `load.Rdata` for loading R data frames.

See `save.Rdata` and `save.data` for saving/writing R data frames.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Toy example
#############################################################################

# load a data frame in the file "data_s3.Rdata" and save this
# as the object "dat.s3"
dat.s3 <- miceadds::load.data( filename="data_s3.Rdata", type="Rdata" )
print(str(dat.s3))
```

```
# load text input with base::readLines() function using the 'load_fun' argument
dat <- miceadds::load.data( "my_output_", type="Rout", load_fun=readLines, path=path)

## End(Not run)
```

`load.Rdata` R *Utilities: Loading* `Rdata` *Files in a Convenient Way*

Description

These functions loads a `Rdata` object saved as a data frame or a matrix in the current R environment. The function `load.Rdata` saves the loaded object in the global environment while `load.Rdata2` loads the object only specified environments. Hence, usage of `load.Rdata2` instead of `load.Rdata` is recommended.

Usage

```
load.Rdata(filename, objname)

load.Rdata2(filename, path=getwd(), RDS=FALSE)
```

Arguments

`filename`	Rdata file (matrix or data frame)
`objname`	Object name. This object will be a global variable in R.
`path`	Directory from which the dataset should be loaded
`RDS`	logical if object is saved as an RDS object

See Also

See also `save.Rdata` for saving data frames in a `Rdata` format.

See also: `base::load`, `base::save`

Examples

```
## Not run:
# load a data frame in the file "data_s3.Rdata" and save this
# as the object "dat.s3"
load.Rdata( filename="data_s3.Rdata", "dat.s3" )
head(dat.s3)

# Alternatively one can use the function
dat.s3 <- miceadds::load.Rdata2( filename="data_s3.Rdata")

## End(Not run)
```

`ma.scale2` *Standardization of a Matrix*

Description

This function performs a z-standardization for a numeric matrix. Note that in a case of a zero standard deviation all matrix entries are divided by a small number such that no `NaN`s occur.

Usage

```
ma.scale2(x, missings=FALSE)
```

Arguments

`x`	A numeric matrix in which missing values are permitted
`missings`	A logical indicating whether missings occur (or could occur) in the dataset

Value

A matrix

See Also

`base::scale`

Examples

```
#############################################################################
# EXAMPLE 1: z-standardization data.internet
#############################################################################

data(data.internet)
dat <- data.internet

# z-standardize all variables in this dataset
zdat <- miceadds::ma.scale2( dat, missings=TRUE )

## Not run:
#############################################################################
# SIMULATED EXAMPLE 2: Speed comparison for many cases and many variables
#############################################################################

set.seed(9786)
# 3000 cases, 200 variables
N <- 3000
p <- 200
# simulate some data
x <- matrix( stats::rnorm( N*p ), N, p )
x <- round( x, 2 )
```

```
# compare computation times for 10 replications
B <- 10
    s1 <- Sys.time()        # scale in R
for (bb in 1:B){
    res <- scale(x)
} ; s2 <- Sys.time() ; d1 <- s2-s1

    s1 <- Sys.time()        # scale in miceadds
for (bb in 1:B){
    res1 <- miceadds::ma.scale2(x)
} ; s2 <- Sys.time() ; d2 <- s2-s1

# scale in miceadds with missing handling
    s1 <- Sys.time()
for (bb in 1:B){
    res1 <- miceadds::ma.scale2(x,missings=TRUE)
} ; s2 <- Sys.time() ; d3 <- s2-s1
d1      # scale in R
d2      # scale in miceadds (no missing handling)
d3      # scale in miceadds (with missing handling)
  ##   > d1      # scale in R
  ##   Time difference of 1.622431 secs
  ##   > d2      # scale in miceadds (no missing handling)
  ##   Time difference of 0.156003 secs
  ##   > d3      # scale in miceadds (with missing handling)
  ##   Time difference of 0.2028039 secs

## End(Not run)
```

`ma.wtd.statNA` *Some Multivariate Descriptive Statistics for Weighted Data in* **miceadds**

Description

Some multivariate descriptive statistics for weighted datasets in **miceadds**. A list of (nested) multiply imputed data sets is also allowed as input.

Usage

```
ma.wtd.meanNA(data, weights=NULL, vars=NULL )

ma.wtd.sdNA(data, weights=NULL, vars=NULL, method="unbiased" )

ma.wtd.covNA(data, weights=NULL, vars=NULL, method="unbiased" )

ma.wtd.corNA(data, weights=NULL, vars=NULL, method="unbiased" )
```

```
ma.wtd.skewnessNA(data, weights=NULL, vars=NULL, method="unbiased" )

ma.wtd.kurtosisNA(data, weights=NULL, vars=NULL, method="unbiased" )

ma.wtd.quantileNA( data, weights=NULL, vars=NULL, type=7,
          probs=seq(0,1,.25) )
```

Arguments

`data`	Numeric data vector or data frame or objects of one of the classes `datlist`, `imputationList`, `mids`, `mids.1chain`, `nested.datlist`, `NestedImputationList` or `BIFIEdata`.
`weights`	Optional vector of sampling weights
`vars`	Optional vector of variable names
`method`	Computation method for covariances. These amount to choosing the divisor $(n-1)$ (`method="unbiased"`) instead of n (`method="ML"`). See `stats::cov.wt` for further details.
`type`	Quantile type. This specification follows `TAM::weighted_quantile`
`probs`	Vector of probabilities used for calculation of quantiles.

Details

Contrary to ordinary R practice, missing values are ignored in the calculation of descriptive statistics.

`ma.wtd.meanNA`	weighted means
`ma.wtd.sdNA`	weighted standard deviations
`ma.wtd.covNA`	weighted covariance matrix
`ma.wtd.corNA`	weighted correlation matrix
`ma.wtd.skewnessNA`	weighted skewness
`ma.wtd.kurtosisNA`	weighted (excess) kurtosis

Value

A vector or a matrix depending on the requested statistic.

Note

If `data` is of class `BIFIEdata` and no weights are specified, sample weights are extracted from the `BIFIEdata` object.

See Also

Some functions for weighted statistics: `stats::weighted.mean`, `stats::cov.wt`, `{Hmisc::wtd.var}`, `TAM::weighted_quantile`, ...

See `micombine.cor` for statistical inference of correlation coefficients.

Examples

```
#############################################################################
# EXAMPLE 1: Weighted statistics for a single dataset data.ma01
#############################################################################

data(data.ma01)
dat <- as.matrix(data.ma01[,-c(1:3)])

# weighted mean
ma.wtd.meanNA( dat, weights=data.ma01$studwgt )

# weighted SD
ma.wtd.sdNA( dat, weights=data.ma01$studwgt )

# weighted covariance for selected variables
ma.wtd.covNA( dat, weights=data.ma01$studwgt, vars=c("books","hisei") )

# weighted correlation
ma.wtd.corNA( dat, weights=data.ma01$studwgt )

## Not run:
# weighted skewness
ma.wtd.skewnessNA( dat[,"books"], weights=data.ma01$studwgt )
# compare with result in TAM
TAM::weighted_skewness( x=dat[,"books"], w=data.ma01$studwgt )

# weighted kurtosis
ma.wtd.kurtosisNA( dat, weights=data.ma01$studwgt, vars=c("books","hisei") )
# compare with TAM
TAM::weighted_kurtosis( dat[,"books"], w=data.ma01$studwgt )
TAM::weighted_kurtosis( dat[,"hisei"], w=data.ma01$studwgt )

#############################################################################
# EXAMPLE 2: Weighted statistics multiply imputed dataset
#############################################################################

library(mitools)
data(data.ma05)
dat <- data.ma05

# do imputations
resp <- dat[, - c(1:2) ]
# object of class mids
imp <- mice::mice( resp, method="norm", maxit=3, m=5 )
# object of class datlist
datlist <- miceadds::mids2datlist( imp )
# object of class imputationList
implist <- mitools::imputationList(datlist)

# weighted means
ma.wtd.meanNA(datlist)
ma.wtd.meanNA(implist)
```

```
ma.wtd.meanNA(imp)

# weighted quantiles
ma.wtd.quantileNA( implist, weights=data.ma05$studwgt, vars=c("manote","Dscore"))

#############################################################################
# EXAMPLE 3: Weighted statistics nested multiply imputed dataset
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2   # list of 5 datasets containing 5 plausible values

#** define imputation method and predictor matrix
data <- datlist[[1]]
V <- ncol(data)
# variables
vars <- colnames(data)
# variables not used for imputation
vars_unused <- miceadds::scan.vec("IDSTUD TOTWGT  JKZONE  JKREP" )
#- define imputation method
impMethod <- rep("norm", V )
names(impMethod) <- vars
impMethod[ vars_unused ] <- ""
#- define predictor matrix
predM <- matrix( 1, V, V )
colnames(predM) <- rownames(predM) <- vars
diag(predM) <- 0
predM[, vars_unused ] <- 0

# object of class mids.nmi
imp1 <- miceadds::mice.nmi( datlist, method=impMethod, predictorMatrix=predM,
                m=4, maxit=3 )
# object of class nested.datlist
datlist <- miceadds::mids2datlist(imp1)
# object of class NestedImputationList
imp2 <- miceadds::NestedImputationList(datlist)

# weighted correlations
vars <- c("books","ASMMAT","likesc")
ma.wtd.corNA( datlist,  vars=vars )
ma.wtd.corNA( imp2,  vars=vars )
ma.wtd.corNA( imp1,  vars=vars )

#############################################################################
# EXAMPLE 4: Multiply imputed datasets in BIFIEdata format
#############################################################################

library(BIFIEsurvey)
data(data.timss1, package="BIFIEsurvey")
data(data.timssrep, package="BIFIEsurvey")

# create BIFIEdata object
```

```
bdat <- BIFIEsurvey::BIFIE.data( data.list=data.timss1, wgt=data.timss1[[1]]$TOTWGT,
            wgtrep=data.timssrep[, -1 ] )
summary(bdat)
# create BIFIEdata object in a compact way
bdat2 <- BIFIEsurvey::BIFIE.data( data.list=data.timss1, wgt=data.timss1[[1]]$TOTWGT,
            wgtrep=data.timssrep[, -1 ], cdata=TRUE)
summary(bdat2)

# compute skewness
ma.wtd.skewnessNA( bdat, vars=c("ASMMAT", "books" ) )
ma.wtd.skewnessNA( bdat2, vars=c("ASMMAT", "books" ) )

#############################################################################
# EXAMPLE 5: Nested multiply imputed datasets in BIFIEdata format
#############################################################################

data(data.timss4, package="BIFIEsurvey")
data(data.timssrep, package="BIFIEsurvey")

# nested imputed dataset, save it in compact format
bdat <- BIFIE.data( data.list=data.timss4, wgt=data.timss4[[1]][[1]]$TOTWGT,
            wgtrep=data.timssrep[, -1 ], NMI=TRUE, cdata=TRUE )
summary(bdat)
# skewness
ma.wtd.skewnessNA( bdat, vars=c("ASMMAT", "books" ) )

## End(Not run)
```

`ma_lme4_formula` *Utility Functions for Working with* **lme4** *Formula Objects*

Description

Utility functions for working with **lme4** formula objects. The function `ma_lme4_formula_terms` decomposes an **lme4** formula into several parts for further processing.

Usage

```
ma_lme4_formula_terms(formula)

ma_lme4_formula_design_matrices(formula, data, start_index=0, formula_terms=NULL,
         only_design_matrices=FALSE)
```

Arguments

`formula`	An R formula object
`data`	Data frame
`start_index`	Starting index for cluster identifiers

`formula_terms`	Optional argument with processed formula terms using the function `ma_lme4_formula_terms`
`only_design_matrices`	Logical indicating whether only design matrices should be created

Value

List with several entries

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Splitting a lme4 formula
#############################################################################

#*** formula for a multilevel model
formula <- y ~ I( miceadds::cwc(x, idcluster)) + z + I(z^2) + I( miceadds::gm(x, idcluster) ) + w +
                        ( x + I(x^2) | idcluster)  + (0 +  w | idcluster ) +
                        ( 0 + I(as.factor(f)) | idcluster)
miceadds::ma_lme4_formula_terms(formula)

#*** formula for a single level model
formula2 <- y ~ I( miceadds::cwc(x, idcluster)) + z + I(z^2) + I( miceadds::gm(x, idcluster) ) + w
miceadds::ma_lme4_formula_terms(formula2)

#############################################################################
# EXAMPLE 2: Design matrices for multilevel model
#############################################################################

data(data.ma07, package="miceadds")
dat <- data.ma07

formula <- x1 ~ x2 + I( miceadds::gm( x2, id2)) + I( miceadds::gm( x2, id3)) + y1 + z1 +
                    ( x2 | id2:id3 ) + ( 1 | id3 ) + ( 0 + x2 | id3 )
res <- miceadds::ma_lme4_formula_design_matrices(formula, data=dat)
str(res)

## End(Not run)
```

`ma_rmvnorm` *Simulating Normally Distributed Data*

Description

Some functions for normally distributed data.

The function `ma_rmvnorm` is like `mvtnorm::rmvnorm`, but allows for a covariance matrix `sigma` which can have zero variances.

Usage

```
ma_rmvnorm(n, mu=NULL, sigma, eps=1e-10)
```

Arguments

n	Sample size
mu	Mean vector
sigma	Covariance matrix
eps	Trimming constant for zero variances

Value

Matrix of simulated values

See Also

`MASS::mvrnorm`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Two-dimensional simulation with zero variance at dimension 1
#############################################################################

sigma <- matrix( c(0,0,0,1), nrow=2, ncol=2)
miceadds::ma_rmvnorm( n=10, sigma=sigma )

## End(Not run)
```

`mi.anova` *Analysis of Variance for Multiply Imputed Data Sets (Using the D_2 Statistic)*

Description

This function combines F values from analysis of variance using the D_2 statistic which is based on combining χ^2 statistics (see Allison, 2001, Grund, Luedtke & Robitzsch, 2016; `micombine.F`, `micombine.chisquare`).

Usage

```
mi.anova(mi.res, formula, type=2)
```

Arguments

`mi.res`	Object of class `mids` or `mids.1chain`
`formula`	Formula for `lm` function. Note that this can be also a string.
`type`	Type for ANOVA calculations. For `type=3`, the `car::Anova` function from the **car** package is used.

Value

A list with the following entries:

`r.squared`	Explained variance R^2
`anova.table`	ANOVA table

References

Allison, P. D. (2002). *Missing data.* Newbury Park, CA: Sage.

Grund, S., Luedtke, O., & Robitzsch, A. (2016). Pooling ANOVA results from multiply imputed datasets: A simulation study. *Methodology, 12*(3), 75-88. doi:10.1027/16142241/a000111

See Also

This function uses `micombine.F` and `micombine.chisquare`.

See `mice::pool.compare` and `mitml::testModels` for model comparisons based on the D_1 statistic. The D_2 statistic is also included in `mitml::testConstraints`.

The D_1, D_2 and D_3 statistics are also included in the **mice** package in functions `mice::D1`, `mice::D2` and `mice::D3`.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: nhanes2 data | two-way ANOVA
#############################################################################

library(mice)
library(car)
data(nhanes2, package="mice")
set.seed(9090)

# nhanes data in one chain and 8 imputed datasets
mi.res <- miceadds::mice.1chain( nhanes2, burnin=4, iter=20, Nimp=8 )
# 2-way analysis of variance (type 2)
an2a <- miceadds::mi.anova(mi.res=mi.res, formula="bmi ~ age * chl" )

# test of interaction effects using mitml::testModels()
mod1 <- with( mi.res, stats::lm( bmi ~ age*chl ) )
mod0 <- with( mi.res, stats::lm( bmi ~ age+chl ) )

mitml::testModels(model=mod1$analyses, null.model=mod0$analyses, method="D1")
```

```
mitml::testModels(model=mod1$analyses, null.model=mod0$analyses, method="D2")

# 2-way analysis of variance (type 3)
an2b <- miceadds::mi.anova(mi.res=mi.res, formula="bmi ~ age * chl", type=3)

#****** analysis based on first imputed dataset

# extract first dataset
dat1 <- mice::complete( mi.res$mids )

# type 2 ANOVA
lm1 <- stats::lm( bmi ~ age * chl, data=dat1 )
summary( stats::aov( lm1 ) )
# type 3 ANOVA
lm2 <- stats::lm( bmi ~ age * chl, data=dat1, contrasts=list(age=contr.sum))
car::Anova(mod=lm2, type=3)

## End(Not run)
```

`mice.1chain` *Multiple Imputation by Chained Equations using One Chain*

Description

This function modifies the `mice::mice` function to multiply impute a dataset using a long chain instead of multiple parallel chains which is the approach employed in `mice::mice`.

Usage

```
mice.1chain(data, burnin=10, iter=20, Nimp=10, method=NULL,
     where=NULL, visitSequence=NULL, blots=NULL, post=NULL,
     defaultMethod=c("pmm", "logreg", "polyreg", "polr"),
     printFlag=TRUE, seed=NA, data.init=NULL, ...)

## S3 method for class 'mids.1chain'
summary(object,...)

## S3 method for class 'mids.1chain'
print(x, ...)

## S3 method for class 'mids.1chain'
plot(x, plot.burnin=FALSE, ask=TRUE, ...)
```

Arguments

`data`	Numeric matrix
`burnin`	Number of burn-in iterations
`iter`	Total number of imputations (larger than `burnin`)

`Nimp`	Number of imputations
`method`	See `mice::mice`
`where`	See `mice::mice`
`visitSequence`	See `mice::mice`
`blots`	See `mice::mice`
`post`	See `mice::mice`
`defaultMethod`	See `mice::mice`
`printFlag`	See `mice::mice`
`seed`	See `mice::mice`
`data.init`	See `mice::mice`
`object`	Object of class `mids.1chain`
`x`	Object of class `mids.1chain`
`plot.burnin`	An optional logical indicating whether burnin iterations should be included in the traceplot
`ask`	An optional logical indicating a user request for viewing next plot
`...`	See `mice::mice`

Value

A list with following entries

`midsobj`	Objects of class `mids`
`datlist`	List of multiply imputed datasets
`datalong`	Original and imputed dataset in the long format
`implist`	List of `mids` objects for every imputation
`chainMpar`	Trace of means for all imputed variables
`chainVarpar`	Trace of variances for all imputed variables

Note

Multiple imputation can also be used for determining causal effects (see Example 3; Schafer & Kang, 2008).

See Also

`mice::mice`

Examples

```
## Not run:

#############################################################################
# EXAMPLE 1: One chain nhanes data
#############################################################################

library(mice)
data(nhanes, package="mice")
set.seed(9090)

# nhanes data in one chain
imp.mi1 <- miceadds::mice.1chain( nhanes, burnin=5, iter=40, Nimp=4,
                 method=rep("norm", 4 ) )
summary(imp.mi1)       # summary of mids.1chain
plot( imp.mi1 ) # trace plot excluding burnin iterations
plot( imp.mi1, plot.burnin=TRUE ) # trace plot including burnin iterations

# select mids object
imp.mi2 <- imp.mi1$midsobj
summary(imp.mi2)    # summary of mids

# apply mice functionality lm.mids
mod <- with( imp.mi2, stats::lm( bmi ~ age ) )
summary( mice::pool( mod ) )

#############################################################################
# EXAMPLE 2: One chain (mixed data: numeric and factor)
#############################################################################

library(mice)
data(nhanes2, package="mice")
set.seed(9090)

# nhanes2 data in one chain
imp.mi1 <- miceadds::mice.1chain( nhanes2, burnin=5, iter=25, Nimp=5 )
# summary
summary( imp.mi1$midsobj )

#############################################################################
# EXAMPLE 3: Multiple imputation with counterfactuals for estimating
#            causal effects (average treatment effects)
# Schafer, J. L., & Kang, J. (2008). Average causal effects from nonrandomized
#    studies: a practical guide and simulated example.
#    Psychological Methods, 13, 279-313.
#############################################################################

data(data.ma01)
dat <- data.ma01[, 4:11]

# define counterfactuals for reading score for students with and
# without migrational background
```

```
dat$read.migrant1 <- ifelse( paste(dat$migrant)==1, dat$read, NA )
dat$read.migrant0 <- ifelse( paste(dat$migrant)==0, dat$read, NA )

# define imputation method
impmethod <- rep("pls", ncol(dat) )
names(impmethod) <- colnames(dat)

# define predictor matrix
pm <- 4*(1 - diag( ncol(dat) ) )    # 4 - use all interactions
rownames(pm) <- colnames(pm) <- colnames(dat)
pm[ c( "read.migrant0", "read.migrant1"), ] <- 0
# do not use counterfactuals for 'read' as a predictor
pm[, "read.migrant0"] <- 0
pm[, "read.migrant1"] <- 0
# define control variables for creation of counterfactuals
pm[ c( "read.migrant0", "read.migrant1"), c("hisei","paredu","female","books") ] <- 4
  ##   > pm
 ##               math read migrant books hisei paredu female urban read.migrant1 read.migrant0
  ##   math           0    4       4     4     4      4      4     4             0             0
  ##   read           4    0       4     4     4      4      4     4             0             0
  ##   migrant        4    4       0     4     4      4      4     4             0             0
  ##   books          4    4       4     0     4      4      4     4             0             0
  ##   hisei          4    4       4     4     0      4      4     4             0             0
  ##   paredu         4    4       4     4     4      0      4     4             0             0
  ##   female         4    4       4     4     4      4      0     4             0             0
  ##   urban          4    4       4     4     4      4      4     0             0             0
  ##   read.migrant1  0    0       0     4     4      4      4     0             0             0
  ##   read.migrant0  0    0       0     4     4      4      4     0             0             0

# imputation using mice function and PLS imputation with
# predictive mean matching method 'pmm6'
imp <- mice::mice( dat, method=impmethod, predictorMatrix=pm,
            maxit=4, m=5, pls.impMethod="pmm5" )

#*** Model 1: Raw score difference
mod1 <- with( imp, stats::lm( read ~ migrant ) )
smod1 <- summary( mice::pool(mod1) )
  ##   > smod1
  ##                  est    se      t     df Pr(>|t|)  lo 95  hi 95 nmis    fmi lambda
  ##   (Intercept) 510.21 1.460 349.37 358.26        0 507.34 513.09   NA 0.1053 0.1004
  ##   migrant     -43.38 3.757 -11.55  62.78        0 -50.89 -35.87  404 0.2726 0.2498

#*** Model 2: ANCOVA - regression adjustment
mod2 <- with( imp, stats::lm( read ~ migrant + hisei + paredu + female + books) )
smod2 <- summary( mice::pool(mod2) )
  ##   > smod2
 ##                   est      se      t      df  Pr(>|t|)    lo 95   hi 95 nmis      fmi   lambda
 ##  (Intercept) 385.1506 4.12027 93.477 3778.66 0.000e+00 377.0725 393.229   NA 0.008678 0.008153
 ##  migrant     -29.1899 3.30263 -8.838   87.46 9.237e-14 -35.7537 -22.626  404 0.228363 0.210917
 ##  hisei         0.9401 0.08749 10.745  160.51 0.000e+00   0.7673   1.113  733 0.164478 0.154132
 ##  paredu        2.9305 0.79081  3.706   41.34 6.190e-04   1.3338   4.527  672 0.339961 0.308780
 ##  female       38.1719 2.26499 16.853 1531.31 0.000e+00  33.7291  42.615    0 0.041093 0.039841
 ##  books        14.0113 0.88953 15.751  154.71 0.000e+00  12.2541  15.768  423 0.167812 0.157123
```

```
#*** Model 3a: Estimation using counterfactuals
mod3a <- with( imp, stats::lm( I( read.migrant1 - read.migrant0) ~ 1 ) )
smod3a <- summary( mice::pool(mod3a) )
  ##   > smod3a
  ##                  est    se      t    df Pr(>|t|)  lo 95  hi 95 nmis    fmi lambda
  ##   (Intercept) -22.54 7.498 -3.007 4.315  0.03602 -42.77 -2.311   NA 0.9652 0.9521

#*** Model 3b: Like Model 3a but using student weights
mod3b <- with( imp, stats::lm( I( read.migrant1 - read.migrant0) ~ 1,
                        weights=data.ma01$studwgt ) )
smod3b <- summary( mice::pool(mod3b) )
  ##   > smod3b
  ##                  est    se      t  df Pr(>|t|)  lo 95  hi 95 nmis    fmi lambda
  ##   (Intercept) -21.88 7.605 -2.877 4.3  0.04142 -42.43 -1.336   NA 0.9662 0.9535

#*** Model 4: Average treatment effect on the treated (ATT, migrants)
#             and non-treated (ATN, non-migrants)
mod4 <- with( imp, stats::lm( I( read.migrant1 - read.migrant0) ~ 0 + as.factor( migrant) )  )
smod4 <- summary( mice::pool(mod4) )
  ##   > smod4
 ##                         est    se      t    df Pr(>|t|)  lo 95   hi 95 nmis    fmi lambda
 ##  as.factor(migrant)0 -23.13 8.664 -2.669  4.27 0.052182 -46.59  0.3416   NA 0.9682 0.9562
 ##  as.factor(migrant)1 -19.95 5.198 -3.837 19.57 0.001063 -30.81 -9.0884   NA 0.4988 0.4501
# ATN=-23.13 and ATT=-19.95

## End(Not run)
```

`mice.impute.2l.contextual.pmm`

Imputation by Predictive Mean Matching or Normal Linear Regression with Contextual Variables

---

Description

This imputation method imputes a variable using linear regression with predictive mean matching as the imputation method. Including a contextual effects means that an aggregated variable at a cluster level is included as a further covariate.

Usage

```
mice.impute.2l.contextual.pmm(y, ry, x, type, imputationWeights=NULL,
     interactions=NULL, quadratics=NULL, pls.facs=NULL, ...)

mice.impute.2l.contextual.norm(y, ry, x, type, ridge=10^(-5),
  imputationWeights=NULL, interactions=NULL, quadratics=NULL, pls.facs=NULL, ...)
```

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`	Type of predictor variables. `type=-2` refers to the cluster variable, `type=2` denotes a variable for which also a contextual effect is included and `type=1` denotes all other variables which are included as 'ordinary' predictors.
`imputationWeights`	Optional vector of sample weights
`interactions`	Vector of variable names used for creating interactions
`quadratics`	Vector of variable names used for creating quadratic terms
`pls.facs`	Number of factors used in partial least dimension reduction (if requested)
`...`	Further arguments to be passed
`ridge`	Ridge parameter in the diagonal of $\boldsymbol{X}'\boldsymbol{X}$

Value

A vector of length `nmis=sum(!ry)` with imputed values.

