Package 'mitml'

October 19, 2015

Type Package

Title Tools for Multiple Imputation in Multilevel Modeling

Version 0.2-4

Date 2015-10-19

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

Maintainer Simon Grund <grund@ipn.uni-kiel.de>

Imports pan, haven, grDevices, graphics, stats, utils

Suggests lme4, nlme, mice

LazyData true

LazyLoad true

Description Provides tools for multiple imputation of missing data in multilevel modeling. Includes a user-friendly interface to the 'pan' package, and several functions for visualization, data management and the analysis of multiply imputed data sets.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2015-10-19 23:25:28

R topics documented:

`mitml-package` *mitml: Tools for multiple imputation in multilevel modeling*

Description

Provides tools for multiple imputation of missing data in multilevel modeling. This package includes a user-friendly interface to the pan algorithm, and several functions for visualizing, managing and the analyzing multiply imputed data sets.

The main interface to pan is the function `panImpute`, which allows specification of a wide array of imputation models. Imputations are efficiently stored in objects of class `mitml`. In order to obtain the completed (i.e., imputed) data sets from those objects, `mitmlComplete` is used, which produces a list of imputed data sets (class `mitml.list`) that can be used in further analyses.

Several functions have been added to allow for convenient analysis of multiply imputed data sets. The functions `with` and `within` may be used for manipulating the data sets and for fitting statistical models. Parameter estimates can be extracted using `testEstimates`. Single- and multi-parameter hypotheses tests can be performed using `testConstraints` and `testModels`.

Data sets can be imported and exported from or to different statistical software packages. Currently, `write.mitmlMplus`, `write.mitmlSAV` and `write.mitmlSPSS` export data sets to M*plus* and SPSS, while other functions import previously imputed data from within R.

Finally, the package supplies tools for summarizing and visualizing imputation models, which is useful for the assessment of convergence and the reporting of results.

Author(s)

Authors: Simon Grund, Alexander Robitzsch, Oliver Luedtke

Maintainer: Simon Grund <grund@ipn.uni-kiel.de>

`as.mitml.list` *Convert a list of data sets to* `mitml.list`

Description

This function adds a `mitml.list` class attribute to a list of data frames. The resulting object can be used with the other functions of this package.

Usage

```
as.mitml.list(x)
```

Arguments

x	A list of data frames.

Value

The original list with an additional class attribute `mitml.list`. The list entries are converted to `data.frame` if necessary, in which case a note is printed.

Author(s)

Simon Grund

See Also

`is.mitml.list`

Examples

```
# data frame with 'imputation' indicator
dat <- data.frame(imputation=rep(1:10,each=20), x=rnorm(200))

# split into a list and convert to 'mitml.list'
l <- split(dat, dat$imputation)
l <- as.mitml.list(l)

is.mitml.list(l)
# TRUE
```

clusterMeans *Calculate cluster means*

Description

Calculates the mean of a given variable within each cluster, possibly conditioning on an additional grouping variable.

Usage

```
clusterMeans(x, cluster, adj=FALSE, group=NULL)
```

Arguments

x	A numeric vector for which to calculate the cluster means. Can also be supplied as a character string denoting a variable in the current environment (see details).
cluster	A numeric vector or a factor denoting the cluster membership of each unit in x. Can also be supplied as a character string (see details).
adj	Logical flag indicating if person-adjusted group means should be calculated. The cluster mean is then calculated for each unit by excluding that unit from calculating the cluster mean. Default is to FALSE.
group	(optional) An grouping factor or a variable that can be interpreted as such. If specified, then cluster means are calculated conditionally on the grouping variable, that is, separately within sub-groups. Can also be supplied as a character string (see details).

Details

This function calculates the mean of a variable within each level of a cluster variable. Any NA are omitted during calculation.

The three main arguments of the function can also be supplied as (single) character strings, denoting the name of the respective variables in the current environment. This is especially useful for calculating several cluster means simultaneously, for example using within.mitml.list (see also Example 2 below).

Value

Returns a numeric vector with the same length as x containing the cluster mean for all units.

Author(s)

Simon Grund, Alexander Robitzsch

See Also

with.mitml.list, within.mitml.list

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

implist <- mitmlComplete(imp, print=1:5)

# * Example 1: single cluster means

# calculate cluster means (for each data set)
with(implist, clusterMeans(ReadAchiev, ID))

# ... person-adjusted cluster means
with(implist, clusterMeans(ReadAchiev, ID, adj=TRUE))

# ... groupwise cluster means
with(implist, clusterMeans(ReadAchiev, ID, group=Sex))

# * Example 2: automated cluster means using 'for' and 'assign'

# calculate multiple cluster means within multiply imputed data sets
within(implist,{
  vars <- c("ReadAchiev","MathAchiev","CognAbility")
  for(i in vars) assign(paste(i,"Mean",sep="."), clusterMeans(i,ID))
  rm(i,vars)
})
```

---

is.mitml.list *Check if an object is of class* mitml.list

Description

This function checks if its argument is a list of class mitml.list.

Usage

```
is.mitml.list(x)
```

Arguments

x	An R object.

Value

TRUE or FALSE. In addition, a warning message is displayed if the contents of x do not appear to be data frames.

Author(s)

Simon Grund

See Also

as.mitml.list

Examples

