

# Package ‘DAPAR’

June 21, 2022

**Type** Package

**Title** Tools for the Differential Analysis of Proteins Abundance with R

**Version** 1.28.1

**Date** 2022-06-17

**Description** This package contains a collection of functions for the visualisation and the statistical analysis of proteomic data.

**License** Artistic-2.0

**VignetteBuilder** knitr

**Depends** R (>= 4.2.0)

**Suggests** BiocGenerics, testthat, BiocStyle, AnnotationDbi, clusterProfiler, diptest, cluster, vioplot, visNetwork, vsn, igraph, FactoMineR, factoextra, dendextend, foreach, parallel, doParallel, Mfuzz, apcluster, forcats, readxl, openxlsx, multcomp, purrr

**Imports** Biobase, MSnbase, tibble, RColorBrewer, stats, preprocessCore, Cairo, png, lattice, reshape2, gplots, pcaMethods, ggplot2, limma, knitr, tmvtnorm, norm, impute, stringr, grDevices, graphics, utils, cp4p (>= 0.3.5), scales, Matrix, imp4p (>= 1.1), methods, DAPARdata (>= 1.23.4), siggenes, graph, lme4, highcharter, dplyr, tidyr, tidyverse, igraph

**biocViews** Proteomics, Normalization, Preprocessing, MassSpectrometry, QualityControl, GO, DataImport

**NeedsCompilation** no

**RoxygenNote** 7.1.2

**Encoding** UTF-8

**URL** <http://www.prostar-proteomics.org/>

**BugReports** <https://github.com/prostarproteomics/DAPAR/issues>

**Collate** agregation.R anova\_analysis.R bioAnalysis.R clustering.R compute.t.tests.R DiffAnalysis.R get\_pep\_prot\_cc.R metacell.R inOutFiles.R limmaAnalysis.R logText.R missingValuesFilter.R

missingValuesImputation\_PeptideLevel.R  
 missingValuesImputation\_ProteinLevel.R normalize.R pepa.R  
 plots\_boxplot.R plots\_compare\_Norm.R plots\_corr\_matrix.R  
 plots\_density.R plots\_density\_CV.R plots\_heatmaps.R plots\_pca.R  
 plots\_violin.R utils.R volcanoPlot.R palette.R hypothesisTest.R  
 metacell\_Plots.R

**git\_url** <https://git.bioconductor.org/packages/DAPAR>

**git\_branch** RELEASE\_3\_15

**git\_last\_commit** 0f29e09

**git\_last\_commit\_date** 2022-06-17

**Date/Publication** 2022-06-21

**Author** Samuel Wieczorek [aut, cre],  
 Florence Combes [aut],  
 Thomas Burger [aut],  
 Vasile-Cosmin Lazar [ctb],  
 Enora Fremy [ctb],  
 Helene Borges [ctb]

**Maintainer** Samuel Wieczorek <samuel.wieczorek@cea.fr>

## R topics documented:

aggregateIter . . . . .	6
aggregateIterParallel . . . . .	7
aggregateMean . . . . .	8
AggregateMetacell . . . . .	8
aggregateSum . . . . .	9
aggregateTopn . . . . .	10
averageIntensities . . . . .	11
barplotEnrichGO_HC . . . . .	12
barplotGroupGO_HC . . . . .	12
boxPlotD_HC . . . . .	13
BuildAdjacencyMatrix . . . . .	14
BuildColumnToProteinDataset . . . . .	15
BuildColumnToProteinDataset_par . . . . .	16
buildGraph . . . . .	17
BuildMetaCell . . . . .	17
check.conditions . . . . .	18
check.design . . . . .	19
checkClusterability . . . . .	20
classic1wayAnova . . . . .	21
compareNormalizationD_HC . . . . .	21
compute_t_tests . . . . .	23
corrMatrixD_HC . . . . .	24
CountPep . . . . .	25
createMSnset . . . . .	25
CVDistD_HC . . . . .	27

dapar_hc_chart . . . . .	28
dapar_hc_ExportMenu . . . . .	28
deleteLinesFromIndices . . . . .	29
densityPlotD_HC . . . . .	30
diffAnaComputeFDR . . . . .	31
diffAnaGetSignificant . . . . .	32
diffAnaSave . . . . .	33
diffAnaVolcanoplot . . . . .	34
diffAnaVolcanoplot_rCharts . . . . .	35
display.CC.visNet . . . . .	37
enrich_GO . . . . .	38
ExtendPalette . . . . .	39
finalizeAggregation . . . . .	40
findMECBlock . . . . .	40
formatLimmaResult . . . . .	41
formatPHResults . . . . .	42
fudge2LRT . . . . .	43
get.pep.prot.cc . . . . .	44
GetCC . . . . .	45
GetColorsForConditions . . . . .	45
GetDetailedNbPeptides . . . . .	46
GetDetailedNbPeptidesUsed . . . . .	47
getIndicesConditions . . . . .	48
getIndicesOfLinesToRemove . . . . .	48
GetIndices_BasedOnConditions . . . . .	49
GetIndices_MetacellFiltering . . . . .	50
GetIndices_WholeLine . . . . .	51
GetIndices_WholeMatrix . . . . .	51
GetKeyId . . . . .	52
getListNbValuesInLines . . . . .	53
GetMatAdj . . . . .	53
GetMetacell . . . . .	54
GetNbPeptidesUsed . . . . .	54
getNumberOf . . . . .	55
getNumberOfEmptyLines . . . . .	56
getPourcentageOfMV . . . . .	56
getProcessingInfo . . . . .	57
getProteinsStats . . . . .	58
getQuantile4Imp . . . . .	58
getTextForAggregation . . . . .	59
getTextForAnaDiff . . . . .	60
getTextForFiltering . . . . .	60
getTextForGOAnalysis . . . . .	61
getTextForHypothesisTest . . . . .	61
getTextForNewDataset . . . . .	62
getTextForNormalization . . . . .	63
getTextForpeptideImputation . . . . .	63
getTextForproteinImputation . . . . .	64

GetTypeofData	65
Get_AllComparisons	65
GlobalQuantileAlignment	66
GOAnalysisSave	67
GraphPepProt	68
group_GO	68
hc_logFC_DensityPlot	69
hc_mvTypePlot2	70
heatmapD	71
heatmapForMissingValues	72
histPValue_HC	73
impute.pa2	74
inner.aggregate.iter	75
inner.aggregate.topn	76
inner.mean	77
inner.sum	77
LH0	78
LH0.lm	78
LH1	79
LH1.lm	80
limmaCompleteTest	80
listSheets	81
LOESS	82
make.contrast	82
make.design	83
make.design.1	84
make.design.2	84
make.design.3	85
match.metacell	86
MeanCentering	86
metacell.def	87
MetaCellFiltering	88
MetacellFilteringScope	89
metacellHisto_HC	89
metacellPerLinesHistoPerCondition_HC	90
metacellPerLinesHisto_HC	91
Metacell_generic	92
Metacell_maxquant	93
Metacell_proline	94
metacombine	95
mvImage	96
my_hc_chart	97
my_hc_ExportMenu	97
nonzero	98
normalizeMethods.dapar	99
NumericalFiltering	99
NumericalgetIndicesOfLinesToRemove	100
pepa.test	101

plotJitter . . . . .	102
plotJitter_rCharts . . . . .	102
plotPCA_Eigen . . . . .	103
plotPCA_Eigen_hc . . . . .	104
plotPCA_Ind . . . . .	104
plotPCA_Var . . . . .	105
postHocTest . . . . .	106
proportionConRev_HC . . . . .	107
QuantileCentering . . . . .	108
rbindMSnset . . . . .	109
readExcel . . . . .	109
reIntroduceMEC . . . . .	110
removeLines . . . . .	111
samLRT . . . . .	111
saveParameters . . . . .	112
scatterplotEnrichGO_HC . . . . .	113
search.metacell.tags . . . . .	114
SetCC . . . . .	114
SetMatAdj . . . . .	115
Set_POV_MEC_tags . . . . .	116
splitAdjacencyMat . . . . .	116
StringBasedFiltering . . . . .	117
StringBasedFiltering2 . . . . .	118
SumByColumns . . . . .	119
SymFilteringOperators . . . . .	119
test.design . . . . .	120
translatedRandomBeta . . . . .	120
univ_AnnotDbPkg . . . . .	121
UpdateMetacell . . . . .	122
violinPlotD . . . . .	122
visualizeClusters . . . . .	123
vsn . . . . .	124
wrapper.compareNormalizationD_HC . . . . .	125
wrapper.corrMatrixD_HC . . . . .	126
wrapper.CVDistD_HC . . . . .	127
wrapper.dapar.impute.mi . . . . .	127
wrapper.heatmapD . . . . .	129
wrapper.impute.detQuant . . . . .	130
wrapper.impute.fixedValue . . . . .	131
wrapper.impute.KNN . . . . .	132
wrapper.impute.mle . . . . .	133
wrapper.impute.pa . . . . .	133
wrapper.impute.pa2 . . . . .	134
wrapper.impute.slsa . . . . .	135
wrapper.mvImage . . . . .	136
wrapper.normalizeD . . . . .	137
wrapper.pca . . . . .	138
wrapperCalibrationPlot . . . . .	139

wrapperClassic1wayAnova . . . . .	140
wrapperRunClustering . . . . .	141
write.excel . . . . .	143
writeMSnsetToCSV . . . . .	144
writeMSnsetToExcel . . . . .	145

<b>Index</b>	<b>146</b>
--------------	------------

---

aggregateIter	xxxx
---------------	------

---

### Description

Method to xxxxx

### Usage

```
aggregateIter(obj.pep, X, init.method = "Sum", method = "Mean", n = NULL)
```

### Arguments

obj.pep	xxxxxx
X	xxxxx
init.method	xxxxxx
method	xxxxxx
n	xxxxx

### Value

A protein object of class MSnset

### Author(s)

Samuel Wieczorek

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
protID <- "Protein_group_IDs"
X <- BuildAdjacencyMatrix(Exp1_R25_pept[1:10], protID, FALSE)
ll.agg <- aggregateIter(Exp1_R25_pept[1:10], X=X)
```

---

`aggregateIterParallel xxx`

---

**Description**

Method to xxxxx

**Usage**