See Also

For imputations at level 2 variables see `mice::mice.impute.2lonly.norm` and `mice::mice.impute.2lonly.pmm`.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Sequential hierarchical imputation for data.ma05 dataset
#############################################################################

data(data.ma05)
dat <- data.ma05

# define predictor matrix
predM <- mice::make.predictorMatrix(data=dat)
# exclude student IDs
predM[, "idstud"] <- 0
# define idclass as the cluster variable (type=-2)
predM[, "idclass" ] <- -2

# initialize with norm method
impMethod <- mice::make.method(data=dat)
names(impMethod) <- names( imp0$method )
impMethod[ c("idstud","idclass")] <- ""

#*****
# STUDENT LEVEL (Level 1)
```

```
# Use a random slope model for Dscore and Mscore as the imputation method.
# Here, variance homogeneity of residuals is assumed (contrary to
# the 2l.norm imputation method in the mice package).
impMethod[ c(”Dscore”, ”Mscore”) ] <- ”2l.pan”
predM[ c(”Dscore”,”Mscore”), ”misei” ] <- 2    # random slopes on 'misei'
predM[, ”idclass” ] <- -2

# For imputing 'manote' and 'denote' use contextual effects (i.e. cluszer means)
# of variables 'misei' and 'migrant'
impMethod[ c(”denote”, ”manote”) ] <- ”2l.contextual.pmm”
predM[ c(”denote”, ”manote”), c(”misei”,”migrant”)] <- 2

# Use no cluster variable 'idclass' for imputation of 'misei'
impMethod[ ”misei”] <- ”norm”
predM[ ”misei”, ”idclass”] <- 0 # use no multilevel imputation model

# Variable migrant: contextual effects of Dscore and misei
impMethod[ ”migrant”] <- ”2l.contextual.pmm”
predM[ ”migrant”, c(”Dscore”,  ”misei” ) ] <- 2
predM[ ”migrant”, ”idclass” ] <- -2

#****
# CLASS LEVEL (Level 2)
# impute 'sprengel' and 'groesse' at the level of classes
impMethod[ ”sprengel”] <- ”2lonly.pmm”
impMethod[ ”groesse”] <- ”2lonly.norm”
predM[ c(”sprengel”,”groesse”), ”idclass” ] <- -2

# do imputation
imp <- mice::mice( dat, predictorMatrix=predM, m=3,  maxit=4,
           method=impMethod, paniter=100)
summary(imp)

#**** imputation model 2 with PLS dimension reduction

# define some interaction effects
interactions <- list( manote=c(”migrant”, ”misei”) )
# number of PLS factors (5 factors)
pls.facs <- list( manote=5 )

# do imputation
imp2 <- mice::mice( dat, predictorMatrix=predM, interactions=interactions,
            pls.facs=pls.facs, method=impMethod, paniter=100)
summary(imp2)

## End(Not run)
```

`mice.impute.2l.latentgroupmean.ml`
Imputation of Latent and Manifest Group Means for Multilevel Data

---

Description

The imputation method `2l.latentgroupmean` imputes a latent group mean assuming an infinite population of subjects within a group (Grund, Luedtke & Robitzsch, 2018; see also Luedtke, Marsh, Robitzsch, Trautwein, Asparouhov & Muthen, 2008 or Croon & van Veldhoven, 2007). Therefore, unreliability of group means when treating subjects as indicators is taken into account.

The imputation method `mice.impute.2l.groupmean` just imputes (i.e. computes) the manifest group mean. See also `mice::mice.impute.2lonly.mean`.

The imputation method `mice.impute.2l.groupmean.elim` computes the group mean eliminating the subject under study from the calculation. Therefore, this imputation method will lead to different values of individuals within the same group.

Usage

```
mice.impute.2l.latentgroupmean.ml(y, ry, x, type, pls.facs=NULL,
    imputationWeights=NULL, interactions=NULL, quadratics=NULL,
    EAP=FALSE, ...)

mice.impute.2l.latentgroupmean.mcmc(y, ry, x, type, pls.facs=NULL,
    imputationWeights=NULL, interactions=NULL, quadratics=NULL,
  mcmc.burnin=100, mcmc.adapt=100, mcmc.iter=1000, draw.fixed=TRUE, EAP=FALSE, ...)

mice.impute.2l.groupmean(y, ry, x, type, grmeanwarning=TRUE, ...)

mice.impute.2l.groupmean.elim(y, ry, x, type, ...)
```

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`	Type of predictor variables. `type=-2` refers to the cluster variable, `type=2` denotes a variable for which also a (latent) group mean should be calculated. Predictors with `type=1` denote all other variables.
`pls.facs`	Number of factors used for PLS regression (optional).
`imputationWeights`	Optional vector of sample weights.
`interactions`	Vector of variable names used for creating interactions
`quadratics`	Vector of variable names used for creating quadratic terms
`draw.fixed`	Optional logical indicating whether parameters for fixed effects should be sampled.
`EAP`	Logical indicating whether EAPs should be used for imputation. The default `FALSE` corresponds to sampling from the posterior distribution.
`mcmc.burnin`	Number of MCMC burn-in iterations.
`mcmc.adapt`	Number of MCMC iterations in adaptation phase.

`mcmc.iter`	Total number of MCMC iterations.
`grmeanwarning`	An optional logical indicating whether some group means cannot be calculated.
`...`	Further arguments to be passed.

Details

The imputation of the latent group mean uses the `lme4::lmer` function of the **lme4** package for `mice.impute.2l.latentgroupmean.ml` and the `MCMCglmm::MCMCglmm` function of the **MCMCglmm** package for `mice.impute.2l.latentgroupmean.ml`. Latent group mean imputation also follows Mislevy (1991).

Value

A vector of length `y` containing imputed group means.

References

Croon, M. A., & van Veldhoven, M. J. (2007). Predicting group-level outcome variables from variables measured at the individual level: a latent variable multilevel model. *Psychological Methods, 12*(1), 45-57. doi:10.1037/1082989X.12.1.45

Grund, S., Luedtke, O., & Robitzsch, A. (2018). Multiple imputation of missing data at level 2: A comparison of fully conditional and joint modeling in multilevel designs. *Journal of Educational and Behavioral Statistics, 43*(3), 316-353. doi:10.3102/1076998617738087

Luedtke, O., Marsh, H. W., Robitzsch, A., Trautwein, U., Asparouhov, T., & Muthen, B. (2008). The multilevel latent covariate model: a new, more reliable approach to group-level effects in contextual studies. *Psychological Methods, 13*(3), 203-229. doi:10.1037/a0012869

Mislevy, R. J. (1991). Randomization-based inference about latent variables from complex samples. *Psychometrika, 56*(2), 177-196. doi:10.1007/BF02294457

See Also

`mice::mice.impute.2lonly.mean`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Two-level imputation data.ma05 dataset with imputation
#            of a latent group mean
#############################################################################

data(data.ma05)
dat <- data.ma05

# include manifest group mean for 'Mscore'
dat$M.Mscore <- NA
# include latent group group for 'Mscore'
dat$LM.Mscore <- NA    #=> LM: latent group mean
```

```
# define predictor matrix
predM <- mice::make.predictorMatrix(data=dat)
# exclude student ISs
predM[, "idstud"] <- 0
# idclass is the cluster identifier
predM[, "idclass" ] <- -2

# define imputation methods
impMethod <- mice::make.method(data=dat)
# initialize with norm
impMethod <- rep( "norm", length(impMethod) )
names(impMethod) <- names( imp$method )
impMethod[ c("idstud","idclass")] <- ""

#*****
# STUDENT LEVEL (Level 1)

# Use a random slope model for Dscore and Mscore as the imputation method.
# Here, variance homogeneity of residuals is assumed (contrary to
# the 2l.norm imputation method in the mice package).
impMethod[ c("Dscore", "Mscore") ] <- "2l.pan"
predM[ c("Dscore","Mscore"), "misei" ] <- 2    # random slopes on 'misei'
predM[, "idclass" ] <- -2

# For imputing 'manote' and 'denote' use contextual effects (i.e. cluster means)
# of variables 'misei' and 'migrant'
impMethod[ c("denote", "manote") ] <- "2l.contextual.pmm"
predM[ c("denote", "manote"), c("misei","migrant")] <- 2

# Use no cluster variable 'idclass' for imputation of 'misei'
impMethod[ "misei"] <- "norm"
predM[ "misei", "idclass"] <- 0 # use no multilevel imputation model

# Variable migrant: contextual effects of Dscore and misei
impMethod[ "migrant"] <- "2l.contextual.pmm"
predM[ "migrant", c("Dscore",  "misei" ) ] <- 2
predM[ "migrant", "idclass" ] <- -2

#****
# CLASS LEVEL (Level 2)
# impute 'sprengel' and 'groesse' at the level of classes
impMethod[ "sprengel"] <- "2lonly.pmm2"
impMethod[ "groesse"] <- "2lonly.norm2"
predM[ c("sprengel","groesse"), "idclass" ] <- -2

# manifest group mean for Mscore
impMethod[ "M.Mscore" ] <- "2l.groupmean"
# latent group mean for Mscore
impMethod[ "LM.Mscore" ] <- "2l.latentgroupmean.ml"
predM[ "M.Mscore", "Mscore" ] <- 2

# covariates for latent group mean of 'Mscore'
predM[ "LM.Mscore", "Mscore" ] <- 2
```

```
predM[ "LM.Mscore", c( "Dscore", "sprengel" ) ] <- 1

# do imputations
imp <- mice::mice( dat, predictorMatrix=predM, m=3,  maxit=4,
         method=impMethod, allow.na=TRUE, pan.iter=100)

## End(Not run)
```

`mice.impute.2lonly.function`

Imputation at Level 2 (in **miceadds***)*

Description

The imputation method `mice.impute.2lonly.function` is a general imputation function for level 2 imputation which allow any defined imputation function at level 1 in **mice**.

Usage

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

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. Only numeric variables are permitted for usage of this function.
`wy`	Logical vector of `length(y)` indicating at which positions imputations should be conducted.
`type`	Cluster identifier can be specified by `-2` for aggregation. However, we recommend to use the argument `cluster_var` for specifying the cluster variable at Level 2. Predictors must be specified by `1`.
`imputationFunction`	Imputation function for **mice**. Any imputation method which is defined at level 1 can be used for level 2 imputation.
`cluster_var`	Cluster identifier for Level 2 units
`...`	Other named arguments.

Value

A vector of length `nmis` with imputations.

See Also

See `mice::mice.impute.2lonly.norm` and the `mice::mice.impute.2lonly.pmm` function.

See also the **jomo** package (`jomo::jomo2`) for joint multilevel imputation of level 1 and level 2 variables.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation of level 2 variables
#############################################################################

#**** Simulate some data
# x,y ... level 1 variables
# v,w ... level 2 variables

set.seed(987)
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( stats::rnorm( G, sd=sqrt(rho)), each=n ) + stats::rnorm(G*n, sd=sqrt(1-rho))
w <- rep( round( stats::rnorm(G ), 2 ), each=n )
v <- rep( round( stats::runif( G, 0, 3 ) ), each=n )
x <-  stats::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 + stats::rnorm( G*n, sd=sqrt( 1 - rho.miss ) ) <
                stats::qnorm(missrate), "y" ] <- NA
dfr[ rep( stats::rnorm(G), each=n ) < stats::qnorm(missrate), "w" ] <- NA
dfr[ rep( stats::rnorm(G), each=n ) < stats::qnorm(missrate), "v" ] <- NA

#* initial predictor matrix and imputation methods
predM <- mice::make.predictorMatrix(data=dfr)
impM <- mice::make.method(data=dfr)

#...
# multilevel imputation
predM1 <- predM
predM1[c("w","v","y"),"group"] <- c(0,0,-2)
predM1["y","x"] <- 1        # fixed x effects imputation
impM1 <- impM
impM1[c("y","w","v")] <- c("2l.continuous", "2lonly.function", "2lonly.function" )
# define imputation functions
imputationFunction <- list( "w"="sample", "v"="pmm5" )
# define cluster variable
cluster_var <- list( "w"="group", "v"="group" )

# impute
```

```
imp1 <- mice::mice( as.matrix(dfr), m=1, predictorMatrix=predM1, method=impM1, maxit=5,
            imputationFunction=imputationFunction, cluster_var=cluster_var )

## End(Not run)
```

`mice.impute.bygroup` *Groupwise Imputation Function*

Description

The function `mice.impute.bygroup` performs groupwise imputation for arbitrary imputation methods defined in **mice**.

Usage

```
mice.impute.bygroup(y, ry, x, wy=NULL, group, imputationFunction, ...)
```

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.
`wy`	Vector of `length(y)` indicating which entries should be imputed.
`group`	Name of grouping variable
`imputationFunction`	Imputation method for **mice**
`...`	More arguments to be passed to imputation function

Value

Vector of imputed values

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Cluster-specific imputation for some variables
#############################################################################

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

# use sub-dataset
dat <- dat[ dat$idschool <=1006, ]
V <- ncol(dat)
```

```
# create initial predictor matrix and imputation methods
predictorMatrix <- matrix( 1, nrow=V, ncol=V)
diag(predictorMatrix) <- 0
rownames(predictorMatrix) <- colnames(predictorMatrix) <- colnames(dat)
predictorMatrix[, c("idstud", "studwgt","urban" ) ] <- 0
method <- rep("norm", V)
names(method) <- colnames(dat)

#** groupwise imputation of variable books
method["books"] <- "bygroup"
# specify name of the grouping variable ('idschool') and imputation method ('norm')
group <- list( "books"="idschool" )
imputationFunction <- list("books"="norm" )

#** conduct multiple imputation in mice
imp <- mice::mice( dat, method=method, predictorMatrix=predictorMatrix,
            m=1, maxit=1, group=group, imputationFunction=imputationFunction )

#############################################################################
# EXAMPLE 2: Group-wise multilevel imputation '2l.pan'
#############################################################################

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

# select data
dat <- dat[, c("idschool","hisei","books","female") ]
V <- ncol(dat)
dat <- dat[ ! is.na( dat$books), ]
# define factor variable

dat$books <- as.factor(dat$books)
# create initial predictor matrix and imputation methods
predictorMatrix <- matrix( 0, nrow=V, ncol=V)
rownames(predictorMatrix) <- colnames(predictorMatrix) <- colnames(dat)
predictorMatrix["idschool", ] <- 0
predictorMatrix[ "hisei", "idschool" ] <- -2
predictorMatrix[ "hisei", c("books","female") ] <- 1
method <- rep("", V)
names(method) <- colnames(dat)
method["hisei"] <- "bygroup"
group <- list( "hisei"="female" )
imputationFunction <- list("hisei"="2l.pan" )

#** conduct multiple imputation in mice
imp <- mice::mice( dat, method=method, predictorMatrix=predictorMatrix,
            m=1, maxit=1, group=group, imputationFunction=imputationFunction )
str(imp)

## End(Not run)
```

`mice.impute.catpmm` *Imputation of a Categorical Variable Using Multivariate Predictive Mean Matching*

Description

Imputes a categorical variable using multivariate predictive mean matching.

Usage

```
mice.impute.catpmm(y, ry, x, donors=5, ridge=10^(-5), ...)
```

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.
`donors`	Number of donors used for random sampling of nearest neighbors in imputation
`ridge`	Numerical constant used for avioding collinearity issues. Noise is added to covariates.
`...`	Further arguments to be passed

Details

The categorical outcome variable is recoded as a vector of dummy variables. A multivariate linear regression is specified for computing predicted values. The L1 distance (i.e., sum of absolute deviations) is utilized for predictive mean matching. Predictive mean matching for categorical variables has been proposed by Meinfelder (2009) using a multinomial regression instead of ordinary linear regression.

Value

A vector of length `nmis=sum(!ry)` with imputed values.

References

Meinfelder, F. (2009). *Analysis of Incomplete Survey Data - Multiple Imputation via Bayesian Bootstrap Predictive Mean Matching*. Dissertation thesis. University of Bamberg, Germany. `https://fis.uni-bamberg.de/handle/uniba/213`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation internat data
#############################################################################

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

#** empty imputation
imp0 <- mice::mice(dat, m=1, maxit=0)
method <- imp0$method
predmat <- imp0$predictorMatrix

#** define factor variable

dat1 <- dat
dat1[,1] <- as.factor(dat1[,1])
method[1] <- "catpmm"

#** impute with 'catpmm''
imp <- mice::mice(dat1, method=method1, m=5)
summary(imp)

## End(Not run)
```

`mice.impute.constant` *Imputation Using a Fixed Vector*

Description

Defines a fixed vector of values for imputation of a variable. The method is particularly useful for the generation of synthetic datasets, see `syn_mice` (Example 1).

Usage

```
mice.impute.constant(y, ry, x, fixed_values, ... )
```

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.
`fixed_values`	Vector containing fixed values
`...`	More arguments to be passed to imputation function

Value

Vector of imputed values

See Also

`syn.constant`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Example with fixed imputed values
#############################################################################

data(nhanes, package="mice")
dat <- nhanes

#* define methods
method <- c(age="", bmi="constant", hyp="norm", chl="pmm")
fixed_values <- list( bmi=rep(27,9) )

#* impute
imp <- mice::mice(dat, method=method, m=1, maxit=3, fixed_values=fixed_values)
table(mice::complete(imp, action=1)$bmi)

## End(Not run)
```

`mice.impute.hotDeck` *Imputation of a Variable Using Probabilistic Hot Deck Imputation*

Description

Imputes a variable under a random draw from a pool of donors defined by a distance function. Uncertainty with respect to the creation of donor pools is introduced by drawing a Bootstrap sample (approximate Bayesian Bootstrap, ABB) from observations with complete data (see Andridge & Little, 2010).

Usage

```
mice.impute.hotDeck(y, ry, x, donors=5, method="Mahalanobis", ...)
```

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.
`donors`	Number of donors used for random sampling of nearest neighbors in imputation

`method`	Method used for computation of weights in distance function. Options are the Mahalanobis metric (`method="Mahalanobis"`), weighted by correlations of covariates with the outcome (`method="cor"`) and weighting by linear regression coefficients (`method="lm"`).
`...`	Further arguments to be passed

Value

A vector of length `nmis=sum(!ry)` with imputed values.

References

Andridge, R. R., & and Little, R. J. A. (2010). A review of hot deck imputation for survey nonresponse. *International Statistical Review, 78*(1), 40-64. doi:10.1111/j.17515823.2010.00103.x

See Also

See also the packages **hot.deck** and **HotDeckImputation**.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Hot deck imputation NHANES dataset
#############################################################################

data(nhanes, package="mice")
dat <- nhanes

#*** prepare imputation method
vars <- colnames(dat)
V <- length(vars)
impMethod <- rep("hotDeck", V)
method <- "cor"

#*** imputation in mice
imp <- mice::mice( data=as.matrix(dat), m=1, method=impMethod, method=method )
summary(imp)

## End(Not run)
```

`mice.impute.imputeR.lmFun`

Wrapper Function to Imputation Methods in the **imputeR** *Package*

Description

The imputation methods `"imputeR.lmFun"` and `"imputeR.cFun"` provide interfaces to imputation methods in the **imputeR** package for continuous and binary data, respectively.

Usage

```
mice.impute.imputeR.lmFun(y, ry, x, Fun=NULL, draw_boot=TRUE, add_noise=TRUE, ... )

mice.impute.imputeR.cFun(y, ry, x, Fun=NULL, draw_boot=TRUE, ... )
```

Arguments

`y`	Incomplete data vector of length `n`
`ry`	Vector of missing data pattern (`FALSE` – missing, `TRUE` – observed)
`x`	Matrix (`n` x `p`) of complete covariates.
`Fun`	Name of imputation functions in **imputeR** package, e.g., `imputeR::ridgeR`, see Details.
`draw_boot`	Logical indicating whether a Bootstrap sample is taken for sampling model parameters
`add_noise`	Logical indicating whether empirical residuals should be added to predicted values
`...`	Further arguments to be passed

Details

Methods for continuous variables:

`imputeR::CubistR`, `imputeR::glmboostR`, `imputeR::lassoR`, `imputeR::pcrR`, `imputeR::plsR`, `imputeR::ridgeR`, `imputeR::stepBackR`, `imputeR::stepBothR`, `imputeR::stepForR`

Methods for binary variables: `imputeR::gbmC`, `imputeR::lassoC`, `imputeR::ridgeC`, `imputeR::rpartC`, `imputeR::stepBackC`, `imputeR::stepBothC`, `imputeR::stepForC`

Value

A vector of length `nmis=sum(!ry)` with imputed values.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Example with binary and continuous variables
#############################################################################

library(mice)
library(imputeR)

data(nhanes, package="mice")
dat <- nhanes
dat$hyp <- as.factor(dat$hyp)

#* define imputation methods
method <- c(age="",bmi="norm",hyp="imputeR.cFun",chl="imputeR.lmFun")
```

```
Fun <- list( hyp=imputeR::ridgeC, chl=imputeR::ridgeR)

#** do imputation
imp <- mice::mice(dat1, method=method, maxit=10, m=4, Fun=Fun)
summary(imp)

## End(Not run)
```

`mice.impute.ml.lmer` *Multilevel Imputation Using* **lme4**

Description

This function is a general imputation function based on the linear mixed effects model as implemented in `lme4::lmer`. The imputation model can be hierarchical or non-hierarchical and can be written in a general form $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \sum_{v=1}^{V} \boldsymbol{Z}_v \boldsymbol{u}_v$ for V multivariate random effects. While predictors can be selected by specifying the rows in the predictor matrix in `mice::mice` (i.e., modification of `type`), the level of random effects can be specified with `levels_id` and random slopes can be selected with `random_slopes`.

The function `mice.impute.ml.lmer` allows the imputation of variables at arbitrary levels. The corresponding level can be specified with `levels_id`. All predictor variables are aggregated to the corresponding level of the variable to be imputed.

Several strategies for the specification of the design matrix $\boldsymbol{X}$ are accommodated. By default, predictors at a lower level are automatically aggregated to the higher level and included as further predictors to maintain the multilevel structure in the data (Grund, Luedtke & Robitzsch, 2018; Enders, Mistler & Keller, 2016; argument `aggregate_automatically=TRUE`). Further, interactions and quadratic effects can be defined by respective arguments `interactions` and `quadratics`. The dimension of the matrix of predictors can be reduced by applying partial least squares regression, see `mice.impute.pls`.

The function now only allows continuous data (`model="continuous"`), ordinal data (`model="pmm"`) or binary data (`model="pmm"` or `model="binary"`). Nominal variables with missing values cannot (yet) be handled.

Usage

```
mice.impute.ml.lmer(y, ry, x, type, levels_id, variables_levels=NULL,
    random_slopes=NULL, aggregate_automatically=TRUE, intercept=TRUE,
    groupcenter.slope=FALSE, draw.fixed=TRUE, random.effects.shrinkage=1e-06,
   glmer.warnings=TRUE, model="continuous", donors=3, match_sampled_pars=FALSE,
    blme_use=FALSE, blme_args=NULL, pls.facs=0, interactions=NULL,
    quadratics=NULL, min.int.cor=0, min.all.cor=0, pls.print.progress=FALSE,
    group_index=NULL, iter_re=0, ...)
```

Arguments

`y`	Incomplete data vector of length `n`
`ry`	Vector of missing data pattern (`FALSE` – missing, `TRUE` – observed)
`x`	Matrix (`n` × `p`) of complete predictors.
`type`	Predictor variables associated with fixed effects.
`levels_id`	Specification of the level identifiers (see Examples)
`variables_levels`	Specification of the level of variables (see Examples)
`random_slopes`	Specification of random slopes (see Examples)
`aggregate_automatically`	Logical indicating whether aggregated effects at higher levels are automatically included.
`intercept`	Optional logical indicating whether the intercept should be included.
`groupcenter.slope`	Optional logical indicating whether covariates should be centered around group means
`draw.fixed`	Optional logical indicating whether fixed effects parameter should be randomly drawn
`random.effects.shrinkage`	Shrinkage parameter for stabilizing the covariance matrix of random effects
`glmer.warnings`	Optional logical indicating whether warnings from `glmer` should be displayed
`model`	Type of model. Can be `"continuous"` for normally distributed data, `"binary"` for dichotomous data specifying a logistic mixed effects model and `"pmm"` for predictive mean matching.
`donors`	Number of donors used for predictive mean matching
`match_sampled_pars`	Logical indicating whether values of nearest neighbors should also be sampled in pmm imputation.
`blme_use`	Logical indicating whether the **blme** package should be used.
`blme_args`	(Prior) Arguments for **blme**, see `blme::blmer` and `blme::bmerDist-class`.
`pls.facs`	Number of factors used in PLS dimension reduction
`interactions`	Specification of predictors with interaction effects
`quadratics`	Specification of predictors with quadratic effects
`min.int.cor`	Minimum absolute value of correlation with outcome for interaction effects to be retained
`min.all.cor`	Minimum absolute value of correlation with outcome for predictors to be retained
`pls.print.progress`	Logical indicating whether progress of algorithm should be displayed
`group_index`	Optional vector for group identifiers (internally used in `mice.impute.bygroup`
`iter_re`	Number of iterations for sampling random effects in random intercept models for continuous outcomes. Using `iter_re>0` is necessary for cross-classified models with not fully balanced designs.
`...`	Further arguments to be passed

Value

Vector of imputed values

References

Enders, C. K., Mistler, S. A., & Keller, B. T. (2016). Multilevel multiple imputation: A review and evaluation of joint modeling and chained equations imputation. *Psychological Methods, 21*(2), 222-240. doi:10.1037/met0000063

Grund, S., Luedtke, O., & Robitzsch, A. (2018). Multiple imputation of multilevel data in organizational research. *Organizational Research Methods, 21*(1), 111-149. doi:10.1177/1094428117703686

See Also

See `mice.impute.2l.continuous` for two-level imputation in **mice** and for several links to other packages which enable multilevel imputation.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation of three-level data with normally distributed residuals
#############################################################################

data(data.ma07, package="miceadds")
dat <- data.ma07

# variables at level 1 (identifier id1): x1 (some missings), x2 (complete)
# variables at level 2 (identifier id2): y1 (some missings), y2 (complete)
# variables at level 3 (identifier id3): z1 (some missings), z2 (complete)

#****************************************************************************
# Imputation model 1

#----- specify levels of variables (only relevent for variables
#      with missing values)
variables_levels <- miceadds:::mice_imputation_create_type_vector( colnames(dat), value="")
 # leave variables at lowest level blank (i.e., "")
variables_levels[ c("y1","y2") ] <- "id2"
variables_levels[ c("z1","z2") ] <- "id3"

#----- specify predictor matrix
predmat <- mice::make.predictorMatrix(data=dat)
predmat[, c("id2", "id3") ] <- 0
# set -2 for cluster identifier for level 3 variable z1
# because "2lonly" function is used
predmat[ "z1", "id3" ] <- -2

#----- specify imputation methods
method <- mice::make.method(data=dat)
method[c("x1","y1")] <- "ml.lmer"
method[c("z1")] <- "2lonly.norm"
```

```
#----- specify hierarchical structure of imputation models
levels_id <- list()
#** hierarchical structure for variable x1
levels_id[["x1"]] <- c("id2", "id3")
#** hierarchical structure for variable y1
levels_id[["y1"]] <- c("id3")

#----- specify random slopes
random_slopes <- list()
#** random slopes for variable x1
random_slopes[["x1"]] <- list( "id2"=c("x2"), "id3"=c("y1") )
# if no random slopes should be specified, the corresponding entry can be left empty
# and only a random intercept is used in the imputation model

#----- imputation in mice
imp1 <- mice::mice( dat, maxit=10, m=5, method=method,
            predictorMatrix=predmat, levels_id=levels_id,  random_slopes=random_slopes,
            variables_levels=variables_levels )
summary(imp1)

#****************************************************************************
# Imputation model 2

#----- impute x1 with predictive mean matching and y1 with normally distributed residuals
model <- list(x1="pmm", y1="continuous")