```
l <- list(data.frame(x=rnorm(20)))
l <- as.mitml.list(l)
is.mitml.list(l)
# TRUE

l <- as.list(1:10)
is.mitml.list(l)
# FALSE

class(l) <- "mitml.list"
is.mitml.list(l)
# TRUE, with a warning
```

---

mids2mitml.list *Convert objects of class* mids *to* mitml.list

---

Description

This function converts a mids class object (from the mice package) to mitml.list, such that it may be managed and analyzed using the functions of this package.

Usage

```
mids2mitml.list(x)
```

Arguments

x	An object of class mids as produced by mice (see the mice package).

Value

A list of imputed data sets with an additional class attribute mitml.list.

Author(s)

Simon Grund

See Also

mitmlComplete

Examples

```
data(studentratings)

# imputation using mice
require(mice)
imp <- mice(studentratings)

implist <- mids2mitml.list(imp)
```

---

mitmlComplete *Extract imputed data sets produced by* panImpute

---

Description

This function extracts imputed data sets from mitml class objects as produced by panImpute.

Usage

```
mitmlComplete(x, print=0, force.list=FALSE)
```

Arguments

x	An object of class mitml as produced by panImpute.
print	Either an integer vector, "list", or "all" denoting which data sets to extract. If set to "all" or "list", then all imputed data sets will be returned as a list. Negative values and zero will return the original (incomplete) data set. Default is to zero.
force.list	(optional) Logical flag indicating if single data sets should be enclosed in a list. Default is to FALSE.

Value

Single data sets are returned as a data frame unless force.list=TRUE. If several data sets are extracted, the result is always a list of data sets with an additional class attribute mitml.list.

Author(s)

Simon Grund

See Also

panImpute

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# extract original (incomplete) data set
mitmlComplete(imp, print=0)

# extract single imputed data set (first one, returned as mitml.list)
mitmlComplete(imp, print=1, force.list=TRUE)

# extract all imputed data sets at once
implist <- mitmlComplete(imp, print=1:5)

## Not run:
# ... alternatives with same results
implist <- mitmlComplete(imp, print="all")
implist <- mitmlComplete(imp, print="list")

## End(Not run)
```

---

panImpute *Impute multilevel missing data using* pan

Description

This function provides a user-friendly interface to the pan package. Imputations can be generated using type or formula, which offer different options for model specification.

Usage

```
panImpute(data, type, formula, n.burn=5000, n.iter=100, m=10, group=NULL,
  prior=NULL, seed=NULL, save.pred=FALSE, silent=FALSE)
```

Arguments

data	A data frame containing incomplete and auxiliary variables, the cluster indicator variable, and any other variables that should be present in the imputed datasets.
type	An integer vector specifying the role of each variable in the imputation model (see details).

formula	A formula specifying the role of each variable in the imputation model. The basic model is constructed by model.matrix, thus allowing to include derived variables in the imputation model using I() (see details and examples).
n.burn	The number of burn-in iterations before any imputations are drawn. Default is to 5,000.
n.iter	The number of iterations between imputations. Default is to 100.
m	The number of imputed data sets to generate.
group	(optional) A character string denoting the name of an additional grouping variable to be used with the formula argument. When specified, the imputation model is run separately within each of these groups.
prior	(optional) A list with components a, Binv, c, and Dinv for specifying prior distributions for the covariance matrix of random effects and the covariance matrix of residuals (see details). Default is to using least-informative priors.
seed	(optional) An integer value initializing pan's random number generator for reproducible results. Default is to using random seeds.
save.pred	(optional) Logical flag indicating if variables derived using formula should be included in the imputed data sets. Default is to FALSE.
silent	(optional) Logical flag indicating if console output should be suppressed. Default is to FALSE.

Details

This function serves as the main interface to the pan algorithm. The imputation model can be specified using either the type or the formula argument.

The type interface is designed to provide quick-and-easy imputations using pan. The type argument must be an integer vector denoting the role of each variable in the imputation model:

- 1: target variables containing missing data
- 2: predictors with fixed effect on all targets (completely observed)
- 3: predictors with random effect on all targets (completely observed)
- -1: grouping variable within which the imputation is run separately
- -2: cluster indicator variable
- 0: variables not featured in the model

At least one target variable and the cluster indicator must be specified. The intercept is automatically included both as a fixed and random effect. If a variable of type -1 is found, then imputations are performed separately within each level of that variable. This is useful if the cluster variable (e.g., schools) is contained in an even larger grouping variable for which imputations models are not deemed comparable (e.g., federal states, educational systems).

The formula argument is intended as more flexible and feature-rich interface to pan. Specifying the formula argument is similar to specifying other formulae in R. Given below is a list of operators that panImpute currently understands:

- ~: separates the target (left-hand) and predictor (right-hand) side of the model
- +: adds target or predictor variables to the model

- `*`: adds an interaction term of two or more predictors
- `|`: denotes cluster-specific random effects and specifies the cluster indicator (i.e., `1|ID`)
- `I()`: defines functions to be interpreted by `model.matrix`

Predictors are allowed to have fixed effects, random effects or both on all target variables. The intercept is automatically included both as a fixed and random effect, but it can be constrained if necessary (see examples). Note that, when specifying random effects other than the intercept, these will *not* be automatically added as fixed effects and must be included explicitly. Any predictors defined by `I()` will be used for imputation but not included in the data set unless `save.pred=TRUE`.