```
aggregateIterParallel(  
  obj.pep,  
  X,  
  init.method = "Sum",  
  method = "Mean",  
  n = NULL  
)
```

**Arguments**

<code>obj.pep</code>	<code>xxxxx</code>
<code>X</code>	<code>xxxx</code>
<code>init.method</code>	<code>xxxxx</code>
<code>method</code>	<code>xxxxx</code>
<code>n</code>	<code>xxxx</code>

**Value**

`xxxxx`

**Author(s)**

Samuel Wieczorek

**Examples**

```
## Not run:  
library(MSnbase)  
utils::data(Exp1_R25_pept, package='DAPARdata')  
protID <- "Protein_group_IDs"  
obj.pep <- Exp1_R25_pept[1:10]  
X <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)  
obj.agg <- aggregateIterParallel(obj.pep, X)  
  
## End(Not run)
```

---

aggregateMean	<i>Compute the intensity of proteins as the mean of the intensities of their peptides.</i>
---------------	--

---

**Description**

This function computes the intensity of proteins as the mean of the intensities of their peptides.

**Usage**

```
aggregateMean(obj.pep, X)
```

**Arguments**

obj.pep	A peptide object of class MSnset
X	An adjacency matrix in which lines and columns correspond respectively to peptides and proteins.

**Value**

A matrix of intensities of proteins

**Author(s)**

Alexia Dorffer

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.pep <- Exp1_R25_pept[1:10]
obj.pep.imp <- wrapper.impute.detQuant(obj.pep, na.type='missing')
protID <- obj.pep@experimentData@other$proteinId
X <- BuildAdjacencyMatrix(obj.pep.imp, protID, FALSE)
ll.agg <- aggregateMean(obj.pep.imp, X)
```

---

AggregateMetacell	<i>Symbolic product of matrices</i>
-------------------	-------------------------------------

---

**Description**

Execute a product two matrices: the first is an adjacency one while the second if a simple dataframe

**Usage**

```
AggregateMetacell(X, obj.pep)
```



**Arguments**

X                    An adjacency matrix between peptides and proteins  
obj.pep             A dataframe of the cell metadata for peptides

**Value**

xxxx

**Author(s)**

Samuel Wieczorek

**Examples**

```
library(MSnbase)
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.pep <- Exp1_R25_pept[1:100]
protID <- "Protein_group_IDs"
X <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)
agg.meta <- AggregateMetacell(X, obj.pep)
```

---

aggregateSum                    *Compute the intensity of proteins with the sum of the intensities of their peptides.*

---

**Description**

This function computes the intensity of proteins based on the sum of the intensities of their peptides.

**Usage**

```
aggregateSum(obj.pep, X)
```

**Arguments**

obj.pep             A matrix of intensities of peptides  
X                    An adjacency matrix in which lines and columns correspond respectively to peptides and proteins.

**Value**

A matrix of intensities of proteins

**Author(s)**

Alexia Dorffer

**Examples**

```
library(MSnbase)
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.pep <- Exp1_R25_pept[1:20]
obj.pep.imp <- wrapper.impute.detQuant(obj.pep, na.type='missing')
protID <- obj.pep@experimentData@other$proteinId
X <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)
ll.agg <- DAPAR::aggregateSum(obj.pep.imp, X)
```

---

aggregateTopn	<i>Compute the intensity of proteins as the sum of the intensities of their n best peptides.</i>
---------------	--

---

**Description**

This function computes the intensity of proteins as the sum of the intensities of their n best peptides.

**Usage**

```
aggregateTopn(obj.pep, X, method = "Mean", n = 10)
```

**Arguments**

obj.pep	A matrix of intensities of peptides
X	An adjacency matrix in which lines and columns correspond respectively to peptides and proteins.
method	xxx
n	The maximum number of peptides used to aggregate a protein.

**Value**

A matrix of intensities of proteins

**Author(s)**

Alexia Dorffer, Samuel Wiczorek

**Examples**

```
## Not run:
library(MSnbase)
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.pep <- Exp1_R25_pept[1:10]
protID <- "Protein_group_IDs"
X <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)
ll.agg <- DAPAR::aggregateTopn(obj.pep, X, n=3)
```

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

---

averageIntensities	<i>Average protein/peptide abundances for each condition studied</i>
--------------------	--

---

### Description

Calculate the average of the abundances for each protein in each condition for an ExpressionSet or MSnSet. Needs to have the array expression data ordered in the same way as the phenotype data (columns of the array data in the same order than the condition column in the phenotype data).

### Usage

```
averageIntensities(ESet_obj)
```

### Arguments

ESet\_obj          ExpressionSet object containing all the data

### Value

a dataframe in wide format providing (in the case of 3 or more conditions) the means of intensities for each protein/peptide in each condition. If there are less than 3 conditions, an error message is returned.

### Author(s)

Helene Borges

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
averageIntensities(obj$new)
```

---

barplotEnrichGO_HC	<i>A barplot that shows the result of a GO enrichment, using the package highcharter</i>
--------------------	--

---

**Description**

A barplot of GO enrichment analysis

**Usage**

```
barplotEnrichGO_HC(ego, maxRes = 5, title = NULL)
```

**Arguments**

ego	The result of the GO enrichment, provides either by the function enrichGO in the package DAPAR or the function enrichGO of the package ‘clusterProfiler’
maxRes	The maximum number of categories to display in the plot
title	The title of the plot

**Value**

A barplot

**Author(s)**

Samuel Wieczorek

---

barplotGroupGO_HC	<i>A barplot which shows the result of a GO classification, using the package highcharter</i>
-------------------	---

---

**Description**

A barplot of GO classification analysis

**Usage**

```
barplotGroupGO_HC(ggo, maxRes = 5, title = "")
```

**Arguments**

ggo	The result of the GO classification, provides either by the function group_GO in the package DAPAR or the function groupGO in the package ‘clusterProfiler’
maxRes	An integer which is the maximum number of classes to display in the plot
title	The title of the plot

**Value**

A barplot

**Author(s)**

Samuel Wieczorek

---

boxPlotD\_HC

*Builds a boxplot from a dataframe using the library highcharter*

---

**Description**

Boxplot for quantitative proteomics data using the library highcharter

**Usage**

```
boxPlotD_HC(  
  obj,  
  conds,  
  keyId = NULL,  
  legend = NULL,  
  pal = NULL,  
  subset.view = NULL  
)
```

**Arguments**

obj	Numeric matrix
conds	xxx
keyId	xxxx
legend	A vector of the conditions (one condition per sample).
pal	A basis palette for the boxes which length must be equal to the number of unique conditions in the dataset.
subset.view	A vector of index indicating which rows to highlight

**Value**

A boxplot

**Author(s)**

Samuel Wieczorek, Anais Courtier, Enora Fremy

## Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot
conds <- legend <- Biobase::pData(obj)$Condition
key <- "Protein_IDs"
boxPlotD_HC(obj, conds, key, legend, NULL, 1:10)
pal <- ExtendPalette(length(unique(conds)))
boxPlotD_HC(obj, conds, key, legend, pal, 1:10)
```

---

BuildAdjacencyMatrix *Function matrix of appartenance group*

---

## Description

Method to create a binary matrix with proteins in columns and peptides in lines on a MSnSet object (peptides)

## Usage

```
BuildAdjacencyMatrix(obj.pep, protID, unique = TRUE)
```

## Arguments

obj.pep	An object (peptides) of class MSnSet.
protID	The name of proteins ID column
unique	A boolean to indicate whether only the unique peptides must be considered (TRUE) or if the shared peptides have to be integrated (FALSE).

## Value

A binary matrix

## Author(s)

Florence Combes, Samuel Wiczorek, Alexia Dorffer

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
protId <- Exp1_R25_pept@experimentData@other$proteinId
BuildAdjacencyMatrix(Exp1_R25_pept[1:10], protId, TRUE)
```

---

BuildColumnToProteinDataset

*creates a column for the protein dataset after agregation by using the previous peptide dataset.*

---

## Description

This function creates a column for the protein dataset after aggregation by using the previous peptide dataset.

## Usage

```
BuildColumnToProteinDataset(peptideData, matAdj, columnName, proteinNames)
```

## Arguments

peptideData	A data.frame of meta data of peptides. It is the fData of the MSnset object.
matAdj	The adjacency matrix used to agregate the peptides data.
columnName	The name of the column in Biobase::fData(peptides_MSnset) that the user wants to keep in the new protein data.frame.
proteinNames	The names of the protein in the new dataset (i.e. rownames)

## Value

A vector

## Author(s)

Samuel Wieczorek

## Examples

```
library(MSnbase)
utils::data(Exp1_R25_pept, package='DAPARdata')
protID <- "Protein_group_IDs"
obj.pep <- Exp1_R25_pept[1:10]
M <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)
data <- Biobase::fData(obj.pep)
protData <- DAPAR::aggregateMean(obj.pep, M)
name <- "Protein_group_IDs"
proteinNames <- rownames(Biobase::fData(protData$obj.prot))
new.col <- BuildColumnToProteinDataset(data, M, name, proteinNames )
```

---

BuildColumnToProteinDataset\_par

*creates a column for the protein dataset after agregation by using the previous peptide dataset.*

---

### Description

This function creates a column for the protein dataset after agregation by using the previous peptide dataset. It is a parallel version of the function BuildColumnToProteinDataset

### Usage

```
BuildColumnToProteinDataset_par(peptideData, matAdj, columnName, proteinNames)
```

### Arguments

peptideData	A data.frame of meta data of peptides. It is the fData of the MSnset object.
matAdj	The adjacency matrix used to agregate the peptides data.
columnName	The name of the column in Biobase::fData(peptides_MSnset) that the user wants to keep in the new protein data.frame.
proteinNames	The names of the protein in the new dataset (i.e. rownames)

### Value

A vector

### Author(s)

Samuel Wieczorek

### Examples

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
protID <- "Protein_group_IDs"
obj.pep <- Exp1_R25_pept[1:10]
M <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)
data <- Biobase::Biobase::fData(obj.pep)
protData <- DAPAR::aggregateSum(obj.pep, M)
name <- "Protein_group_IDs"
proteinNames <- rownames(Biobase::fData(protData$obj.prot))
BuildColumnToProteinDataset_par(data, M, name,proteinNames )

## End(Not run)
```



---

buildGraph	<i>Display a CC</i>
------------	---------------------

---

**Description**

Display a CC

**Usage**

```
buildGraph(The.CC, X)
```

**Arguments**

The.CC	A cc (a list)
X	xxxxx

**Value**

A plot

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
X <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", FALSE)
ll <- get.pep.prot.cc(X)
g <- buildGraph(ll[[1]], X)
```

---

BuildMetaCell	xxxx
---------------	------

---

**Description**

xxxxxx

**Usage**

