Package 'mi4p'

May 8, 2026

Type Package

Title Multiple Imputation for Proteomics

Version 1.3

Date 2025-09-09

Depends R (>= 3.5.0)

Imports emmeans, foreach, imp4p, impute, limma, mice, stringr

Suggests Biobase, knitr, R.rsp, markdown, rmarkdown, DAPAR, ProteoMM, testthat (>= 3.0.0)

Author Marie Chion [aut] (ORCID: <https://orcid.org/0000-0001-8956-8388>), Christine Carapito [aut] (ORCID: <https://orcid.org/0000-0002-0079-319X>), Frederic Bertrand [cre, aut] (ORCID: <https://orcid.org/0000-0002-0837-8281>), Gordon Smyth [ctb], Davis McCarthy [ctb], Hélène Borges [ctb], Thomas Burger [ctb], Quentin Giai-Gianetto [ctb], Samuel Wieczorek [ctb]

Maintainer Frederic Bertrand <frederic.bertrand@lecnam.net>

Contact Marie Chion <marie.chion@protonmail.fr>, Frederic Bertrand <frederic.bertrand@lecnam.net>

Description A framework for multiple imputation for proteomics is proposed by Marie Chion, Christine Carapito and Frederic Bertrand (2021) <doi:10.1371/journal.pcbi.1010420>. It is dedicated to dealing with multiple imputation for proteomics.

License GPL (>= 2)

Copyright See the file COPYRIGHTS

Encoding UTF-8

Classification/MSC 62J05, 62J07, 62J99, 92C42

VignetteBuilder knitr

RoxygenNote 7.3.3

URL https://mariechion.github.io/mi4p/,
https://github.com/mariechion/mi4p/

BugReports https://github.com/mariechion/mi4p/issues/

Config/testthat/edition 3

NeedsCompilation no

Repository CRAN

Date/Publication 2025-09-19 23:40:08 UTC

Contents

check.conditions *Check if the design is valid*

Description

This function checks the validity of the conditions.

This function was included from the check.conditions function in the DAPAR package, since DAPAR was to be removed from Bioconductor >= 3.15.

Usage

```
check.conditions(conds)
```

Arguments

conds	A vector

Value

A list

Author(s)

Samuel Wieczorek as the author of check.conditions in the DAPAR package.

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
check.conditions(Biobase::pData(Exp1_R25_pept)$Condition)

## End(Not run)
```

check.design *Check if the design is valid*

Description

This function checks the validity of the experimental design.

This function was included from the check.design function in the DAPAR package, since DAPAR was to be removed from Bioconductor >= 3.15.

Usage

```
check.design(sTab)
```

Arguments

sTab	The data.frame which correspond to the pData function of MSnbase

Value

A boolean

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek as the authors of `check.design` in the `DAPAR` package.

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
check.design(Biobase::pData(Exp1_R25_pept)[,1:3])

## End(Not run)
```

---

datasim *A single simulated dataset*

Description

This dataset was simulated using the default values of the values of the options of the protdatasim function and the set.seed value set to 4619.

Format

A data frame with 200 observations on the following 11 variables.

id.obs a numeric vector

X1 a numeric vector

X2 a numeric vector

X3 a numeric vector

X4 a numeric vector

X5 a numeric vector

X6 a numeric vector

X7 a numeric vector

X8 a numeric vector

X9 a numeric vector

X10 a numeric vector

Author(s)

M. Chion, Ch. Carapito and F. Bertrand.

Source

We simulated the data.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
data(datasim)
str(datasim)
```

`eBayes.mod` *MI-aware Modifed eBayes Function*

Description

Modified eBayes function to be used instead of the one in the limma package

Usage