In order to run separate imputation models for an additional grouping variable, the `group` argument may be used. The variable name must be specified without quotation marks and must be present in the data set.

As a default prior, `panImpute` uses least informative inverse-Wishart priors for the covariance matrices of random effects and of residuals, that is, with minimum degrees of freedom (largest dispersion) and identity matrices for scale. For better control, the `prior` argument may be used for specifying alternative prior distributions. These must be supplied as a list containing the following components:

- `a`: degrees of freedom for the residual covariance matrix
- `Binv`: scale matrix for the residual covariance matrix
- `c`: degrees of freedom for the covariance matrix of random effects
- `Dinv`: scale matrix for the covariance matrix of random effects

A sensible choice for a diffuse non-default prior is to set the degrees of freedom to the lowest value possible, and the scale matrices according to a prior guess of the corresponding covariance matrices (see Schafer & Yucel, 2002).

Value

Returns an object of class `mitml`. A `mitml` class object is a list, each containing the following components:

`data` The original (incomplete) data set that has been sorted according to the cluster variable and (if given) the grouping variable. An attribute `"sort"` contains the original row order. An attribute `"group"` contains the optional grouping variable.

`replacement.mat` A matrix containing the multiple replacements (i.e., imputations) for each missing value. The replacement matrix contains one row for each missing value and one one column for each imputed data set.

`index.mat` A matrix containing the row and column index for each missing value. The index matrix is used to *link* the missing values in the data set with their corresponding rows in the replacement matrix.

`call` The matched function call.

`model` A list containing the names of the cluster variable, the target variables, and the predictor variables with fixed and random effects, respectively.

`prior` The prior parameters used in the imputation model.

iter	A list containing the number of burn-in iterations, the number of iterations between imputations, and the number of imputed data sets.
par.burnin	A multi-dimensional array containing the parameters of the imputation model from the burn-in phase.
par.imputation	A multi-dimensional array containing the parameters of the imputation model from the imputation phase.

Note

For objects of class mitml, methods for the generic functions print, summary and plot have been defined. mitmlComplete is used to extract the imputed data sets.

Author(s)

Simon Grund, Alexander Robitzsch, Oliver Luedtke

References

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

See Also

mitmlComplete, summary.mitml, plot.mitml

Examples

```
data(studentratings)

# *** ..............................
# the 'type' interface
#

# * Example 1.1: 'ReadDis' and 'SES', predicted by 'ReadAchiev' and
# 'CognAbility', with random slope for 'ReadAchiev'

type <- c(-2,0,0,0,0,0,3,1,2,0)
names(type) <- colnames(studentratings)
type

imp <- panImpute(studentratings, type=type, n.burn=1000, n.iter=100, m=5)

# * Example 1.2: 'ReadDis' and 'SES' groupwise for 'FedState',
# and predicted by 'ReadAchiev'

type <- c(-2,-1,0,0,0,0,2,1,0,0)
names(type) <- colnames(studentratings)
type

imp <- panImpute(studentratings, type=type, n.burn=1000, n.iter=100, m=5)
```

```
# *** ..............................
# the 'formula' interface
#

# * Example 2.1: imputation of 'ReadDis', predicted by 'ReadAchiev'
# (random intercept)

fml <- ReadDis ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# ... the intercept can be suppressed using '0' or '-1' (here for fixed intercept)
fml <- ReadDis ~ 0 + ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# * Example 2.2: imputation of 'ReadDis', predicted by 'ReadAchiev'
# (random slope)

fml <- ReadDis ~ ReadAchiev + (1+ReadAchiev|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# * Example 2.3: imputation of 'ReadDis', predicted by 'ReadAchiev',
# groupwise for 'FedState'

fml <- ReadDis ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, group="FedState", n.burn=1000,
n.iter=100, m=5)

# * Example 2.4: imputation of 'ReadDis', predicted by 'ReadAchiev'
# including the cluster mean of 'ReadAchiev' as an additional predictor

fml <- ReadDis ~ ReadAchiev + I(clusterMeans(ReadAchiev,ID)) + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# ... using 'save.pred' to save the calculated cluster means in the data set
fml <- ReadDis ~ ReadAchiev + I(clusterMeans(ReadAchiev,ID)) + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5,
save.pred=TRUE)

head(mitmlComplete(imp,1))
```

`plot.mitml` *Print diagnostic plots*

Description

Generates diagnostic plots for assessing the convergence and autocorrelation behavior of pan's Gibbs sampler.

Usage

```
## S3 method for class 'mitml'
plot(x, print=c("beta","psi","sigma"), group="all",
  trace=c("imputation","burnin","all"), smooth=3,
  export=c("none","png","pdf"), dev.args=list(), ...)
```

Arguments

x	An object of class mitml as produced by panImpute.
print	A character vector containing one or several of "beta", "psi" or "sigma" denoting which parameters should be plotted. Default is to plot all parameters.
group	Either "all" or an integer denoting for which group plots should be generated. Used only when group has been specified in panImpute for group-wise imputation.
trace	One of "imputation", "burnin" or "all" denoting which part of the parameter chain should be used for the trace plot. Default plots only the iterations after burn-in.
smooth	A numeric value denoting the smoothing factor for the trend line in trace plots. Higher values correspond to less smoothing. Default is 3. If set to 0 or NULL, the trend line is suppressed.
export	(optional) A character string specifying if plots should be exported to file. If "png" or "pdf", then plots are printed into a folder named "panPlots" in the current directory using either the png or pdf device. Default is to "none", which does not export files.
dev.args	(optional) A named list containing additional arguments that are passed to the graphics device.
...	Parameters passed to the plotting functions.