```
BuildMetaCell(from, level, qdata = NULL, conds = NULL, df = NULL)
```

**Arguments**

from	xxx
level	xxx
qdata	An object of class MSnSet
conds	xxx
df	A list of integer xxxxxxxx

**Value**

xxxxxx

**Author(s)**

Samuel Wieczorek

**Examples**

```
file <- system.file("extdata", "Exp1_R25_pept.txt", package="DAPARdata")
data <- read.table(file, header=TRUE, sep="\t", stringsAsFactors = FALSE)
metadataFile <- system.file("extdata", "samples_Exp1_R25.txt",
package="DAPARdata")
metadata <- read.table(metadataFile, header=TRUE, sep="\t", as.is=TRUE,
stringsAsFactors = FALSE)
conds <- metadata$Condition
qdata <- data[,56:61]
df <- data[, 43:48]
df <- BuildMetaCell(from = 'maxquant', level='peptide', qdata = qdata,
conds = conds, df = df)
df <- BuildMetaCell(from = 'proline', level='peptide', qdata = qdata,
conds = conds, df = df)
```

---

check.conditions

*Check if the design is valid*

---

**Description**

This function check the validity of the conditions

**Usage**

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

**Arguments**

conds	A vector
-------	----------

**Value**

A list

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
check.conditions(Biobase::pData(Exp1_R25_pept)$Condition)
```

---

check.design	<i>Check if the design is valid</i>
--------------	-------------------------------------

---

**Description**

This function check the validity of the experimental design

**Usage**

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

**Arguments**

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

**Value**

A boolean

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
check.design(Biobase::pData(Exp1_R25_pept)[,1:3])
```

---

checkClusterability    *Prepare the data for subsequent clustering*

---

### Description

The first step is to standardize the data (with the Mfuzz package). Then the function checks that these data are clusterizable or not (use of [dipetest::dip.test()] to determine whether the distribution is unimodal or #' multimodal). Finally, it determines the "optimal" k by the Gap statistic approach.

### Usage

```
checkClusterability(standards, b = 500)
```

### Arguments

standards	a matrix or dataframe containing only the standardized mean intensities returned by the function [standardiseMeanIntensities()]
b	Parameter B of the function [gap_cluster()]

### Value

a list of 2 elements: \* dip\_test: the result of the clusterability of the data \* gap\_cluster: the gap statistic obtained with the function [cluster::clusGap()].

### Author(s)

Helene Borges

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:100]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
averaged_means <- averageIntensities(obj$new)
only_means <- dplyr::select_if(averaged_means, is.numeric)
only_features <- dplyr::select_if(averaged_means, is.character)
means <- purrr::map(purrr::array_branch(as.matrix(only_means), 1), mean)
centered <- only_means - unlist(means)
centered_means <- dplyr::bind_cols(feature = dplyr::as_tibble(only_features),
dplyr::as_tibble(centered))
checkClust <- checkClusterability(centered_means, b=100)
```

---

classic1wayAnova	<i>Function to perform a One-way Anova statistical test on a MsnBase dataset</i>
------------------	--

---

**Description**

Function to perform a One-way Anova statistical test on a MsnBase dataset

**Usage**

```
classic1wayAnova(current_line, conditions)
```

**Arguments**

current_line	The line currently treated from the quantitative data to perform the ANOVA
conditions	The conditions represent the different classes of the studied factor

**Value**

A named vector containing all the different values of the aov model

**Author(s)**

Hélène Borges

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
anova_tests <- t(apply(Biobase::exprs(obj$new), 1, classic1wayAnova,
conditions=as.factor(Biobase::pData(obj$new)$Condition)))
```

---

compareNormalizationD\_HC

*Builds a plot from a dataframe. Same as compareNormalizationD but uses the library highcharter*

---

**Description**

Plot to compare the quantitative proteomics data before and after normalization using the library highcharter

**Usage**

```
compareNormalizationD_HC(
  qDataBefore,
  qDataAfter,
  keyId = NULL,
  conds = NULL,
  pal = NULL,
  subset.view = NULL,
  n = 100,
  type = "scatter"
)
```

**Arguments**

qDataBefore	A dataframe that contains quantitative data before normalization.
qDataAfter	A dataframe that contains quantitative data after normalization.
keyId	xxx
conds	A vector of the conditions (one condition per sample).
pal	xxx
subset.view	xxx
n	An integer that is equal to the maximum number of displayed points. This number must be less or equal to the size of the dataset. If it is less than it, it is a random selection
type	scatter or line

**Value**

A plot

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot
qDataBefore <- Biobase::exprs(obj)
conds <- Biobase::pData(obj)[,"Condition"]
id <- Biobase::fData(obj)[,obj@experimentData@other$proteinId]
pal <- ExtendPalette(2)
objAfter <- wrapper.normalizeD(obj, method = "QuantileCentering",
  conds =conds, type = "within conditions")
compareNormalizationD_HC(qDataBefore=qDataBefore,
  qDataAfter=Biobase::exprs(objAfter), keyId = id, conds=conds, n=1000)

compareNormalizationD_HC(qDataBefore=qDataBefore,
  qDataAfter=Biobase::exprs(objAfter), keyId = id, pal=pal, subset.view=1:4,
```

```
conds=conds, n=100)
```

---

```
compute_t_tests      xxxxxx
```

---

## Description

This function is xxxxxx

## Usage

```
compute_t_tests(obj, contrast = "OnevsOne", type = "Student")
```

## Arguments

obj	A matrix of quantitative data, without any missing values.
contrast	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).
type	xxxxx

## Value

A list of two items : logFC and P\_Value; both are dataframe. 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)

Florence Combes, Samuel Wiczorek

## Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
ttest <- compute_t_tests(obj$new)
```

---

corrMatrixD_HC	<i>Displays a correlation matrix of the quantitative data of the Biobase::exprs() table.</i>
----------------	--

---

### Description

Correlation matrix based on a MSnSet object.

### Usage

```
corrMatrixD_HC(object, samplesData = NULL, rate = 0.5, showValues = TRUE)
```

### Arguments

object	The result of the cor function.
samplesData	A dataframe in which lines correspond to samples and columns to the meta-data for those samples.
rate	The rate parameter to control the exponential law for the gradient of colors
showValues	xxx

### Value

A colored correlation matrix

### Author(s)

Samuel Wieczorek

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
samplesData <- Biobase::pData(Exp1_R25_pept)
res <- cor(qData,use = 'pairwise.complete.obs')
corrMatrixD_HC(res, samplesData)
```



---

CountPep	<i>Compute the number of peptides used to aggregate proteins</i>
----------	--

---

**Description**

This function computes the number of peptides used to aggregate proteins.

**Usage**

```
CountPep(M)
```

**Arguments**

M                    A "valued" adjacency matrix in which lines and columns correspond respectively to peptides and proteins.

**Value**

A vector of boolean which is the adjacency matrix but with NA values if they exist in the intensity matrix.

**Author(s)**

Alexia Dorffer

**Examples**

```
library(DAPARdata)
utils::data(Exp1_R25_pept)
protID <- "Protein_group_IDs"
M <- BuildAdjacencyMatrix(Exp1_R25_pept[1:10], protID, FALSE)
CountPep(M)
```

---

createMSnset	<i>Creates an object of class MSnSet from text file</i>
--------------	---

---

**Description**

Builds an object of class MSnSet from a single tabulated-like file for quantitative and meta-data and a dataframe for the samples description. It differs from the original MSnSet builder which requires three separated files tabulated-like quantitative proteomic data into a MSnSet object, including meta-data.

**Usage**

```
createMSnset(
  file,
  metadata = NULL,
  indExpData,
  colnameForID = NULL,
  indexForMetacell = NULL,
  logData = FALSE,
  replaceZeros = FALSE,
  pep_prot_data = NULL,
  proteinId = NULL,
  software = NULL
)
```

**Arguments**

file	The name of a tab-separated file that contains the data.
metadata	A dataframe describing the samples (in lines).
indExpData	A vector of string where each element is the name of a column in designTable that have to be integrated in the Biobase::fData() table of the MSnSet object.
colnameForID	The name of the column containing the ID of entities (peptides or proteins)
indexForMetacell	xxxxxxxxxxxx
logData	A boolean value to indicate if the data have to be log-transformed (Default is FALSE)
replaceZeros	A boolean value to indicate if the 0 and NaN values of intensity have to be replaced by NA (Default is FALSE)
pep_prot_data	A string that indicates whether the dataset is about
proteinId	xxxx
software	xxx

**Value**

An instance of class MSnSet.

**Author(s)**

Florence Combes, Samuel Wieczorek

**Examples**

```
require(Matrix)
exprsFile <- system.file("extdata", "Exp1_R25_pept.txt", package="DAPARdata")
metadataFile <- system.file("extdata", "samples_Exp1_R25.txt",
  package="DAPARdata")
metadata = read.table(metadataFile, header=TRUE, sep="\t", as.is=TRUE)
indExpData <- c(56:61)
```

```

colnameForID <- 'id'
obj <- createMSnset(exprsFile, metadata, indExpData, colnameForID,
indexForMetacell = c(43:48), pep_prot_data = "peptide", software = 'maxquant')

exprsFile <- system.file("extdata", "Exp1_R25_pept.txt", package="DAPARdata")
metadataFile <- system.file("extdata", "samples_Exp1_R25.txt", package="DAPARdata")
metadata = read.table(metadataFile, header=TRUE, sep="\t", as.is=TRUE)
indExpData <- c(56:61)
colnameForID <- 'AutoID'
obj <- createMSnset(exprsFile, metadata, indExpData, colnameForID,
indexForMetacell = c(43:48), pep_prot_data = "peptide", software = 'maxquant')

```

CVDistD\_HC

*Distribution of CV of entities***Description**

Builds a densityplot of the CV of entities in the `Biobase::exprs()` table of a object. The CV is calculated for each condition present in the dataset (see the slot 'Condition' in the `Biobase::pData()` table)

**Usage**

```
CVDistD_HC(qData, conds = NULL, pal = NULL)
```

**Arguments**

<code>qData</code>	A dataframe that contains quantitative data.
<code>conds</code>	A vector of the conditions (one condition per sample).
<code>pal</code>	xxx

**Value**

A density plot

**Author(s)**

Samuel Wieczorek

**Examples**

```

utils::data(Exp1_R25_pept, package='DAPARdata')
conds <- Biobase::pData(Exp1_R25_pept)[,"Condition"]
CVDistD_HC(Biobase::exprs(Exp1_R25_pept), conds)
pal <- ExtendPalette(2, 'Dark2')
CVDistD_HC(Biobase::exprs(Exp1_R25_pept), conds, pal)