```
eBayes.mod(
  fit,
  VarRubin,
  proportion = 0.01,
  stdev.coef.lim = c(0.1, 4),
  trend = FALSE,
  robust = FALSE,
  winsor.tail.p = c(0.05, 0.1)
)
```

Arguments

`fit`	an MArrayLM fitted model object produced by lmFit or contrasts.fit. For ebayes only, fit can alternatively be an unclassed list produced by lm.series, gls.series or mrlm containing components coefficients, stdev.unscaled, sigma and df.residual.
`VarRubin`	a variance-covariance matrix.
`proportion`	numeric value between 0 and 1, assumed proportion of genes which are differentially expressed

`stdev.coef.lim`	numeric vector of length 2, assumed lower and upper limits for the standard deviation of log2-fold-changes for differentially expressed genes
`trend`	logical, should an intensity-trend be allowed for the prior variance? Default is that the prior variance is constant.
`robust`	logical, should the estimation of df.prior and var.prior be robustified against outlier sample variances?
`winsor.tail.p`	numeric vector of length 1 or 2, giving left and right tail proportions of x to Winsorize. Used only when robust=TRUE.

Value

eBayes produces an object of class MArrayLM (see MArrayLM-class) containing everything found in `fit` plus the following added components:

t numeric matrix of moderated t-statistics.

p.value numeric matrix of two-sided p-values corresponding to the t-statistics.

lods numeric matrix giving the log-odds of differential expression (on the natural log scale).

s2.prior estimated prior value for sigma^2. A row-wise vector if covariate is non-NULL, otherwise a single value.

df.prior degrees of freedom associated with s2.prior. A row-wise vector if robust=TRUE, otherwise a single value.

df.total row-wise numeric vector giving the total degrees of freedom associated with the t-statistics for each gene. Equal to df.prior+df.residual or sum(df.residual), whichever is smaller.

s2.post row-wise numeric vector giving the posterior values for sigma^2.

var.prior column-wise numeric vector giving estimated prior values for the variance of the log2-fold-changes for differentially expressed gene for each constrast. Used for evaluating lods.

F row-wise numeric vector of moderated F-statistics for testing all contrasts defined by the columns of fit simultaneously equal to zero.

F.p.value row-wise numeric vector giving p-values corresponding to F.

The matrices t, p.value and lods have the same dimensions as the input object fit, with rows corresponding to genes and columns to coefficients or contrasts. The vectors s2.prior, df.prior, df.total, F and F.p.value correspond to rows, with length equal to the number of genes. The vector var.prior corresponds to columns, with length equal to the number of contrasts. If s2.prior or df.prior have length 1, then the same value applies to all genes.

s2.prior, df.prior and var.prior contain empirical Bayes hyperparameters used to obtain df.total, s2.post and lods.

Author(s)

Modified by M. Chion and F. Bertrand. Original by Gordon Smyth and Davis McCarthy

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
VarRubin.matrix <- rubin2.all(datasim_imp[1:5,,],
attr(datasim,"metadata")$Condition)
set.seed(2016)
sigma2 <- 0.05 / rchisq(100, df=10) * 10
y <- datasim_imp[,,1]
design <- cbind(Intercept=1,Group=as.numeric(
attr(datasim,"metadata")$Condition)-1)
fit.model <- limma::lmFit(y,design)
eBayes.mod(fit=fit.model,VarRubin.matrix[[1]])
```

`formatLimmaResult` *Format a Result from Limma*

Description

It is not exported by DAPAR and has to be reproduced here.

Usage

```
formatLimmaResult(fit, conds, contrast)
```

Arguments

`fit`	Limma fit
`conds`	Condition vector
`contrast`	Contrast vector

Value

A list of two dataframes : logFC and P_Value. The first one contains the logFC values of all the comparisons (one column for one comparison), the second one contains the pvalue of all the comparisons (one column for one comparison). The names of the columns for those two dataframes are identical and correspond to the description of the comparison.

Author(s)

Adapted from the code of Samuel Wieczorek in the DAPAR package as it is an object that is not exported by the DAPAR package.

Examples

```
# library(DAPAR)
set.seed(2016)
data(qData)
data(sTab)
contrast=1
sTab.old <- sTab
conds <- factor(sTab$Condition, levels = unique(sTab$Condition))
sTab <- sTab[unlist(lapply(split(sTab, conds), function(x) {
  x["Sample.name"]
})), ]
qData <- qData[, unlist(lapply(split(sTab.old, conds), function(x) {
  x["Sample.name"]
}))]
conds <- conds[order(conds)]
res.l <- NULL
design.matrix <- mi4p::make.design(sTab)
contra <- mi4p::make.contrast(design.matrix, condition = conds,
                                  contrast)
cmtx <- limma::makeContrasts(contrasts = contra, levels = make.names(colnames(design.matrix)))
fit <- limma::eBayes(limma::contrasts.fit(limma::lmFit(qData,
                                                       design.matrix), cmtx))
res.l <- mi4p::formatLimmaResult(fit, conds, contrast)
```