Details

The plot method generates a series of plots for the parameters of the pan model that can be used for diagnostic purposes.

Setting print to "beta", "psi" and "sigma" will plot the fixed effects, the variances and covariances of random effects, and the variances and covariances of residuals, respectively. Each plotting window contains a trace plot (upper left), an autocorrelation plot (lower left), a kernel density approximation of the posterior distribution (upper right), and a posterior summary (lower right). The trace and smooth arguments can be used to influence how the trace plot is drawn, and what part of the chain should be used for it.

The plots are presented one at a time. To proceed with the next plot, the user may left-click in the plotting window or press "enter" while in the R console. No plots are displayed when exporting to file.

Value

None (invisible NULL).

Author(s)

Simon Grund

See Also

panImpute

Examples

```
## Not run:
data(studentratings)

# * Example 1: simple imputation

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

plot all parameters (default)
plot(imp)
plot(imp, print=c("beta","psi","sigma"))

plot fixed effects only
plot(imp, print="beta")

# export plots to file (using pdf device)
plot(imp, export="pdf", dev.args=list(width=9, height=4, pointsize=12))

# * Example 2: groupwise imputation

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, group=FedState, n.burn=1000,
       n.iter=100, m=5)

plot fixed effects for all groups (default for 'group')
plot(imp, print="beta", group="all")

plot fixed effects for first group only
plot(imp, print="beta", group=1)

## End(Not run)
```

`read.mitml` *Read* `mitml` *objects from file*

Description

This function loads `mitml` class objects from native R formats.

Usage

```
read.mitml(filename)
```

Arguments

`filename`	Name of the file to read, to be specified with file extension (e.g., `.R`, `.Rdata`).

Value

Returns the saved `mitml` class object.

Author(s)

Simon Grund

See Also

`panImpute`, `write.mitml`

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# write 'mitml' object
write.mitml(imp, filename="imputation.R")

# read previously saved 'mitml' object
previous.imp <- read.mitml("imputation.R")

class(previous.imp)
previous.imp
```

`studentratings` *Example data set on student's ratings and achievement*

Description

This is data set has been simulated to mimic typical data from educational research. The data set features student's ratings of their teachers' behavior (i.e., disciplinary problems in mathematics and reading class) and their general learning environment (school climate), as well as mathematics and reading achievement scores, and scores for socio-economic state and cognitive ability.

In addition, the data set features the ID of 50 different schools (i.e., clusters), the biological sex of all students, and a broad, additional grouping factor. Different amounts of missing data have been inserted into the data set in a completely random fashion.

All data are completely fictional, and thus should not be interpreted as if they were real.

Usage

```
data(studentratings)
```

Format

A data frame containing 750 observations on 10 variables.

---

summary.mitml *Summary measures for imputation models*

Description

Provides summary statistics and additional information regarding the imputation process.

Usage

```
## S3 method for class 'mitml'
summary(object, n.Rhat=3, goodness.of.appr=FALSE, ...)
```

Arguments

`object`	An object of class `mitml` as produced by `panImpute`.
`n.Rhat`	(optional) An integer denoting the number of sequences used for calculating the potential scale reduction factor. Default is to `3`.
`goodness.of.appr`	(optional) A logical flag indicating if the goodness of approximation should be printed. Default is to `FALSE` (see Note).
`...`	Not being used.

Details

The `summary` method calculates summary statistics for objects of class `mitml`, gives information on the imputation process, and the amount of missing data per variable.

The output includes the potential scale reduction factor (PSRF, or $\hat{R}$), which is calculated for each parameter of the imputation model as a measure of convergence (Gelman and Rubin, 1992). Calculation of the PSRFs can be suppressed by setting `n.Rhat=NULL`.

Value

Returns an object of class `summary.mitml`. A print method is used for better readable console output.

Note

The PSRFs are not computed from different chains, but by dividing each chain from the imputation phase into a number of sequences as denoted by `n.Rhat`. This is slightly different from the original method proposed by Gelman and Rubin.

The goodness of approximation indicates what proportion of the posterior standard deviation is due to simulation error. For multiple imputation, goodness of approximation is not essential; it should be considered only if posterior summaries, such as the EAP, are of interest.

Author(s)

Simon Grund

References

Gelman, A., and Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science, 7*, 457-472.

Hoff, P. D. (2009). *A first course in Bayesian statistical methods*. New York, NY: Springer.

See Also

`panImpute`

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# print summary
summary(imp)
```

---

testConstraints *Test functions and constraints of model parameters*

Description

Performs hypothesis tests for arbitrary functions of a parameter vector using the Delta method.

Usage

```
testConstraints(model, qhat, uhat, constraints, method=c("D1","D2"), df.com=NULL)
```

Arguments

model	A list of fitted statistical models (see examples).
qhat, uhat	Two matrices/arrays or lists containing estimates of the parameter vector and its covariance matrix, respectively, for each imputed data set (see examples).
constraints	A character vector specifying constraints or functions of the vector of model parameters to be tested.
method	A character string denoting the method by which the test is performed. Can be either "D1" or "D2" (see details). Default is to "D1".
df.com	(optional) A single number or a numeric vector denoting the complete-data degrees of freedom for the hypothesis test. Only used if method="D1".