```

---

dapar\_hc\_chart      *Customised resetZoomButton of highcharts plots*

---

### Description

Customised resetZoomButton of highcharts plots

### Usage

```
dapar_hc_chart(hc, chartType, zoomType = "None", width = 0, height = 0)
```

### Arguments

hc	A highcharter object
chartType	The type of the plot
zoomType	The type of the zoom (one of "x", "y", "xy", "None")
width	xxx
height	xxx

### Value

A highchart plot

### Author(s)

Samuel Wiczorek

### Examples

```
library("highcharter")
hc <- highchart()
hc <- dapar_hc_chart(hc, chartType='line', zoomType='x')
hc_add_series(hc, data = c(29, 71, 40))
```

---

dapar\_hc\_ExportMenu      *Customised contextual menu of highcharts plots*

---

### Description

Customised contextual menu of highcharts plots

### Usage

```
dapar_hc_ExportMenu(hc, filename)
```

**Arguments**

hc                    A highcharter object  
 filename            The filename under which the plot has to be saved

**Value**

A contextual menu for highcharts plots

**Author(s)**

Samuel Wieczorek

**Examples**

```
library("highcharter")
hc <- highchart()
hc_chart(hc,type = "line")
hc_add_series(hc,data = c(29, 71, 40))
dapar_hc_ExportMenu(hc,filename='foo')
```

---

deleteLinesFromIndices

*Delete the lines in the matrix of intensities and the metadata table given their indice.*

---

**Description**

Delete the lines of Biobase::exprs() table identified by their indice.

**Usage**

```
deletelinesFromIndices(obj, deleteThat = NULL, processText = "")
```

**Arguments**

obj                    An object of class MSnSet containing quantitative data.  
 deleteThat           A vector of integers which are the indices of lines to delete.  
 processText          A string to be included in the MSnSet object for log.

**Value**

An instance of class MSnSet that have been filtered.

**Author(s)**

Florence Combes, Samuel Wieczorek

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- deleteLinesFromIndices(Exp1_R25_pept[1:100], c(1:10))
```

---

densityPlotD_HC	<i>Builds a densityplot from a dataframe</i>
-----------------	--

---

## Description

Densityplot of quantitative proteomics data over samples.

## Usage

```
densityPlotD_HC(obj, legend = NULL, pal = NULL)
```

## Arguments

obj	xxx
legend	A vector of the conditions (one condition per sample).
pal	xxx

## Value

A density plot

## Author(s)

Samuel Wieczorek

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
densityPlotD_HC(Exp1_R25_pept)
conds <- Biobase::pData(Exp1_R25_pept)$Condition
pal <- ExtendPalette(2, 'Dark2')
densityPlotD_HC(Exp1_R25_pept, pal=pal)
```

---

diffAnaComputeFDR	<i>Computes the FDR corresponding to the p-values of the differential analysis using</i>
-------------------	--

---

### Description

This function is a wrapper to the function `adjust.p` from the `cp4p` package. It returns the FDR corresponding to the p-values of the differential analysis. The FDR is computed with the function `p.adjust{stats}`.

### Usage

```
diffAnaComputeFDR(
  logFC,
  pval,
  threshold_PVal = 0,
  threshold_LogFC = 0,
  pi0Method = 1
)
```

### Arguments

<code>logFC</code>	The result (logFC values) of the differential analysis processed by <a href="#">limmaCompleteTest</a>
<code>pval</code>	The result (p-values) of the differential analysis processed by <a href="#">limmaCompleteTest</a>
<code>threshold_PVal</code>	The threshold on p-value to distinguish between differential and non-differential data
<code>threshold_LogFC</code>	The threshold on log(Fold Change) to distinguish between differential and non-differential data
<code>pi0Method</code>	The parameter <code>pi0.method</code> of the method <code>adjust.p</code> in the package <code>cp4p</code>

### Value

The computed FDR value (floating number)

### Author(s)

Samuel Wieczorek

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
```

```

qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
limma <- limmaCompleteTest(qData,sTab)
diffAnaComputeFDR(limma$logFC[,1],limma$P_Value[,1])

```

---

`diffAnaGetSignificant` *Returns a MSnSet object with only proteins significant after differential analysis.*

---

### Description

Returns a MSnSet object with only proteins significant after differential analysis.

### Usage

```
diffAnaGetSignificant(obj)
```

### Arguments

`obj` An object of class MSnSet.

### Value

A MSnSet

### Author(s)

Alexia Dorffer

### Examples

```

utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
allComp <- limmaCompleteTest(qData,sTab)
data <- list(logFC=allComp$logFC[1], P_Value = allComp$P_Value[1])
obj$new <- diffAnaSave(obj$new, allComp, data)
signif <- diffAnaGetSignificant(obj$new)

```



---

diffAnaSave	Returns a MSnSet object with the results of the differential analysis performed with <a href="#">limma</a> package.
-------------	---

---

## Description

This method returns a class MSnSet object with the results of differential analysis.

## Usage

```
diffAnaSave(obj, allComp, data = NULL, th_pval = 0, th_logFC = 0)
```

## Arguments

obj	An object of class MSnSet.
allComp	A list of two items which is the result of the function wrapper.limmaCompleteTest or xxxx
data	The result of the differential analysis processed by <a href="#">limmaCompleteTest</a>
th_pval	xxx
th_logFC	xxx

## Value

A MSnSet

## Author(s)

Alexia Dorffer, Samuel Wieczorek

## Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
allComp <- limmaCompleteTest(qData,sTab)
data <- list(logFC=allComp$logFC[1], P_Value = allComp$P_Value[1])
diffAnaSave(obj$new, allComp, data)
```

---

diffAnaVolcanoplot      *Volcanoplot of the differential analysis*

---

### Description

Plots a volcano plot after the differential analysis. Typically, the log of Fold Change is represented on the X-axis and the log<sub>10</sub> of the p-value is drawn on the Y-axis. When the `threshold_pVal` and the `threshold_logFC` are set, two lines are drawn respectively on the y-axis and the X-axis to visually distinguish between differential and non differential data.

### Usage

```
diffAnaVolcanoplot(  
  logFC = NULL,  
  pVal = NULL,  
  threshold_pVal = 1e-60,  
  threshold_logFC = 0,  
  conditions = NULL,  
  colors = NULL  
)
```

### Arguments

<code>logFC</code>	A vector of the log(fold change) values of the differential analysis.
<code>pVal</code>	A vector of the p-value values returned by the differential analysis.
<code>threshold_pVal</code>	A floating number which represents the p-value that separates differential and non-differential data.
<code>threshold_logFC</code>	A floating number which represents the log of the Fold Change that separates differential and non-differential data.
<code>conditions</code>	A list of the names of condition 1 and 2 used for the differential analysis.
<code>colors</code>	xxx

### Value

A volcano plot

### Author(s)

Florence Combes, Samuel Wieczorek

**Examples**

```

utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
limma <- limmaCompleteTest(qData, sTab)
diffAnaVolcanoplot(limma$logFC[,1], limma$P_Value[,1])

```

---

diffAnaVolcanoplot\_rCharts

*Volcanoplot of the differential analysis*


---

**Description**

Plots an interactive volcanoplot after the differential analysis. Typically, the log of Fold Change is represented on the X-axis and the log10 of the p-value is drawn on the Y-axis. When the `threshold_pVal` and the `threshold_logFC` are set, two lines are drawn respectively on the y-axis and the X-axis to visually distinguish between differential and non differential data. With the use of the package `Highcharter`, a customizable tooltip appears when the user put the mouse's pointer over a point of the scatter plot.

**Usage**

```

diffAnaVolcanoplot_rCharts(
  df,
  threshold_pVal = 1e-60,
  threshold_logFC = 0,
  conditions = NULL,
  clickFunction = NULL,
  pal = NULL
)

```

**Arguments**

`df` A dataframe which contains the following slots : `x` : a vector of the log(fold change) values of the differential analysis, `y` : a vector of the p-value values returned by the differential analysis. `index` : a vector of the rownames of the data. This dataframe must has been built with the option `stringsAsFactors` set to `FALSE`. There may be additional slots which will be used to show informations in the tooltip. The name of these slots must begin with the prefix `"tooltip_"`. It will be automatically removed in the plot.

`threshold_pVal` A floating number which represents the p-value that separates differential and non-differential data.

threshold_logFC	A floating number which represents the log of the Fold Change that separates differential and non-differential data.
conditions	A list of the names of condition 1 and 2 used for the differential analysis.
clickFunction	A string that contains a JavaScript function used to show info from slots in df. The variable <code>this.index</code> refers to the slot named <code>index</code> and allows to retrieve the right row to show in the tooltip.
pal	xxx

**Value**

An interactive volcano plot

**Author(s)**

Samuel Wieczorek

**Examples**

```
## Not run:
library(highcharter)
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
data <- limmaCompleteTest(qData,sTab)
df <- data.frame(x=data$logFC, y = -log10(data$P_Value),
index = as.character(rownames(obj$new)))
colnames(df) <- c("x", "y", "index")
tooltipSlot <- c("Sequence", "Score")
df <- cbind(df, Biobase::fData(obj)[, tooltipSlot])
colnames(df) <- gsub(".", "_", colnames(df), fixed=TRUE)
if (ncol(df) > 3){
  colnames(df)[4:ncol(df)] <-
  paste("tooltip_", colnames(df)[4:ncol(df)], sep="")}
hc_clickFunction <- JS("function(event) {
Shiny.onInputChange('eventPointClicked',
[this.index]+'_'+ [this.series.name]);}")
cond <- c("25fmol", "10fmol")
diffAnaVolcanoplot_rCharts(df, 2.5, 1, cond,hc_clickFunction)

## End(Not run)
```

---

display.CC.visNet      *Display a CC*

---

**Description**

Display a CC

**Usage**

```
display.CC.visNet(  
  g,  
  layout = layout_nicely,  
  obj = NULL,  
  prot.tooltip = NULL,  
  pept.tooltip = NULL  
)
```

**Arguments**

g	A cc (a list)
layout	xxxxx
obj	xxx
prot.tooltip	xxx
pept.tooltip	xxx

**Value**

A plot

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
X <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", FALSE)  
ll <- get.pep.prot.cc(X)  
g <- buildGraph(ll[[1]], X)  
display.CC.visNet(g)
```

---

enrich_GO	<i>Calculates GO enrichment classes for a given list of proteins/genes ID. It results an enrichResult instance.</i>
-----------	---

---

### Description

This function is a wrapper to the function `enrichGO` from the package ‘clusterProfiler’. Given a vector of genes/proteins, it returns an `enrichResult` instance.

### Usage