---

hid.ebayes *MI-aware Modifed eBayes Function*

Description

Modified eBayes function to be used instead of the one, .ebayes, implemented in the limma package

Usage

```
hid.ebayes(
  fit,
  VarRubin,
  mod = TRUE,
  proportion = 0.01,
  stdev.coef.lim = c(0.1, 4),
  trend = FALSE,
  robust = FALSE,
  winsor.tail.p = c(0.05, 0.1),
  legacy = TRUE
)
```

Arguments

`fit`	an MArrayLM fitted model object produced by lmFit or contrasts.fit. For ebayes only, fit can alternatively be an unclassed list produced by lm.series, gls.series or mrlm containing components coefficients, stdev.unscaled, sigma and df.residual.
`VarRubin`	a variance-covariance matrix.
`mod`	TRUE (not used at the moment)
`proportion`	numeric value between 0 and 1, assumed proportion of genes which are differentially expressed
`stdev.coef.lim`	numeric vector of length 2, assumed lower and upper limits for the standard deviation of log2-fold-changes for differentially expressed genes
`trend`	logical, should an intensity-trend be allowed for the prior variance? Default is that the prior variance is constant.
`robust`	logical, should the estimation of df.prior and var.prior be robustified against outlier sample variances?
`winsor.tail.p`	numeric vector of length 1 or 2, giving left and right tail proportions of x to Winsorize. Used only when robust=TRUE.
`legacy`	boolean to stick to former way to squeeze variances. Defaults to TRUE.

Value

eBayes produces an object of class MArrayLM (see MArrayLM-class) containing everything found in `fit` plus the following added components:

t numeric matrix of moderated t-statistics.

p.value numeric matrix of two-sided p-values corresponding to the t-statistics.

lods numeric matrix giving the log-odds of differential expression (on the natural log scale).

s2.prior estimated prior value for sigma^2. A row-wise vector if covariate is non-NULL, otherwise a single value.

df.prior degrees of freedom associated with s2.prior. A row-wise vector if robust=TRUE, otherwise a single value.

df.total row-wise numeric vector giving the total degrees of freedom associated with the t-statistics for each gene. Equal to df.prior+df.residual or sum(df.residual), whichever is smaller.

s2.post row-wise numeric vector giving the posterior values for sigma^2.

var.prior column-wise numeric vector giving estimated prior values for the variance of the log2-fold-changes for differentially expressed gene for each constrast. Used for evaluating lods.

F row-wise numeric vector of moderated F-statistics for testing all contrasts defined by the columns of fit simultaneously equal to zero.

F.p.value row-wise numeric vector giving p-values corresponding to F.

The matrices t, p.value and lods have the same dimensions as the input object fit, with rows corresponding to genes and columns to coefficients or contrasts. The vectors s2.prior, df.prior, df.total, F and F.p.value correspond to rows, with length equal to the number of genes. The vector var.prior corresponds to columns, with length equal to the number of contrasts. If s2.prior or df.prior have length 1, then the same value applies to all genes.

s2.prior, df.prior and var.prior contain empirical Bayes hyperparameters used to obtain df.total, s2.post and lods.

Author(s)

Modified by M. Chion and F. Bertrand. Original by Gordon Smyth and Davis McCarthy

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
VarRubin.matrix <- rubin2.all(datasim_imp[1:5,,],
attr(datasim,"metadata")$Condition)
set.seed(2016)
sigma2 <- 0.05 / rchisq(100, df=10) * 10
y <- datasim_imp[,,1]
design <- cbind(Intercept=1,Group=as.numeric(
attr(datasim,"metadata")$Condition)-1)
fit.model <- limma::lmFit(y,design)
hid.ebayes(fit=fit.model,VarRubin.matrix[[1]])
```

---

limmaCompleteTest.mod *Computes a hierarchical differential analysis*

Description

Modified version of the limmaCompleteTest function from the DAPAR package to return both the fit and the results.

Usage

```
limmaCompleteTest.mod(qData, sTab, comp.type = "OnevsOne")
```

Arguments

qData	A matrix of quantitative data, without any missing values.
sTab	A dataframe of experimental design (pData()).
comp.type	A string that corresponds to the type of comparison. Values are: 'anova1way', 'OnevsOne' and 'OnevsAll'; default is 'OnevsOne'.

Value

A list of two dataframes : logFC and P_Value. The first one contains the logFC values of all the comparisons (one column for one comparison), the second one contains the pvalue of all the comparisons (one column for one comparison). The names of the columns for those two dataframes are identical and correspond to the description of the comparison.

Author(s)

Adapted from Hélène Borges, Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek

Examples