#----- assume only random intercepts
random_slopes <- NULL

#---- create interactions with z2 for all predictors in imputation models for x1 and y1
interactions <- list("x1"="z2", "y1"="z2")

#----- imputation in mice
imp2 <- mice::mice( dat, method=method, predictorMatrix=predmat,
                levels_id=levels_id, random_slopes=random_slopes,
              variables_levels=variables_levels, model=model, interactions=interactions)
summary(imp2)

## End(Not run)
```

`mice.impute.plausible.values` *Plausible Value Imputation using Classical Test Theory and Based on Individual Likelihood*

Description

This imputation function performs unidimensional plausible value imputation if (subject-wise) measurement errors or the reliability of the scale is known (Mislevy, 1991; see also Asparouhov &

Muthen, 2010; Blackwell, Honaker & King, 2011, 2017a, 2017b). The function also allows the input of an individual likelihood obtained by fitting an item response model.

Usage

```
mice.impute.plausible.values(y, ry, x, type, alpha=NULL,
    alpha.se=0, scale.values=NULL, sig.e.miss=1e+06,
    like=NULL, theta=NULL, normal.approx=NULL,
    pviter=15, imputationWeights=rep(1, length(y)), plausible.value.print=TRUE,
    pls.facs=NULL, interactions=NULL, quadratics=NULL, extract_data=TRUE,
    control_latreg=list( progress=FALSE, ridge=1e-5 ),  ...)
```

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.
`type`	Type of predictor variables. `type=3` refers to items belonging to a scale to be imputed. A cluster (grouping) variable is defined by `type=-2`. If for some predictors, the cluster means should also be included as predictors, then specify `type=2` (see Imputation Model 3 of Example 1).
`alpha`	A known reliability estimate. An optional standard error of the estimate can be provided in `alpha.se`
`alpha.se`	Optional numeric value of the standard error of the `alpha` reliability estimate if in every iteration a new reliability should be sampled.
`scale.values`	A list consisting of scale values of scale values and its corresponding standard errors (see Example 1).
`sig.e.miss`	A standard error of measurement for cases with missing values on a scale
`like`	Individual likelihood evaluated at `theta`
`theta`	Grid of unidimensional latent variable
`normal.approx`	Logical indicating whether the individual posterior should be approximated by a normal distribution
`pviter`	Number of iterations in each imputation which should be run until the plausible values are drawn
`imputationWeights`	Optional vector of sample weights
`plausible.value.print`	An optional logical indicating whether some information about the plausible value imputation should be printed at the console
`pls.facs`	Number of PLS factors if PLS dimension reduction is used
`interactions`	Vector of variable names used for creating interactions
`quadratics`	Vector of variable names used for creating quadratic terms
`extract_data`	Logical indicating whether input data should be extracted from parent environment within `mice::mice` routine
`control_latreg`	Control arguments for `TAM::tam.latreg`
`...`	Further objects to be passed

Details

The linear model is assumed for drawing plausible values of a variable Y contaminated by measurement error. Assuming $Y = \theta + e$ and a linear regression model for θ

$$\theta = \boldsymbol{X}\beta + \epsilon$$

(plausible value) imputations from the posterior distribution $P(\theta|Y, \boldsymbol{X})$ are drawn. See Mislevy (1991) for details.

Value

A vector of length `nrow(x)` containing imputed plausible values.

Note

Plausible value imputation is also known as multiple overimputation (Blackwell, Honaker & King, 2016a, 2016b) which is implemented in the **Amelia** package, see `Amelia::moPrep` and `Amelia::amelia`.

References

Asparouhov, T., & Muthen, B. (2010). *Plausible values for latent variables using Mplus*. Technical Report. `https://www.statmodel.com/papers.shtml`

Blackwell, M., Honaker, J., & King, G. (2011). *Multiple overimputation: A unified approach to measurement error and missing data*. Technical Report.

Blackwell, M., Honaker, J., & King, G. (2017a). A unified approach to measurement error and missing data: Overview and applications. *Sociological Methods & Research, 46*(3), 303-341.

Blackwell, M., Honaker, J., & King, G. (2017b). A unified approach to measurement error and missing data: Details and extensions. *Sociological Methods & Research, 46*(3), 342-369.

Mislevy, R. J. (1991). Randomization-based inference about latent variables from complex samples. *Psychometrika, 56*, 177-196.

See Also

See `TAM::tam.latreg` for fitting latent regression models.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Plausible value imputation for data.ma04 | 2 scales
#############################################################################

data(data.ma04, package="miceadds")
dat <- data.ma04

# Scale 1 consists of items A1,...,A4
# Scale 2 consists of items B1,...,B5
dat$scale1 <- NA
dat$scale2 <- NA
```

```
#** inits imputation method and predictor matrix
res <- miceadds::mice_inits(dat, ignore=c("group") )
predM <- res$predictorMatrix
impMethod <- res$method
impMethod <- gsub("pmm", "norm", impMethod )

# look at missing proportions
colSums( is.na(dat) )

# redefine imputation methods for plausible value imputation
impMethod[ "scale1" ] <- "plausible.values"
predM[ "scale1",  ] <- 1
predM[ "scale1", c("A1", "A2",  "A3", "A4" ) ] <- 3
    # items corresponding to a scale should be declared by a 3 in the predictor matrix
impMethod[ "scale2" ] <- "plausible.values"
predM[,"scale2"  ] <- 0
predM[ "scale2",  c("A2","A3","A4","V6","V7") ] <- 1
diag(predM) <- 0

# use imputed scale values as predictors for V5, V6 and V7
predM[ c("V5","V6","V7"), c("scale1","scale2" ) ] <- 1
# exclude for V5, V6 and V7 the items of scales A and B as predictors
predM[ c("V5","V6","V7"), c( paste0("A",2:4), paste0("B",1:5) ) ] <- 0
# exclude 'group' as a predictor
predM[,"group"] <- 0

# look at imputation method and predictor matrix
impMethod
predM

#-------------------------------
# Parameter for imputation
#***
# scale 1 (A1,...,A4)
# known Cronbach's Alpha
alpha <- NULL
alpha <- list( "scale1"=.8 )
alpha.se <- list( "scale1"=.05 )  # sample alpha with a standard deviation of .05

#***
# scale 2 (B1,...,B5)
# means and SE's of scale scores are assumed to be known
M.scale2 <- rowMeans( dat[, paste("B",1:5,sep="")  ] )
# M.scale2[ is.na( m1) ] <- mean( M.scale2, na.rm=TRUE )
SE.scale2 <- rep( sqrt( stats::var(M.scale2,na.rm=T)*(1-.8) ), nrow(dat) )
#=> heterogeneous measurement errors are allowed
scale.values <- list( "scale2"=list( "M"=M.scale2, "SE"=SE.scale2 ) )

#*** Imputation Model 1: Imputation four using parallel chains
imp1 <- mice::mice( dat, predictorMatrix=predM, m=4, maxit=5,
          alpha.se=alpha.se, method=impMethod,  allow.na=TRUE, alpha=alpha,
          scale.values=scale.values  )
```

```
summary(imp1)

# extract first imputed dataset
dat11 <- mice::complete( imp, 1 )

#*** Imputation Model 2: Imputation using one long chain
imp2 <- miceadds::mice.1chain( dat, predictorMatrix=predM, burnin=10, iter=20, Nimp=4,
          alpha.se=alpha.se, method=impMethod,  allow.na=TRUE, alpha=alpha,
          scale.values=scale.values )
summary(imp2)

#-------------
#*** Imputation Model 3: Imputation including  group level variables

# use group indicator for plausible value estimation
predM[ "scale1", "group" ] <- -2
# V7 and B1 should be aggregated at the group level
predM[ "scale1", c("V7","B1") ] <- 2
predM[ "scale2", "group" ] <- -2
predM[ "scale2", c("V7","A1") ] <- 2

# perform single imputation (m=1)
imp <- mice::mice( dat, predictorMatrix=predM, m=1, maxit=10,
            method=impMethod,  allow.na=TRUE, alpha=alpha,
            scale.values=scale.values )
dat10 <- mice::complete(imp)

# multilevel model
library(lme4)
mod <- lme4::lmer( scale1 ~ ( 1 | group), data=dat11 )
summary(mod)

mod <- lme4::lmer( scale1 ~ ( 1 | group), data=dat10)
summary(mod)

#############################################################################
# EXAMPLE 2: Plausible value imputation with chained equations
#############################################################################

# - simulate a latent variable theta and dichotomous item responses
# - two covariates X in which the second covariate has measurement error

library(sirt)
library(TAM)
library(lavaan)

set.seed(7756)
N <- 2000    # number of persons
I <- 10     # number of items

# simulate covariates
X <- MASS::mvrnorm( N, mu=c(0,0), Sigma=matrix( c(1,.5,.5,1),2,2 ) )
colnames(X) <- paste0("X",1:2)
```

```
# second covariate with measurement error with variance var.err
var.err <- .3
X.err <- X
X.err[,2] <- X[,2] + stats::rnorm(N, sd=sqrt(var.err) )
# simulate theta
theta <- .5*X[,1] + .4*X[,2] + stats::rnorm( N, sd=.5 )
# simulate item responses
itemdiff <- seq( -2, 2, length=I)  # item difficulties
dat <- sirt::sim.raschtype( theta, b=itemdiff )

#***********************
#*** Model 0: Regression model with true variables
mod0 <- stats::lm( theta ~ X )
summary(mod0)

#**********************
# plausible value imputation for abilities and error-prone
# covariates using the mice package

# creating the likelihood for plausible value for abilities
mod11 <- TAM::tam.mml( dat )
likePV <- IRT.likelihood(mod11)
# creating the likelihood for error-prone covariate X2
# The known measurement error variance is 0.3.
lavmodel <- "
  X2true=~ 1*X2
  X2 ~~ 0.3*X2
    "
mod12 <- lavaan::cfa( lavmodel, data=as.data.frame(X.err) )
summary(mod12)
likeX2 <- IRTLikelihood.cfa( data=X.err, cfaobj=mod12)
str(likeX2)

#-- create data input for mice package
data <- data.frame( "PVA"=NA, "X1"=X[,1], "X2"=NA  )
vars <- colnames(data)
V <- length(vars)
predictorMatrix <- 1 - diag(V)
rownames(predictorMatrix) <- colnames(predictorMatrix) <- vars
method <- rep("norm", V )
names(method) <- vars
method[c("PVA","X2")] <- "plausible.values"

#-- create argument lists for plausible value imputation
# likelihood and theta grid of plausible value derived from IRT model
like <- list( "PVA"=likePV, "X2"=likeX2 )
theta <- list( "PVA"=attr(likePV,"theta"),
                "X2"=attr(likeX2, "theta") )
#-- initial imputations
data.init <- data
data.init$PVA <- mod11$person$EAP
data.init$X2 <- X.err[,"X2"]
```

```
#-- imputation using the mice and miceadds package
imp1 <- mice::mice( as.matrix(data), predictorMatrix=predictorMatrix, m=4,
            maxit=6, method=method,  allow.na=TRUE,
            theta=theta, like=like, data.init=data.init )
summary(imp1)

# compute linear regression
mod4a <- with( imp1, stats::lm( PVA ~ X1 + X2 ) )
summary( mice::pool(mod4a) )

#############################################################################
# EXAMPLE 3: Plausible value imputation with known error variance
#############################################################################

#---- simulate data
set.seed(987)
N <- 1000         # number of persons
var_err <- .4     # error variance
dat <- data.frame( x1=stats::rnorm(N), x2=stats::rnorm(N) )
dat$theta <- .3 * dat$x1 - .5*dat$x2 + stats::rnorm(N)
dat$y <- dat$theta + stats::rnorm( N, sd=sqrt(var_err) )

#-- linear regression for measurement-error-free data
mod0a <- stats::lm( theta ~ x1 + x2, data=dat )
summary(mod0a)
#-- linear regression for data with measurement error
mod0b <- stats::lm( y ~ x1 + x2, data=dat )
summary(mod0b)

#-- process data for imputation

dat1 <- dat
dat1$theta <- NA
scale.values <- list( "theta"=list( "M"=dat$y, "SE"=rep(sqrt(var_err),N )))
dat1$y <- NULL

cn <- colnames(dat1)
V <- length(cn)
method <- rep("", length(cn) )
names(method) <- cn
method["theta"] <- "plausible.values"

#-- imputation in mice
imp <- mice::mice( dat1, maxit=1, m=5, allow.na=TRUE, method=method,
            scale.values=scale.values )
summary(imp)

#-- inspect first dataset
summary( mice::complete(imp, action=1) )

#-- linear regression based on imputed datasets
mod1 <- with(imp, stats::lm( theta ~ x1 + x2 ) )
summary( mice::pool(mod1) )
```

```
## End(Not run)
```

`mice.impute.pls` *Imputation using Partial Least Squares for Dimension Reduction*

Description

This function imputes a variable with missing values using PLS regression (Mevik & Wehrens, 2007) for a dimension reduction of the predictor space.

Usage

```
mice.impute.pls(y, ry, x, type, pls.facs=NULL,
  pls.impMethod="pmm", donors=5, pls.impMethodArgs=NULL, pls.print.progress=TRUE,
   imputationWeights=rep(1, length(y)), pcamaxcols=1E+09,
   min.int.cor=0, min.all.cor=0, N.largest=0, pls.title=NULL, print.dims=TRUE,
   pls.maxcols=5000, use_boot=FALSE, envir_pos=NULL, extract_data=TRUE,
   remove_lindep=TRUE, derived_vars=NULL, ...)

mice.impute.2l.pls2(y, ry, x, type, pls.facs=NULL, pls.impMethod="pmm",
  pls.print.progress=TRUE, imputationWeights=rep(1, length(y)), pcamaxcols=1E+09,
   tricube.pmm.scale=NULL, min.int.cor=0, min.all.cor=0, N.largest=0,
  pls.title=NULL, print.dims=TRUE, pls.maxcols=5000, envir_pos=parent.frame(), ...)
```

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`	`type=1` – variable is used as a predictor, `type=4` – create interactions with the specified variable with all other predictors, `type=5` – create a quadratic term of the specified variable `type=6` – if some interactions are specified, ignore the variables with entry `6` when creating interactions `type=-2` – specification of a cluster variable. The cluster mean of the outcome `y` (when eliminating the subject under study) is included as a further predictor in the imputation.
`pls.facs`	Number of factors used in PLS regression. This argument can also be specified as a list defining different numbers of factors for all variables to be imputed.
`pls.impMethod`	Imputation method used for in PLS estimation. Any imputation method can be used except if `imputationWeights` is provided. Imputation weights are available for `norm` and `pmm`. Categorical variables can be imputed with the method `catpmm` (see `mice.impute.catpmm`). For the method `catpmm`, multivariate PLS regression is employed for dummy-coded categories of the outcome variable. The method `xplsfacs` creates only PLS factors of the regression model.

`donors`	Number of donors if predictive mean matching is used (`pls.impMethod="pmm"`).
`pls.impMethodArgs`	Arguments for imputation method `pls.impMethod`.
`pls.print.progress`	Print progress during PLS regression.
`imputationWeights`	Vector of sample weights to be used in imputation models.
`pcamaxcols`	Amount of variance explained by principal components (must be a number between 0 and 1) or number of factors used in PCA (an integer larger than 1).
`min.int.cor`	Minimum absolute correlation for an interaction of two predictors to be included in the PLS regression model
`min.all.cor`	Minimum absolute correlation for inclusion in the PLS regression model.
`N.largest`	Number of variable to be included which do have the largest absolute correlations.
`pls.title`	Title for progress print in console output.
`print.dims`	An optional logical indicating whether dimensions of inputs should be printed.
`pls.maxcols`	Maximum number of interactions to be created.
`use_boot`	Logical whether Bayesian bootstrap should be used for drawing regression parameters
`envir_pos`	Position of the environment from which the data should be extracted.
`extract_data`	Logical indicating whether input data should be extracted from parent environment within `mice::mice` routine
`remove_lindep`	Logical indicating whether linear dependencies should be automatically detected and some predictors are removed
`derived_vars`	Optional list containing formulas with derived variables for inclusion in PLS dimension reduction
`...`	Further arguments to be passed.
`tricube.pmm.scale`	Scale factor for tricube PMM imputation.

Value

A vector of length `nmis=sum(!ry)` with imputations if `pls.impMethod !="xplsfacs"`. In case of `pls.impMethod=="xplsfacs"` a matrix with PLS factors is computed.

Note

The `mice.impute.2l.pls2` function is just included for reasons of backward compatibility to former **miceadds** versions.

References

Mevik, B. H., & Wehrens, R. (2007). The **pls** package: Principal component and partial least squares regression in R. *Journal of Statistical Software, 18*, 1-24. doi:10.18637/jss.v018.i02

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: PLS imputation method for internet data
#############################################################################

data(data.internet)
dat <- data.internet

# specify predictor matrix
predictorMatrix <- matrix( 1, ncol(dat), ncol(dat) )
rownames(predictorMatrix) <- colnames(predictorMatrix) <- colnames(dat)
diag( predictorMatrix) <- 0

# use PLS imputation method for all variables
impMethod <- rep( "pls", ncol(dat) )
names(impMethod) <- colnames(dat)

# define predictors for interactions (entries with type 4 in predictorMatrix)
predictorMatrix[c("IN1","IN15","IN16"),c("IN1","IN3","IN10","IN13")] <- 4
# define predictors which should appear as linear and quadratic terms (type 5)
predictorMatrix[c("IN1","IN8","IN9","IN10","IN11"),c("IN1","IN2","IN7","IN5")] <- 5

# use 9 PLS factors for all variables
pls.facs <- as.list( rep( 9, length(impMethod) ) )
names(pls.facs) <- names(impMethod)
pls.facs$IN1 <- 15   # use 15 PLS factors for variable IN1

# choose norm or pmm imputation method
pls.impMethod <- as.list( rep("norm", length(impMethod) ) )
names(pls.impMethod) <- names(impMethod)
pls.impMethod[ c("IN1","IN6")] <- "pmm"

# some arguments for imputation method
pls.impMethodArgs <- list( "IN1"=list( "donors"=10 ),
                           "IN2"=list( "ridge2"=1E-4 ) )

# Model 1: Three parallel chains
imp1 <- mice::mice(data=dat, method=impMethod,
     m=3, maxit=5, predictorMatrix=predictorMatrix,
     pls.facs=pls.facs, # number of PLS factors
     pls.impMethod=pls.impMethod,  # Imputation Method in PLS imputation
     pls.impMethodArgs=pls.impMethodArgs, # arguments for imputation method
     pls.print.progress=TRUE, ls.meth="ridge" )
summary(imp1)

# Model 2: One long chain
imp2 <- miceadds::mice.1chain(data=dat, method=impMethod,
     burnin=10, iter=21, Nimp=3, predictorMatrix=predictorMatrix,
     pls.facs=pls.facs, pls.impMethod=pls.impMethod,
     pls.impMethodArgs=pls.impMethodArgs, ls.meth="ridge" )
summary(imp2)
```

```
# Model 3: inclusion of additional derived variables

# define derived variables for IN1
derived_vars <- list( "IN1"=~I( ifelse( IN2>IN3, IN2, IN3 ) ) + I( sin(IN2) ) )

imp3 <- miceadds::mice.1chain(data=dat, method=impMethod, derived_vars=derived_vars,
     burnin=10, iter=21, Nimp=3, predictorMatrix=predictorMatrix,
     pls.facs=pls.facs, pls.impMethod=pls.impMethod,
     pls.impMethodArgs=pls.impMethodArgs, ls.meth="ridge" )
summary(imp3)

#*** example for using imputation function at the level of a variable

# extract first imputed dataset
imp1 <- mice::complete(imp1, action=1)
data_imp1[ is.na(dat$IN1), "IN1" ] <- NA

# define variables
y <- data_imp1$IN1
x <- data_imp1[, -1 ]
ry <- ! is.na(y)
cn <- colnames(dat)
p <- ncol(dat)
type <- rep(1,p)
names(type) <- cn
type["IN1"] <- 0

# imputation of variable 'IN1'
imp0 <- miceadds::mice.impute.pls(y=y, x=x, ry=ry, type=type, pls.facs=10, pls.impMethod="norm",
             ls.meth="ridge", extract_data=FALSE )

## End(Not run)
```

`mice.impute.pmm3` *Imputation by Predictive Mean Matching (in* **miceadds***)*

Description

This function imputes values by predictive mean matching like the `mice::mice.impute.pmm` method in the **mice** package.

Usage

```
mice.impute.pmm3(y, ry, x, donors=3, noise=10^5, ridge=10^(-5), ...)
mice.impute.pmm4(y, ry, x, donors=3, noise=10^5, ridge=10^(-5), ...)
mice.impute.pmm5(y, ry, x, donors=3, noise=10^5, ridge=10^(-5), ...)
mice.impute.pmm6(y, ry, x, donors=3, noise=10^5, ridge=10^(-5), ...)
```

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.
`donors`	Number of donors used for imputation
`noise`	Numerical value to break ties
`ridge`	Ridge parameter in the diagonal of $\boldsymbol{X}'\boldsymbol{X}$
`...`	Further arguments to be passed

Details

The imputation method `pmm3` imitates `mice::mice.impute.pmm` imputation method in **mice**.

The imputation method `pmm4` ignores ties in predicted y values. With many predictors, this does not probably implies any substantial problem.

The imputation method `pmm5` suffers from the same problem. Contrary to the other PMM methods, it searches D donors (specified by `donors`) smaller than the predicted value and D donors larger than the predicted value and randomly samples a value from this set of $2 \cdot D$ donors.

The imputation method `pmm6` is just the **Rcpp** implementation of `pmm5`.

Value

A vector of length `nmis=sum(!ry)` with imputed values.

See Also

See `data.largescale` and `data.smallscale` for speed comparisons of different functions for predictive mean matching.

Examples

```
## Not run:
#############################################################################
# SIMULATED EXAMPLE 1: Two variables x and y with missing y
#############################################################################
set.seed(1413)

rho <- .6   # correlation between x and y
N <- 6800    # number of cases
x <- stats::rnorm(N)
My <- .35   # mean of y
y.com <- y <- My + rho * x + stats::rnorm(N, sd=sqrt( 1 - rho^2 ) )

# create missingness on y depending on rho.MAR parameter
rho.mar <- .4    # correlation response tendency z and x
missrate <- .25  # missing response rate
# simulate response tendency z and missings on y
z <- rho.mar * x + stats::rnorm(N, sd=sqrt( 1 - rho.mar^2 ) )
```

```
y[ z < stats::qnorm( missrate ) ] <- NA
dat <- data.frame(x, y )

# mice imputation
impmethod <- rep("pmm", 2 )
names(impmethod) <- colnames(dat)

# pmm (in mice)
imp1 <- mice::mice( as.matrix(dat), m=1, maxit=1, method=impmethod)
# pmm3 (in miceadds)
imp3 <- mice::mice( as.matrix(dat), m=1, maxit=1,
           method=gsub("pmm","pmm3",impmethod)  )
# pmm4 (in miceadds)
imp4 <- mice::mice( as.matrix(dat), m=1, maxit=1,
           method=gsub("pmm","pmm4",impmethod)  )
# pmm5 (in miceadds)
imp5 <- mice::mice( as.matrix(dat), m=1, maxit=1,
           method=gsub("pmm","pmm5",impmethod)  )
# pmm6 (in miceadds)
imp6 <- mice::mice( as.matrix(dat), m=1, maxit=1,
           method=gsub("pmm","pmm6",impmethod)  )

dat.imp1 <- mice::complete( imp1, 1 )
dat.imp3 <- mice::complete( imp3, 1 )
dat.imp4 <- mice::complete( imp4, 1 )
dat.imp5 <- mice::complete( imp5, 1 )
dat.imp6 <- mice::complete( imp6, 1 )

dfr <- NULL
# means
dfr <- rbind( dfr, c( mean( y.com ), mean( y, na.rm=TRUE ), mean( dat.imp1$y),
    mean( dat.imp3$y), mean( dat.imp4$y), mean( dat.imp5$y),  mean( dat.imp6$y)  ) )
# SD
dfr <- rbind( dfr, c( stats::sd( y.com ), stats::sd( y, na.rm=TRUE ),
      stats::sd( dat.imp1$y), stats::sd( dat.imp3$y), stats::sd( dat.imp4$y),
      stats::sd( dat.imp5$y), stats::sd( dat.imp6$y) ) )
# correlations
dfr <- rbind( dfr, c( stats::cor( x,y.com ),
    stats::cor( x[ ! is.na(y) ], y[ ! is.na(y) ] ),
    stats::cor( dat.imp1$x, dat.imp1$y), stats::cor( dat.imp3$x, dat.imp3$y),
    stats::cor( dat.imp4$x, dat.imp4$y), stats::cor( dat.imp5$x, dat.imp5$y),
    stats::cor( dat.imp6$x, dat.imp6$y)
        ) )
rownames(dfr) <- c("M_y", "SD_y", "cor_xy" )
colnames(dfr) <- c("compl", "ld", "pmm", "pmm3", "pmm4", "pmm5","pmm6")
##           compl     ld    pmm   pmm3   pmm4   pmm5   pmm6
##   M_y    0.3306 0.4282 0.3314 0.3228 0.3223 0.3264 0.3310
##   SD_y   0.9910 0.9801 0.9873 0.9887 0.9891 0.9882 0.9877
##   cor_xy 0.6057 0.5950 0.6072 0.6021 0.6100 0.6057 0.6069

## End(Not run)
```

`mice.impute.rlm` *Imputation of a Linear Model by Bayesian Bootstrap*

Description

These functions impute from linear models using the functions `stats::lm`, `MASS::rlm` or `MASS::lqs`. The method `mice.impute.lm_fun` allows the definition of a general linear regression fitting function for which the methods `predict` and `residuals` are defined.

Parameters of the model are estimated by Bayesian bootstrap. Predicted values are computed and residuals are randomly drawn from the empirical distribution of residuals of observed data.

Usage

```
mice.impute.lm(y, ry, x, wy=NULL, lm_args=NULL, trafo=NULL, antitrafo=NULL, ...)

mice.impute.rlm(y, ry, x, wy=NULL, lm_args=NULL, trafo=NULL, antitrafo=NULL, ...)

mice.impute.lqs(y, ry, x, wy=NULL, lm_args=NULL, trafo=NULL, antitrafo=NULL, ...)

mice.impute.lm_fun(y, ry, x, wy=NULL, lm_args=NULL, lm_fun="lm", trafo=NULL,
               antitrafo=NULL, ...)
```

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.
`wy`	Vector of logicals indicating which entries should be imputed
`lm_args`	List of arguments for `stats::lm`, `MASS::rlm`, `MASS::lqs` or a user-defined function.
`lm_fun`	Linear regression fitting function, e.g. `stats::lm` for which S3 methods `predict` and `residuals` are defined.
`trafo`	Optional function for transforming the outcome values
`antitrafo`	Optional function which is the inverse function of `trafo`
`...`	Further arguments to be passed

Value

A vector of length `nmis=sum(!ry)` with imputed values.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Some toy example illustrating the methods
#############################################################################

library(MASS)
library(mice)

#-- simulate data
set.seed(98)
N <- 1000
x <- stats::rnorm(N)
z <- 0.5*x + stats::rnorm(N, sd=.7)
y <- stats::rnorm(N, mean=.3*x - .2*z, sd=1 )
dat <- data.frame(x,z,y)
dat[ seq(1,N,3), c("x","y") ] <- NA
dat[ seq(1,N,4), "z" ] <- NA

#-- define imputation methods
imp <- mice::mice(dat, maxit=0)
method <- imp$method
method["x"] <- "rlm"
method["z"] <- "lm"
method["y"] <- "lqs"

#-- impute data
imp <- mice::mice(dat, method=method)
summary(imp)

#--- example using transformations
dat1$x <- exp(dat1$x)
dat1$z <- stats::plogis(dat1$z)

trafo <- list(x=log, z=stats::qlogis)
antitrafo <- list(x=exp, z=stats::plogis)

#- impute with transformations
imp2 <- mice::mice(dat1, method=method, m=1, maxit=3, trafo=trafo, antitrafo=antitrafo)
print(imp2)

## End(Not run)
```

`mice.impute.simputation` *Wrapper Function to Imputation Methods in the* **simputation** *Package*

Description

This imputation method provides a wrapper function to univariate imputation methods in the **simputation** package.