Details

This function is similar in functionality to testModels but extended to arbitrary functions (or constraints) of the model parameters. The function is based on the Delta method (e.g., Casella & Berger, 2002) according to which any function of the model parameters can be tested using Wald-like methods. It is assumed that the parameters can be extracted using coef and vcov methods (or similar; e.g., regression coefficients, fixed effects in multilevel models) Alternatively, hypothesis tests can be carried out using user-supplied matrices/arrays or lists (qhat and uhat, see examples).

Constraints and functions of the model parameters can be specified in the constraints argument. The constraints must be supplied as a character vector, where each string denotes a function or a constraint to be tested (see examples).

As in testModels, the parameter vector is assumed to follow a multivariate normal distribution. The Wald-like tests can be aggregated across data sets either by method D_1 (Li, Raghunathan & Rubin, 1991) or D_2 (Li, Meng, Raghunathan & Rubin, 1991), where D_1 operates on the constrained estimates and standard errors, and D_2 operates on the Wald-statistics.

For D_1, the complete-data degrees of freedom can be adjusted for smaller samples by specifying df.com.

Currently, the procedure supports statistical models that define coef and vcov methods (e.g., lm) and multilevel models estimated with lme4 or nlme. The arguments qhat and uhat allow for quite general hypothesis tests regardless of model class. Support for further models may be added in future releases.

Value

Returns a list containing the results of the model comparison, the constrained estimates and standard errors, and the relative increase in variance due to nonresponse (Rubin, 1987). A print method is used for better readable console output.

Author(s)

Simon Grund

References

Casella, G., & Berger, R. L. (2002). *Statistical inference (2nd. Ed.)*. Pacific Grove, CA: Duxbury.

Li, K.-H., Meng, X.-L., Raghunathan, T. E., & Rubin, D. B. (1991). Significance levels from repeated p-values with multiply-imputed data. *Statistica Sinica, 1*, 65-92.

Li, K. H., Raghunathan, T. E., & Rubin, D. B. (1991). Large-sample significance levels from multiply imputed data using moment-based statistics and an F reference distribution. *Journal of the American Statistical Association, 86*, 1065-1073.

See Also

testModels, with.mitml.list

Examples

```
data(studentratings)

fml <- MathDis + ReadDis + SchClimate ~ (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

implist <- mitmlComplete(imp, print=1:5)

# fit simple regression model
fit.lm <- with(implist, lm(SchClimate ~ ReadDis + MathDis))

# apply Rubin's rules
testEstimates(fit.lm)

# * Example 1: test 'identity' function of two parameters
# multi-parameter hypothesis test, equivalent to model comparison

cons <- c("ReadDis","MathDis")
testConstraints(fit.lm, constraints=cons)

# ... adjusting for finite samples
testConstraints(fit.lm, constraints=cons, df.com=749)

# ... using D2
testConstraints(fit.lm, constraints=cons, method="D2")

# * Example 2: test for equality of two effects
# tests the hypothesis that the effects of 'ReadDis' and 'MathDis'
# are equal (ReadDis=MathDis)

cons <- c("ReadDis-MathDis")
testConstraints(fit.lm, constraints=cons)

# * Example 3: test against a fixed value
# tests the hypothesis that the effect of "ReadDis" is one (ReadDis=1)

cons <- c("ReadDis-1")
testConstraints(fit.lm, constraints=cons)
```

```
# * Example 4: test 'identity' using arrays and list

fit.lm <- with(implist, lm(SchClimate ~ ReadDis + MathDis))

cons <- c("ReadDis","MathDis")
qhat <- sapply(fit.lm, coef)
uhat <- sapply(fit.lm, function(x) vcov(x), simplify="array")
testConstraints(qhat=qhat, uhat=uhat, constraints=cons)
```

`testEstimates` *Compute final estimates and inferences*

Description

Computes final parameter estimates and inferences from multiply imputed data sets.

Usage

```
testEstimates(model, qhat, uhat, var.comp=FALSE, df.com=NULL)
```

Arguments

`model`	A list of fitted statistical models (see examples).
`qhat, uhat`	Two matrices or lists containing point and variances estimates, respectively, for each imputed data set (see examples).
`var.comp`	A logical flag indicating if estimates for variance components should be calculated. Default is to `FALSE`.
`df.com`	(optional) A numeric vector denoting the complete-data degrees of freedom for the hypothesis test.

Details

This function calculates final parameter estimates and inferences as suggested by Rubin (1987, "Rubin's rules") for each parameter of the fitted model. In other words, `testEstimates` aggregates estimates and standard errors across multiply imputed data sets. The parameters and standard errors can either be supplied as fitted statistical models (`model`), or as two matrices or lists (`qhat` and `uhat`, see examples).

Rubin's original method assumes that the complete-data degrees of freedom are infinite, which is reasonable in larger samples. Alternatively, the degrees of freedom can be adjusted for smaller samples by specifying `df.com` (Barnard & Rubin, 1999). The `df.com` argument can either be a single number if the degrees of freedom are equal for all tests, or a numeric vector with one element per test.