```
enrich_GO(data, idFrom, orgdb, ont, readable = FALSE, pval, universe)
```

### Arguments

<code>data</code>	A vector of ID (among ENSEMBL, ENTREZID, GENENAME, REFSEQ, UNIGENE, UNIPROT -can be different according to organisms)
<code>idFrom</code>	character indicating the input ID format (among ENSEMBL, ENTREZID, GENENAME, REFSEQ, UNIGENE, UNIPROT)
<code>orgdb</code>	annotation Bioconductor package to use (character format)
<code>ont</code>	One of "MF", "BP", and "CC" subontologies
<code>readable</code>	TRUE or FALSE (default FALSE)
<code>pval</code>	The qvalue cutoff (same parameter as in the function <code>enrichGO</code> of the package ‘clusterProfiler’)
<code>universe</code>	a list of ID to be considered as the background for enrichment calculation

### Value

A `groupGOResult` instance.

### Author(s)

Florence Combes

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot
univ<-univ_AnnotDbPkg("org.Sc.sgd.db") #univ is the background
ego<-enrich_GO(data=Biobase::fData(obj)$Protein.IDs, idFrom="UNIPROT",
orgdb="org.Sc.sgd.db",ont="MF", pval=0.05, universe = univ)
```

---

ExtendPalette	<i>Extends a base-palette of the package RColorBrewer to n colors.</i>
---------------	--

---

### Description

The colors in the returned palette are always in the same order

### Usage

```
ExtendPalette(n = NULL, base = "Set1")
```

### Arguments

n	The number of desired colors in the palette
base	The name of the palette of the package RColorBrewer from which the extended palette is built. Default value is 'Set1'.

### Value

A vector composed of n color code.

### Author(s)

Samuel Wieczorek

### Examples

```
ExtendPalette(12)
nPalette <- 10
par(mfrow=c(nPalette,1))
par(mar=c(0.5, 4.5, 0.5, 0.5))
for (i in 1:nPalette){
  pal <- ExtendPalette(n=i, base = 'Dark2')
  barplot(1:length(pal), col=pal)
  print(pal)
}
```

finalizeAggregation     *Finalizes the aggregation process*

---

**Description**

Method to finalize the aggregation process

**Usage**

```
finalizeAggregation(obj.pep, pepData, protData, protMetacell, X)
```

**Arguments**

obj.pep	A peptide object of class MSnset
pepData	xxxx
protData	xxxxx
protMetacell	xxx
X	An adjacency matrix in which lines and columns correspond respectively to peptides and proteins.

**Value**

A protein object of class MSnset

**Author(s)**

Samuel Wieczorek

---

findMECBlock     *Finds the LAPALA into a MSnSet object*

---

**Description**

This method finds the LAPALA in a dataset.

**Usage**

```
findMECBlock(obj)
```

**Arguments**

obj	An object of class MSnSet.
-----	----------------------------

**Value**

A data.frame that contains the indexes of LAPALA



**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:100]
lapala <- findMECBlock(obj)
```

---

formatLimmaResult	xxxx
-------------------	------

---

**Description**

xxxx

**Usage**

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

**Arguments**

fit	xxxx
conds	xxxx
contrast	xxxx

**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)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
limma <- limmaCompleteTest(qData, sTab)
```

---

formatPHResults	<i>Extract logFC and raw pvalues from multiple post-hoc models summaries</i>
-----------------	--

---

### Description

Extract logFC and raw pvalues from multiple post-hoc models summaries

### Usage

```
formatPHResults(post_hoc_models_summaries)
```

### Arguments

post\_hoc\_models\_summaries  
a list of summaries of post-hoc models.

### Value

a list of 2 dataframes containing the logFC values and pvalues for each comparison.

### Author(s)

Hélène Borges

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
anova_tests <- t(apply(Biobase::exprs(obj$new),1, classic1wayAnova,
conditions=as.factor(Biobase::pData(obj$new)$Condition)))
names(anova_tests) <- rownames(Biobase::exprs(obj$new))
tms <- lapply(anova_tests,
              function(x) summary(multcomp::glht(x,
linfct = multcomp::mcp(conditions = "Tukey"),
test = multcomp::adjusted("none"))))
res <- formatPHResults(tms)
```

---

fudge2LRT	<i>Heuristic to choose the value of the hyperparameter (fudge factor) used to regularize the variance estimator in the likelihood ratio statistic</i>
-----------	---

---

### Description

fudge2LRT: heuristic to choose the value of the hyperparameter (fudge factor) used to regularize the variance estimator in the likelihood ratio statistic (as implemented in samLRT). We follow the heuristic described in [1] and adapt the code of the fudge2 function in the siggene R package. [1] Tusher, Tibshirani and Chu, Significance analysis of microarrays applied to the ionizing radiation response, PNAS 2001 98: 5116-5121, (Apr 24).

### Usage

```
fudge2LRT(
  lmm.res.h0,
  lmm.res.h1,
  cc,
  n,
  p,
  s,
  alpha = seq(0, 1, 0.05),
  include.zero = TRUE
)
```

### Arguments

lmm.res.h0	a vector of object containing the estimates (used to compute the statistic) under H0 for each connected component. If the fast version of the estimator was used (as implemented in this package), lmm.res.h0 is a vector containing averages of squared residuals. If a fixed effect model was used, it is a vector of lm objects and if a mixed effect model was used it is a vector or lmer object.
lmm.res.h1	similar to lmm.res.h0, a vector of object containing the estimates (used to compute the statistic) under H1 for each protein.
cc	a list containing the indices of peptides and proteins belonging to each connected component.
n	the number of samples used in the test
p	the number of proteins in the experiment
s	a vector containing the maximum likelihood estimate of the variance for the chosen model. When using the fast version of the estimator implemented in this package, this is the same thing as the input lmm.res.h1. For other models (e.g. mixed models) it can be obtained from samLRT.
alpha	A vector of proportions used to build candidate values for the regularizer. We use quantiles of s with these proportions. Default to seq(0, 1, 0.05)
include.zero	logical value indicating if 0 should be included in the list of candidates. Default to TRUE.

**Value**

(same as the fudge2 function of siggene): s.zero: the value of the fudge factor s0. alpha.hat: the optimal quantile of the 's' values. If s0=0, 'alpha.hat' will not be returned. vec.cv: the vector of the coefficients of variations. Following Tusher et al. (2001), the optimal 'alpha' quantile is given by the quantile that leads to the smallest CV of the modified test statistics. msg: a character string summarizing the most important information about the fudge factor.

**Author(s)**

Thomas Burger, Laurent Jacob

---

get.pep.prot.cc

*Build the list of connex component of the adjacency matrix*

---

**Description**

Method to build the list of connex component of the adjacency matrix

**Usage**

```
get.pep.prot.cc(X)
```

**Arguments**

X                    An adjacency matrix

**Value**

A list of CC

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
X <- BuildAdjacencyMatrix(Exp1_R25_pept, "Protein_group_IDs", FALSE)
ll <- get.pep.prot.cc(X)
```

---

GetCC	<i>Returns the contains of the slot processing of an object of class MSnSet</i>
-------	---

---

**Description**

Returns the contains of the slot processing of an object of class MSnSet

**Usage**

```
GetCC(obj)
```

**Arguments**

obj                    An object (peptides) of class MSnSet.

**Value**

A list of connected components

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
Xshared <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", FALSE)
Xunique <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", TRUE)
l1.X <- list(matWithSharedPeptides = Xshared, matWithUniquePeptides = Xunique)
Exp1_R25_pept <- SetMatAdj(Exp1_R25_pept, l1.X)
l11 <- get.pep.prot.cc(GetMatAdj(Exp1_R25_pept)$matWithSharedPeptides)
l12 <- DAPAR::get.pep.prot.cc(GetMatAdj(Exp1_R25_pept)$matWithUniquePeptides)
cc <- list(allPep = l11, onlyUniquePep = l12)
Exp1_R25_pept <- SetCC(Exp1_R25_pept, cc)
l1.cc <- GetCC(Exp1_R25_pept)
```

---

GetColorsForConditions
------------------------

---

*Builds a complete color palette for the conditions given in argument*

---

**Description**

xxxx

**Usage**

```
GetColorsForConditions(conds, pal = NULL)
```

**Arguments**

conds	The extended vector of samples conditions
pal	A vector of HEX color code that form the basis palette from which to build the complete color vector for the conditions.

**Value**

A vector composed of HEX color code for the conditions

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
conditions <- Biobase::pData(Exp1_R25_pept)$Condition
GetColorsForConditions(conditions, ExtendPalette(2))
```

---

GetDetailedNbPeptides *Computes the detailed number of peptides for each protein*

---

**Description**

Method to compute the detailed number of quantified peptides for each protein

**Usage**

```
GetDetailedNbPeptides(X)
```

**Arguments**

X	An adjacency matrix
---	---------------------

**Value**

A data.frame

**Author(s)**

Samuel Wiczorek

## Examples

```
library(MSnbase)
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.pep <- Exp1_R25_pept[1:10]
protID <- "Protein_group_IDs"
X <- BuildAdjacencyMatrix(obj.pep, protID, FALSE)
n <- GetDetailedNbPeptidesUsed(X)
```

---

GetDetailedNbPeptidesUsed

*Computes the detailed number of peptides used for aggregating each protein*

---

## Description

Method to compute the detailed number of quantified peptides used for aggregating each protein

## Usage

```
GetDetailedNbPeptidesUsed(X, qdata.pep)
```

## Arguments

X	An adjacency matrix
qdata.pep	A data.frame of quantitative data

## Value

A list of two items

## Author(s)

```
Samuel Wieczorek library(MSnbase) utils::data(Exp1_R25_pept, package='DAPARdata') protID
<- "Protein_group_IDs" X <- BuildAdjacencyMatrix(Exp1_R25_pept[1:10], protID, FALSE) ll.n
<- GetDetailedNbPeptidesUsed(X, Biobase::exprs(Exp1_R25_pept[1:10]))
```

---

`getIndicesConditions` *Gets the conditions indices.*

---

**Description**

Returns a list for the two conditions where each slot is a vector of indices for the samples.