Usage

```
mice.impute.simputation(y, ry, x, Fun=NULL, Fun_args=NULL, ... )
```

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.
`Fun`	Name of imputation functions in **simputation** package, e.g., `imputeR::impute_lm`, see Details.
`Fun_args`	Optional argument list for `Fun`
`...`	Further arguments to be passed

Details

Selection of imputation methods included in the **simputation** package:

linear regression: `simputation::impute_lm`,
robist linear regression with M-estimators: `simputation::impute_rlm`,
regularized regression with lasso/elasticnet/ridge regression: `simputation::impute_en`,
CART models or random forests: `simputation::impute_cart`, `simputation::impute_rf`,
Hot deck imputation: `simputation::impute_rhd`, `simputation::impute_shd`,
Predictive mean matching: `simputation::impute_pmm`,
k-nearest neighbours imputation: `simputation::impute_knn`

Value

A vector of length `nmis=sum(!ry)` with imputed values.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Nhanes example
#############################################################################

library(mice)
library(simputation)

data(nhanes, package="mice")
dat <- nhanes

#** imputation methods
method <- c(age="",bmi="norm", hyp="norm", chl="simputation")
Fun <- list( chl=simputation::impute_lm)
Fun_args <- list( chl=list(add_residual="observed") )

#** do imputations
imp <- mice::mice(dat, method=method, Fun=Fun, Fun_args=Fun_args)
summary(imp)
```

```
## End(Not run)
```

`mice.impute.smcfcs` *Substantive Model Compatible Multiple Imputation (Single Level)*

Description

Computes substantive model compatible multiple imputation (Bartlett et al., 2015; Bartlett & Morris, 2015). Several regression functions are allowed (see `dep_type`).

Usage

```
mice.impute.smcfcs(y, ry, x, wy=NULL, sm, dep_type="norm", sm_type="norm",
       fac_sd_proposal=1, mh_iter=20, ...)
```

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.
`wy`	Logical vector indicating positions where imputations should be conducted.
`sm`	Formula for substantive model.
`dep_type`	Distribution type for variable which is imputed. Possible choices are `"norm"` (normal distribution), `"lognorm"` (lognormal distribution), `"yj"` (Yeo-Johnson distribution, see `mdmb::yjt_regression`), `"bc"` (Box-Cox distribution, see `mdmb::bct_regression`), `"logistic"` (logistic distribution).
`sm_type`	Distribution type for dependent variable in substantive model. One of the distribution mentioned in `dep_type` can be chosen.
`fac_sd_proposal`	Starting value for factor of standard deviation in Metropolis-Hastings sampling.
`mh_iter`	Number iterations in Metropolis-Hasting sampling
`...`	Further arguments to be passed

Details

Imputed values are drawn based on a Metropolis-Hastings sampling algorithm in which the standard deviation of the proposal distribution is adaptively tuned.

Value

A vector of length `nmis=sum(!ry)` with imputed values.

References

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

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

See Also

See the **smcfcs** package for an alternative implementation of substantive model multiple imputation in a fully conditional specification approach.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Substantive model with interaction effects
#############################################################################

library(mice)
library(mdmb)

#--- simulate data
set.seed(98)
N <- 1000
x <- stats::rnorm(N)
z <- 0.5*x + stats::rnorm(N, sd=.7)
y <- stats::rnorm(N, mean=.3*x - .2*z + .7*x*z, sd=1 )
dat <- data.frame(x,z,y)
dat[ seq(1,N,3), c("x","y") ] <- NA


#--- define imputation methods
imp <- mice::mice(dat, maxit=0)
method <- imp$method
method["x"] <- "smcfcs"

# define substantive model
sm <- y ~ x*z
# define formulas for imputation models
formulas <- as.list( rep("",ncol(dat)))
names(formulas) <- colnames(dat)
formulas[["x"]] <- x ~ z
formulas[["y"]] <- sm
formulas[["z"]] <- z ~ 1

#- Yeo-Johnson distribution for x
dep_type <- list()
dep_type$x <- "yj"
```

```
#-- do imputation
imp <- mice::mice(dat, method=method, sm=sm, formulas=formulas, m=1, maxit=10,
                    dep_type=dep_type)
summary(imp)

#############################################################################
# EXAMPLE 2: Substantive model with quadratic effects
#############################################################################

#** simulate data with missings
set.seed(50)
n <- 1000
x <- stats::rnorm(n)
z <- stats::rnorm(n)
y <- 0.5 * z + x + x^2 + stats::rnorm(n)
mm <- stats::runif(n)
x[sample(1:n, size=370, prob=mm)] <- NA
z[sample(1:n, size=480, prob=mm)] <- NA
y[sample(1:n, size=500, prob=mm)] <- NA

df <- data.frame(x=x,y=y,z=z)

#** imputation
imp <- mice::mice(df, method="smcfcs", sm=y ~ z + x + I(x^2), m=6, maxit=10)
summary(imp)

#** analysis
summary(mice::pool(with(imp, stats::lm(y ~ z + x + I(x^2)))))

#** imputation using the smcfcs package
df$x_sq <- df$x^2
nonmice <- smcfcs::smcfcs(df, smtype="lm", smformula=y ~ z + x + x_sq,
             method=c("norm", "", "norm", "x^2"))
mice::pool(lapply(nonmice$impDatasets, function(x) stats::lm(y ~ z + x + x_sq, data=x)))

## End(Not run)
```

`mice.impute.synthpop` *Using a* **synthpop** *Synthesizing Method in the* **mice** *Package*

Description

The function allows to use a **synthpop** synthesizing method to be used in the `mice::mice` function of the **mice** package.

Usage

```
mice.impute.synthpop(y, ry, x, synthpop_fun="norm", synthpop_args=list(),
     proper=TRUE, ...)
```

Arguments

`y`	Incomplete data vector of length `n`
`ry`	Vector of missing data pattern (`FALSE` – missing, `TRUE` – observed)
`x`	Matrix (`n` x `p`) of complete covariates.
`synthpop_fun`	Synthesizing method in the **synthpop** package
`synthpop_args`	Function arguments of `syn_fun`
`proper`	Logical value specifying whether proper synthesis should be conducted.
`...`	Further arguments to be passed

Value

A vector of length `nmis=sum(!ry)` with imputed values.

See Also

See `syn.mice` for using a **mice** imputation method in the **synthpop** package.

See `synthpop::syn` for generating synthetic datasets with the **synthpop** package.

Examples

```
## Not run: 
#############################################################################
# EXAMPLE 1: Imputation of NHANES data using the 'syn.normrank' method
#############################################################################

library(synthpop)
data(nhanes, package="mice")
dat <- nhanes

#* empty imputation
imp0 <- mice::mice(dat, maxit=0)
method <- imp0$method

#* define synthpop method 'normrank' for variable 'chl'
method["chl"] <- "synthpop"
synthpop_fun <- list( chl="normrank" )
synthpop_args <- list( chl=list(smoothing="density") )

#* conduct imputation
imp <- mice::mice(dat, method=method, m=1, maxit=3, synthpop_fun=synthpop_fun,
            synthpop_args=synthpop_args)
summary(imp)

## End(Not run)
```

`mice.impute.tricube.pmm` *Imputation by Tricube Predictive Mean Matching*

Description

This function performs tricube predictive mean matching (see `Hmisc::aregImpute`) in which donors are weighted according to distances of predicted values. Three donors are chosen.

Usage

```
mice.impute.tricube.pmm(y, ry, x, tricube.pmm.scale=0.2, tricube.boot=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.
`tricube.pmm.scale`	A scaling factor for tricube matching. The default is 0.2.
`tricube.boot`	A logical indicating whether tricube matching should be performed using a bootstrap sample
`...`	Further arguments to be passed

Value

A vector of length `nmis=sum(!ry)` with imputed values.

See Also

`Hmisc::aregImpute`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Tricube predictive mean matching for nhanes data
#############################################################################

library(mice)
data(nhanes, package="mice")
set.seed(9090)

#*** Model 1: Use default of tricube predictive mean matching
varnames <- colnames(nhanes)
VV <- length(varnames)
method <- rep("tricube.pmm", VV )
```

```
names(method) <- varnames
# imputation with mice
imp.mi1 <- mice::mice( nhanes, m=5, maxit=4, method=method )

#*** Model 2: use item-specific imputation methods
iM2 <- method
iM2["bmi"] <- "pmm6"
# use imputation method 'tricube.pmm' for hyp and chl
# select different scale parameters for these variables
tricube.pmm.scale1 <- list( "hyp"=.15, "chl"=.30 )
imp.mi2 <- miceadds::mice.1chain( nhanes, burnin=5, iter=20, Nimp=4,
               method=iM2, tricube.pmm.scale=tricube.pmm.scale1  )

## End(Not run)
```

`mice.impute.weighted.pmm`

Imputation by Weighted Predictive Mean Matching or Weighted Normal Linear Regression

---

Description

Imputation by predictive mean matching or normal linear regression using sampling weights.

Usage

```
mice.impute.weighted.pmm(y, ry, x, wy=NULL, imputationWeights=NULL,
      pls.facs=NULL, interactions=NULL, quadratics=NULL, donors=5, ...)

mice.impute.weighted.norm(y, ry, x, wy=NULL, ridge=1e-05, pls.facs=NULL,
     imputationWeights=NULL, interactions=NULL, quadratics=NULL, ...)
```

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.
`wy`	Logical vector of length `length(y)`. A `TRUE` value indicates locations in `y` for which imputations are created.
`imputationWeights`	Optional vector of sampling weights
`pls.facs`	Number of factors in PLS regression (if used). The default is `NULL` which means that no PLS regression is used for dimension reduction.
`interactions`	Optional vector of variables for which interactions should be created
`quadratics`	Optional vector of variables which should also be included as quadratic effects.
`donors`	Number of donors
`...`	Further arguments to be passed
`ridge`	Ridge parameter in the diagonal of $\boldsymbol{X}'\boldsymbol{X}$

Value

A vector of length `nmis=sum(!ry)` with imputed values.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation using sample weights
#############################################################################

data( data.ma01)
set.seed(977)

# select subsample
dat <- as.matrix(data.ma01)
dat <- dat[ 1:1000, ]

# empty imputation
imp0 <- mice::mice( dat, maxit=0)

# redefine imputation methods
meth <- imp0$method
meth[ meth=="pmm"  ] <- "weighted.pmm"
meth[ c("paredu", "books", "migrant" ) ] <- "weighted.norm"
# redefine predictor matrix
pm <- imp0$predictorMatrix
pm[, 1:3 ] <- 0
# do imputation
imp <- mice::mice( dat, predictorMatrix=pm, method=meth,
           imputationWeights=dat[,"studwgt"], m=3, maxit=5)

## End(Not run)
```

`mice.nmi` *Nested Multiple Imputation*

Description

Performs nested multiple imputation (Rubin, 2003) for the functions `mice::mice` and `mice.1chain`. The function `mice.nmi` generates an object of class `mids.nmi`.

Usage

```
mice.nmi(datlist, type="mice", ...)

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

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

Arguments

`datlist`	List of datasets for which nested multiple imputation should be applied
`type`	Imputation model: `type="mice"` for `mice::mice` or `type="mice.1chain"` for `mice.1chain`.
`...`	Arguments to be passed to `mice::mice` or `mice.1chain`.
`object`	Object of class `mids.nmi`.
`x`	Object of class `mids.nmi`.

Value

Object of class `mids.nmi` with entries

`imp`	List of nested multiply imputed datasets whose entries are of class `mids` or `mids.1chain`.
`Nimp`	Number of between and within imputations.

References

Rubin, D. B. (2003). Nested multiple imputation of NMES via partially incompatible MCMC. *Statistica Neerlandica, 57*(1), 3-18. doi:10.1111/14679574.00217

See Also

For imputation models see `mice::mice` and `mice.1chain`.

Functions for analyses of nested multiply imputed datasets: `complete.mids.nmi`, `with.mids.nmi`, `pool.mids.nmi`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Nested multiple imputation for TIMSS data
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
   # list of 5 datasets containing 5 plausible values

#** define imputation method and predictor matrix
data <- datlist[[1]]
V <- ncol(data)
# variables
vars <- colnames(data)
# variables not used for imputation
vars_unused <- miceadds::scan.vec("IDSTUD TOTWGT  JKZONE  JKREP" )

#- define imputation method
impMethod <- rep("norm", V )
```

```
names(impMethod) <- vars
impMethod[ vars_unused ] <- ""

#- define predictor matrix
predM <- matrix( 1, V, V )
colnames(predM) <- rownames(predM) <- vars
diag(predM) <- 0
predM[, vars_unused ] <- 0

#****************
# (1) nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist, method=impMethod, predictorMatrix=predM,
                m=4, maxit=3 )
summary(imp1)

#****************
# (2) nested multiple imputation using mice.1chain
imp2 <- miceadds::mice.nmi( datlist, method=impMethod, predictorMatrix=predM,
            Nimp=4, burnin=10,iter=22, type="mice.1chain")
summary(imp2)

## End(Not run)
```

`miceadds-defunct` *Defunct* **miceadds** *Functions*

Description

These functions have been removed or replaced in the **miceadds** package.

Usage

```
fast.groupmean(...)
fast.groupsum(...)
mice.impute.2l.plausible.values(...)
mice.impute.2l.pls(...)
mice.impute.2lonly.norm2(...)
mice.impute.2lonly.pmm2(...)
mice.impute.tricube.pmm2(...)
```

Arguments

`...`	Arguments to be passed.

Details

The `fast.groupmean` function has been replaced by `GroupMean`.

The `fast.groupsum` function has been replaced by `GroupSum`.

The `mice.impute.2l.plausible.values` function has been replaced by `mice.impute.plausible.values`.

The `mice.impute.2l.pls2` function has been replaced by `mice.impute.pls`.

The `mice.impute.2lonly.norm2` and `mice.impute.2lonly.pmm2` functions can be safely replaced by the `mice::mice.impute.2lonly.norm` and `mice::mice.impute.2lonly.pmm` functions in the **mice** package.

The `mice.impute.tricube.pmm2` function has been replaced by `mice.impute.tricube.pmm`.

`miceadds-utilities` *Utility Functions in* **miceadds**

Description

Utility functions in **miceadds**.

Usage

```
## searches for objects in parent environments
ma_exists_get( x, pos, n_index=1:8)
ma_exists( x, pos, n_index=1:8)
mice_imputation_get_states( pos=parent.frame(n=1), n_index=1:20 )
```

Arguments

`x`	Object name (character)
`pos`	Environment
`n_index`	Levels in `parent.frame` in which object is searched

Details

The function `ma_exists_get` is used in `miceadds:::mice_imputation_get_states`.

`mice_imputation_2l_lmer` *Imputation of a Continuous or a Binary Variable From a Two-Level Regression Model using* **lme4** *or* **blme**

Description

The function `mice.impute.2l.continuous` imputes values of continuous variables with a linear mixed effects model using `lme4::lmer` or `blme::blmer`. The `lme4::lmer` or `blme::blmer` function is also used for predictive mean matching where the match is based on predicted values which contain the fixed and (sampled) random effects. Binary variables can be imputed from a two-level logistic regression model fitted with the `lme4::glmer` or `blme::bglmer` function. See Snijders and Bosker (2012) and Zinn (2013) for details.

Usage

```
mice.impute.2l.continuous(y, ry, x, type, intercept=TRUE,
    groupcenter.slope=FALSE, draw.fixed=TRUE, random.effects.shrinkage=1E-6,
    glmer.warnings=TRUE, blme_use=FALSE, blme_args=NULL, ... )

mice.impute.2l.pmm(y, ry, x, type, intercept=TRUE,
    groupcenter.slope=FALSE, draw.fixed=TRUE, random.effects.shrinkage=1E-6,
    glmer.warnings=TRUE, donors=5, match_sampled_pars=TRUE,
    blme_use=FALSE, blme_args=NULL, ... )

mice.impute.2l.binary(y, ry, x, type, intercept=TRUE,
    groupcenter.slope=FALSE, draw.fixed=TRUE, random.effects.shrinkage=1E-6,
    glmer.warnings=TRUE, blme_use=FALSE, blme_args=NULL, ... )
```

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 predictors.
`type`	Type of predictor variable. The cluster identifier has type `-2`, fixed effects predictors without a random slope type `1` and predictors with fixed effects and random effects have type `2`. If the cluster mean should be included for a covariate, `3` should be chosen. The specification `4` includes the cluster mean, the fixed effect and the random effect.
`intercept`	Optional logical indicating whether the intercept should be included.
`groupcenter.slope`	Optional logical indicating whether covariates should be centered around group means
`draw.fixed`	Optional logical indicating whether fixed effects parameter should be randomly drawn
`random.effects.shrinkage`	Shrinkage parameter for stabilizing the covariance matrix of random effects
`glmer.warnings`	Optional logical indicating whether warnings from `glmer` should be displayed
`blme_use`	Logical indicating whether the **blme** package should be used.
`blme_args`	(Prior) Arguments for **blme**, see `blme::blmer` and `blme::bmerDist-class`.
`donors`	Number of donors used for predictive mean matching
`match_sampled_pars`	Logical indicating whether values of nearest neighbors should also be sampled in pmm imputation.
`...`	Further arguments to be passed

Value

A vector of length `nmis=sum(!ry)` with imputed values.

References

Snijders, T. A. B., & Bosker, R. J. (2012). *Multilevel analysis: An introduction to basic and advanced multilevel modeling*. Thousand Oaks, CA: Sage.

Zinn, S. (2013). *An imputation model for multilevel binary data*. NEPS Working Paper No 31.

See Also

See `mice.impute.ml.lmer` for imputation for datasets with more than two levels (e.g., three-level datasets or cross-classified datasets).

Variables at a higher level (e.g. at level 2) can be imputed using `2lonly` functions, for example the `mice::mice.impute.2lonly.norm` function in the **mice** package or the general `mice.impute.2lonly.function` function in the **miceadds** package which using an already defined imputation method at level 1. If a level-2 variable for 3-level data should be imputed, then `mice.impute.ml.lmer` can also be used to impute this variable with a two-level imputation model in which level 1 corresponds to the original level-2 units and level 2 corresponds to the original level-3 units.

See `mice::mice.impute.2l.norm` and `mice::mice.impute.2l.pan` for imputation functions in the **mice** package under fully conditional specification for normally distributed variables. The function `mice::mice.impute.2l.norm` allows for residual variances which are allowed to vary across groups while `mice::mice.impute.2l.pan` assumes homogeneous residual variances.

The **micemd** package provides further imputation methods for the **mice** package for imputing multilevel data with fully conditional specification. The function `micemd::mice.impute.2l.jomo` has similar functionality like `mice::mice.impute.2l.pan` and imputes normally distributed two-level data with a Bayesian MCMC approach, but relies on the **jomo** package instead of the **pan** package. The functions `mice::mice.impute.2l.lmer` and `micemd::mice.impute.2l.glm.norm` have similar functionality like `mice.impute.2l.continuous` and imputes normally distributed two-level data. The function `{micemd::mice.impute.2l.glm.bin}` has similar functionality like `mice.impute.2l.binary` and imputes binary two-level data.

The **hmi** package imputes single-level and multilevel data and is also based on fully conditional specification. The package relies on the MCMC estimation implemented in the **MCMCglmm** package. The imputation procedure can be run with the `hmi::hmi` function.

See the **pan** (`pan::pan`) and the **jomo** (`jomo::jomo`) package for joint multilevel imputation. See `mitml::panImpute` and `mitml::jomoImpute` for wrapper functions to these packages in the **mitml** package.

Imputation by chained equations can also be conducted in blocks of multivariate conditional distributions since **mice** 3.0.0 (see the `blocks` argument in `mice::mice`). The `mitml::panImpute` and `mitml::jomoImpute` functions can be used with `mice::mice` by specifying imputation methods `"panImpute"` (see `mice::mice.impute.panImpute`) and `"jomoImpute"` (see `mice::mice.impute.jomoImpute`)).

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation of a binary variable
#############################################################################

#--- simulate missing values
set.seed(976)
```

```
G <- 30        # number of groups
n <- 8        # number of persons per group
iccx <- .2    # intra-class correlation X
iccy <- .3    # latent intra-class correlation binary outcome
bx <- .4    # regression coefficient
threshy <- stats::qnorm(.70)  # threshold for y
x <- rep( rnorm( G, sd=sqrt( iccx) ), each=n )  +
            rnorm(G*n, sd=sqrt( 1 - iccx) )
y <- bx * x + rep( rnorm( G, sd=sqrt( iccy) ), each=n )  +
                rnorm(G*n, sd=sqrt( 1 - iccy) )
y <- 1 * ( y > threshy )
dat <- data.frame( group=100+rep(1:G, each=n), x=x, y=y )

#* create some missings
dat1 <- dat
dat1[ seq( 1, G*n, 3 ),"y" ]  <- NA
dat1[ dat1$group==2, "y" ] <- NA

#--- prepare imputation in mice
vars <- colnames(dat1)
V <- length(vars)
#* predictor matrix
predmat <- matrix( 0, nrow=V, ncol=V)
rownames(predmat) <- colnames(predmat) <- vars
predmat["y", ] <- c(-2,2,0)
#* imputation methods
impmeth <- rep("",V)
names(impmeth) <- vars
impmeth["y"] <- "2l.binary"

#** imputation with logistic regression ('2l.binary')
imp1 <- mice::mice( data=as.matrix(dat1), method=impmeth,
                predictorMatrix=predmat, maxit=1, m=5 )

#** imputation with predictive mean matching ('2l.pmm')
impmeth["y"] <- "2l.pmm"
imp2 <- mice::mice( data=as.matrix(dat1), method=impmeth,
                predictorMatrix=predmat, maxit=1, m=5 )

#** imputation with logistic regression using blme package
blme_args <- list( "cov.prior"="invwishart")
imp3 <- mice::mice( data=as.matrix(dat1), method=impmeth,
                predictorMatrix=predmat, maxit=1, m=5,
                blme_use=TRUE, blme_args=blme_args )

## End(Not run)
```

`mice_inits` *Arguments for* `mice::mice` *Function*

Description

Defines initial arguments of imputation method and predictor matrix for `mice::mice` function.

Usage

```
mice_inits(dat, ignore=NULL)
```

Arguments

`dat`	Dataset
`ignore`	Vector of variables which should be ignored in imputation

Value

List with entries

`method`	Imputation method
`predictorMatrix`	Predictor matrix

See Also

See `mice::make.predictorMatrix` and `mice::make.method` for generating an initial predictor matrix and a vector of imputation methods.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Inits for mice imputation
#############################################################################

data(data.ma04, package="miceadds")
dat <- data.ma04

res <- miceadds::mice_inits(dat, ignore=c("group") )
str(res)

## End(Not run)
```

`micombine.chisquare` *Combination of Chi Square Statistics of Multiply Imputed Datasets*

Description

This function does inference for the χ^2 statistic based on multiply imputed datasets (see e.g. Enders, 2010, p. 239 ff.; Allison, 2002). This function is also denoted as the D_2 statistic.

Usage

```
micombine.chisquare(dk, df, display=TRUE, version=1)
```

Arguments

`dk`	Vector of chi square statistics
`df`	Degrees of freedom of χ^2 statistic
`display`	An optional logical indicating whether results should be printed at the R console.
`version`	Integer indicating which calculation formula should be used. The default `version=1` refers to the correct formula as in Enders (2010), while `version=0` uses an incorrect formula as printed in Allison (2001). The incorrect calculation `version=0` was included in **miceadds** versions smaller than version 2.0. See also *http://statisticalhorizons.com/wp-content/uploads/2012/01/combchi.sas*.

Value

A vector with following entries

`D`	Combined D_2 statistic which is approximately F-distributed with (`df`, `df2`) degrees of freedom
`p`	The p value corresponding to `D`
`df`	Numerator degrees of freedom
`df2`	Denominator degrees of freedom

References

Allison, P. D. (2002). *Missing data.* Newbury Park, CA: Sage.

Enders, C. K. (2010). *Applied missing data analysis.* Guilford Press.

See Also

See also `mice::pool.compare` for a Wald test to compare two fitted models in the **mice** package.

Examples

```
#############################################################################
# EXAMPLE 1: Chi square values of analyses from 7 multiply imputed datasets
#############################################################################

# Vector of 7 chi square statistics
dk <- c(24.957, 18.051, 18.812, 17.362, 21.234, 18.615, 19.84)
dk.comb <- miceadds::micombine.chisquare(dk=dk, df=4 )
  ##  Combination of Chi Square Statistics for Multiply Imputed Data
  ##  Using 7 Imputed Data Sets
  ##  F(4, 482.06)=4.438     p=0.00157
```

`micombine.cor` *Inference for Correlations and Covariances for Multiply Imputed Datasets*

Description

Statistical inference for correlations and covariances for multiply imputed datasets

Usage

```
micombine.cor(mi.res, variables=NULL, conf.level=0.95,
     method="pearson", nested=FALSE, partial=NULL )

micombine.cov(mi.res, variables=NULL, conf.level=0.95,
     nested=FALSE )
```

Arguments

`mi.res`	Object of class `mids` or `mids.1chain`
`variables`	Indices of variables for selection
`conf.level`	Confidence level
`method`	Method for calculating correlations. Must be one of `"pearson"` or `"spearman"`. The default is the calculation of the Pearson correlation.
`nested`	Logical indicating whether the input dataset stems from a nested multiple imputation.
`partial`	Formula object for computing partial correlations. The terms which should be residualized are written in the formula object `partial`. Alternatively, it can be a vector of variables.

Value

A data frame containing the coefficients (`r`, `cov`) and its corresponding standard error (`rse`, `cov_se`), fraction of missing information (`fmi`) and a t value (`t`).

The corresponding coefficients can also be obtained as matrices by requesting `attr(result,"r_matrix")`.

See Also

See `stats::cor.test` for testing correlation coefficients.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: nhanes data | combination of correlation coefficients
#############################################################################
```

```
library(mice)
data(nhanes, package="mice")
set.seed(9090)

# nhanes data in one chain
imp.mi <- miceadds::mice.1chain( nhanes, burnin=5, iter=20, Nimp=4,
                  method=rep("norm", 4) )

# correlation coefficients of variables 4, 2 and 3 (indexed in nhanes data)
res <- miceadds::micombine.cor(mi.res=imp.mi, variables=c(4,2,3) )
  ##     variable1 variable2       r    rse fisher_r fisher_rse    fmi       t      p
  ##   1       chl       bmi  0.2458 0.2236   0.2510     0.2540 0.3246  0.9879 0.3232
  ##   2       chl       hyp  0.2286 0.2152   0.2327     0.2413 0.2377  0.9643 0.3349
  ##   3       bmi       hyp -0.0084 0.2198  -0.0084     0.2351 0.1904 -0.0358 0.9714
  ##     lower95 upper95
  ##   1 -0.2421  0.6345
  ##   2 -0.2358  0.6080
  ##   3 -0.4376  0.4239

# extract matrix with correlations and its standard errors
attr(res, "r_matrix")
attr(res, "rse_matrix")

# inference for covariance
res2 <- miceadds::micombine.cov(mi.res=imp.mi, variables=c(4,2,3) )

# inference can also be conducted for non-imputed data
res3 <- miceadds::micombine.cov(mi.res=nhanes, variables=c(4,2,3) )

# partial correlation residualizing bmi and chl
res4 <- miceadds::micombine.cor(mi.res=imp.mi, variables=c("age","hyp" ),
                  partial=~bmi+chl )
res4
# alternatively, 'partial' can also be defined as c('age','hyp')

#############################################################################
# EXAMPLE 2: nhanes data | comparing different correlation coefficients
#############################################################################

library(psych)
library(mitools)

# imputing data
imp1 <- mice::mice( nhanes,  method=rep("norm", 4 ) )
summary(imp1)

#*** Pearson correlation
res1 <- miceadds::micombine.cor(mi.res=imp1, variables=c(4,2) )

#*** Spearman rank correlation
res2 <- miceadds::micombine.cor(mi.res=imp1, variables=c(4,2),  method="spearman")

#*** Kendalls tau
```

```
# test of computation of tau for first imputed dataset
dat1 <- mice::complete(imp1, action=1)
tau1 <- psych::corr.test(x=dat1[,c(4,2)], method="kendall")
tau1$r[1,2]    # estimate
tau1$se     # standard error

# results of Kendalls tau for all imputed datasets
res3 <- with( data=imp1,
        expr=psych::corr.test( x=cbind( chl, bmi ), method="kendall") )
# extract estimates
betas <- lapply( res3$analyses, FUN=function(ll){ ll$r[1,2] } )
# extract variances
vars <- lapply( res3$analyses, FUN=function(ll){ (ll$se[1,2])^2 } )
# Rubin inference
tau_comb <- mitools::MIcombine( results=betas, variances=vars )
summary(tau_comb)

#############################################################################
# EXAMPLE 3: Inference for correlations for nested multiply imputed datasets
#############################################################################

library(BIFIEsurvey)
data(data.timss4, package="BIFIEsurvey" )
datlist <- data.timss4

# object of class nested.datlist
datlist <- miceadds::nested.datlist_create(datlist)
# inference for correlations
res2 <- miceadds::micombine.cor(mi.res=datlist, variables=c("lang", "migrant", "ASMMAT"))

## End(Not run)
```

`micombine.F` *Combination of F Statistics for Multiply Imputed Datasets Using a Chi Square Approximation*

Description

Several F statistics from multiply imputed datasets are combined using an approximation based on χ^2 statistics (see `micombine.chisquare`).