Using the `var.comp` argument, final estimates for variance components and related parameters can be requested. These will be shown as a separate table within the console output. Accessing variance

components is highly dependent on the model being estimated and not implemented for all models. Users may prefer calculating these estimates manually using `with.mitml.list` (see Example 3). No inferences are calculated for variance components.

Currently, the procedure supports statistical models that define `coef` and `vcov` methods (e.g., `lm`) and multilevel models estimated with `lme4` or `nlme`. The arguments `qhat` and `uhat` allow for quite general calculation of final estimates regardless of model class. Support for further models may be added in future releases.

Value

Returns a list containing the final parameter and inferences as well as the relative increase in variance due to nonresponse and the fraction of missing information (Rubin, 1987). A `print` method is used for better readable console output.

Author(s)

Simon Grund

References

Barnard, J., & Rubin, D. B. (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika, 86*, 948-955.

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys*. Hoboken, NJ: Wiley.

See Also

`with.mitml.list`

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

implist <- mitmlComplete(imp, print=1:5)

# fit multilevel model using lme4
require(lme4)
fit.lmer <- with(implist, lmer(SES ~ (1|ID)))

# * Example 1: combine estimates using model recognition
# final estimates and inferences sperately for each parameter (Rubin's rules)
testEstimates(fit.lmer)

# ... adjusted df for finite samples
testEstimates(fit.lmer, df.com=49)

# ... with additional table for variance components and ICCs
testEstimates(fit.lmer, var.comp=TRUE)
```

```
# * Example 2: combine estimates using matrices or lists
fit.lmer <- with(implist, lmer(SES ~ ReadAchiev + (1|ID)))

qhat <- sapply(fit.lmer, fixef)
uhat <- sapply(fit.lmer, function(x) diag(vcov(x)))
testEstimates(qhat=qhat, uhat=uhat)
```

`testModels` *Test multiple parameters and compare nested models*

Description

Performs multi-parameter hypothesis tests for a vector of statistical parameters and compares nested statistical models obtained from multiply imputed data sets.

Usage

```
testModels(model, null.model, method=c("D1","D2","D3"),
  use=c("wald","likelihood"), df.com=NULL)
```

Arguments

`model`	A list of fitted statistical models (see examples).
`null.model`	A list of fitted (more restrictive) statistical models.
`method`	A character string denoting the method by which the test is performed. Can be either `"D1"`, `"D2"` or `"D3"` (see details). Default is to `"D1"`.
`use`	A character string denoting Wald- or likelihood-based based tests. Can be either `"wald"` or `"likelihood"`. Only used if `method="D2"`.
`df.com`	(optional) A single number or a numeric vector denoting the complete-data degrees of freedom for the hypothesis test. Only used if `method="D1"`.

Details

This function compares two nested statistical models which differ by one or several parameters. In other words, the function performs Wald-like and likelihood-ratio hypothesis tests for the statistical parameters by which the two models differ.

The basic approach to Wald-like inference for multi-dimensional estimands was introduced Rubin (1987) and further developed by Li, Raghunathan and Rubin (1991). This procedure is commonly referred to as D_1 and can be called using `method="D1"`. D_1 is the multi-parameter equivalent of `testEstimates`, that is, it tests multiple parameters simultaneously. For D_1, the complete-data degrees of freedom are assumed to be infinite, but they can be adjusted for smaller samples by supplying `df.com` (Reiter, 2007).

An alternative method for Wald-like hypothesis tests was suggested by Li, Meng, Raghunathan and Rubin (1991). The procedure is often called D_2 and can be used by setting `method="D2"`. D_2

calculates the Wald-test directly for each data set and then aggregates the resulting χ^2 values. The source of these values is specified by the `use` argument. If `use="wald"` (the default), then a Wald-like hypothesis test similar to D_1 is performed. If `use="likelihood"`, then the two models are compared by their likelihood.

A third method relying on likelihood-based comparisons was suggested by Meng and Rubin (1992). This procedure is referred to as D_3 and can be used by setting `method="D3"`. D_3 compares the two models by aggregating the likelihood-ratio test across multiply imputed data sets.

In general, Wald-like hypothesis tests (D_1 and D_2) are appropriate if the parameters can be assumed to follow a multivariate normal distribution (e.g., regression coefficients, fixed effects in multilevel models). Likelihood-based comparisons (D_2 and D_3) may also be used for variance components.

The function supports different classes of statistical models depending on which `method` is chosen. D_1 supports quite general models as long as they define `coef` and `vcov` methods (or similar) for extracting the parameter estimates and their estimated covariance matrix. D_2 can be used for the same models (if `use="wald"`, or alternatively, for models that define a `logLik` method (if `use="likelihood"`). Finally, D_3 supports linear models and linear mixed-effects models with a single cluster variable as estimated by `lme4` or `nlme` (see Laird, Lange, & Stram, 1987). Support for other statistical models may be added in future releases.

Value

Returns a list containing the results of the model comparison as well as the relative increase in variance due to nonresponse (Rubin, 1987). A `print` method is used for better readable console output.

Note

The D_3 method and the likelihood-based D_2 are currently only supported for maximum likelihood (ML) fitted models. Support for REML fitted models may be added in future releases.

Author(s)

Simon Grund

References

Meng, X.-L., & Rubin, D. B. (1992). Performing likelihood ratio tests with multiply-imputed data sets. *Biometrika, 79*, 103-111.

Laird, N., Lange, N., & Stram, D. (1987). Maximum likelihood computations with repeated measures: Application of the em algorithm. *Journal of the American Statistical Association, 82*, 97-105.

Li, K.-H., Meng, X.-L., Raghunathan, T. E., & Rubin, D. B. (1991). Significance levels from repeated p-values with multiply-imputed data. *Statistica Sinica, 1*, 65-92.

Li, K. H., Raghunathan, T. E., & Rubin, D. B. (1991). Large-sample significance levels from multiply imputed data using moment-based statistics and an F reference distribution. *Journal of the American Statistical Association, 86*, 1065-1073.

Reiter, J. P. (2007). Small-sample degrees of freedom for multi-component significance tests with multiple imputation for missing data. *Biometrika, 94*, 502-508.

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys*. Hoboken, NJ: Wiley.

See Also

testEstimates, with.mitml.list

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

implist <- mitmlComplete(imp, print=1:5)

# * Example 1: multiparameter hypothesis test for 'ReadDis' and 'SES'
# This tests the hypothesis that both effects are zero.

require(lme4)
fit0 <- with(implist, lmer(ReadAchiev ~ (1|ID), REML=FALSE))
fit1 <- with(implist, lmer(ReadAchiev ~ ReadDis + (1|ID), REML=FALSE))

# apply Rubin's rules
testEstimates(fit1)

# multiparameter hypothesis test using D1 (default)
testModels(fit1, fit0)

# ... adjusting for finite samples
testModels(fit1, fit0, df.com=47)

# ... using D2 ("wald", using estimates and covariance-matrix)
testModels(fit1, fit0, method="D2")

# ... using D2 ("likelihood", using likelihood-ratio test)
testModels(fit1, fit0, method="D2", use="likelihood")

# ... using D3 (likelihood-ratio test, requires ML fit)
testModels(fit1, fit0, method="D3")

## Not run:
# * Example 2: multiparameter test using D3 with nlme

# for D3 to be calculable, the 'data' argument for 'lme' must be
# can be constructed manually

require(nlme)
fit0 <- with(implist, lme(ReadAchiev~1, random=~1|ID,
  data=data.frame(ReadAchiev,ID), method="ML"))
fit1 <- with(implist, lme(ReadAchiev ~ 1 + ReadDis, random=~ 1|ID,
  data=data.frame(ReadAchiev,ReadDis,ID), method="ML"))

# multiparameter hypothesis test using D3
testModels(fit1, fit0, method="D3")
```