```
set.seed(2016)
data(qData)
data(sTab)
limma <- limmaCompleteTest.mod(qData, sTab, comp.type='OnevsOne')
```

`make.contrast` *Builds the contrast matrix*

Description

This function builds the contrast matrix

Usage

```
make.contrast(design, condition, contrast = 1)
```

Arguments

`design`	The data.frame which correspond to the pData function of MSnbase
`condition`	xxxxx
`contrast`	An integer that Indicates if the test consists of the comparison of each biological condition versus each of the other ones (Contrast=1; for example H0:"C1=C2" vs H1:"C1!=C2", etc.) or each condition versus all others (Contrast=2; e.g. H0:"C1=(C2+C3)/2" vs H1:"C1!=(C2+C3)/2", etc. if there are three conditions).

Value

A constrat matrix

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek originally in the `DAPAR` package. Included in this package since `DAPAR` was to be removed from Bioconductor >= 3.15.

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
design <- make.design(Biobase::pData(Exp1_R25_pept))
conds <- Biobase::pData(Exp1_R25_pept)$Condition
make.contrast(design, conds)

## End(Not run)
```

make.design *Builds the design matrix*

Description

This function builds the design matrix

Usage

```
make.design(sTab)
```

Arguments

sTab	The data.frame which correspond to the pData function of MSnbase

Value

A design matrix

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
make.design(Biobase::pData(Exp1_R25_pept))

## End(Not run)
```

make.design.1 *Builds the design matrix for designs of level 1*

Description

This function builds the design matrix for design of level 1

Usage

```
make.design.1(sTab)
```

Arguments

sTab	The data.frame which correspond to the pData function of MSnbase

Value

A design matrix

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
make.design.1(Biobase::pData(Exp1_R25_pept))

## End(Not run)
```

---

make.design.2 *Builds the design matrix for designs of level 2*

Description

This function builds the design matrix for design of level 2

Usage

```
make.design.2(sTab)
```

Arguments

sTab	The data.frame which correspond to the pData function of MSnbase

Value

A design matrix

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
make.design.2(Biobase::pData(Exp1_R25_pept))

## End(Not run)
```

make.design.3 *Builds the design matrix for designs of level 3*

Description

This function builds the design matrix for design of level 3

Usage

```
make.design.3(sTab)
```

Arguments

sTab	The data.frame which correspond to the pData function of MSnbase

Value

A design matrix

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek originally in the DAPAR package. Included in this package since DAPAR was to be removed from Bioconductor >= 3.15.

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
sTab <-cbind(Biobase::pData(Exp1_R25_pept), Tech.Rep=1:6)
make.design.3(sTab)

## End(Not run)
```

meanImp_emmeans *Multiple Imputation Estimate*

Description

Computes the multiple imputation parameter estimate using the emmeans package.

Usage

```
meanImp_emmeans(ind, peptide = 1, tabdata, metacond)
```

Arguments

ind	index
peptide	name of the peptide
tabdata	dataset
metacond	a factor to specify the groups

Value

A vector.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
meanImp_emmeans(1,1,datasim_imp,attr(datasim,"metadata")$Condition)
```

---

mi4limma *Differential analysis after multiple imputation*

Description

This function performs hierarchical differential analysis using a moderated t-test statistic, which accounts for multiple imputation variability if applicable.

Usage

```
mi4limma(qData, sTab, VarRubin, comp.type = "OnevsOne", robust = FALSE)
```

Arguments

qData	A matrix of quantitative data, without any missing values. It should be the averaged matrix from the array resulting from multi.impute.
sTab	The experimental matrix, also corresponding to the pData function of MSnbase.
VarRubin	A numerical vector, resulting from proj_matrix. It denotes the vector of projected variance-covariance matrices. It should be of length the number of peptides or proteins considered.
comp.type	A string that corresponds to the type of comparison. Values are: 'anova1way', 'OnevsOne' and 'OnevsAll'; default is 'OnevsOne'.
robust	logical, should the estimation of df.prior and var.prior be robustified against outlier sample variances? (as in limma's eBayes)

Value

A list of two dataframes : logFC and P_Value. The first one contains the logFC values of all the comparisons (one column for one comparison), the second one contains the pvalue of all the comparisons (one column for one comparison). The names of the columns for those two dataframes are identical and correspond to the description of the comparison.

Author(s)

Adapted by Marie Chion, from `limmaCompleteTest` of the `DAPAR` package by Hélène Borges, Thomas Burger, Quentin Giai-Gianetto and Samuel Wieczorek.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
set.seed(2016)
data(qData)
data(sTab)
fit.limma <- mi4limma(qData, sTab, diag(1,2))
```