Usage

```
micombine.F(Fvalues, df1, display=TRUE, version=1)
```

Arguments

`Fvalues`	Vector containing F values.
`df1`	Degrees of freedom of the numerator. Degrees of freedom of the numerator are approximated by ∞ (large number of degrees of freedom).

`display`	A logical indicating whether results should be displayed at the console
`version`	Integer indicating which calculation formula should be used. The default `version=1` refers to the correct formula as in Enders (2010), while `version=0` uses an incorrect formula as printed in Allison (2001). The incorrect calculation `version=0` was included in **miceadds** versions smaller than version 2.0. See also *http://statisticalhorizons.com/wp-content/uploads/2012/01/combchi.sas*.

Value

The same output as in `micombine.chisquare`

References

Allison, P. D. (2002). *Missing data*. Newbury Park, CA: Sage.

Enders, C. K. (2010). *Applied missing data analysis*. Guilford Press.

Grund, S., Luedtke, O., & Robitzsch, A. (2016). Pooling ANOVA results from multiply imputed datasets: A simulation study. *Methodology, 12*(3), 75-88. doi:10.1027/16142241/a000111

See Also

`micombine.chisquare`

Examples

```
#############################################################################
# EXAMPLE 1: F statistics for 5 imputed datasets
#############################################################################

Fvalues <- c( 6.76, 4.54, 4.23, 5.45, 4.78 )
micombine.F(Fvalues, df1=4 )
  ##  Combination of Chi Square Statistics for Multiply Imputed Data
  ##  Using 5 Imputed Data Sets
  ##  F(4, 52.94)=3.946     p=0.00709
```

`mids2datlist` *Converting a* `mids`, `mids.1chain` *or* `mids.nmi` *Object in a Dataset List*

Description

Converts a `mids`, `mids.1chain` or `mids.nmi` object in a dataset list.

Usage

```
mids2datlist(midsobj, X=NULL)
```

Arguments

`midsobj`	Object of class `mids`, `mids.1chain` or `mids.nmi`
`X`	Optional data frame of variables to be included in imputed datasets.

Value

List of multiply imputed datasets of classes `datlist` or `nested.datlist`.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputing nhanes data and convert result into a dataset list
#############################################################################

data(nhanes,package="mice")

#**** imputation using mice
imp1 <- mice::mice( nhanes, m=3, maxit=5 )
# convert mids object into list
datlist1 <- miceadds::mids2datlist( imp1 )

#**** imputation using mice.1chain
imp2 <- miceadds::mice.1chain( nhanes, burnin=4, iter=20, Nimp=5 )
# convert mids.1chain object into list
datlist2 <- miceadds::mids2datlist( imp2 )

#############################################################################
# EXAMPLE 2: Nested multiple imputation and datalist conversion
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
   # list of 5 datasets containing 5 plausible values

# remove first four variables
M <- length(datlist)
for (ll in 1:M){
    datlist[[ll]] <- datlist[[ll]][, -c(1:4) ]
                }

#***************
# (1) nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist,  m=4, maxit=3 )
summary(imp1)

#***************
# (2) nested multiple imputation using mice.1chain
imp2 <- miceadds::mice.nmi( datlist, Nimp=4, burnin=10,iter=22, type="mice.1chain")
summary(imp2)
```

```
#**************
# conversion into a datalist
datlist.i1 <- miceadds::mids2datlist( imp1 )
datlist.i2 <- miceadds::mids2datlist( imp2 )

#############################################################################
# EXAMPLE 3: mids object conversion and inclusion of further variables
#############################################################################

data(data.ma05)
dat <- data.ma05

# imputation
resp <- dat[, - c(1:2) ]
imp <- mice::mice( resp, method="norm", maxit=2, m=3 )

# convert mids object into datalist
datlist0 <- miceadds::mids2datlist( imp )
# convert mids object into datalist and include some id variables
datlist1 <- miceadds::mids2datlist( imp, X=dat[,c(1,2) ] )

## End(Not run)
```

`mids2mlwin` *Export* `mids` *object to MLwiN*

Description

Converts a `mids` object into a format recognized by the multilevel software MLwiN.

Usage

```
mids2mlwin(imp, file.prefix, path=getwd(), sep=" ", dec=".", silent=FALSE,
   X=NULL)
```

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.
`X`	Optional data frame of variables to be included in imputed datasets.

Value

The return value is `NULL`.

Author(s)

Thorsten Henke

Examples

```
## Not run:
# imputation nhanes data
data(nhanes)
imp <- mice::mice(nhanes)
# write files to MLwiN
mids2mlwin(imp, file.prefix="nhanes" )

## End(Not run)
```

`mi_dstat` *Cohen's d Effect Size for Missingness Indicators*

Description

Computes Cohen's d effect size indicating whether missingness on a variable is related to other variables (covariates).

Usage

```
mi_dstat(dat)
```

Arguments

`dat`	Data frame

Value

A matrix. Missingness indicators refer to rows and covariates to columns.

Examples

```
#############################################################################
# EXAMPLE 1: d effect size for missingness indicators data.ma01
#############################################################################

data(data.ma01)
dat <- data.ma01

# compute d effect sizes
md <- miceadds::mi_dstat(dat)
round( md, 3 )
```

---

ml_mcmc *MCMC Estimation for Mixed Effects Model*

Description

Fits a mixed effects model via MCMC. The outcome can be normally distributed or ordinal (Goldstein, 2011; Goldstein, Carpenter, Kenward & Levin, 2009).

Usage

```
ml_mcmc( formula, data, iter=3000, burnin=500, print_iter=100, outcome="normal",
     nu0=NULL, s0=1, psi_nu0_list=NULL, psi_S0_list=NULL, inits_lme4=FALSE,
     thresh_fac=5.8, ridge=1e-5)

## S3 method for class 'ml_mcmc'
summary(object, digits=4, file=NULL, ...)

## S3 method for class 'ml_mcmc'
plot(x, ask=TRUE, ...)

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

## S3 method for class 'ml_mcmc'
vcov(object, ...)

ml_mcmc_fit(y, X, Z_list, beta, Psi_list, sigma2, alpha, u_list, idcluster_list,
   onlyintercept_list, ncluster_list, sigma2_nu0, sigma2_sigma2_0, psi_nu0_list,
   psi_S0_list, est_sigma2, est_probit, parameter_index, est_parameter, npar, iter,
   save_iter, verbose=TRUE, print_iter=500, parnames0=NULL, K=9999, est_thresh=FALSE,
    thresh_fac=5.8, ridge=1e-5, parm_summary=TRUE )

## exported Rcpp functions
miceadds_rcpp_ml_mcmc_sample_beta(xtx_inv, X, Z_list, y, u_list, idcluster_list, sigma2,
     onlyintercept_list, NR, ridge)
miceadds_rcpp_ml_mcmc_sample_u(X, beta, Z_list, y, ztz_list, idcluster_list,
    ncluster_list, sigma2, Psi_list, onlyintercept_list, NR, u0_list, ridge)
miceadds_rcpp_ml_mcmc_sample_psi(u_list, nu0_list, S0_list, NR, ridge)
miceadds_rcpp_ml_mcmc_sample_sigma2(y, X, beta, Z_list, u_list, idcluster_list,
     onlyintercept_list, nu0, sigma2_0, NR, ridge)
miceadds_rcpp_ml_mcmc_sample_latent_probit(X, beta, Z_list, u_list, idcluster_list, NR,
     y_int, alpha, minval, maxval)
miceadds_rcpp_ml_mcmc_sample_thresholds(X, beta, Z_list, u_list, idcluster_list, NR, K,
     alpha, sd_proposal, y_int)
miceadds_rcpp_ml_mcmc_predict_fixed_random(X, beta, Z_list, u_list, idcluster_list, NR)
miceadds_rcpp_ml_mcmc_predict_random_list(Z_list, u_list, idcluster_list, NR, N)
miceadds_rcpp_ml_mcmc_predict_random(Z, u, idcluster)
```

```
miceadds_rcpp_ml_mcmc_predict_fixed(X, beta)
miceadds_rcpp_ml_mcmc_subtract_fixed(y, X, beta)
miceadds_rcpp_ml_mcmc_subtract_random(y, Z, u, idcluster, onlyintercept)
miceadds_rcpp_ml_mcmc_compute_ztz(Z, idcluster, ncluster)
miceadds_rcpp_ml_mcmc_compute_xtx(X)
miceadds_rcpp_ml_mcmc_probit_category_prob(y_int, alpha, mu1, use_log)
miceadds_rcpp_pnorm(x, mu, sigma)
miceadds_rcpp_qnorm(x, mu, sigma)
miceadds_rcpp_rtnorm(mu, sigma, lower, upper)
```

Arguments

`formula`	An R formula in **lme4**-like specification
`data`	Data frame
`iter`	Number of iterations
`burnin`	Number of burnin iterations
`print_iter`	Integer indicating that every `print_iter`th iteration progress should be displayed
`outcome`	Outcome distribution: `"normal"` or `"probit"`
`nu0`	Prior sample size
`s0`	Prior guess for variance
`inits_lme4`	Logical indicating whether initial values should be obtained from fitting the model in the **lme4** package
`thresh_fac`	Factor for proposal variance for estimating thresholds which is determined as `thresh_fac`$/N$ ($5.8/N$ as default).
`ridge`	Ridge parameter for covariance matrices in sampling
`object`	Object of class `ml_mcmc`
`digits`	Number of digits after decimal used for printing
`file`	Optional file name
`...`	Further arguments to be passed
`x`	Object of class `ml_mcmc`
`ask`	Logical indicating whether display of the next plot should be requested via clicking
`y`	Outcome vector
`X`	Design matrix fixed effects
`Z_list`	Design matrices random effects
`beta`	Initial vector fixed coefficients
`Psi_list`	Initial covariance matrices random effects
`sigma2`	Initial residual covariance matrix
`alpha`	Vector of thresholds
`u_list`	List with initial values for random effects

`idcluster_list`	List with cluster identifiers for random effects
`onlyintercept_list`	List of logicals indicating whether only random intercepts are used for a corresponding random effect
`ncluster_list`	List containing number of clusters for each random effect
`sigma2_nu0`	Prior sample size residual variance
`sigma2_sigma2_0`	Prior guess residual variance
`psi_nu0_list`	List of prior sample sizes for covariance matrices of random effects
`psi_S0_list`	List of prior guesses for covariance matrices of random effects
`est_sigma2`	Logical indicating whether residual variance should be estimated
`est_probit`	Logical indicating whether probit model for ordinal outcomes should be estimated
`parameter_index`	List containing integers for saving parameters
`est_parameter`	List of logicals indicating which parameter type should be estimated
`npar`	Number of parameters
`save_iter`	Vector indicating which iterations should be used
`verbose`	Logical indicating whether progress should be displayed
`parnames0`	Optional vector of parameter names
`K`	Number of categories
`est_thresh`	Logical indicating whether thresholds should be estimated
`parm_summary`	Logical indicating whether a parameter summary table should be computed
`xtx_inv`	Matrix
`NR`	Integer
`ztz_list`	List containing design matrices for random effects
`u0_list`	List containing random effects
`nu0_list`	List with prior sample sizes
`S0_list`	List with prior guesses
`sigma2_0`	Numeric
`y_int`	Integer vector
`minval`	Numeric
`maxval`	Numeric
`sd_proposal`	Numeric vector
`N`	Integer
`Z`	Matrix
`u`	Matrix containing random effects
`idcluster`	Integer vector
`onlyintercept`	Logical

`ncluster`	Integer
`mu1`	Vector
`use_log`	Logical
`mu`	Vector
`sigma`	Numeric
`lower`	Vector
`upper`	Vector

Details

Fits a linear mixed effects model $y = \boldsymbol{Xbeta} + \boldsymbol{Zu} + e$ with MCMC sampling. In case of ordinal data, the ordinal variable y is replaced by an underlying latent normally distributed variable y^* and the residual variance is fixed to 1.

Value

List with following entries (selection)

`sampled_values`	Sampled values
`par_summary`	Parameter summary

References

Goldstein, H. (2011). *Multilevel statistical models*. New York: Wiley. doi:10.1002/9780470973394

Goldstein, H., Carpenter, J., Kenward, M., & Levin, K. (2009). Multilevel models with multivariate mixed response types. *Statistical Modelling, 9*(3), 173-197. doi:10.1177/1471082X0800900301

See Also

See also the **MCMCglmm** package for MCMC estimation and the **lme4** package for maximum likelihood estimation.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Multilevel model continuous data
#############################################################################

library(lme4)

#*** simulate data
set.seed(9097)

# number of clusters and units within clusters
K <- 150
n <- 15
iccx <- .2
idcluster <- rep( 1:K, each=n )
```

```
w <- stats::rnorm( K )
x <- rep( stats::rnorm( K, sd=sqrt(iccx) ), each=n) +
              stats::rnorm( n*K, sd=sqrt( 1 - iccx ))
X <- data.frame(int=1, "x"=x, xaggr=miceadds::gm(x, idcluster),
        w=rep( w, each=n ) )
X <- as.matrix(X)
Sigma <- diag( c(2, .5 ) )
u <- MASS::mvrnorm( K, mu=c(0,0), Sigma=Sigma )
beta <- c( 0, .3, .7, 1 )
Z <- X[, c("int", "x") ]
ypred <- as.matrix(X) %*% beta + rowSums( Z * u[ idcluster, ] )
y <- ypred[,1] + stats::rnorm( n*K, sd=1 )
data <- as.data.frame(X)
data$idcluster <- idcluster
data$y <- y

#*** estimate mixed effects model with miceadds::ml_mcmc() function
formula <- y ~ x + miceadds::gm(x, idcluster) + w + ( 1 + x | idcluster)
mod1 <- miceadds::ml_mcmc( formula=formula, data=data)
plot(mod1)
summary(mod1)

#*** compare results with lme4 package
mod2 <- lme4::lmer(formula=formula, data=data)
summary(mod2)

#############################################################################
# EXAMPLE 2: Multilevel model for ordinal outcome
#############################################################################

#*** simulate data
set.seed(456)
# number of clusters and units within cluster
K <- 500
n <- 10
iccx <- .2
idcluster <- rep( 1:K, each=n )
w <- rnorm( K )
x <- rep( stats::rnorm( K, sd=sqrt(iccx)), each=n) +
                stats::rnorm( n*K, sd=sqrt( 1 - iccx ))
X <- data.frame("int"=1, "x"=x, "xaggr"=miceadds::gm(x, idcluster),
        w=rep( w, each=n ) )
X <- as.matrix(X)
u <- matrix( stats::rnorm(K, sd=sqrt(.5) ), ncol=1)
beta <- c( 0, .3, .7, 1 )
Z <- X[, c("int") ]
ypred <- as.matrix(X) %*% beta + Z * u[ idcluster, ]
y <- ypred[,1] + stats::rnorm( n*K, sd=1 )
data <- as.data.frame(X)
data$idcluster <- idcluster
alpha <- c(-Inf, -.4, 0, 1.7,  Inf)
data$y <- cut( y, breaks=alpha, labels=FALSE ) - 1
```

```
#*** estimate model
formula <- y ~ miceadds::cwc(x, idcluster) + miceadds::gm(x,idcluster) + w + ( 1 | idcluster)
mod <- miceadds::ml_mcmc( formula=formula, data=data, iter=2000, burnin=500,
                outcome="probit", inits_lme4=FALSE)
summary(mod)
plot(mod)

## End(Not run)
```

NestedImputationList *Functions for Analysis of Nested Multiply Imputed Datasets*

Description

The function `NestedImputationList` takes a list of lists of datasets and converts this into an object of class `NestedImputationList`.

Statistical models can be estimated with the function `with.NestedImputationList`.

The `mitools::MIcombine` method can be used for objects of class `NestedImputationResultList` which are the output of `with.NestedImputationList`.

Usage

```
NestedImputationList( datasets )

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

## S3 method for class 'NestedImputationResultList'
MIcombine(results, ...)
```

Arguments

`datasets`	List of lists of datasets which are created by nested multiple imputation.
`x`	Object of class `NestedImputationResultsList`
`results`	Object of class `NestedImputationResultsList`
`...`	Further arguments to be passed.

Value

Function `NestedImputationList`: Object of class `NestedImputationList`.

Function `MIcombine.NestedImputationList`: Object of class `mipo.nmi`.

See Also

`with.NestedImputationList`, `within.NestedImputationList`, `pool.mids.nmi`, `NMIcombine`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Nested multiple imputation and conversion into an object of class
#            NestedImputationList
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2

# remove first four variables
M <- length(datlist)
for (ll in 1:M){
    datlist[[ll]] <- datlist[[ll]][, -c(1:4) ]
                }

# nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist,  m=3, maxit=2 )
summary(imp1)

# create object of class NestedImputationList
datlist1 <- miceadds::mids2datlist( imp1 )
datlist1 <- miceadds::NestedImputationList( datlist1 )

# estimate linear model using with
res1 <- with( datlist1, stats::lm( ASMMAT ~ female + migrant ) )
# pool results
mres1 <- mitools::MIcombine( res1 )
summary(mres1)
coef(mres1)
vcov(mres1)

## End(Not run)
```

`nestedList2List` *Converting a Nested List into a List (and Vice Versa)*

Description

Converts a nested list into a list (and vice versa).

Usage

```
nestedList2List(nestedList)

List2nestedList(List, N_between, N_within=NULL, loop_within=TRUE)
```

Arguments

`nestedList`	A nested list
`List`	A list
`N_between`	Number of between list elements
`N_within`	Number of within list elements
`loop_within`	Optional logical indicating whether looping should start from within list

Value

A list or a nested list

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: List conversions using a small example
#############################################################################

# define a nestedList
nestedList <- as.list(1:3)
nestedList[[1]] <- as.list( 2:4 )
nestedList[[2]] <- as.list( 34 )
nestedList[[3]] <- as.list( 4:9 )

# convert a nested list into a list
v2 <- miceadds::nestedList2List( nestedList)

## reconvert list v2 into a nested list, looping within first
v3 <- miceadds::List2nestedList(v2, N_between=5)
# looping between first
v4 <- miceadds::List2nestedList(v2, N_between=5, loop_within=FALSE)

## End(Not run)
```

`NMIwaldtest` *Wald Test for Nested Multiply Imputed Datasets*

Description

Performs a Wald test for nested multiply imputed datasets (`NMIwaldtest`) and ordinary multiply imputed datasets (`MIwaldtest`), see Reiter and Raghunathan (2007). The corresponding statistic is also called the D_1 statistic.

The function `create.designMatrices.waldtest` is a helper function for the creation of design matrices.

Usage

```
NMIwaldtest(qhat, u, Cdes=NULL, rdes=NULL, testnull=NULL)

MIwaldtest(qhat, u, Cdes=NULL, rdes=NULL, testnull=NULL)

## S3 method for class 'NMIwaldtest'
summary(object, digits=4,...)

## S3 method for class 'MIwaldtest'
summary(object, digits=4,...)

create.designMatrices.waldtest(pars, k)
```

Arguments

`qhat`	List or array of estimated parameters
`u`	List or array of estimated covariance matrices of parameters
`Cdes`	Design matrix C for parameter test (see Details)
`rdes`	Constant vector r (see Details)
`testnull`	Vector containing names of parameters which should be tested for a parameter value of zero.
`object`	Object of class `NMIwaldtest`
`digits`	Number of digits after decimal for print
`...`	Further arguments to be passed
`pars`	Vector of parameter names
`k`	Number of linear hypotheses which should be tested

Details

The Wald test is performed for a linear hypothesis $C\boldsymbol{\theta} = r$ for a parameter vector $\boldsymbol{\theta}$.

Value

List with following entries

`stat`	Data frame with test statistic
`qhat`	Transformed parameter according to linear hypothesis
`u`	Covariance matrix of transformed parameters

Note

The function `create.designMatrices.waldtest` is a helper function for the creation of design matrices.

References

Reiter, J. P. and Raghunathan, T. E. (2007). The multiple adaptations of multiple imputation. *Journal of the American Statistical Association, 102*(480), 1462-1471. doi:10.1198/016214507000000932

See Also

`NMIcombine`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Nested multiple imputation and Wald test | TIMSS data
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
# remove first four variables
M <- length(datlist)
for (ll in 1:M){
    datlist[[ll]] <- datlist[[ll]][, -c(1:4) ]
}

#***************
# (1) nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist,  m=3, maxit=2 )
summary(imp1)

#**** Model 1: Linear regression with interaction effects
res1 <- with( imp1, stats::lm( likesc ~ female*migrant + female*books  ) )
pres1 <- miceadds::pool.mids.nmi( res1 )
summary(pres1)

# test whether both interaction effects equals zero
pars <- dimnames(pres1$qhat)[[3]]
des <- miceadds::create.designMatrices.waldtest( pars=pars, k=2)
Cdes <- des$Cdes
rdes <- des$rdes
Cdes[1, "female:migrant"] <- 1
Cdes[2, "female:books"] <- 1
wres1 <- miceadds::NMIwaldtest( qhat=pres1$qhat, u=pres1$u, Cdes=Cdes, rdes=rdes )
summary(wres1)

# a simpler specification is the use of "testnull"
testnull <- c("female:migrant", "female:books")
wres1b <- miceadds::NMIwaldtest( qhat=qhat, u=u, testnull=testnull )
summary(wres1b)

#**** Model 2: Multivariate linear regression
res2 <- with( imp1, stats::lm( cbind( ASMMAT, ASSSCI ) ~
                           0 + I(1*(female==1)) + I(1*(female==0))   ) )
```

```
pres2 <- miceadds::pool.mids.nmi( res2 )
summary(pres2)

# test whether both gender differences equals -10 points
pars <- dimnames(pres2$qhat)[[3]]
  ##  > pars
  ##  [1] "ASMMAT:I(1 * (female==1))" "ASMMAT:I(1 * (female==0))"
  ##  [3] "ASSSCI:I(1 * (female==1))" "ASSSCI:I(1 * (female==0))"

des <- miceadds::create.designMatrices.waldtest( pars=pars, k=2)
Cdes <- des$Cdes
rdes <- c(-10,-10)
Cdes[1, "ASMMAT:I(1*(female==1))"] <- 1
Cdes[1, "ASMMAT:I(1*(female==0))"] <- -1
Cdes[2, "ASSSCI:I(1*(female==1))"] <- 1
Cdes[2, "ASSSCI:I(1*(female==0))"] <- -1

wres2 <- miceadds::NMIwaldtest( qhat=pres2$qhat, u=pres2$u, Cdes=Cdes, rdes=rdes )
summary(wres2)

# test only first hypothesis
wres2b <- miceadds::NMIwaldtest( qhat=pres2$qhat, u=pres2$u, Cdes=Cdes[1,,drop=FALSE],
                         rdes=rdes[1] )
summary(wres2b)

#############################################################################
# EXAMPLE 2: Multiple imputation and Wald test | TIMSS data
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
dat <- data.timss2[[1]]
dat <- dat[, - c(1:4) ]

# perform multiple imputation
imp <- mice::mice( dat, m=6, maxit=3 )

# define analysis model
res1 <- with( imp, lm( likesc ~ female*migrant + female*books  ) )
pres1 <- mice::pool( res1 )
summary(pres1)

# Wald test for zero interaction effects
qhat <- mitools::MIextract(res1$analyses, fun=coef)
u <- mitools::MIextract(res1$analyses, fun=vcov)
pars <- names(qhat[[1]])
des <- miceadds::create.designMatrices.waldtest( pars=pars, k=2)
Cdes <- des$Cdes
rdes <- des$rdes
Cdes[1, "female:migrant"] <- 1
Cdes[2, "female:books"] <- 1

# apply MIwaldtest function
```

```
wres1 <- miceadds::MIwaldtest( qhat, u, Cdes, rdes )
summary(wres1)

# use again "testnull"
testnull <- c("female:migrant", "female:books")
wres1b <- miceadds::MIwaldtest( qhat=qhat, u=u, testnull=testnull )
summary(wres1b)

#***** linear regression with cluster robust standard errors

# convert object of class mids into a list object
datlist_imp <- miceadds::mids2datlist( imp )
# define cluster
idschool <- as.numeric( substring( data.timss2[[1]]$IDSTUD, 1, 5 ) )
# linear regression
res2 <- lapply( datlist_imp, FUN=function(data){
         miceadds::lm.cluster( data=data, formula=likesc ~ female*migrant + female*books,
                               cluster=idschool ) } )
# extract parameters and covariance matrix
qhat <- lapply( res2, FUN=function(rr){ coef(rr) } )
u <- lapply( res2, FUN=function(rr){ vcov(rr) } )
# perform Wald test
wres2 <- miceadds::MIwaldtest( qhat, u, Cdes, rdes )
summary(wres2)

## End(Not run)
```

`nnig_sim` *Simulation of Multivariate Linearly Related Non-Normal Variables*

Description

Simulates multivariate linearly related non-normally distributed variables (Foldnes & Olsson, 2016). For marginal distributions, skewness and (excess) kurtosis values are provided and the values are simulated according to the Fleishman power transformation (Fleishman, 1978; see `fleishman_sim`).

The function `nnig_sim` simulates data from a multivariate random variable $\boldsymbol{Y}$ which is related to a number of independent variables $\boldsymbol{X}$ (independent generators; Foldnes & Olsson, 2016) which are Fleishman power normally distributed. In detail, it holds that $\boldsymbol{Y} = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{X}$ where the covariance matrix $\boldsymbol{\Sigma}$ is decomposed according to a Cholesky decomposition $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T$.

Usage

```
# determine coefficients
nnig_coef(mean=NULL, Sigma, skew, kurt)

# simulate values
nnig_sim(N, coef)
```

Arguments

`mean`	Vector of means. The default is a vector containing zero means.
`Sigma`	Covariance matrix
`skew`	Vector of skewness values
`kurt`	Vector of (excess) kurtosis values
`N`	Number of cases
`coef`	List of parameters generated by `nnig_coef`

Value

A list of parameter values (`nnig_coef`) or a data frame with simulated values (`nnig_sim`).

References

Fleishman, A. I. (1978). A method for simulating non-normal distributions. *Psychometrika, 43*(4), 521-532. doi:10.1007/BF02293811

Foldnes, N., & Olsson, U. H. (2016). A simple simulation technique for nonnormal data with prespecified skewness, kurtosis, and covariance matrix. *Multivariate Behavioral Research, 51*(2-3), 207-219. doi:10.1080/00273171.2015.1133274

Vale, D. C., & Maurelli, V. A. (1983). Simulating multivariate nonnormal distributions. *Psychometrika, 48*(3), 465-471. doi:10.1007/BF02293687

See Also

See `fungible::monte1` for simulating multivariate linearly related non-normally distributed variables generated by the method of Vale and Morelli (1983). See also the `MultiVarMI::MVNcorr` function in the **MultiVarMI** package and the **SimMultiCorrData** package.

The **MultiVarMI** also includes an imputation function `MultiVarMI::MI` for non-normally distributed variables.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Simulating data with nnig_sim function
#############################################################################

#* define input parameters
Sigma <- matrix( c(1,.5, .2,
                  .5, 1,.7,
                  .2, .7, 1), 3, 3 )
skew <- c(0,1,1)
kurt <- c(1,3,3)

#* determine coefficients
coeff <- miceadds::nnig_coef( Sigma=Sigma, skew=skew, kurt=kurt )
print(coeff)
```

```
#* simulate data
set.seed(2018)
Y <- miceadds::nnig_sim( N=2000, coef=coeff)

#* check descriptive statistics
apply(Y, 2, TAM::weighted_skewness )
apply(Y, 2, TAM::weighted_kurtosis )

## End(Not run)
```

`output.format1` R *Utilities: Formatting R Output on the* R *Console*

Description

This function does some formatting of output.