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

`with.mitml.list` *Evaluate an expression in a list of imputed data set*

Description

This function evaluates an R expression in a list of multiply imputed data sets.

Usage

```
## S3 method for class 'mitml.list'
with(data, expr, ...)
## S3 method for class 'mitml.list'
within(data, expr, ...)
```

Arguments

`data`	A list of imputed data sets with class `mitml.list` as produced by `mitmlComplete`.
`expr`	An R expression to be evaluated for each data set.
`...`	Not being used.

Details

The two functions are `with` and `within` methods for objects of class `mitml.list`. Both `with` and `within` evaluate an R expression in each of the imputed data sets. However, the two functions return different values (see below).

Value

- `with`: Returns the evaluated expression from each data set as a list. This is particularly useful for fitting statistical models to multiply imputed data. The list of fitted models can be analyzed using `testEstimates`, `testModels`, and `testConstraints`.
- `within`: Evaluates the R expression for each data set and returns the altered data sets as a list. This is useful for manipulating the data prior to data analysis (e.g., centering, calculating cluster means, etc.).

Author(s)

Simon Grund

See Also

`mitmlComplete`, `clusterMeans`

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

implist <- mitmlComplete(imp, print=1:5)

# * Example 1: data manipulation

# calculate and save cluster means
new1.implist <- within(implist, Means.ReadAchiev <- clusterMeans(ReadAchiev, ID))

# calculate interaction terms
new2.implist <- within(implist, int.SesCa <- SES*CognAbility )

# * Example 2: fitting statistical models

# fit regression model
fit.lm <- with(implist, lm(ReadAchiev ~ ReadDis))

# fit multilevel model using lme4
require(lme4)
fit.lmer <- with(implist, lmer(ReadAchiev ~ ReadDis + (1|ID)))

# * Example 3: manual extraction of variance estimates
require(lme4)
fit.lmer <- with(implist, lmer(SES ~ (1|ID)))

# extract level-1 and level-2 variances
var.l1 <- sapply(fit.lmer, function(z) attr(VarCorr(z),"sc")^2)
var.l2 <- sapply(fit.lmer, function(z) VarCorr(z)$ID[1,1])

# calculate final estimate of the intraclass correlation
ICC <- mean( var.l2 / (var.l2+var.l1) )
```

`write.mitml` *Write* `mitml` *objects to file*

Description

This function saves objects of class `mitml` in native R formats.

Usage

```
write.mitml(x, filename, drop=FALSE)
```

Arguments

x	An object of class mitml as produced by panImpute.
filename	Name of the destination file, to be specified with file extension (e.g., .R, .Rdata).
drop	Logical flag indicating if the parameters of the imputation model should be dropped in favor for lower file size. Default is to FALSE.

Value

None (invisible NULL).

Author(s)

Simon Grund

See Also

panImpute, read.mitml

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# write full 'mitml' object (default)
write.mitml(imp, filename="imputation.R")

# drop parameters of the imputation model
write.mitml(imp, filename="imputation.R", drop=TRUE)
```

---

write.mitmlMplus *Write* mitml *objects to Mplus format*

Description

Saves objects of class mitml as a series of text files which can be processed by the statistical software M*plus* (Muthen & Muthen, 2012).