**Usage**

```
getIndicesConditions(conds, cond1, cond2)
```

**Arguments**

<code>conds</code>	A vector of strings containing the column "Condition" of the <code>Biobase::pData()</code> .
<code>cond1</code>	A vector of Conditions (a slot in the <code>Biobase::pData()</code> table) for the condition 1.
<code>cond2</code>	A vector of Conditions (a slot in the <code>Biobase::pData()</code> table) for the condition 2.

**Value**

A list with two slots `iCond1` and `iCond2` containing respectively the indices of samples in the `Biobase::pData()` table of the dataset.

**Author(s)**

Florence Combes, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
conds <- Biobase::pData(Exp1_R25_pept)[,"Condition"]
getIndicesConditions(conds, "25fmol", "10fmol")
```

---

`getIndicesOfLinesToRemove`

*Get the indices of the lines to delete, based on a prefix string*

---

**Description**

This function returns the indice of the lines to delete, based on a prefix string

**Usage**

```
getIndicesOfLinesToRemove(obj, idLine2Delete = NULL, prefix = NULL)
```



**Arguments**

obj	An object of class MSnSet.
idLine2Delete	The name of the column that correspond to the data to filter
prefix	A character string that is the prefix to find in the data

**Value**

A vector of integers.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
ind <- getIndicesOfLinesToRemove(Exp1_R25_pept[1:100], "Potential_contaminant",
prefix="+")
```

---

GetIndices\_BasedOnConditions

*Search lines which respects request on one or more conditions.*

---

**Description**

This function looks for the lines that respect the request in either all conditions or at least one condition.

**Usage**

```
GetIndices_BasedOnConditions(metacell.mask, type, conds, percent, op, th)
```

**Arguments**

metacell.mask	xxx
type	Available values are: * 'AllCond' (the query is valid in all the conditions), * 'AtLeatOneCond' (the query is valid in at least one condition).
conds	xxx
percent	xxx
op	String for operator to use. List of operators is available with SymFilteringOperators().

```

th          The threshold to apply
           #' @examples utils::data(Exp1_R25_pept, package='DAPARdata') obj <- Exp1_R25_pept[1:10]
           level <- GetTypeofData(obj) pattern <- 'missing' metacell.mask <- match.metacell(metadata=GetMetacell
           pattern=pattern, level=level) type <- 'AllCond' conds <- Biobase::pData(obj)$Condition
           op <- '>=' th <- 0.5 percent <- TRUE ind <- GetIndices_BasedOnConditions(metacell.mask,
           type, conds, percent, op, th)

```

---

GetIndices\_MetacellFiltering

*Delete the lines in the matrix of intensities and the metadata table given their indice.*

---

### Description

Delete the lines in the matrix of intensities and the metadata table given their indice.

### Usage

```
GetIndices_MetacellFiltering(obj, level, pattern, type, percent, op, th)
```

### Arguments

obj	An object of class MSnSet containing quantitative data.
level	A vector of integers which are the indices of lines to delete.
pattern	A string to be included in the MSnSet object for log.
type	xxx
percent	xxx
op	xxx
th	xxx

### Value

An instance of class MSnSet that have been filtered.

### Author(s)

Samuel Wieczorek

### Examples

```

utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10,]
level <- GetTypeofData(obj)
pattern <- 'missing'
type <- 'AllCond'
percent <- FALSE

```

```
op <- '=='  
th <- 2  
indices <- GetIndices_MetacellFiltering(obj, level, pattern, type, percent, op, th )
```

---

GetIndices\_WholeLine *Search lines which respects query on all their elements.*

---

### Description

This function looks for the lines where each element respect the query.

### Usage

```
GetIndices_WholeLine(metacell.mask)
```

### Arguments

metacell.mask xxx

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
obj <- Exp1_R25_pept[20:30]  
level <- obj@experimentData@other$typeOfData  
pattern <- 'missing POV'  
metacell.mask <- match.metacell(metadata=GetMetacell(obj), pattern=pattern, level=level)  
ind <- GetIndices_WholeLine(metacell.mask)
```

---

GetIndices\_WholeMatrix

*Search lines which respects request on one or more conditions.*

---

### Description

This function looks for the lines that respect the request in either all conditions or at least one condition.

### Usage

```
GetIndices_WholeMatrix(metacell.mask, op = "==", percent = FALSE, th = 0)
```

**Arguments**

metacell.mask	xxx
op	String for operator to use. List of operators is available with <code>SymFilteringOperators()</code> .
percent	A boolean to indicate whether the threshold represent an absolute value (percent = FALSE) or a percentage (percent=TRUE).
th	A floating number which is in the interval [0, 1]

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10]
level <- obj@experimentData@other$typeOfData
pattern <- 'missing'
metacell.mask <- match.metacell(metadata=GetMetacell(obj), pattern=pattern, level=level)
percent <- FALSE
th <- 3
op <- '>='
ind <- GetIndices_WholeMatrix(metacell.mask, op, percent, th)
```

---

GetKeyId	xxxx
----------	------

---

**Description**

xxxx

**Usage**

```
GetKeyId(obj)
```

**Arguments**

obj	xxx
-----	-----

---

`getListNbValuesInLines`*Returns the possible number of values in lines in the data*

---

**Description**

Returns the possible number of values in lines in a matrix.

**Usage**

```
getListNbValuesInLines(obj, type)
```

**Arguments**

<code>obj</code>	An object of class MSnSet
<code>type</code>	xxxxxxx

**Value**

An integer

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
getListNbValuesInLines(Exp1_R25_pept, 'WholeMatrix')
```

---

`GetMatAdj`*Returns the contains of the slot processing of an object of class MSnSet*

---

**Description**

Returns the contains of the slot processing of an object of class MSnSet

**Usage**

```
GetMatAdj(obj)
```

**Arguments**

<code>obj</code>	An object (peptides) of class MSnSet.
------------------	---------------------------------------

**Value**

The slot processing of obj@processingData

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
Xshared <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", FALSE)
Xunique <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", TRUE)
ll.X <- list(matWithSharedPeptides = Xshared, matWithUniquePeptides = Xunique)
Exp1_R25_pept <- SetMatAdj(Exp1_R25_pept, ll.X)
ll.X <- GetMatAdj(Exp1_R25_pept)
```

---

GetMetacell	.xxx
-------------	------

---

**Description**

xxxx

**Usage**

```
GetMetacell(obj)
```

**Arguments**

obj	xxxx
-----	------

---

GetNbPeptidesUsed	<i>Computes the number of peptides used for aggregating each protein</i>
-------------------	--

---

**Description**

Method to compute the number of quantified peptides used for aggregating each protein

**Usage**

```
GetNbPeptidesUsed(X, pepData)
```

**Arguments**

X	An adjacency matrix
pepData	A data.frame of quantitative data

**Value**

A data.frame

**Author(s)**

Samuel Wieczorek

---

getNumberOf                      *Number of lines with prefix*

---

**Description**

Returns the number of lines, in a given column, where content matches the prefix.

**Usage**

```
getNumberOf(obj, name = NULL, prefix = NULL)
```

**Arguments**

obj	An object of class MSnSet.
name	The name of a column.
prefix	A string

**Value**

An integer

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
getNumberOf(Exp1_R25_pept[1:100], "Potential_contaminant", "+")
```

getNumberOfEmptyLines *Returns the number of empty lines in the data*

---

**Description**

Returns the number of empty lines in a matrix.

**Usage**

```
getNumberOfEmptyLines(qData)
```

**Arguments**

qData                    A matrix corresponding to the quantitative data.

**Value**

An integer

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
getNumberOfEmptyLines(qData)
```

---

getPourcentageOfMV        *Percentage of missing values*

---

**Description**

Returns the percentage of missing values in the quantitative data (Biobase::exprs() table of the dataset).

**Usage**

```
getPourcentageOfMV(obj)
```

**Arguments**

obj                      An object of class MSnSet.



**Value**

A floating number

**Author(s)**

Florence Combes, Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
getPourcentageOfMV(Exp1_R25_pept[1:100,])
```

---

`getProcessingInfo`      *Returns the contains of the slot processing of an object of class MSnSet*

---

**Description**

Returns the contains of the slot processing of an object of class MSnSet

**Usage**

```
getProcessingInfo(obj)
```

**Arguments**

`obj`                    An object (peptides) of class MSnSet.

**Value**

The slot processing of `obj@processingData`

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
getProcessingInfo(Exp1_R25_pept)
```

---

getProteinsStats	<i>Computes the number of proteins that are only defined by specific peptides, shared peptides or a mixture of two.</i>
------------------	---

---

**Description**

This function computes the number of proteins that are only defined by specific peptides, shared peptides or a mixture of two.

**Usage**

```
getProteinsStats(matShared)
```

**Arguments**

matShared      The adjacency matrix with both specific and shared peptides.

**Value**

A list

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
protID <- "Protein_group_IDs"
obj <- Exp1_R25_pept[1:20]
MShared <- BuildAdjacencyMatrix(obj, protID, FALSE)
getProteinsStats(matShared=MShared)
```

---

getQuantile4Imp	<i>Quantile imputation value definition</i>
-----------------	---

---

**Description**

This method returns the q-th quantile of each column of an expression set, up to a scaling factor

**Usage**

```
getQuantile4Imp(qdata, qval = 0.025, factor = 1)
```

**Arguments**

qdata            An expression set containing quantitative values of various replicates  
qval            The quantile used to define the imputation value  
factor           A scaling factor to multiply the imputation value with

**Value**

A list of two vectors, respectively containing the imputation values and the rescaled imputation values

**Author(s)**

Thomas Burger

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
qdata <- Biobase::exprs(Exp1_R25_pept)  
quant <- getQuantile4Imp(qdata)
```

---

*getTextForAggregation* *Build the text information for the Aggregation process*

---

**Description**

Builds the text information for the Aggregation process

**Usage**

```
getTextForAggregation(l.params)
```

**Arguments**

l.params            A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wiczorek

**Examples**

```
params <- list()  
getTextForAggregation(params)
```

---

getTextForAnaDiff      *Build the text information for the Aggregation process*

---

**Description**

Build the text information for the differential Analysis process

**Usage**

```
getTextForAnaDiff(l.params)
```

**Arguments**

l.params      A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wieczorek

**Examples**

```
getTextForAnaDiff(list(design="OnevsOne",method="Limma"))
```

---

getTextForFiltering      *Build the text information for the filtering process*

---

**Description**

Build the text information for the filtering process

**Usage**

```
getTextForFiltering(l.params)
```

**Arguments**

l.params      A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wiczorek

---

*getTextForGOAnalysis*    *Build the text information for the Aggregation process*