Usage

```
output.format1(stringtype, label, rep.N=1,stringlength=70)
```

Arguments

`stringtype`	Type of string for display, e.g. `”*”`, `”-”`, ...
`label`	Some comment which should be displayed at the console
`rep.N`	Number of lines which shall be left blank
`stringlength`	Length of vector with `label`

Value

Generates a string output at the R console

Examples

```
output.format1(stringtype=”*'”, label=”HELLO WORLD”, stringlength=20)
##   *'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'*'
##   HELLO WORLD
```

`pca.covridge` *Principal Component Analysis with Ridge Regularization*

Description

Performs a principal component analysis for a dataset while a ridge parameter is added on the diagonal of the covariance matrix.

Usage

```
pca.covridge(x, ridge=1E-10, wt=NULL )
```

Arguments

`x`	A numeric matrix
`ridge`	Ridge regularization parameter for the covariance matrix
`wt`	Optional vector of weights

Value

A list with following entries:

`loadings`	Matrix of factor loadings
`scores`	Matrix of principal component scores
`sdev`	Vector of standard deviations of factors (square root of eigenvalues)

See Also

Principal component analysis in **stats**: `stats::princomp`

For calculating first eigenvalues of a symmetric matrix see also `sirt::sirt_eigenvalues` in the **sirt** package.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: PCA on imputed internet data
#############################################################################

library(mice)
data(data.internet)
dat <- as.matrix( data.internet)

# single imputation in mice
imp <- mice::mice( dat, m=1, maxit=10 )

# apply PCA
```

```
pca.imp <- miceadds::pca.covridge( complete(imp) )
  ##   > pca.imp$sdev
  ##      Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7
  ##   3.0370905 2.3950176 2.2106816 2.0661971 1.8252900 1.7009921 1.6379599

# compare results with princomp
pca2.imp <- stats::princomp( complete(imp) )
  ##   > pca2.imp
  ##   Call:
  ##   stats::princomp(x=complete(imp))
  ##
  ##   Standard deviations:
  ##      Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7
  ##   3.0316816 2.3907523 2.2067445 2.0625173 1.8220392 1.6979627 1.6350428

## End(Not run)
```

`pool.mids.nmi` *Pooling for Nested Multiple Imputation*

Description

Statistical inference for scalar parameters for nested multiply imputed datasets (Rubin, 2003; Harel & Schafer, 2002, 2003; Reiter & Raghanuthan, 2007; Harel, 2007).

The `NMIcombine` (`pool_nmi` as a synonym) and `NMIextract` functions are extensions of `mitools::MIcombine` and `mitools::MIextract`.

Usage

```
pool.mids.nmi(object, method="largesample")

NMIcombine( qhat, u=NULL, se=NULL, NMI=TRUE, comp_cov=TRUE, is_list=TRUE,
       method=1)

pool_nmi( qhat, u=NULL, se=NULL, NMI=TRUE, comp_cov=TRUE, is_list=TRUE,
       method=1)

NMIextract(results, expr, fun)

## S3 method for class 'mipo.nmi'
summary(object, digits=4, ...)

## S3 method for class 'mipo.nmi'
coef(object, ...)

## S3 method for class 'mipo.nmi'
vcov(object, ...)
```

Arguments

`object`	Object of class `mids.nmi`. For `summary` it must be an object of class `mipo.nmi`.
`method`	For `pool.mids.nmi`: Method for calculating degrees of freedom. Until now, only the method `"largesample"` is available. For `NMIcombine` and `pool_nmi`: Computation method of fraction of missing information. `method=1` is due to Harel and Schafer (2003) or Shen (2007). `method=2` is due to Harel and Schafer (2002) and is coherent to the calculation for multiply imputed datasets, while the former method is not.
`qhat`	List of lists of parameter estimates. In case of an ordinary imputation it can only be a list.
`u`	Optional list of lists of covariance matrices of parameter estimates
`se`	Optional vector of standard errors. This argument overwrites `u` if it is provided.
`NMI`	Optional logical indicating whether the `NMIcombine` function should be applied for results of nested multiply imputed datasets. It is set to `FALSE` if only a list results of multiply imputed datasets is available.
`comp_cov`	Optional logical indicating whether covariances between parameter estimates should be estimated.
`is_list`	Optional logical indicating whether `qhat` and `u` are provided as lists as an input. If `is_list=FALSE`, appropriate arrays can be used as input.
`results`	A list of objects
`expr`	An expression
`fun`	A function of one argument
`digits`	Number of digits after decimal for printing results in `summary`.
`...`	Further arguments to be passed.

Value

Object of class `mipo.nmi` with following entries

`qhat`	Estimated parameters in all imputed datasets
`u`	Estimated covariance matrices of parameters in all imputed datasets
`qbar`	Estimated parameter
`ubar`	Average estimated variance within imputations
`Tm`	Total variance of parameters
`df`	Degrees of freedom
`lambda`	Total fraction of missing information
`lambda_Between`	Fraction of missing information of between imputed datasets (first stage imputation)
`lambda_Within`	Fraction of missing information of within imputed datasets (second stage imputation)

References

Harel, O., & Schafer, J. (2002). *Two stage multiple imputation*. Joint Statistical Meetings - Biometrics Section.

Harel, O., & Schafer, J. (2003). *Multiple imputation in two stages*. In Proceedings of Federal Committee on Statistical Methodology 2003 Conference.

Harel, O. (2007). Inferences on missing information under multiple imputation and two-stage multiple imputation. *Statistical Methodology, 4*(1), 75-89. doi:10.1016/j.stamet.2006.03.002

Reiter, J. P. and Raghunathan, T. E. (2007). The multiple adaptations of multiple imputation. *Journal of the American Statistical Association, 102*(480), 1462-1471. doi:10.1198/016214507000000932

Rubin, D. B. (2003). Nested multiple imputation of NMES via partially incompatible MCMC. *Statistica Neerlandica, 57*(1), 3-18. doi:10.1111/14679574.00217

See Also

`mice::pool`, `mitools::MIcombine`, `mitools::MIextract`

`mice.nmi`, `MIcombine.NestedImputationResultList`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Nested multiple imputation and statistical inference
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
# remove first four variables
M <- length(datlist)
for (ll in 1:M){
    datlist[[ll]] <- datlist[[ll]][, -c(1:4) ]
               }

#***************
# (1) nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist,  m=3, maxit=2 )
summary(imp1)

#***************
# (2) first linear regression: ASMMAT ~ migrant + female
res1 <- with( imp1, stats::lm( ASMMAT ~ migrant + female ) ) # fit
pres1 <- miceadds::pool.mids.nmi( res1 )  # pooling
summary(pres1)  # summary
coef(pres1)
vcov(pres1)

#***************
# (3) second linear regression: likesc ~ migrant + books
res2 <- with( imp1, stats::lm( likesc ~ migrant + books  ) )
```

```
pres2 <- miceadds::pool.mids.nmi( res2 )
summary(pres2)

#***************
# (4) some descriptive statistics using the mids.nmi object
res3 <- with( imp1, c( "M_lsc"=mean(likesc), "SD_lsc"=stats::sd(likesc) ) )
pres3 <- miceadds::NMIcombine( qhat=res3$analyses )
summary(pres3)

#*************
# (5) apply linear regression based on imputation list

# convert mids object to datlist
datlist2 <- miceadds::mids2datlist( imp1 )
str(datlist2, max.level=1)

# double application of lapply to the list of list of nested imputed datasets
res4 <- lapply( datlist2, FUN=function(dl){
    lapply( dl, FUN=function(data){
            stats::lm( ASMMAT ~ migrant + books, data=data )
                                } )
                }  )

# extract coefficients
qhat <- lapply( res4, FUN=function(bb){
            lapply( bb, FUN=function(ww){
                    coef(ww)
                        } )
                } )
# shorter function
NMIextract( results=res4, fun=coef )

# extract covariance matrices
u <- lapply( res4, FUN=function(bb){
            lapply( bb, FUN=function(ww){
                    vcov(ww)
                        } )
                } )
# shorter function
NMIextract( results=res4, fun=vcov )

# apply statistical inference using the NMIcombine function
pres4 <- miceadds::NMIcombine( qhat=qhat, u=u )
summary(pres4)

#--- statistical inference if only standard errors are available
# extract standard errors
se <- lapply( res4, FUN=function(bb){
            lapply( bb, FUN=function(ww){
                # ww <- res4[[1]][[1]]
                sww <- summary(ww)
                sww$coef[,"Std. Error"]
                        } )
```

```
                } )
se
# apply NMIcombine function
pres4b <- miceadds::NMIcombine( qhat=qhat, se=se )
# compare results
summary(pres4b)
summary(pres4)

#############################################################################
# EXAMPLE 2: Some comparisons for a multiply imputed dataset
#############################################################################

library(mitools)
data(data.ma02)

# save dataset as imputation list
imp <- mitools::imputationList( data.ma02 )
print(imp)
# save dataset as an mids object
imp1 <- miceadds::datlist2mids( imp )

# apply linear model based on imputationList
mod <- with( imp, stats::lm( read ~ hisei + female ) )
# same linear model based on mids object
mod1 <- with( imp1, stats::lm( read ~ hisei + female ) )

# extract coefficients
cmod <- mitools::MIextract( mod, fun=coef)
# extract standard errors
semod <- lapply( mod, FUN=function(mm){
                smm <- summary(mm)
                smm$coef[,"Std. Error"]
                        } )
# extract covariance matrix
vmod <- mitools::MIextract( mod, fun=vcov)

#*** pooling with NMIcombine with se (1a) and vcov (1b) as input
pmod1a <- miceadds::NMIcombine( qhat=cmod, se=semod, NMI=FALSE )
pmod1b <- miceadds::NMIcombine( qhat=cmod, u=vmod, NMI=FALSE )
# use method 2 which should conform to MI inference of mice::pool
pmod1c <- miceadds::NMIcombine( qhat=cmod, u=vmod, NMI=FALSE, method=2)

#*** pooling with mitools::MIcombine function
pmod2 <- mitools::MIcombine( results=cmod, variances=vmod )
#*** pooling with mice::pool function
pmod3a <- mice::pool( mod1 )
pmod3b <- mice::pool( mod1, method="Rubin")

#--- compare results
summary(pmod1a)   # method=1  (the default)
summary(pmod1b)   # method=1  (the default)
summary(pmod1c)   # method=2
summary(pmod2)
```

```
summary(pmod3a)
summary(pmod3b)

## End(Not run)
```

---

pool_mi *Statistical Inference for Multiply Imputed Datasets*

Description

Statistical inference for multiply imputed datasets. See `mitools::MIcombine` or `mice::pool` for functions of the same functionality.

Usage

```
pool_mi(qhat, u=NULL, se=NULL, dfcom=1e+07, method="smallsample")

## S3 method for class 'pool_mi'
summary(object, alpha=0.05, ...)

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

## S3 method for class 'pool_mi'
vcov(object, ...)
```

Arguments

`qhat`	List of parameter vectors
`u`	List of covariance matrices
`se`	List of vector of standard errors. Either `u` or `se` must be provided.
`dfcom`	Degrees of freedom of statistical analysis
`method`	The default is the small sample inference (`"smallsample"`). Any other input provides large sample inference.
`object`	Object of class `pool_mi`
`alpha`	Confidence level
`...`	Further arguments to be passed

Value

Object of with similar output as produced by the `mice::pool` function.

See Also

`mitools::MIcombine`, `mice::pool`, `mitml::testEstimates`

For statistical inference for nested multiply imputed datasets see `NMIcombine`.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Statistical inference for models based on imputationList
#############################################################################

library(mitools)
library(mice)
library(Zelig)
library(mitml)
library(lavaan)
library(semTools)
data(data.ma02)

# save dataset as imputation list
imp <- mitools::imputationList( data.ma02 )
# mids object
imp0 <- miceadds::datlist2mids( imp )
# datlist object
imp1 <- miceadds::datlist_create(data.ma02)

#--- apply linear model based on imputationList
mod <- with( imp, stats::lm( read ~ hisei + female ) )
#--- apply linear model for mids object
mod0 <- with( imp0, stats::lm( read ~ hisei + female ) )
# extract coefficients
cmod <- mitools::MIextract( mod, fun=coef)
# extract standard errors
semod <- lapply( mod, FUN=function(mm){
    smm <- summary(mm)
    smm$coef[,"Std. Error"]
} )
# extract covariance matrix
vmod <- mitools::MIextract( mod, fun=vcov)

#*** pooling based on covariance matrices
res1 <- miceadds::pool_mi( qhat=cmod, u=vmod )
summary(res1)
coef(res1)
vcov(res1)

#*** pooling based on standard errors
res2 <- miceadds::pool_mi( qhat=cmod, se=semod )

#*** pooling with MIcombine
res3 <- mitools::MIcombine( results=cmod, variances=vmod )

#*** pooling with pool function in mice
res4 <- mice::pool( mod0 )

#*** analysis in Zelig
# convert datalist into object of class amelia
```

```
mi02 <- list( "imputations"=data.ma02)
class(mi02) <- "amelia"
res5 <- Zelig::zelig( read ~ hisei + female, model="ls", data=mi02 )

#*** analysis in lavaan
lavmodel <- "
     read ~ hisei + female
     read ~~ a*read
     read ~ 1
     # residual standard deviation
     sde :=sqrt(a)
       "
# analysis for first imputed dataset
mod6a <- lavaan::sem( lavmodel, data=imp1[[1]] )
summary(mod6a)
# analysis based on all datasets using with
mod6b <- lapply( imp1, FUN=function(data){
           res <- lavaan::sem( lavmodel, data=data )
           return(res)
                } )
# extract parameters and covariance matrices
qhat0 <- lapply( mod6b, FUN=function(ll){  coef(ll) } )
u0 <- lapply( mod6b, FUN=function(ll){  vcov(ll) } )
res6b <- mitools::MIcombine( results=qhat0, variances=u0 )

# extract informations for all parameters
qhat <- lapply( mod6b, FUN=function(ll){
        h1 <- lavaan::parameterEstimates(ll)
        parnames <- paste0( h1$lhs, h1$op, h1$rhs )
        v1 <- h1$est
        names(v1) <- parnames
        return(v1)
     } )
se <- lapply( mod6b, FUN=function(ll){
        h1 <- lavaan::parameterEstimates(ll)
        parnames <- paste0( h1$lhs, h1$op, h1$rhs )
        v1 <- h1$se
        names(v1) <- parnames
        return(v1)
     } )
res6c <- miceadds::pool_mi( qhat=qhat, se=se )

# function runMI in semTools package
res6d <- semTools::runMI(model=lavmodel, data=imp1, m=length(imp1) )
  # semTools version 0.4-9 provided an error message
# perform inference with mitml package
se2 <- lapply( se, FUN=function(ss){  ss^2  } )  # input variances
res6e <- mitml::testEstimates(qhat=qhat, uhat=se2)

#*** complete model estimation and inference in mitml

# convert into object of class mitml.list
ml02 <- mitml::as.mitml.list( data.ma02)
```

```
# estimate regression
mod7 <- with( ml02, stats::lm( read ~ hisei + female ) )
# inference
res7 <- mitml::testEstimates( mod7 )

#*** model comparison
summary(res1)
summary(res2)
summary(res3)
summary(res4)
summary(res5)
summary(res6b)
summary(res6c)
print(res6e)
print(res7)

## End(Not run)
```

`Reval` R *Utilities: Evaluates a String as an Expression in* R

Description

This function evaluates a string as an R expression.

Usage

```
Reval(Rstring, print.string=TRUE, n.eval.parent=1)

# Reval( print(Rstring) )
Revalpr(Rstring, print.string=TRUE)

#  Reval( print(str(Rstring)) )
Revalprstr(Rstring, print.string=TRUE)

#  Reval( print(round(Rstring, digits)) )
Revalpr_round( Rstring, digits=5, print.string=TRUE)

#  Reval( print(max(abs(Rstring_x - Rstring_y)) ) )
Revalpr_maxabs( Rstring_x, Rstring_y, print.string=TRUE, na.rm=FALSE)
```

Arguments

`Rstring`	String which shall be evaluated in R
`print.string`	Should the string printed on the console?
`n.eval.parent`	Index of parent environment in which the R command should be evaluated.
`digits`	Number of digits after decimal.

`Rstring_x`	String corresponding to an R object
`Rstring_y`	String corresponding to an R object
`na.rm`	Logical indicating whether missing values should be removed from calculation

Details

The string is evaluated in the parent environment. See `base::eval` for the definition of environments in R.

Examples

```
# This function is simply a shortage function
# See the definition of this function:
Reval <- function( Rstring, print.string=TRUE){
    if (print.string){ cat( paste( Rstring ), "\n"  ) }
        eval.parent( parse( text=paste( Rstring )), n=1 )
            }

Reval( "a <- 2^3" )
  ## a <- 2^3
a
  ## [1] 8
```

`Rfunction_include_argument_values`

Utility Functions for Writing R Functions

Description

Utility functions for writing R functions.

Usage

```
## include argument values in a function input
Rfunction_include_argument_values(string, maxlen=70)

## assign objects to entries in a list
Rfunction_output_list_result_function(string, mid=" <- res$")

## delete declaration of Rcpp and RcppArmadillo object classes
Rcppfunction_remove_classes(string, maxlen=70, remove=TRUE)
```

Arguments

`string`	String
`maxlen`	Maximal string length for output
`mid`	Middle term in the output
`remove`	Logical indicating whether object classes should be removed

Value

String

Examples

```
#############################################################################
# EXAMPLE 1: Toy examples
#############################################################################

##**** extend missing arguments

string <- "
          mice.impute.2l.pls2(y, ry, x, type, pls.facs=pls.facs  ))
          "
cat( miceadds::Rfunction_include_argument_values(string) )
  ##    mice.impute.2l.pls2( y=y, ry=ry, x=x, type=type, pls.facs=pls.facs )

##**** assignment to objects as entries in a list

string <- "
          list( vname=vname, p, type=type, data=data, levels_id )
          "
cat( miceadds::Rfunction_output_list_result_function( string ) )
  ##
  ##  vname <- res$vname
  ##  p <- res$p
  ##  type <- res$type
  ##  data <- res$data
  ##  levels_id <- res$levels_id


string <- "
arma::colvec miceadds_rcpp_rtnorm2( arma::colvec mu,
            double sigma0, arma::colvec lower, arma::colvec upper,
            double minval, double maxval)
    "

cat( miceadds::Rcppfunction_remove_classes(string, maxlen=70) )
cat( miceadds::Rcppfunction_remove_classes(string, maxlen=70, remove=FALSE) )
```

`Rhat.mice` *Rhat Convergence Statistic of a* `mice` *Imputation*

Description

Computes the Rhat statistic for a `mids` object.

Usage

```
Rhat.mice(mice.object)
```

Arguments

`mice.object`	Object of class `mids`

Value

Data frame containing the Rhat statistic for mean and variances for all variables of the Markov chains used for imputation

References

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

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Rhat statistic for nhanes data
#############################################################################

library(mice)
data(nhanes, package="mice")
set.seed(9090)

# nhanes 3 parallel chains
imp1 <- mice::mice( nhanes, m=3, maxit=10, method=rep("norm", 4 ))
miceadds::Rhat.mice( imp1 )
  ##     variable MissProp Rhat.M.imp Rhat.Var.imp
  ##   1      bmi       36  1.0181998     1.155807
  ##   2      hyp       32  1.0717677     1.061174
  ##   3      chl       40  0.9717109     1.318721

## End(Not run)
```

`round2` R *Utilities: Rounding DIN 1333 (Kaufmaennisches Runden)*

Description

This is a rounding function which rounds up for all numbers according to the rule of 'kaufmaennisches Runden' (DIN 1333).

Usage

```
round2(vec, digits=0)
```

Arguments

`vec`	Numeric vector
`digits`	Number of digits after decimal for rounding

Value

Vector with rounded values

Examples

```
#############################################################################
# EXAMPLE 1:
#############################################################################

vec <- c( 1.5, 2.5, 3.5, 1.51,  1.49)
vec
round(vec)
round2(vec)
  ##   > vec
  ##   [1] 1.50 2.50 3.50 1.51 1.49
  ##   > round(vec)
  ##   [1] 2 2 4 2 1
  ##   > miceadds::round2(vec)
  ##   [1] 2 3 4 2 1

#############################################################################
# EXAMPLE 2:
#############################################################################

vec <- - c( 1.5, 2.5, 3.5, 1.51,  1.49)
vec
round(vec)
round2(vec)
  ##   > vec
  ##   [1] -1.50 -2.50 -3.50 -1.51 -1.49
  ##   > round(vec)
  ##   [1] -2 -2 -4 -2 -1
  ##   > miceadds::round2(vec)
  ##   [1] -2 -3 -4 -2 -1

#############################################################################
# EXAMPLE 3:
#############################################################################

vec <- c(8.4999999, 8.5, 8.501, 7.4999999, 7.5, 7.501 )
round(vec)
round2( vec )
round2( vec, digits=1)
round2( -vec )
  ##   > round(vec)
  ##   [1] 8 8 9 7 8 8
  ##   > miceadds::round2( vec )
  ##   [1] 8 9 9 7 8 8
  ##   > miceadds::round2( vec, digits=1)
  ##   [1] 8.5 8.5 8.5 7.5 7.5 7.5
  ##   > miceadds::round2( -vec )
  ##   [1] -8 -9 -9 -7 -8 -8
```

`Rsessinfo` R *Utilities:* R *Session Information*

Description

Informs about current R session.

Usage

```
Rsessinfo()
```

Value

A string containing reduced information about R session info

Examples

```
Rsessinfo()
  ##   > miceadds::Rsessinfo()
 ##  [1] "R version 2.15.2 (2012-10-26) x86_64, mingw32 | nodename=SD70 | login=robitzsch"
```

`save.data` R *Utilities: Saving/Writing Data Files using* **miceadds**

Description

This function is a wrapper function for saving or writing data frames or matrices.

Usage

```
save.data( data, filename, type="Rdata", path=getwd(), row.names=FALSE, na=NULL,
      suffix=NULL, suffix_space="__", index=FALSE, systime=FALSE, ...)
```

Arguments

`data`	Data frame or matrix to be saved
`filename`	Name of data file
`type`	The type of file in which the data frame or matrix should be loaded. This can be `Rdata` (for R binary format, using `base::save`, `csv` (using `utils::write.csv2`), `csv` (using `utils::write.csv`), `table` (using `utils::write.table`), `sav` (using `sjlabelled::write_spss`), `RDS` (using `saveRDS`). `type` can also be a vector if the data frame should be saved in multiple formats.
`path`	Directory from which the dataset should be loaded
`row.names`	Optional logical indicating whether row names should be included in saved `csv` or `csv2` files.

`na`	Missing value handling. The default is `""` for `type="csv"` and `type="csv2"` and is `"."` for `type="table"`.
`suffix`	Optional suffix in file name.
`suffix_space`	Optional place holder if a suffix is used.
`index`	Optional logical indicating whether an index should be included in the first column using the function `index.dataframe`.
`systime`	If `index=TRUE`, this optional logical indicates whether a time stamp should be included in the second column.
`...`	Further arguments to be passed to `save`, `write.csv2`, `write.csv`, `write.table` or `sjlabelled::write_spss`.