`mm_peptides` *mm_peptides - peptide-level intensities for mouse*

Description

A dataset containing the protein and petide information and peptide-level intensities for 6 samples: 3 CG and 3 mCG groups. There are 69 proteins. The columns are as follows:

Usage

```
data(mm_peptides)
```

Format

A data frame with 1102 rows and 13 columnns, compiring 7 columns of metadata and 6 columns of peptide intensities. 69 proteins.

Details

- Sequence - peptide sequence - randomly chosen from a larger list of sequences
- MatchedID - numeric ID that links proteins in the two datasets, unnecessary if datasets are for the same species
- ProtID - protein ID, artificial protein ID, eg. Prot1, Prot2, ...

- GeneID - gene ID, artificial gene ID, eg. Gene1, Gene2, ...
- ProtName - artificial Protein Name
- ProtIDLong - long protein ID, full protein name, here artificially simulated
- GeneIDLong - long gene ID, full gene name, here artificially simulated
- CG1 - raw intensity column for sample 1 in CG group
- CG2 - raw intensity column for sample 2 in CG group
- CG3 - raw intensity column for sample 3 in CG group
- mCG1 - raw intensity column for sample 1 in mCG group
- mCG2 - raw intensity column for sample 2 in mCG group
- mCG3 - raw intensity column for sample 3 in mCG group

`multi.impute` *Multiple imputation of quantitative proteomics datasets*

Description

`multi.impute` performs multiple imputation on a given quantitative proteomics dataset.

Usage

```
multi.impute(data, conditions, nb.imp = NULL, method, parallel = FALSE)
```

Arguments

`data`	A quantitative matrix to be imputed, with proteins/peptides in rows and samples in columns.
`conditions`	A vector of length the number of samples where each element corresponds to the condition the sample belongs to.
`nb.imp`	The number of imputation to perform.
`method`	A single character string describing the imputation method to be used. See details.
`parallel`	Logical, whether or not use parallel computing (with `foreach`).

Details

Multiple imputation consists in imputing several times a given dataset using a given method. Here, imputation methods can be chosen either from `mice`, `imp4p-package` or `impute.knn`:

- "pmm", "midastouch", "sample", "cart", "rf","mean", "norm", "norm.nob", "norm.boot", "norm.predict": imputation methods as described in `mice`.
- "RF" imputes missing values using random forests algorithm as described in `impute.RF`.
- "MLE" imputes missing values using maximum likelihood estimation as described in `impute.mle`.
- "PCA" imputes missing values using principal component analysis as described in `impute.PCA`.
- "SLSA" imputes missing values using structured least squares algorithm as described in `impute.slsa`.
- "kNN" imputes missing values using k nearest neighbors as described in `impute.knn`.

Value

A numeric array of dimension c(dim(data),nb.imp).

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. arxiv:2108.07086. https://arxiv.org/abs/2108.07086.

Examples

```
library(mi4p)
data(datasim)
multi.impute(data = datasim[,-1], conditions = attr(datasim,"metadata")$Condition, method = "MLE")
```

`MVgen` *Amputation of a dataset*

Description

This function is designed to ampute datasets.