Usage

```
write.mitmlMplus(x, filename, suffix="list", sep="\t", dec=".", na.value=-999)
```

Arguments

x	An object of class mitml or mitml.list (i.e., either produced by panImpute or mitmlComplete).
filename	Basic file name for the text files containing the imputed data sets, to be specified without file extension.
suffix	File name suffix for the index file.
sep	The field separator.
dec	The decimal separator.
na.value	A numeric value coding the missing data in the resulting data files.

Details

The native M*plus* format for multiply imputed data sets comprises a series of text files, each containing one imputed data set, and an index file containing the names of all data files. During export, factors and character variables are converted to numeric. Therefore, write.mitmlMplus produces a log file which contains information about the data set and the factors that have been converted.

In addition, a basic M*plus* input file is generated that can be used for setting up the subsequent analysis models.

Value

None (invisible NULL).

Author(s)

Simon Grund

References

Muthen, L. K., & Muthen, B. O. (2012). *Mplus User's Guide. Seventh Edition.* Los Angeles, CA: Muthen & Muthen.

See Also

panImpute

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# write imputation files, index file, and log file
write.mitmlMplus(imp, filename="imputation", suffix="list", na.value=-999)
```

`write.mitmlSAV` *Write* `mitml` *objects to native SPSS format*

Description

Saves objects of class `mitml` in the `.sav` format used by the statistical software SPSS (IBM Corp., 2013). The function serves as a front-end for `write_sav` from the `haven` package.

Usage

```
write.mitmlSAV(x, filename)
```

Arguments

`x`	An object of class `mitml` or `mitml.list` (i.e., either produced by `panImpute` or `mitmlComplete`).
`filename`	Name of the destination file, to be specified with or without file extension. The file extension (`.sav`) is appended if necessary.

Details

This function exports multiply imputed data sets to a single `.sav` file, in which an `Imputation_` variable separates the original data and the various imputed data sets. Thus, `write.mitmlSAV` exports directly to the native SPSS format.

Alternatively, `write.mitmlSPSS` may be used for creating separate text and SPSS syntax files; an option that offers more control over the data format.

Value

None (invisible `NULL`).

Author(s)

Simon Grund

References

IBM Corp. (2013). *IBM SPSS Statistics for Windows, Version 22.0*. Armonk, NY: IBM Corp

See Also

`panImpute`, `mitmlComplete`, `write.mitmlSPSS`

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# write data file and SPSS syntax
write.mitmlSAV(imp, filename="imputation")
```

---

write.mitmlSPSS *Write* mitml *objects to SPSS compatible format*

Description

Saves objects of class mitml as a text and a syntax file which can be processed by the statistical software SPSS (IBM Corp., 2013).

Usage

```
write.mitmlSPSS(x, filename, sep="\t", dec=".", na.value=-999, syntax=TRUE,
  locale=NULL)
```

Arguments

x	An object of class mitml or mitml.list (i.e., either produced by panImpute or mitmlComplete).
filename	Basic file name of the data and syntax files, to be specified without file extension.
sep	The field separator.
dec	The decimal separator.
na.value	A numeric value coding the missing data in the resulting data file.
syntax	A logical flag indicating if an SPSS syntax file should be generated. This file contains instructions for SPSS for reading in the data file. Default is to TRUE.
locale	(optional) A character string specifying the localization to be used in SPSS (e.g., "en_US", "de_DE"). This argument may be specified if SPSS reads the data incorrectly due to conflicting locale settings.

Details

Multiply imputed data sets in SPSS are contained in a single file, in which an Imputation_ variable separates the original data and the various imputed data sets. During export, factors are converted to numeric, whereas character variables are left "as is".

By default, write.mitmlSPSS generates a raw text file containing the data, along with a syntax file containing instructions for SPSS. This syntax file mimics SPSS's functionality to read text files but

circumvents certain problems that may occur when using the GUI. In order to read in the data, the syntax file must be opened and executed using SPSS. The syntax file may be altered manually if problems occur, for example, if the file path of the data file is not correctly represented in the syntax.

Alternatively, `write.mitmlSAV` may be used for exporting directly to the SPSS native `.sav` format. However, this may offer less control over the data format.

Value

None (invisible `NULL`).

Author(s)

Simon Grund

References

IBM Corp. (2013). *IBM SPSS Statistics for Windows, Version 22.0.* Armonk, NY: IBM Corp

See Also

`panImpute`, `write.mitmlSAV`

Examples

```
data(studentratings)

fml <- ReadDis + SES ~ ReadAchiev + (1|ID)
imp <- panImpute(studentratings, formula=fml, n.burn=1000, n.iter=100, m=5)

# write data file and SPSS syntax
write.mitmlSPSS(imp, filename="imputation", sep="\t", dec=".", na.value=-999,
locale="en_US")
```

Index