---

**Description**

Build the text information for the Aggregation process

**Usage**

```
getTextForGOAnalysis(l.params)
```

**Arguments**

l.params            A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wiczorek

**Examples**

```
getTextForGOAnalysis(list())
```

---

*getTextForHypothesisTest*  
*Build the text information for the hypothesis test process*

---

**Description**

Builds the text information for the hypothesis test process

**Usage**

```
getTextForHypothesisTest(l.params)
```

**Arguments**

l.params            A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wiczorek

**Examples**

```
params <- list(design='OnevsOne', method='limma')
getTextForHypothesisTest(params)
```

---

getTextForNewDataset *Build the text information for a new dataset*

---

**Description**

Build the text information for a new dataset

**Usage**

```
getTextForNewDataset(l.params)
```

**Arguments**

l.params      A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wiczorek

**Examples**

```
getTextForNewDataset(list(filename="foo.msnset"))
```

---

`getTextForNormalization`

*Build the text information for the Normalization process*

---

**Description**

Build the text information for the Normalization process

**Usage**

```
getTextForNormalization(l.params)
```

**Arguments**

`l.params`            A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wieczorek

**Examples**

```
getTextForNormalization(list(method="SumByColumns"))
```

---

`getTextForpeptideImputation`

*Build the text information for the peptide Imputation process*

---

**Description**

Build the text information for the peptide Imputation process

**Usage**

```
getTextForpeptideImputation(l.params)
```

**Arguments**

`l.params`            A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wieczorek

**Examples**

```
params <- list()
getTextForpeptideImputation(params)
```

---

getTextForproteinImputation

*Build the text information for the protein Imputation process*

---

**Description**

Build the text information for the Protein Imputation process

**Usage**

```
getTextForproteinImputation(l.params)
```

**Arguments**

l.params            A list of parameters related to the process of the dataset

**Value**

A string

**Author(s)**

Samuel Wieczorek

**Examples**

```
params <- list()
getTextForproteinImputation(params)
```



---

GetTypeofData	.xxx
---------------	------

---

**Description**

xxxx

**Usage**

GetTypeofData(obj)

**Arguments**

obj            xxx

---

Get_AllComparisons	<i>Returns list that contains a list of the statistical tests performed with DAPAR and recorded in an object of class MSnSet.</i>
--------------------	---

---

**Description**

This method returns a list of the statistical tests performed with DAPAR and recorded in an object of class MSnSet.

**Usage**

Get\_AllComparisons(obj)

**Arguments**

obj            An object of class MSnSet.

**Value**

A list of two slots: logFC and P\_Value

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- GetTypeofData(obj)
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
allComp <- limmaCompleteTest(qData,sTab)
data <- list(logFC=allComp$logFC[1], P_Value = allComp$P_Value[1])
obj$new <- diffAnaSave(obj$new, allComp, data)
ll <- Get_AllComparisons(obj$new)
```

---

GlobalQuantileAlignment

*Normalisation GlobalQuantileAlignement*

---

**Description**

Normalisation GlobalQuantileAlignement

**Usage**

```
GlobalQuantileAlignment(qData)
```

**Arguments**

qData                    xxxx

**Value**

A normalized numeric matrix

**Author(s)**

Samuel Wieczorek, Thomas Burger, Helene Borges, Anais Courtier, Enora Fremy

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
normalized <- GlobalQuantileAlignment(qData)
```

---

GOAnalysisSave	<i>Returns an MSnSet object with the results of the GO analysis performed with the functions enrichGO and/or groupGO of the 'clusterProfiler' package.</i>
----------------	--

---

### Description

This method returns an MSnSet object with the results of the Gene Ontology analysis.

### Usage

```
GOAnalysisSave(
  obj,
  ggo_res = NULL,
  ego_res = NULL,
  organism,
  ontology,
  levels,
  pvalueCutoff,
  typeUniverse
)
```

### Arguments

obj	An object of the class MSnSet
ggo_res	The object returned by the function group_GO of the package DAPAR or the function groupGO of the package 'clusterProfiler'
ego_res	The object returned by the function enrich_GO of the package DAPAR or the function enrichGO of the package 'clusterProfiler'
organism	The parameter OrgDb of the functions bitr, groupGO and enrichGO
ontology	One of "MF", "BP", and "CC" subontologies
levels	A vector of the different GO grouping levels to save
pvalueCutoff	The qvalue cutoff (same parameter as in the function enrichGO of the package 'clusterProfiler')
typeUniverse	The type of background to be used. Values are 'Entire Organism', 'Entire dataset' or 'Custom'. In the latter case, a file should be uploaded by the user

### Value

An object of the class MSnSet

### Author(s)

Samuel Wieczorek

---

GraphPepProt	<i>Function to create a histogram that shows the repartition of peptides w.r.t. the proteins</i>
--------------	--

---

**Description**

Method to create a plot with proteins and peptides on a MSnSet object (peptides)

**Usage**

```
GraphPepProt(mat)
```

**Arguments**

mat                    An adjacency matrix.

**Value**

A histogram

**Author(s)**

Alexia Dorffer, Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
mat <- BuildAdjacencyMatrix(Exp1_R25_pept[1:10], "Protein_group_IDs")
GraphPepProt(mat)
```

---

group_GO	<i>Calculates the GO profile of a vector of genes/proteins at a given level of the Gene Ontology</i>
----------	--

---

**Description**

This function is a wrapper to the function groupGO from the package 'clusterProfiler'. Given a vector of genes/proteins, it returns the GO profile at a specific level. It returns a groupGOResult instance.

**Usage**

```
group_GO(data, idFrom, orgdb, ont, level, readable = FALSE)
```

**Arguments**

data	A vector of ID (among ENSEMBL, ENTREZID, GENENAME, REFSEQ, UNIGENE, UNIPROT -can be different according to organisms)
idFrom	character indicating the input ID format (among ENSEMBL, ENTREZID, GENENAME, REFSEQ, UNIGENE, UNIPROT)
orgdb	annotation Bioconductor package to use (character format)
ont	on which ontology to perform the analysis (MF, BP or CC)
level	level of the ontology to perform the analysis
readable	TRUE or FALSE (default FALSE)

**Value**

GO profile at a specific level

**Author(s)**

Florence Combes

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot
ggo<-group_GO(data=Biobase::fData(obj)$Protein.IDs, idFrom="UNIPROT",
orgdb="org.Sc.sgd.db", ont="MF", level=2)
```

---

hc\_logFC\_DensityPlot *Density plots of logFC values*

---

**Description**

This function show the density plots of Fold Change (the same as calculated by limma) for a list of the comparisons of conditions in a differential analysis.

**Usage**

```
hc_logFC_DensityPlot(df_logFC, threshold_LogFC = 0, pal = NULL)
```

**Arguments**

df_logFC	A dataframe that contains the logFC values
threshold_LogFC	The threshold on log(Fold Change) to distinguish between differential and non-differential data
pal	xxx

**Value**

A highcharts density plot

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
res <- limmaCompleteTest(qData, sTab, comp.type = "OnevsAll")
hc_logFC_DensityPlot(res$logFC, threshold_LogFC=1)
pal <- ExtendPalette(2, 'Dark2')
hc_logFC_DensityPlot(res$logFC, threshold_LogFC=1, pal=pal)
```

---

hc\_mvTypePlot2

*Distribution of Observed values with respect to intensity values*

---

**Description**

This method shows density plots which represents the repartition of Partial Observed Values for each replicate in the dataset. The colors correspond to the different conditions (slot Condition in in the dataset of class MSnSet). The x-axis represent the mean of intensity for one condition and one entity in the dataset (i. e. a protein) whereas the y-axis count the number of observed values for this entity and the considered condition.

**Usage**

```
hc_mvTypePlot2(obj, pal = NULL, pattern, typeofMV = NULL, title = NULL)
```

**Arguments**

obj	xxx
pal	The different colors for conditions
pattern	xxx
typeofMV	xxx
title	The title of the plot

**Value**

Density plots

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:100]
hc_mvTypePlot2(obj, pattern = 'missing MEC', title="POV distribution")
conds <- Biobase::pData(obj)$Condition
pal <- ExtendPalette(length(unique(conds)), 'Dark2')
hc_mvTypePlot2(obj, pattern = 'missing MEC', title="POV distribution", pal=pal)
```

---

heatmapD

*This function is a wrapper to [heatmap.2](#) that displays quantitative data in the `Biobase::exprs()` table of an object of class `MSnSet`*

---

**Description**

Heatmap of the quantitative proteomic data of a `MSnSet` object

**Usage**

```
heatmapD(
  qData,
  conds,
  distance = "euclidean",
  cluster = "complete",
  dendro = FALSE
)
```

**Arguments**

qData	A dataframe that contains quantitative data.
conds	A vector containing the conditions
distance	The distance used by the clustering algorithm to compute the dendrogram. See <code>help(heatmap.2)</code>
cluster	the clustering algorithm used to build the dendrogram. See <code>help(heatmap.2)</code>
dendro	A boolean to indicate if the dendrogram has to be displayed

**Value**

A heatmap

**Author(s)**

Florence Combes, Samuel Wiczorek, Enor Fremy

**Examples**

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10,]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeLine(metacell.mask)
qData <- Biobase::exprs(obj)
conds <- Biobase::pData(obj)[['Condition']]
heatmapD(qData, conds)

## End(Not run)
```

---

**heatmapForMissingValues**

*This function is inspired from the function [heatmap.2](#) that displays quantitative data in the `Biobase::exprs()` table of an object of class `MSet`. For more information, please refer to the help of the `heatmap.2` function.*

---

**Description**

Heatmap inspired by the `heatmap.2` function.

**Usage**

```
heatmapForMissingValues(
  x,
  col = heat.colors(100),
  srtCol = NULL,
  labCol = NULL,
  labRow = NULL,
  key = TRUE,
  key.title = NULL,
  main = NULL,
  ylab = NULL
)
```

**Arguments**

`x` A dataframe that contains quantitative data.  
`col` colors used for the image. Defaults to `heat.colors` (`heat.colors`).



srtCol	angle of column conds, in degrees from horizontal
labCol	character vectors with column conds to use.
labRow	character vectors with row conds to use.
key	logical indicating whether a color-key should be shown.
key.title	main title of the color key. If set to NA no title will be plotted.
main	main title; default to none.
ylab	y-axis title; default to none.

**Value**

A heatmap

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeLine(metacell.mask)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
heatmapForMissingValues(qData)
```