Usage

```
MVgen(dataset, prop_NA)
```

Arguments

`dataset`	dataset to be amputed
`prop_NA`	desired proportion of missing values in the amputed dataset

Value

A dataset with missing values.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_amp <- MVgen(datasim, .2)
sum(is.na(datasim_amp))/prod(dim(datasim_amp))
```

`norm.200.m100.sd1.vs.m200.sd1.list`
A list of simulated datasets.

Description

This list of 100 datasets was simulated using the default values of the options of the protdatasim function and the set.seed value set to 4619.

Format

The format is: List of 100 data.frames.

data.frame 200 obs. of 11 variables

- **id.obs** int [1:200] 1 2 3 4 5 6 7 8 9 10 ...
- **X1** num [1:200] 99.6 99.9 100.2 99.8 100.4 ...
- **X2** num [1:200] 97.4 101.3 100.3 100.2 101.7 ...
- **X3** num [1:200] 100.3 100.9 99.1 101.2 100.6 ...
- **X4** num [1:200] 99.4 99.2 98.5 99.1 99.5 ...
- **X5** num [1:200] 98.5 99.7 100 100.2 100.7 ...
- **X6** num [1:200] 200 199 199 200 199 ...
- **X7** num [1:200] 200 200 202 199 199 ...
- **X8** num [1:200] 202 199 200 199 201 ...
- **X9** num [1:200] 200 200 199 201 200 ...
- **X10** num [1:200] 200 198 200 201 199 ...

attr(*, "metadata") 'data.frame': 10 obs. of 3 variables:

- **Sample.name** chr [1:10] "X1" "X2" "X3" "X4" ...
- **Condition** Factor w/ 2 levels "A","B": 1 1 1 1 1 2 2 2 2 2
- **Bio.Rep** int [1:10] 1 2 3 4 5 6 7 8 9 10

... ...

Author(s)

M. Chion, Ch. Carapito and F. Bertrand.

Source

We simulated the data.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
data(norm.200.m100.sd1.vs.m200.sd1.list)
str(norm.200.m100.sd1.vs.m200.sd1.list)
```

`proj_matrix` *Variance-Covariance Matrix Projection*

Description

Use a projection of the given variance-covariance matrix.

Usage

```
proj_matrix(VarRubin.matrix, metadata)
```

Arguments

`VarRubin.matrix`	A variance-covariance matrix.
`metadata`	Metadata of the experiment.

Value

A list of variance-covariance matrices.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
VarRubin.matrix <- rubin2.all(datasim_imp[1:5,,],
attr(datasim,"metadata")$Condition)
proj_matrix(VarRubin.matrix, attr(datasim,"metadata"))
```

protdatasim *Data simulation function*

Description

Function to simulate benchmark datasets.

Usage

```
protdatasim(
  iii = 1,
  nobs = 200,
  nobs1 = 10,
  ng1 = 5,
  ng2 = 5,
  mg1 = 100,
  mg2 = 200,
  dispg1 = 1,
  dispg2 = 1
)
```

Arguments

iii	A parameter useful to loop over for simulated lists of datasets. It has no effect.
nobs	Number of peptides
nobs1	Number of peptides with differential expressions between the two conditions
ng1	Number of biological replicates in condition A
ng2	Number of biological replicates in condition B
mg1	Mean in condition A
mg2	Mean in condition B
dispg1	Dispersion in condition A
dispg2	Dispersion in condition B

Value

A data frame with the simulated and attribute metadata.

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
data_sim <- protdatasim()
attr(data_sim,"metadata")

norm.200.m100.sd1.vs.m200.sd1_list <- lapply(1:100, protdatasim)
attr(norm.200.m100.sd1.vs.m200.sd1_list[[1]],"metadata")
```

---

qData *Extract of the abundances of Exp1_R25_pept dataset*

Description

The data frame qData contains the first 500 rows of six columns that are the quantitation of peptides for the six replicates. They were obtained using the code exprs(Exp1_R25_pept)[1:500,].

Format

The format is: num [1:500, 1:6] 24.8 24.7 24.6 NA 24.5 ... - attr(*, "dimnames")=List of 2 ..$: chr [1:500] "0" "1" "2" "3" ... ..$: chr [1:6] "Intensity_C_R1" "Intensity_C_R2" "Intensity_C_R3" "Intensity_D_R1" ...

Details

The DAPARdata's Exp1_R25_pept dataset is the final outcome of a quantitative mass spectrometry-based proteomic analysis of two samples containing different concentrations of 48 human proteins (UPS1 standard from Sigma-Aldrich) within a constant yeast background (see Giai Gianetto et al. (2016) for details). It contains the abundance values of the different human and yeast peptides identified and quantified in these two conditions. The two conditions represent the measured abundances of peptides when respectively 25 fmol and 10 fmol of UPS1 human proteins were mixed with the yeast extract before mass spectrometry analyses. This results in a concentration ratio of 2.5. Three technical replicates were acquired for each condition.