See Also

See `load.Rdata` and `load.data` for saving/writing R data frames.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Save dataset data.ma01
#############################################################################

#*** use data.ma01 as an example for writing data files using save.data
data(data.ma01)
dat <- data.ma01

# set a working directory
pf2 <- "P:/ARb/temp_miceadds"

# save data in Rdata format
miceadds::save.data( dat, filename="ma01data", type="Rdata", path=pf2)

# save data in table format without row and column names
miceadds::save.data( dat, filename="ma01data", type="table", path=pf2,
            row.names=FALSE, na=".", col.names=FALSE)

# save data in csv2 format, including time stamp in file name
# and row index and time stamp in saved data
miceadds::save.data( dat, filename="ma01data", type="csv2", path=pf2,
            row.names=FALSE, na="", suffix=systime()[5],
            index=TRUE, systime=TRUE )

# save data in sav format
miceadds::save.data( dat, filename="ma02data", type="sav",  path=pf2 )

# save data file in different formats
types <- c("Rdata", "csv2", "sav")
sapply( types, FUN=function(type){
    miceadds::save.data( dat, filename="ma02data", type=type,  path=pf2,
               suffix=miceadds::systime()[3], row.names=TRUE  )
```

```
                                    } )

# save data frame in multiple file formats (sav, table and csv2)
miceadds::save.data( dat, filename="ma03data", type=c("sav","table","csv2"),  path=pf2,
            suffix=miceadds::systime()[7]  )

## End(Not run)
```

---

save.Rdata — R *Utilities: Save a Data Frame in* Rdata *Format*

Description

This function saves a data frame in a `Rdata` format.

Usage

```
save.Rdata(dat, name, path=NULL, part.numb=1000)
```

Arguments

`dat`	Data frame
`name`	Name of the R object to be saved
`path`	Directory for saving the object
`part.numb`	Number of rows of the data frame which should also be saved in csv format. The default is saving 1000 rows.

Examples

```
## Not run:
dfr <- matrix( 2*1:12-3, 4,3 )
save.Rdata( dfr, "dataframe_test" )

## End(Not run)
```

---

scale_datlist — *Adding a Standardized Variable to a List of Multiply Imputed Datasets or a Single Datasets*

Description

Adds a standardized variable to a list of multiply imputed datasets or a single dataset. This function extends `base::scale` for a data frame to a list of multiply imputed datasets.

Usage

```
scale_datlist(datlist, orig_var, trafo_var, weights=NULL, M=0, SD=1,
    digits=NULL)
```

Arguments

`datlist`	A data frame, a list of multiply imputed datasets of one of the classes `datlist` or `imputationList` or a list of nested multiply imputed datasets of one of the classes `nested_datlist` or `NestedImputationList`.
`orig_var`	Vector with names of the variables to be transformed
`trafo_var`	Vector with names of the standardized variables
`weights`	Optional vector of sample weights. Alternatively, the `weights` can also be a string indicating the variable used from `datlist`.
`M`	Mean of the transformed variable
`SD`	Standard deviation of the transformed variable
`digits`	Number of digits used for rounding the standardized variable

Value

A vector or a matrix

See Also

`base::scale`, `ma.scale2`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Standardized variables in list of multiply imputed datasets
#############################################################################

data(data.ma02)
datlist <- data.ma02

#--- object of class 'datlist'
datlist <- miceadds::datlist_create( datlist )

# mean and SD of variable hisei
miceadds::ma.wtd.meanNA(data=datlist, weights=datlist[[1]]$studwgt, vars="hisei" )
mean( unlist( lapply( datlist, FUN=function(data){
        stats::weighted.mean( data$hisei, data$studwgt )  } ) ) )
miceadds::ma.wtd.sdNA(data=datlist, weights=datlist[[1]]$studwgt, vars="hisei" )
mean( unlist( lapply( datlist, FUN=function(data){
        sqrt( Hmisc::wtd.var( data$hisei, data$studwgt ) ) } ) ) )

# standardize variable hisei to M=100 and SD=15
datlist1a <- miceadds::scale_datlist( datlist=datlist, orig_var="hisei",
               trafo_var="hisei100", weights=datlist[[1]]$studwgt, M=100, SD=15 )
```

```
# check mean and SD
miceadds::ma.wtd.meanNA(data=datlist1a, weights=datlist[[1]]$studwgt, vars="hisei100")
miceadds::ma.wtd.sdNA(data=datlist1a, weights=datlist[[1]]$studwgt, vars="hisei100")

#--- do standardization for unweighted sample with books <=3
# -> define a weighting variable at first
datlist0 <- mitools::imputationList( datlist )
datlist2a <- miceadds::within.imputationList( datlist0, {
             # define weighting variable
                 wgt_books <- 1 * ( books <=3 )
                    } )

# standardize variable hisei to M=100 and SD=15 with respect to weighting variable
datlist2b <- miceadds::scale_datlist( datlist=datlist2a, orig_var="hisei", trafo_var="hisei100",
         weights="wgt_books", M=100, SD=15 )

# check mean and SD (groupwise)
miceadds::ma.wtd.meanNA(data=datlist1a, weights=datlist[[1]]$studwgt, vars="hisei100")
miceadds::ma.wtd.sdNA(data=datlist1a, weights=datlist[[1]]$studwgt, vars="hisei100")

#--- transformation for a single dataset
dat0 <- datlist[[1]]
dat0a <- miceadds::scale_datlist( datlist=dat0, orig_var="hisei", trafo_var="hisei100",
                    weights=dat0$studwgt, M=100, SD=15 )
stats::weighted.mean( dat0a[,"hisei"],  w=dat0a$studwgt )
stats::weighted.mean( dat0a[,"hisei100"],  w=dat0a$studwgt )
sqrt( Hmisc::wtd.var( dat0a[,"hisei100"],  weights=dat0a$studwgt ) )

#--- Standardizations for objects of class imputationList
datlist2 <- mitools::imputationList(datlist)   # object class conversion
datlist2a <- miceadds::scale_datlist( datlist=datlist2, orig_var="hisei",
                 trafo_var="hisei100", weights=datlist[[1]]$studwgt, M=100, SD=15 )

#############################################################################
# EXAMPLE 2: Standardized variables in list of nested multiply imputed datasets
#############################################################################

# nested multiply imputed dataset in BIFIEsurvey package
data(data.timss4, package="BIFIEsurvey")
datlist <- data.timss4
wgt <- datlist[[1]][[1]]$TOTWGT

# class nested.datlist
imp1 <- miceadds::nested.datlist_create( datlist )
# class NestedImputationList
imp2 <- miceadds::NestedImputationList( datlist )

# standardize variable scsci
imp1a <- miceadds::scale_datlist( datlist=imp1, orig_var="scsci", trafo_var="zscsci", weights=wgt)
# check descriptives
miceadds::ma.wtd.meanNA( imp1a, weights=wgt, vars=c("scsci", "zscsci" ) )
miceadds::ma.wtd.sdNA( imp1a, weights=wgt, vars=c("scsci", "zscsci" ) )
```

```
##############################################################################
# EXAMPLE 3: Standardization of variables for imputed data in mice package
##############################################################################

data(nhanes, package="mice")
set.seed(76)

#--- impute nhanes data
imp <- mice::mice(nhanes)
#--- convert into datlist
datlist <- miceadds::mids2datlist(imp)
#--- scale datlist (all variables)
vars <- colnames(nhanes)
sdatlist <- miceadds::scale_datlist(datlist, orig_var=vars, trafo_var=paste0("z",vars) )
#--- reconvert to mids object
imp2 <- miceadds::datlist2mids(sdatlist)
#*** compare descriptive statistics of objects
round( miceadds::mean0( mice::complete(imp, action=1) ), 2 )
round( miceadds::mean0( mice::complete(imp2, action=1) ), 2 )

## End(Not run)
```

`scan.vec` *R Utilities: Scan a Character Vector*

Description

The function `scan.vec` function splits a string into a character vector. The function `scan0` is the `base::scan` function using the default `what="character"`.

Usage

```
scan.vec(vec)
scan.vector(vec)

scan0(file="", ...)
```

Arguments

`vec`	A string which should be split according to blanks
`file`	File to be scanned. See `base::scan`.
`...`	Further arguments to be passed. See `base::scan`.

See Also

`base::scan`

Examples

```
#############################################################################
# EXAMPLE 1: Example scan.vec | reading a string
#############################################################################


vars <- miceadds::scan.vector( "urbgrad \n  groesse  \t  Nausg  grpgroesse   privat  ")
vars
  ## [1] "urbgrad"    "groesse"    "Nausg"      "grpgroesse"
  ## [6] "privat"

## the next lines are only commented out to fulfill CRAN checks
## vars2 <- miceadds::scan0()
##     female urbgrad  groesse  Nausg    grpgroesse   privat
```

`source.all` R *Utilities: Source all R or* **Rcpp** *Files within a Directory*

Description

The function `source.all` sources all R files within a specified directory and is based on `base::source`.

The function `source.Rcpp.all` sources all **Rcpp** files within a specified directory and is based on `Rcpp::sourceCpp`.

The function `rcpp_create_header_file` creates a cpp header file for a **Rcpp** file.

Usage

```
source.all( path, grepstring="\\.R",  print.source=TRUE, file_sep="__"  )

source.Rcpp.all( path, file_names=NULL, ext="\\.cpp", excl="RcppExports",
   remove_temp_file=FALSE )

rcpp_create_header_file(file_name, pack=NULL, path=getwd() )
```

Arguments

`path`	Path where the files are located
`grepstring`	Which strings should be looked for? `grepstring` can also be a vector.
`print.source`	An optional logical whether the source process printed on the console?
`file_sep`	String at which file name should be split for looking for most recent files
`file_names`	Optional vector of (parts of) file names
`ext`	File extension for **Rcpp** files
`excl`	String indicating which files should be omitted from sourcing
`remove_temp_file`	Logical indicating whether temporary **Rcpp** files should be removed.
`file_name`	File name
`pack`	Optional string for package

Details

For loading header files, the line `// [include_header_file]` has to be included before loading the header file using a line of the form `#include "my_function.h"`.

Examples

```
## Not run:
# define path
path <- "c:/myfiles/"
# source all files containing the string 'Rex'
source.all( path, "Rex" )

## End(Not run)
```

`stats0` *Descriptive Statistics for a Vector or a Data Frame*

Description

Applies descriptive statistics to a vector or a data frame. The function `stats0` is a general function. This function is used for extending the basic descriptive statistics functions from the **base** and **stats** package. The function `prop_miss` computes the proportion of missing data for each variable.

Usage

```
stats0(x, FUN, na.rm=TRUE,...)

max0(x, na.rm=TRUE)
mean0(x, na.rm=TRUE)
min0(x, na.rm=TRUE)
quantile0(x, probs=seq(0, 1, 0.25), na.rm=TRUE)
sd0(x, na.rm=TRUE)
var0(x, na.rm=TRUE)

prop_miss(x)
```

Arguments

`x`	Vector or a data frame
`FUN`	Function which is applied to `x`
`na.rm`	Logical indicating whether missing data should be removed
`probs`	Probabilities
`...`	Further arguments to be passed

Value

A vector or a matrix

See Also

`base::max`, `base::mean`, `base::min`, `stats::quantile`, `stats::sd`, `stats::var`

Examples

```
#############################################################################
# EXAMPLE 1: Descriptive statistics toy datasets
#############################################################################

#--- simulate vector y and data frame dat
set.seed(765)
N <- 25    # number of observations
y <- stats::rnorm(N)
V <- 4    # number of variables
dat <- matrix( stats::rnorm( N*V ), ncol=V )
colnames(dat) <- paste0("V",1:V)

#-- standard deviation
apply( dat, 2, stats::sd )
sd0( dat )
#-- mean
apply( dat, 2, base::mean )
mean0( dat )
#-- quantile
apply( dat, 2, stats::quantile )
quantile0( dat )
#-- minimum and maximum
min0(dat)
max0(dat)

#*** apply functions to missing data
dat1 <- dat
dat1[ cbind( c(2,5),2) ] <- NA

#-- proportion of missing data
prop_miss( dat1 )
#-- MAD statistic
stats0( dat, FUN=stats::mad )
#-- SD
sd0(y)
```

`str_C.expand.grid` R *Utilities: String Paste Combined with* `expand.grid`

Description

String paste combined with `expand.grid`

Usage

```
str_C.expand.grid(xlist, indices=NULL)
```

Arguments

`xlist`	A list of character vectors
`indices`	Optional vector of indices to be permuted in `xlist`

Value

A character vector

Examples

```
#############################################################################
# EXAMPLE 1: Some toy examples
#############################################################################

x1 <- list( c("a","b" ), c("t", "r","v") )
str_C.expand.grid( x1 )
  ##   [1] "at" "bt" "ar" "br" "av" "bv"

x1 <- list( c("a","b" ), paste0("_", 1:4 ), c("t", "r","v") )
str_C.expand.grid( x1, indices=c(2,1,3) )
  ##    [1] "_1at" "_1bt" "_2at" "_2bt" "_3at" "_3bt" "_4at" "_4bt" "_1ar" "_1br"
  ##   [11] "_2ar" "_2br" "_3ar" "_3br" "_4ar" "_4br" "_1av" "_1bv" "_2av" "_2bv"
  ##   [21] "_3av" "_3bv" "_4av" "_4bv"

## Not run:
##***************************************************************************
## The function 'str_C.expand.grid' is currently defined as
function( xlist, indices=NULL )
{
     xeg <- expand.grid( xlist)
     if ( ! is.null(indices) ){    xeg <- xeg[, indices ]}
     apply( xeg, 1, FUN=function(vv){ paste0( vv, collapse="") } )
}
##***************************************************************************

## End(Not run)
```

`subset_datlist` *Subsetting Multiply Imputed Datasets and Nested Multiply Imputed Datasets*

Description

Returns a subsets of multiply imputed datasets or nested multiply imputed datasets. These function allows choosing parts of the imputed datasets using the `index` argument for multiply imputed datasets and `index_between` and `index_within` for nested multiply imputed datasets as well as the application of the `base::subset` S3 method for selecting cases and variables in datasets.

Usage

```
subset_datlist(datlist, subset=TRUE, select=NULL, expr_subset=NULL,
        index=NULL, toclass="datlist")

## S3 method for class 'datlist'
subset(x, subset, select=NULL, expr_subset=NULL,
                     index=NULL, ...)
## S3 method for class 'imputationList'
subset(x, subset, select=NULL, expr_subset=NULL,
                     index=NULL, ...)
## S3 method for class 'mids'
subset(x, subset, select=NULL, expr_subset=NULL,
                     index=NULL, ...)
## S3 method for class 'mids.1chain'
subset(x, subset, select=NULL, expr_subset=NULL,
                     index=NULL, ...)

subset_nested.datlist( datlist, subset=TRUE, select=NULL, expr_subset=NULL,
      index_between=NULL, index_within=NULL, toclass="nested.datlist",
          simplify=FALSE )

## S3 method for class 'nested.datlist'
subset(x, subset, select=NULL, expr_subset=NULL,
                index_between=NULL, index_within=NULL, simplify=FALSE, ...)
## S3 method for class 'NestedImputationList'
subset(x, subset, select=NULL, expr_subset=NULL,
                index_between=NULL, index_within=NULL, simplify=FALSE, ...)
```

Arguments

`datlist`	For `subset_datlist` it is a list of datasets or an object of class `datlist`, `imputationList`, `mids` or `mids.1chain`. For `subset_nested.datlist` it is a list of datasets or an object of class `nested.datlist` or `NestedImputationList`.
`subset`	Logical expression indicating elements or rows to keep, see `base::subset`. `subset` can also be a numeric vector containing row indices.
`select`	Expression indicating columns to select from a data frame
`expr_subset`	Expression indicating a selection criterion for selection rows.
`index`	Vector of indices indicating which of the multiply imputed datasets should be selected.

`toclass`	The object class in which the datasets should be saved.
`index_between`	Index for between nest datasets
`index_within`	Index for within nest datasets
`simplify`	Optional logical indicating whether a nested multiply imputed dataset should be simplified to a multiplied imputed dataset.
`x`	Object containing multiply imputed or nested multiply imputed datasets
`...`	Further arguments to be passed.

Value

For multiply imputed datasets: Object of class `datlist`, `imputationList` or `mids`
For nested multiply imputed datasets: Object of class `nested.datlist` or `NestedImputationList`.

Note

If subsetting is applied to objects of class `mids` (or `mids.1chain`), then informations about the imputation procedure are lost.

See Also

`base::subset`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Subsetting and selection of multiply imputed datasets
#############################################################################

data(data.ma02)

# define original list of datasets
datlist1a <- data.ma02
# object of class datlist
datlist1b <- miceadds::datlist_create(datlist1a)
datlist1b
# object of class imputationList
datlist1c <- mitools::imputationList(datlist1a)
datlist1c
# object of class mids
datlist1d <- miceadds::datlist2mids(datlist1a)
datlist1d

# select some imputed datasets
datlist2a <- miceadds::subset_datlist( datlist1a, index=c(5,3,7) )
datlist2a
# convert to class imputationList
datlist2b <- miceadds::subset_datlist( datlist1a, index=c(5,3,7),
                      toclass="imputationList")
datlist2b
```

```
# convert to class mids
datlist2c <- miceadds::subset_datlist( datlist1a, index=1:3, toclass="mids")
datlist2c

# select some variables
datlist3a <- miceadds::subset_datlist( datlist1a, select=c("idstud", "books")  )
datlist3a
# Because datlist1b is a datlist it is equivalent to
datlist3b <- subset( datlist1b, select=c("idstud", "books")  )
datlist3b
# operating on imputationList class
datlist3c <- miceadds::subset_datlist( datlist1c, select=c("idstud", "books")  )
datlist3c
# operating on mids class
datlist3d <- miceadds::subset_datlist( datlist1d, select=c("idstud", "books")  )
datlist3d
# selection of rows and columns in multiply imputed datasets
datlist4a <- miceadds::subset_datlist( datlist1a, index=1:5,
                 subset=datlist1a[[1]]$idschool < 1067,
                 select=c("idstud", "idschool","hisei") )
datlist4a
# convert to class mids
datlist4b <- miceadds::subset_datlist( datlist1a, index=1:5,
                 subset=datlist1a[[1]]$idschool < 1067,
                 select=c("idstud", "idschool","hisei"), toclass="mids" )
datlist4b
# The same functionality, but now applying to object of class mids datlist1d
datlist4c <- miceadds::subset_datlist( datlist1d, index=1:5,
               subset=datlist1a[[1]]$idschool < 1067,
               select=c("idstud", "idschool","hisei") )
datlist4c

# expression for selecting rows specific in each data frame
# which can result in differently sized datasets (because the variable
# migrant is imputed)
datlist5a <- miceadds::subset_datlist( datlist1a,  expr_subset=expression(migrant==1) )
datlist5a

# select the first 100 cases
datlist6a <- miceadds::subset_datlist( datlist1a, select=c("idstud", "books"),
                       subset=1:100 )
datlist6a

#############################################################################
# EXAMPLE 2: Subsetting and selection of nested multiply imputed datasets
#############################################################################

library(BIFIEsurvey)
data(data.timss4, package="BIFIEsurvey")
dat <- data.timss4

# create object of class 'nested.datlist'
datlist1a <- miceadds::nested.datlist_create( dat )
```

```
# create object of class 'NestedImputationList'
datlist1b <- miceadds::NestedImputationList(dat)

# select some between datasets
datlist2a <- subset_nested.datlist( datlist1a, index_between=c(1,3,4) )
datlist2a
# shorter version
datlist2b <- subset( datlist1a, index_between=c(1,3,4) )
datlist2b
# conversion of a NestedImputationList
datlist2c <- subset( datlist1b, index_between=c(1,3,4))
datlist2c
# select rows and columns
sel_cases <- datlist1a[[1]][[1]]$JKZONE <=42
datlist3a <- subset( datlist1a, subset=sel_cases,
                 select=c("IDSTUD","books", "ASMMAT") )
datlist3a
# remove within nest
datlist4a <- subset( datlist1a, index_within=1 )
datlist4a
# remove within nest and simplify structure
datlist4b <- subset( datlist1a, index_within=1, simplify=TRUE)
datlist4b
datlist4c <- subset( datlist1b, index_within=1, simplify=TRUE)
datlist4c
# remove between nest
datlist5a <- subset( datlist1a, index_between=1, simplify=TRUE)
datlist5a
datlist5b <- subset( datlist1b, index_between=1, simplify=TRUE)
datlist5b

## End(Not run)
```

`sumpreserving.rounding` *Sum Preserving Rounding*

Description

This function implements sum preserving rounding. If the supplied data is a matrix, then the sum of all row entries is preserved.

Usage

```
sumpreserving.rounding(data, digits=0, preserve=TRUE)
```

Arguments

`data`	Vector or data frame
`digits`	Number of digits to be round
`preserve`	Should the sum be preserved?

Examples

```
#############################################################################
# EXAMPLE 1:
#############################################################################

# define example data
data <- c( 1455, 1261, 1067, 970, 582, 97 )
data <- 100 * data / sum(data)

( x1 <- round( data ) )
sum(x1)
(x2 <- miceadds::sumpreserving.rounding( data ) )
sum(x2)

  ##   > ( x1 <- round( data ) )
  ##   [1] 27 23 20 18 11  2
  ##   > sum(x1)
  ##   [1] 101
  ##   > (x2 <- miceadds::sumpreserving.rounding( data ) )
  ##   [1] 27 23 20 18 10  2
  ##   > sum(x2)
  ##   [1] 100

#############################################################################
# EXAMPLE 2:
#############################################################################

# matrix input
data <- rbind( data, data )
( x1 <- round( data ) )
rowSums(x1)
(x2 <- miceadds::sumpreserving.rounding( data ) )
rowSums(x2)

#############################################################################
# EXAMPLE 3:
#############################################################################

x2 <- c( 1.4, 1.4, 1.2 )
round(x2)
sumpreserving.rounding(x2)
  ##   > round(x2)
  ##   [1] 1 1 1
  ##   > miceadds::sumpreserving.rounding(x2)
  ##   [1] 1 2 1
```

`syn.constant` *Synthesizing Method for Fixed Values by Design in* **synthpop**

Description

Defines a synthesizing method for fixed values of a variable by design in the **synthpop** package.

Usage

```
syn.constant(y, x, xp, fixed_values, ...)
```

Arguments

`y`	Original data vector of length n
`x`	Matrix ($n \times p$) of original covariates
`xp`	Matrix ($k \times p$) of synthesised covariates
`fixed_values`	Vector containing fixed values
`...`	Further arguments to be passed

Details

When using the synthesis method `"mice"` in `synthpop::syn`, the function argument has to appear as `rf.fixed_values` (convention in **synthpop**).

Value

A vector of length `k` with synthetic values of `y`.

See Also

`synthpop::syn`, `mice.impute.constant`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: SD2011 | Fixed values for variable sex
#############################################################################

library(synthpop)

#** selection of dataset
data(SD2011, package="synthpop")
vars <- c("sex","age","ls","smoke")
dat  <- SD2011[1:1000, vars]
dat$ls <- as.numeric(dat$ls)

#** default synthesis
imp0 <- synthpop::syn(dat)
pred <- imp0$predictor.matrix
method <- imp0$method

#** constant vector
method["sex"] <- "constant"
```

```
fixed_values <- data.frame( sex=rep(dat$sex[c(1,2)], each=1000) )
imp <- synthpop::syn( dat, method=method, k=2000, m=1,
                rf.fixed_values=fixed_values)
table(imp$syn$sex)

## End(Not run)
```

`syn.formula` *Synthesizing Method for* **synthpop** *Using a Formula Interface*

Description

Defines a synthesizing method for for **synthpop** using a formula interface.

Usage

```
syn.formula(y, x, xp, proper=FALSE, syn_formula, syn_fun, syn_args, ...)
```

Arguments

`y`	Original data vector of length n
`x`	Matrix $(n \times p)$ of original covariates
`xp`	Matrix $(k \times p)$ of synthesised covariates
`proper`	Logical value specifying whether proper synthesis should be conducted.
`syn_formula`	A formula object
`syn_fun`	Synthesizing method in **synthpop** package
`syn_args`	Function arguments of `syn_fun`
`...`	Further arguments to be passed

Details

When using the synthesis method `"mice"` in `synthpop::syn`, the function arguments have to appear as `rf.syn_formula`, `rf.syn_fun` and `rf.syn_args` (convention in **synthpop**).

Value

A vector of length `k` with synthetic values of `y`.

See Also

`synthpop::syn`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: SD2011 | using a formula for defining the regression model
#############################################################################

library(synthpop)

#** selection of dataset
data(SD2011, package="synthpop")
vars <- c("sex","age","ls","smoke")
dat  <- SD2011[1:1000, vars]
dat$ls <- as.numeric(dat$ls)

#** default synthesis
imp0 <- synthpop::syn(dat)
pred <- imp0$predictor.matrix
method <- imp0$method

#** use synthesizing method 'formula'
method["ls"] <- "formula"
syn_fun <- list( ls="normrank" )
syn_args <- list( ls=list( smoothing="density" ) )
syn_formula <- list( ls=~ sex + age + I(age^2) + I(age>50) )

#* synthesize data
imp <- synthpop::syn( dat, method=method, predictor.matrix=pred, k=2000, m=1,
            rf.syn_fun=syn_fun, rf.syn_args=syn_args, rf.syn_formula=syn_formula)
summary(imp)

## End(Not run)
```

`syn.mice` *Using a* **mice** *Imputation Method in the* **synthpop** *Package*

Description

The function allows to use a **mice** imputation method to be used in the `synthpop::syn` function of the **synthpop** package (Nowok, Raab, & Dibben, 2016).

Usage

```
syn.mice(y, x, xp, mice_fun, mice_args, ...)
```

Arguments

y	Original data vector of length n
x	Matrix ($n \times p$) of original covariates
xp	Matrix ($k \times p$) of synthesised covariates

`mice_fun`	Name of imputation method for **mice**
`mice_args`	Optional list of arguments for `mice_fun`, see Examples.
`...`	Further arguments to be passed

Details

When using the synthesis method `"mice"` in `synthpop::syn`, the function arguments have to appear as `rf.mice_fun` and `rf.mice_arg` (convention in **synthpop**).

Value

A vector of length `k` with synthetic values of `y`.

References

Nowok, B., Raab, G., & Dibben, C. (2016). **synthpop**: Bespoke creation of synthetic data in R. *Journal of Statistical Software, 74*(11), 1-26. doi:10.18637/jss.v074.i11

See Also

`synthpop::syn`, `syn_mice`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: SD2011 | Minimal example for using a mice imputation method
#############################################################################

library(synthpop)

#** selection of dataset
data(SD2011, package="synthpop")
vars <- c("sex","age","ls","smoke")
dat  <- SD2011[1:1000, vars]
dat$ls <- as.numeric(dat$ls)
dat$smoke <- 1*(paste(dat$smoke)=="YES")

#** default synthesis
imp0 <- synthpop::syn(dat)
pred <- imp0$predictor.matrix
method <- imp0$method

#** use mice imputation method 'rlm' for variable 'ls'
method[c("ls","smoke")] <- c("mice","mice")
mice_fun <- list( ls="rlm", smoke="pmm")
mice_args <- list( ls=list( trafo=log, antitrafo=exp) )

#* synthesize data
imp <- synthpop::syn( dat, method=method, predictor.matrix=pred, k=2000, m=1,
            rf.mice_fun=mice_fun, rf.mice_args=mice_args)
```

```
summary(imp)

## End(Not run)
```

`syn_da` *Generation of Synthetic Data Utilizing Data Augmentation*

Description

This function generates synthetic data utilizing data augmentation (Jiang et al., 2022; Grund et al., 2022). Continuous and ordinal variables can be handled. The order of the synthesized variables can be defined using the argument `syn_vars`.