Source

The DAPARdata package.

References

Cox J., Hein M.Y., Luber C.A., Paron I., Nagaraj N., Mann M. Accurate proteome-wide label-free quantification by delayed normalization and maximal peptide ratio extraction, termed MaxLFQ. Mol Cell Proteomics. 2014 Sep, 13(9):2513-26.

Giai Gianetto, Q., Combes, F., Ramus, C., Bruley, C., Coute, Y., Burger, T. (2016). Calibration plot for proteomics: A graphical tool to visually check the assumptions underlying FDR control in quantitative experiments. Proteomics, 16(1), 29-32.

Examples

```
data(qData)
str(qData)
pairs(qData)
```

`rubin1.all` *First Rubin rule (all peptides)*

Description

Computes the first Rubin's rule for all the peptides.

Usage

```
rubin1.all(
  data,
  metacond,
  funcmean = meanImp_emmeans,
  is.parallel = FALSE,
  verbose = FALSE
)
```

Arguments

`data`	dataset
`metacond`	a factor to specify the groups
`funcmean`	function that should be used to compute the mean
`is.parallel`	Logical, whether or not use parallel computing (with `foreach`).
`verbose`	Logical, should messages be displayed?

Value

A vector of estimated parameters.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
rubin1.all(datasim_imp[1:5,,],funcmean = meanImp_emmeans,
attr(datasim,"metadata")$Condition)
```

`rubin1.one` *First Rubin rule (a given peptide)*

Description

Computes the first Rubin's rule for a given peptide.

Usage

```
rubin1.one(peptide, data, funcmean = meanImp_emmeans, metacond)
```

Arguments

`peptide`	peptide for which the variance-covariance matrix should be derived.
`data`	dataset
`funcmean`	function that should be used to compute the mean
`metacond`	a factor to specify the groups

Value

A vector of estimated parameters.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. arxiv:2108.07086. https://arxiv.org/abs/2108.07086.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
rubin1.one(1,datasim_imp,funcmean = meanImp_emmeans,
attr(datasim,"metadata")$Condition)
```

rubin2.all *Computes the 2nd Rubin's rule (all peptides)*

Description

Computes the total variance-covariance component in the 2nd Rubin's rule for all peptides.

Usage

```
rubin2.all(
  data,
  metacond,
  funcmean = meanImp_emmeans,
  funcvar = within_variance_comp_emmeans,
  is.parallel = FALSE,
  verbose = FALSE
)
```

Arguments

data	dataset
metacond	a factor to specify the groups
funcmean	function that should be used to compute the mean
funcvar	function that should be used to compute the variance
is.parallel	should parallel computing be used?
verbose	should messages be displayed?

Value

List of variance-covariance matrices.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
rubin2.all(datasim_imp[1:5,,],attr(datasim,"metadata")$Condition)
```

`rubin2bt.all` *2nd Rubin's rule Between-Imputation component (all peptides)*

Description

Computes the between-imputation component in the 2nd Rubin's rule for all peptides.

Usage

```
rubin2bt.all(
  data,
  funcmean = meanImp_emmeans,
  metacond,
  is.parallel = FALSE,
  verbose = FALSE
)
```

Arguments

`data`	dataset
`funcmean`	function that should be used to compute the mean
`metacond`	a factor to specify the groups
`is.parallel`	should parallel computing be used?
`verbose`	should messages be displayed?

Value

List of variance-covariance matrices.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
rubin2bt.all(datasim_imp[1:5,,],funcmean = meanImp_emmeans,
attr(datasim,"metadata")$Condition)
```

rubin2bt.one *2nd Rubin's rule Between-Imputation Component (a given peptide)*

Description

Computes the between-imputation component in the 2nd Rubin's rule for a given peptide.

Usage

```
rubin2bt.one(peptide, data, funcmean, metacond)
```

Arguments

peptide	peptide for which the variance-covariance matrix should be derived.
data	dataset
funcmean	function that should be used to compute the mean
metacond	a factor to specify the groups

Value

A variance-covariance matrix.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
rubin2bt.one(1,datasim_imp,funcmean = meanImp_emmeans,
attr(datasim,"metadata")$Condition)
```

`rubin2wt.all` *2nd Rubin's rule Within-Variance Component (all peptides)*

Description

Computes the within-variance component in the 2nd Rubin's rule for all peptides.

Usage

```
rubin2wt.all(
  data,
  funcvar = mi4p::within_variance_comp_emmeans,
  metacond,
  is.parallel = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	dataset
`funcvar`	function that should be used to compute the variance
`metacond`	a factor to specify the groups
`is.parallel`	should parallel computing be used?
`verbose`	should messages be displayed?