Usage

```
syn_da(dat, syn_vars=NULL, fix_vars=NULL, ord_vars=NULL, da_noise=0.5,
   use_pls=TRUE, ncomp=20, exact_regression=TRUE, exact_marginal=TRUE,
   imp_maxit=5)
```

Arguments

`dat`	Original dataset
`syn_vars`	Vector with variable names that should be synthesized
`fix_vars`	Vector with variable names that are held fixed in the synthesis
`ord_vars`	Vector with ordinal variables that are treated as factors when modeled as predictors in the regression model
`da_noise`	Proportion of variance (i.e., unreliability) that is added as noise in data augmentation. The argument can be numeric or a vector, depending on whether it is made variable-specific.
`use_pls`	Logical indicating whether partial least squares (PLS) should be used for dimension reduction
`ncomp`	Number of PLS factors
`exact_regression`	Logical indicating whether residuals are forced to be uncorrelated with predictors in the synthesis model
`exact_marginal`	Logical indicating whether marginal distributions of the variables should be preserved
`imp_maxit`	Number of iterations in the imputation if the original dataset contains missing values

Value

A list with entries

`dat_syn`	generated synthetic data
`dat2`	Data frame containing original and synthetic data
`...`	more entries

References

Grund, S., Luedtke, O., & Robitzsch, A. (2022). Using synthetic data to improve the reproducibility of statistical results in psychological research. *Psychological Methods*. Epub ahead of print. doi:10.1037/met0000526

Jiang, B., Raftery, A. E., Steele, R. J., & Wang, N. (2022). Balancing inferential integrity and disclosure risk via model targeted masking and multiple imputation. *Journal of the American Statistical Association, 117*(537), 52-66. doi:10.1080/01621459.2021.1909597

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Generate synthetic data with item responses and covariates
#############################################################################

data(data.ma09, package="miceadds")
dat <- data.ma09

# fixed variables in synthesis
fix_vars <- c("PV1MATH", "SEX","AGE")
# ordinal variables in synthesis
ord_vars <- c("FISCED", "MISCED", items)
# variables that should be synthesized
syn_vars <- c("HISEI", "FISCED", "MISCED", items)

#-- synthesize data
mod <- miceadds::syn_da( dat=dat0, syn_vars=syn_vars, fix_vars=fix_vars,
            ord_vars=ord_vars, da_noise=0.5, imp_maxit=2, use_pls=TRUE, ncomp=20,
            exact_regression=TRUE, exact_marginal=TRUE)
#- extract synthetic dataset
mod$dat_syn

## End(Not run)
```

`syn_mice` *Constructs Synthetic Dataset with* **mice** *Imputation Methods*

Description

Constructs synthetic dataset with **mice** imputation methods. The functionality is very similar to the functionality of `synthpop::syn` in the **synthpop** package (Nowok, Raab, & Dibben, 2016). Methods defined in **synthpop** are accessible via `mice.impute.synthpop` (see Examples).

Usage

```
syn_mice(data, m=5, k=NULL, syn_check=TRUE, ...)
```

Arguments

`data`	Original data frame
`m`	Number of synthetic datasets
`k`	Number of observations in synthetic data
`syn_check`	Logical indicating whether checks in `synthpop::syn` should be performed.
`...`	Further arguments to be passed, with conventions in `mice::mice`

Value

Object of class synds, see `synthpop::syn`.

References

Nowok, B., Raab, G., & Dibben, C. (2016). **synthpop**: Bespoke creation of synthetic data in R. *Journal of Statistical Software, 74*(11), 1-26. doi:10.18637/jss.v074.i11

See Also

`mice::mice`, `synthpop::syn`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Synthesization of SD2011 using mice functionality
#############################################################################

library(synthpop)

#** selection of dataset
data(SD2011, package="synthpop")
vars <- c("sex","age","ls","smoke")
dat  <- SD2011[1:1000, vars]
dat$ls <- as.numeric(dat$ls)

#** default synthesis
imp0 <- synthpop::syn(dat)
pred0 <- imp0$predictor.matrix
method0 <- imp0$method

#* define imputation methods
method <- c(sex="synthpop", age="synthpop", ls="synthpop", smoke="logreg")
# only for smoke, an original mice imputation method is used

#- define synthpop functions
synthpop_fun <- list(sex="constant", age="constant", ls="cart")

#- arguments for 'syn.cart' method
synthpop_args <- list(ls=list(smoothing="density"))
```

```
#- fixed values for 'syn.constant' method
fixed_values <- dat[,1:2]

#- do synthesization
imp <- miceadds::syn_mice(dat, m=1, synthpop_fun=synthpop_fun, method=method,
        pedictorMatrix=pred0, rf.fixed_values=fixed_values, synthpop_args=synthpop_args)
summary(imp)

## End(Not run)
```

`systime` R *Utilities: Various Strings Representing System Time*

Description

This function generates system time strings in several formats.

Usage

```
systime()
```

Value

A vector with entries of system time (see Examples).

Examples

```
#############################################################################
# EXAMPLE 1: Output of systime
#############################################################################

systime()
  ##
  ##  > miceadds::systime()
  ##  [1] "2016-02-29 10:25:44"
  ##  [2] "2016-02-29"
  ##  [3] "20160229"
  ##  [4] "2016-02-29_1025"
  ##  [5] "2016-02-29_1000"
  ##  [6] "20160229_102544"
  ##  [7] "20160229102544"
  ##  [8] "IPNERZW-C014_20160229102544"
```

`tw.imputation` *Two-Way Imputation*

Description

Two-way imputation using the simple method of Sijtsma and van der Ark (2003) and the MCMC based imputation of van Ginkel, van der Ark, Sijtsma and Vermunt (2007).

Usage

```
tw.imputation(data, integer=FALSE)

tw.mcmc.imputation(data, iter=100, integer=FALSE)
```

Arguments

`data`	Matrix of item responses corresponding to a scale
`integer`	A logical indicating whether imputed values should be integers. The default is `FALSE`.
`iter`	Number of iterations

Details

For persons p and items i, the two-way imputation is conducted by posing a linear model of tau-equivalent measurements:

$$X_{pi} = \theta_p + b_i + \varepsilon_{ij}$$

If the score X_{pi} is missing then it is imputed by

$$\hat{X}_{pi} = \tilde{X}_p + b_i$$

where $\tilde{X}_p$ is the person mean of person p of the remaining items with observed responses.

The two-way imputation can also be seen as a scaling procedure to obtain a scale score which takes different item means into account.

Value

A matrix with original and imputed values

References

Sijtsma, K., & Van der Ark, L. A. (2003). Investigation and treatment of missing item scores in test and questionnaire data. *Multivariate Behavioral Research, 38*(4), 505-528. doi:10.1207/s15327906mbr3804_4

Van Ginkel, J. R., Van der Ark, A., Sijtsma, K., & Vermunt, J. K. (2007). Two-way imputation: A Bayesian method for estimating missing scores in tests and questionnaires, and an accurate approximation. *Computational Statistics & Data Analysis, 51*(8), 4013-4027. doi:10.1016/j.csda.2006.12.022

See Also

The two-way imputation method is also implemented in the `TestDataImputation::Twoway` function of the **TestDataImputation** package.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Two-way imputation data.internet
#############################################################################

data(data.internet)
data <- data.internet

#***
# Model 1: Two-way imputation method of Sijtsma and van der Ark (2003)
set.seed(765)
dat.imp <- miceadds::tw.imputation( data )
dat.imp[ 278:281,]
  ##       IN9     IN10    IN11     IN12
  ##   278   5 4.829006 5.00000 4.941611
  ##   279   5 4.000000 4.78979 4.000000
  ##   280   7 4.000000 7.00000 7.000000
  ##   281   4 3.000000 5.00000 5.000000

#***
# Model 2: Two-way imputation method using MCMC
dat.imp <- miceadds::tw.mcmc.imputation( data, iter=3)
dat.imp[ 278:281,]
  ##       IN9     IN10     IN11     IN12
  ##   278   5 6.089222 5.000000 3.017244
  ##   279   5 4.000000 5.063547 4.000000
  ##   280   7 4.000000 7.000000 7.000000
  ##   281   4 3.000000 5.000000 5.000000

## End(Not run)
```

`VariableNames2String` *Stringing Variable Names with Line Breaks*

Description

Stringing variable names with line breaks.

Usage

```
VariableNames2String(vars, breaks=80, sep=" ")
```

Arguments

`vars`	Vector with variable names
`breaks`	Numeric value for line break of variable string
`sep`	Separator

Value

String with line breaks

Examples

```
#############################################################################
# EXAMPLE 1: Toy example
#############################################################################

data(data.ma01)
# extract variable names
vars <- colnames(data.ma01)
# convert into a long string with line breaks at column 25
vars2 <- miceadds::VariableNames2String(vars, breaks=25)
vars
  ##   [1] "idstud"   "idschool" "studwgt"  "math"     "read"     "migrant"
  ##   [7] "books"    "hisei"    "paredu"   "female"   "urban"
vars2
  ##  idstud idschool studwgt
  ##  math read migrant books
  ##  hisei paredu female
  ##  urban
```

`visitSequence.determine`

Automatic Determination of a Visit Sequence in `mice`

Description

This function automatically determines a visit sequence for a specified model in `mice::mice` when passive variables are defined as imputation methods. Note that redundant visits could be computed and a user should check the plausibility of the result.

Usage

```
visitSequence.determine(impMethod, vis, data, maxit=10)
```

Arguments

`impMethod`	Vector with imputation methods
`vis`	Initial vector of visit sequence
`data`	Data frame to be used for multiple imputations
`maxit`	Maximum number of iteration for computation of the updated visit sequence

Value

Updated vector of the visit sequence

See Also

Used in the `mice::mice` function as an argument. The function `mice::make.visitSequence` creates a visit sequence.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Visit sequence for a small imputation model
#############################################################################

data( data.smallscale )
# select a small number of variables
dat <- data.smallscale[, paste0("v",1:4) ]
V <- ncol(dat)

# define initial vector of imputation methods
impMethod <- rep("norm", V)
names(impMethod) <- colnames(dat)
# define variable names and imputation method for passive variables in a data frame
dfr.impMeth <- data.frame( "variable"=NA,
                  "impMethod"=NA )
dfr.impMeth[1,] <- c("v1_v1", "~ I(v1^2)" )
dfr.impMeth[2,] <- c("v2_v4", "~ I(v2*v4)" )
dfr.impMeth[3,] <- c("v4log", "~ I( log(abs(v4)))" )
dfr.impMeth[4,] <- c("v12", "~ I( v1 + v2 + 3*v1_v1 - v2_v4 )" )
# add variables to dataset and imputation methods
VV <- nrow(dfr.impMeth)
for (vv in 1:VV){
    impMethod[ dfr.impMeth[vv,1] ] <- dfr.impMeth[vv,2]
    dat[, dfr.impMeth[vv,1] ] <- NA
}

# run empty imputation model to obtain initial vector of visit sequence
imp0 <- mice::mice( dat, m=1, method=impMethod, maxit=0 )
imp0$vis

# update visit sequence
vis1 <- miceadds::visitSequence.determine( impMethod=impMethod, vis=imp0$vis, data=dat)

# imputation with updated visit sequence
imp <- mice::mice( dat, m=1, method=impMethod, visitSequence=vis1, maxit=2)

## End(Not run)
```

`with.miceadds` *Evaluates an Expression for (Nested) Multiply Imputed Datasets*

Description

Evaluates an expression for (nested) multiply imputed datasets. These functions extend the following functions: `mice::with.mids`, `base::with`, `base::within.data.frame`, `mitools::with.imputationList`.

The `withPool` functions try to pool estimates (by simple averaging) obtained by `with` or a list of results of imputed datasets.

Usage

```
## S3 method for class 'mids.1chain'
with(data, expr, ...)
## S3 method for class 'datlist'
with(data, expr, fun, ...)

## S3 method for class 'mids.nmi'
with(data, expr, ...)
## S3 method for class 'nested.datlist'
with(data, expr, fun, ...)
## S3 method for class 'NestedImputationList'
with(data, expr, fun, ...)

## S3 method for class 'datlist'
within(data, expr, ...)
## S3 method for class 'imputationList'
within(data, expr, ...)

## S3 method for class 'nested.datlist'
within(data, expr, ...)
## S3 method for class 'NestedImputationList'
within(data, expr, ...)

withPool_MI(x, ...)

withPool_NMI(x, ...)

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

Arguments

`data`	Object of class `mids.1chain`, `mids.nmi`, `imputationList` or `NestedImputationList`
`expr`	Expression with a formula object.
`fun`	A function taking a data frame argument

`...`	Additional parameters to be passed to `expr`.
`object`	Object of class `mira.nmi`.
`x`	List with vectors or matrices as results of an analysis for (nested) multiply imputed datasets.

Value

`with.mids.1chain`: List of class `mira`.

`with.mids.nmi`: List of class `mira.nmi`.

`with.datlist`: List of class `imputationResultList`.

`with.NestedImputationList` or `with.nested.datlist`: List of class `NestedImputationResultList`.

`within.imputationList`: List of class `imputationList`.

`within.NestedImputationList`: List of class `NestedImputationList`.

`withPool_MI` or `withPool_NMI`: Vector or matrix with pooled estimates

Author(s)

Slightly modified code of `mice::with.mids`, `mice::summary.mira`, `base::within.data.frame`

See Also

See the corresponding functionality in **base**, **mice**, **mitools** and **mitml** packages:
`mice::with.mids`, `mitools::with.imputationList`, `mitml::with.mitml.list`, `base::with`

`base::within.data.frame`, `mitml::within.mitml.list`,

`mice::summary.mira`,

Imputation functions in **miceadds**: `mice.1chain`, `mice.nmi`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: One chain nhanes data | application of 'with' and 'within'
#############################################################################

library(mice)
data(nhanes, package="mice")
set.seed(9090)

# nhanes data in one chain
imp <- miceadds::mice.1chain( nhanes, burnin=5, iter=40, Nimp=4 )
# apply linear regression
res <- with( imp, expr=stats::lm( hyp ~ age + bmi  ) )
summary(res)
# pool results
summary( mice::pool(res))

# calculate some descriptive statistics
```

```
res2 <- with( imp, expr=c("M1"=mean(hyp), "SD_age"=stats::sd(age) ) )
# pool estimates
withPool_MI(res2)

# with method for datlist
imp1 <- miceadds::datlist_create(imp)
res2b <- with( imp1, fun=function(data){
                    dfr <- data.frame("M"=colMeans(data),
                             "Q5"=apply( data, 2, stats::quantile, .05 ),
                             "Q95"=apply( data, 2, stats::quantile, .95 ) )
                    return(dfr)
                        } )
withPool_MI(res2b)

# convert mids object into an object of class imputationList
datlist <- miceadds::mids2datlist( imp )
datlist <- mitools::imputationList(datlist)

# define formulas for modification of the data frames in imputationList object
datlist2 <- within( datlist, {
                     age.D3 <- 1*(age==3)
                     hyp_chl <- hyp * chl
                        } )
# look at modified dataset
head( datlist2$imputations[[1]] )

# convert into a datlist
datlist2b <- miceadds::datlist_create( datlist2 )

# apply linear model using expression
mod1a <- with( datlist2, expr=stats::lm( hyp ~ age.D3 ) )
# do the same but now with a function argument
mod1b <- with( datlist2, fun=function(data){
                    stats::lm( data$hyp ~ data$age.D3 )
                        } )
# apply the same model for object datlist2b
mod2a <- with( datlist2b, expr=lm( hyp ~ age.D3 ) )
mod2b <- with( datlist2b, fun=function(data){
                    stats::lm( data$hyp ~ data$age.D3 )
                        } )

mitools::MIcombine(mod1a)
mitools::MIcombine(mod1b)
mitools::MIcombine(mod2a)
mitools::MIcombine(mod2b)

#############################################################################
# EXAMPLE 2: Nested multiple imputation and application of with/within methods
#############################################################################

library(BIFIEsurvey)
data(data.timss2, package="BIFIEsurvey" )
datlist <- data.timss2
```

```
# remove first four variables
M <- length(datlist)
for (ll in 1:M){
    datlist[[ll]] <- datlist[[ll]][, -c(1:4) ]
                }

# nested multiple imputation using mice
imp1 <- miceadds::mice.nmi( datlist,  m=4, maxit=3 )
summary(imp1)
# apply linear model and use summary method for all analyses of imputed datasets
res1 <- with( imp1, stats::lm( ASMMAT ~ migrant + female ) )
summary(res1)

# convert mids.nmi object into an object of class NestedImputationList
datlist1 <- miceadds::mids2datlist( imp1 )
datlist1 <- miceadds::NestedImputationList( datlist1 )
# convert into nested.datlist object
datlist1b <- miceadds::nested.datlist_create(datlist1)

# use with function
res1b <- with( datlist1, stats::glm( ASMMAT ~ migrant + female ) )
# apply for nested.datlist
res1c <- with( datlist1b, stats::glm( ASMMAT ~ migrant + female ) )

# use within function for data transformations
datlist2 <- within( datlist1, {
                highsc <- 1*(ASSSCI > 600)
                books_dum <- 1*(books>=3)
                rm(scsci)   # remove variable scsci
                    } )

# include random number in each dataset
N <- attr( datlist1b, "nobs")
datlist3 <- within( datlist1b, {
                rn <- stats::runif( N, 0, .5 )
                    } )

#-- some applications of withPool_NMI
# mean and SD
res3a <- with( imp1, c( "m1"=mean(ASMMAT), "sd1"=stats::sd(ASMMAT) ) )
withPool_NMI(res3a)
# quantiles
vars <- c("ASMMAT", "lang", "scsci")
res3b <- with( datlist1b, fun=function(data){
                dat <- data[,vars]
                res0 <- sapply( vars, FUN=function(vv){
                    stats::quantile( dat[,vv], probs=c(.25, .50, .75) )
                                    } )
                t(res0)
                    } )
withPool_NMI(res3b)
```

```
## End(Not run)
```

---

write.datlist *Write a List of Multiply Imputed Datasets*

Description

Writes a list of multiply imputed datasets.

Usage

```
write.datlist(datlist, name, include.varnames=TRUE, type="csv2",
     separate=TRUE, Mplus=FALSE, round=NULL, Rdata=TRUE,
     subdir=TRUE, ...)
```

Arguments

`datlist`	List of imputed datasets. Can also be an object of class `mids`, `mids.1chain` or `imputationList`
`name`	Name of files to be saved
`include.varnames`	Logical indicating whether variables should be saved
`type`	File type of datasets to be saved, see `save.data`.
`separate`	Logical indicating whether imputed datasets should be written in separate files.
`Mplus`	Logical indicating whether files should be written for usage in Mplus software
`round`	Number of digits to round after decimal. The default is no rounding.
`Rdata`	Logical indicating whether `datlist` should also be saved in R binary format.
`subdir`	Logical indicating whether results should be written into a subdirectory.
`...`	Further arguments to be passed to `save.data`.

See Also

See also `mice::mids2mplus`, `mice::mids2spss` and `write.mice.imputation` for writing objects of class `mids`.

See also `Amelia::write.amelia` for writing imputed datasets in **Amelia**.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Write data list imputed in mice
#############################################################################

data(data.ma01)
dat <- as.matrix(data.ma01)
```

```
# start with empty imputation
imp0 <- mice::mice( dat, maxit=0)

# modify predictor matrix
predM <- imp0$predictorMatrix
predM[, c("idschool", "idstud" ) ] <- 0
# modify imputation method
impMeth <- imp0$method
impMeth[ impMeth=="pmm" ] <- "norm"

# do imputations in mice
imp <- mice::mice( dat, predictorMatrix=predM, method=impMeth, m=3, maxit=4 )

# write imputed data in format "csv2" and round after 4 digits
write.datlist( datlist=imp, name="mice_imp_csv2", round=4 )
# write imputed data in R binary format
write.datlist( datlist=imp, name="mice_imp_Rdata", type="Rdata")
# write data for Mplus usage
write.datlist( datlist=imp, name="mice_imp_Mplus", Mplus=TRUE, round=5)

## End(Not run)
```

`write.fwf2` *Reading and Writing Files in Fixed Width Format*

Description

Reads and writes files in fixed width format. The functions are written for being more efficient than `utils::read.fwf`.

Usage

```
write.fwf2(dat, format.full, format.round, file)

read.fwf2( file, format.full, variables=NULL)
```

Arguments

`dat`	Data frame (or matrix). Variables can be numeric or strings. However, string length of string variables are not allowed to be larger than what is specified in `format.full`.
`format.full`	Vector with fixed width variable lengths
`format.round`	Vector with digits after decimals
`file`	File name
`variables`	Optional vector with variable names

See Also

`utils::read.fwf`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Write and read a file in fixed width format
#############################################################################

# set working directory
path <- "P:/ARb/temp"
setwd(path)

# define a data frame
set.seed(9876)
dat <- data.frame( "x"=seq( 1, 21, len=5), "y"=stats::runif( 5 ),
            "z"=stats::rnorm( 5 ) )

# save data frame in fixed width format
format.full <- c(6, 6, 8 )
format.round <- c( 0, 2, 3 )
write.fwf2( dat, format.full=format.full, format.round=format.round,
                file="testdata" )

# read the data
dat1 <- miceadds::read.fwf2( file="testdata.dat", format.full=c(6,6,8),
               variables=c("x","y","z") )
# check differences between data frames
dat - dat1

#############################################################################
# EXAMPLE 2: Write datasets containing some string variables in fwf format
#############################################################################

n <- 5
dat <- data.frame( "x"=stats::runif(n, 0, 9 ), "y"=LETTERS[1:n] )
write.fwf2(dat, format.full=c(4,2), format.round=c(2,0),  file="testdata")

## End(Not run)
```

`write.mice.imputation` *Export Multiply Imputed Datasets from a* `mids` *Object*

Description

Exports multiply imputed datasets and information about the imputation. Objects of class `mids` (generated by `mice::mice`) and `mids.1chain` (generated by `mice.1chain`) are supported.

Usage

```
write.mice.imputation(mi.res, name, include.varnames=TRUE,
      long=TRUE, mids2spss=TRUE, spss.dec=",", dattype=NULL)
```

Arguments

`mi.res`	Object of class `mids` or `mids.1chain`
`name`	Name of created subdirectory and datasets
`include.varnames`	An optional logical indicating whether variable names should be included in the imputed dataset. The default is `TRUE`.
`long`	An optional logical indicating whether the dataset should also be saved in a long format?
`mids2spss`	An optional logical indicating whether a syntax for reading imputed datasets in SPSS should be included
`spss.dec`	SPSS decimal separator (can be `","` or `"."`)
`dattype`	Format of the saved dataset: `csv` or `csv2`

Value

Several files are saved using `impxxx` (the `name`) as the prefix:

`impxxx.Rdata`	Saved object of class `mids`
`impxxx__DATALIST.Rdata`	Saved object of a list containing multiply imputed datasets
`impxxx__IMP_LIST`	File with list of multiply imputed datasets
`impxxx__IMP_SUMMARY`	Summary file of the imputation
`impxxx__IMPDATA_nn`	Imputed datasets `nn`
`impxxx__IMPMETHOD`	File containing imputation methods
`impxxx__LEGENDE`	File with variable names of the dataset
`impxxx__LONG`	Imputed datasets in long format
`impxxx__PREDICTORMATRIX`	File containing the predictor matrix
`impxxx__SPSS.sps`	SPSS syntax for reading the corresponding `txt` file into SPSS format.

See Also

See also `mice::mids2mplus` and `mice::mids2spss`

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Imputation of nhanes data and write imputed datasets on disk
#############################################################################

data(nhanes,package="mice")

#**********
# Model 1: Imputation using mice
imp1 <- mice::mice( nhanes, m=3, maxit=5 )
# write results
write.mice.imputation(mi.res=imp1, name="mice_imp1" )

#**********
# Model 2: Imputation using mice.1chain
imp2 <- miceadds::mice.1chain( nhanes, burnin=10, iter=20, Nimp=4 )
# write results
write.mice.imputation(mi.res=imp2, name="mice_imp2" )

## End(Not run)
```

`write.pspp` *Writing a Data Frame into SPSS Format Using PSPP Software*

Description

Writes a data frame into SPSS format using the *PSPP* software. To use this function, download and install PSPP at first: http://www.gnu.org/software/pspp/pspp.html.

Usage

```
write.pspp(data, datafile, pspp.path, decmax=6,
   as.factors=TRUE, use.bat=FALSE)
```

Arguments

`data`	Data frame
`datafile`	Name of the output file (without file ending)
`pspp.path`	Path where the PSPP executable is located, e.g. `"C:/Program Files (x86)/PSPP/bin/"`
`decmax`	Maximum number of digits after decimal
`as.factors`	A logical indicating whether all factors and string entries should be treated as factors in the output file.
`use.bat`	A logical indicating whether PSPP executed via a batch file in the DOS mode (`TRUE`) or directly invoked via the `system` command from within R (`FALSE`).

Value

A dataset in *sav* format (SPSS format).

Author(s)

The code was adapted from https://stat.ethz.ch/pipermail/r-help/2006-January/085941.html

See Also

See also `foreign::write.foreign`.

For writing sav files see also `haven::write_sav` and `sjlabelled::write_spss`.

For convenient viewing *sav* files we recommend the freeware program *ViewSav*, see http://www.asselberghs.dds.nl/stuff.htm.

Examples

```
## Not run:
#############################################################################
# EXAMPLE 1: Write a data frame into SPSS format
#############################################################################

#****
# (1) define data frame
data <- data.frame( "pid"=1000+1:5, "height"=round(rnorm( 5 ),4),
                "y"=10*c(1,1,1,2,2), "r2"=round( rnorm(5),2),
                "land"=as.factor( c( rep("A",1), rep("B", 4 ) ) ) )
#****
# (2) define variable labels
v1 <- rep( "", ncol(data) )
names(v1) <-  colnames(data)
attr( data, "variable.labels" ) <- v1
attr(data,"variable.labels")["pid"] <- "Person ID"
attr(data,"variable.labels")["height"] <- "Height of a person"
attr(data,"variable.labels")["y"] <- "Gender"

#****
# (3) define some value labels
v1 <- c(10,20)
names(v1) <- c("male", "female" )
attr( data$y, "value.labels" ) <- v1

#****
# (4a) run PSPP to produce a sav file
write.pspp( data, datafile="example_data1",
        pspp.path="C:/Program Files (x86)/PSPP/bin/" )

#****
# (4b) produce strings instead of factors
write.pspp( data, datafile="example_data2",
        pspp.path="C:/Program Files (x86)/PSPP/bin/", as.factors=FALSE )
```

```
#****
# write sav file using haven package
library(haven)
haven::write_sav( data, "example_data1a.sav" )

#****
# write sav file using sjlabelled package
library(sjlabelled)
data <- sjlabelled::set_label( data, attr(data, "variable.labels") )
sjlabelled::write_spss( data, "example_data1b.sav" )

## End(Not run)
```

Index