Value

List of variance-covariance matrices.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1],
conditions = attr(datasim,"metadata")$Condition, method = "MLE")
rubin2wt.all(datasim_imp[1:5,,],funcvar = within_variance_comp_emmeans,
attr(datasim,"metadata")$Condition)
```

rubin2wt.one *2nd Rubin's rule Within-Variance Component (a given peptide)*

Description

Computes the within-variance component in the 2nd Rubin's rule for a given peptide.

Usage

```
rubin2wt.one(peptide, data, funcvar, metacond)
```

Arguments

peptide	peptide for which the variance-covariance matrix should be derived.
data	dataset
funcvar	function that should be used to compute the variance
metacond	a factor to specify the groups

Value

A variance-covariance matrix.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
rubin2wt.one(1,datasim_imp,funcvar=within_variance_comp_emmeans,
attr(datasim,"metadata")$Condition)
```

sTab *Experimental design for the Exp1_R25_pept dataset*

Description

The data frame sTab contains the experimental design and gives few informations about the samples. They were obtained using the code pData(Exp1_R25_pept).

Format

A data frame with 6 observations on the following 3 variables.

Sample.name a character vector

Condition a character vector

Bio.Rep a numeric vector

Details

The DAPARdata's Exp1_R25_pept dataset is the final outcome of a quantitative mass spectrometry-based proteomic analysis of two samples containing different concentrations of 48 human proteins (UPS1 standard from Sigma-Aldrich) within a constant yeast background (see Giai Gianetto et al. (2016) for details). It contains the abundance values of the different human and yeast peptides identified and quantified in these two conditions. The two conditions represent the measured abundances of peptides when respectively 25 fmol and 10 fmol of UPS1 human proteins were mixed with the yeast extract before mass spectrometry analyses. This results in a concentration ratio of 2.5. Three technical replicates were acquired for each condition.

Source

The DAPARdata package.

References

Cox J., Hein M.Y., Luber C.A., Paron I., Nagaraj N., Mann M. Accurate proteome-wide label-free quantification by delayed normalization and maximal peptide ratio extraction, termed MaxLFQ. Mol Cell Proteomics. 2014 Sep, 13(9):2513-26.

Giai Gianetto, Q., Combes, F., Ramus, C., Bruley, C., Coute, Y., Burger, T. (2016). Calibration plot for proteomics: A graphical tool to visually check the assumptions underlying FDR control in quantitative experiments. Proteomics, 16(1), 29-32.

Examples

```
data(sTab)
str(sTab)
```

test.design *Check if xxxxxx*

Description

This function check xxxxx

Usage

```
test.design(tab)
```

Arguments

tab	A data.frame which correspond to xxxxxx

Value

A list of two items

Author(s)

Thomas Burger, Quentin Giai-Gianetto, Samuel Wieczorek originally in the DAPAR package. Included in this package since DAPAR was to be removed from Bioconductor >= 3.15.

Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
test.design(Biobase::pData(Exp1_R25_pept)[,1:3])

## End(Not run)
```

---

within_variance_comp_emmeans
Multiple Imputation Within Variance Component

Description

Computes the multiple imputation within variance component using the emmeans package.

Usage

```
within_variance_comp_emmeans(ind, peptide, data, metacond)
```

Arguments

ind	index
peptide	name of the peptide
data	dataset
metacond	a factor to specify the groups

Value

A variance-covariance matrix.

Author(s)

Frédéric Bertrand

References

M. Chion, Ch. Carapito and F. Bertrand (2021). *Accounting for multiple imputation-induced variability for differential analysis in mass spectrometry-based label-free quantitative proteomics*. doi:10.1371/journal.pcbi.1010420.

Examples

```
library(mi4p)
data(datasim)
datasim_imp <- multi.impute(data = datasim[,-1], conditions =
attr(datasim,"metadata")$Condition, method = "MLE")
within_variance_comp_emmeans(1,1,datasim_imp,
attr(datasim,"metadata")$Condition)
```

Index