---

histPValue_HC	<i>Plots a histogram ov p-values</i>
---------------	--------------------------------------

---

**Description**

This function plots a histogram ov p-values

**Usage**

```
histPValue_HC(pval_ll, bins = 80, pi0 = 1)
```

**Arguments**

pval_ll	xxx
bins	xxx
pi0	xxx

**Value**

A plot

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
qData <- Biobase::exprs(obj$new)
sTab <- Biobase::pData(obj$new)
allComp <- limmaCompleteTest(qData,sTab)
histPValue_HC(allComp$P_Value[1])
```

---

impute.pa2

*Missing values imputation from a MSnSet object*

---

**Description**

This method is a variation to the function `impute.pa` from the package `imp4p`.

**Usage**

```
impute.pa2(
  tab,
  conditions,
  q.min = 0,
  q.norm = 3,
  eps = 0,
  distribution = "unif"
)
```

**Arguments**

<code>tab</code>	An object of class <code>MSnSet</code> .
<code>conditions</code>	A vector of conditions in the dataset
<code>q.min</code>	A quantile value of the observed values allowing defining the maximal value which can be generated. This maximal value is defined by the quantile <code>q.min</code> of the observed values distribution minus <code>eps</code> . Default is 0 (the maximal value is the minimum of observed values minus <code>eps</code> ).

q.norm	A quantile value of a normal distribution allowing defining the minimal value which can be generated. Default is 3 (the minimal value is the maximal value minus $qn * \text{median}(\text{sd}(\text{observed values}))$ where sd is the standard deviation of a row in a condition).
eps	A value allowing defining the maximal value which can be generated. This maximal value is defined by the quantile q.min of the observed values distribution minus eps. Default is 0.
distribution	The type of distribution used. Values are unif or beta.

**Value**

The object obj which has been imputed

**Author(s)**

Thomas Burger, Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.imp <- wrapper.impute.pa2(Exp1_R25_pept[1:100], distribution="beta")
```

---

```
inner.aggregate.iter  xxxx
```

---

**Description**

Method to xxxxx

**Usage**

```
inner.aggregate.iter(
  pepData,
  X,
  init.method = "Sum",
  method = "Mean",
  n = NULL
)
```

**Arguments**

pepData	xxxxx
X	xxxx
init.method	xxx
method	xxx
n	xxxx

**Value**

xxxxx

**Author(s)**

Samuel Wieczorek library(MSNbase) utils::data(Exp1\_R25\_pept, package='DAPARdata') protID  
 <- "Protein\_group\_IDs" X <- BuildAdjacencyMatrix(Exp1\_R25\_pept[1:10], protID, FALSE) qdata.agg  
 <- DAPAR::inner.aggregate.iter(Biobase::exprs(Exp1\_R25\_pept[1:10]), X)

---

 inner.aggregate.topn xxxx
 

---

**Description**

xxxx

**Usage**

```
inner.aggregate.topn(pepData, X, method = "Mean", n = 10)
```

**Arguments**

pepData	A data.frame of quantitative data
X	An adjacency matrix
method	xxxxx
n	xxxxx

**Value**

xxxxx

**Author(s)**

Samuel Wieczorek

---

inner.mean	.xxx
------------	------

---

**Description**

xxxx

**Usage**

inner.mean(pepData, X)

**Arguments**

pepData	A data.frame of quantitative data
X	An adjacency matrix

**Value**

xxxxx

**Author(s)**

Samuel Wieczorek

---

inner.sum	.xxx
-----------	------

---

**Description**

xxxx

**Usage**

inner.sum(pepData, X)

**Arguments**

pepData	A data.frame of quantitative data
X	An adjacency matrix

**Value**

A matrix

**Author(s)**

Samuel Wieczorek

---

LH0	xxxxxx
-----	--------

---

**Description**

This function is xxxxxxxx

**Usage**

LH0(X, y1, y2)

**Arguments**

X	an n.pep*n.prot indicator matrix.
y1	n.pep*n.samples matrice giving the observed counts for
y2	n.pep*n.samples matrice giving the observed counts for

**Value**

xxxxxxxxxxx..

**Author(s)**

Thomas Burger, Laurent Jacob

---

LH0.lm	xxxxxx
--------	--------

---

**Description**

This function is xxxxxxxx

**Usage**

LH0.lm(X, y1, y2)

**Arguments**

X	an n.pep*n.prot indicator matrix.
y1	n.pep*n.samples matrice giving the observed counts for each peptide in each sample from the condition 1
y2	n.pep*n.samples matrice giving the observed counts for each peptide in each sample from the condition 2

**Value**

xxxxxxxxxx..

**Author(s)**

Thomas Burger, Laurent Jacob

---

LH1

xxxxxx

---

**Description**

This function is xxxxxxxx

**Usage**

LH1(X, y1, y2, j)

**Arguments**

X            an n.pep\*n.prot indicator matrix.  
y1           n.pep\*n.samples matrice giving the observed counts for  
y2           n.pep\*n.samples matrice giving the observed counts for  
j            the index of the protein being tested, ie which has different

**Value**

xxxxxxxxxx..

**Author(s)**

Thomas Burger, Laurent Jacob

---

LH1.lm                      xxxxxx

---

**Description**

This function is xxxxxxxx

**Usage**

LH1.lm(X, y1, y2, j)

**Arguments**

X	an n.pep*n.prot indicator matrix.
y1	n.pep*n.samples matrix giving the observed counts for
y2	n.pep*n.samples matrix giving the observed counts for
j	the index of the protein being tested, ie which has different

**Value**

xxxxxxxxxxx..

**Author(s)**

Thomas Burger, Laurent Jacob

---

limmaCompleteTest            *Computes a hierarchical differential analysis*

---

**Description**

This function is a limmaCompleteTest

**Usage**

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

**Arguments**

qData	A matrix of quantitative data, without any missing values.
sTab	A dataframe of experimental design (Biobase::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)**

Hélène Borges, Thomas Burger, Quentin Gaii-Gianetto, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept
qData <- Biobase::exprs(obj)
sTab <- Biobase::pData(obj)
limma <- limmaCompleteTest(qData, sTab, comp.type='anova1way')
```

---

listSheets

*This function returns the list of the sheets names in a Excel file.*

---

**Description**

This function lists all the sheets of an Excel file.

**Usage**

```
listSheets(file)
```

**Arguments**

file                    The name of the Excel file.

**Value**

A vector

**Author(s)**

Samuel Wiczorek

---

LOESS

*Normalisation LOESS*

---

### Description

Normalisation LOESS

### Usage

```
LOESS(qData, conds, type = "overall", span = 0.7)
```

### Arguments

qData	A numeric matrix.
conds	xxx
type	"overall" (shift all the sample distributions at once) or "within conditions" (shift the sample distributions within each condition at a time).
span	xxx

### Value

A normalized numeric matrix

### Author(s)

Thomas Burger, Helene Borges, Anais Courtier, Enora Fremy

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
conds <- Biobase::pData(Exp1_R25_pept)$Condition
normalized <- LOESS(qData, conds, type="overall")
```

---

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 Wiczorek

**Examples**

```
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)
```

---

make.design	<i>Builds the design matrix</i>
-------------	---------------------------------

---

**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 Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
make.design(Biobase::pData(Exp1_R25_pept))
```

---

make.design.1	<i>Builds the design matrix for designs of level 1</i>
---------------	--

---

**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 Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
make.design.1(Biobase::pData(Exp1_R25_pept))
```

---

make.design.2	<i>Builds the design matrix for designs of level 2</i>
---------------	--

---

**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 Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
make.design.2(Biobase::pData(Exp1_R25_pept))
```

---

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 Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
sTab <-cbind(Biobase::pData(Exp1_R25_pept), Tech.Rep=1:6)
make.design.3(sTab)
```

---

match.metacell	<i>Similar to the function is.na but focused on the equality with the paramter 'type'.</i>
----------------	--

---

**Description**

Similar to the function `is.na` but focused on the equality with the paramter 'type'.

**Usage**

```
match.metacell(metadata, pattern, level)
```

**Arguments**

metadata	A data.frame
pattern	The value to search in the dataframe
level	xxx

**Value**

A boolean dataframe

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10,]
metadata <- GetMetacell(obj)
m <- match.metacell(metadata, pattern="missing", level = 'peptide')
```

---

MeanCentering	<i>Normalisation MeanCentering</i>
---------------	------------------------------------

---

**Description**

Normalisation MeanCentering

**Usage**

```
MeanCentering(
  qData,
  conds,
  type = "overall",
  subset.norm = NULL,
  scaling = FALSE
)
```

**Arguments**

qData	xxx
conds	xxx
type	"overall" (shift all the sample distributions at once) or "within conditions" (shift the sample distributions within each condition at a time).
subset.norm	A vector of index indicating rows to be used for normalization
scaling	A boolean that indicates if the variance of the data have to be forced to unit (variance reduction) or not.

**Value**

A normalized numeric matrix

**Author(s)**

Samuel Wieczorek, Thomas Burger, Helene Borges, Anais Courtier, Enora Fremy

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
conds <- Biobase::pData(Exp1_R25_pept)$Condition
normalized <- MeanCentering(qData, conds, type="overall")
```

---

metacell.def

*Metadata vocabulary for entities*


---

**Description**

This function gives the vocabulary used for the metadata of each entity in each condition. Peptide-level vocabulary

|- 1.0 Quantitative Value |||- 1.1 Identified (color 4, white) |||- 1.2 Recovered (color 3, lightgrey)  
|- 2.0 Missing value (no color) |||- 2.1 Missing POV (color 1) |||- 2.2 Missing MEC (color 2)  
|- 3.0 Imputed value |||- 3.1 Imputed POV (color 1) |||- 3.2 Imputed MEC (color 2)

Protein-level vocabulary:

|- 1.0 Quantitative Value |||- 1.1 Identified (color 4, white) |||- 1.2 Recovered (color 3, lightgrey)  
 |- 2.0 Missing value |||- 2.1 Missing POV (color 1) |||- 2.2 Missing MEC (color 2) |- 3.0  
 Imputed value |||- 3.1 Imputed POV (color 1) |||- 3.2 Imputed MEC (color 2) |- 4.0 Combined  
 value (color 3bis, light-lightgrey)

**Usage**

```
metacell.def(level)
```

**Arguments**

level	A string designing the type of entity/pipeline. Available values are: 'peptide', 'protein'
-------	--

**Author(s)**

Thomas Burger, Samuel Wiczorek

---

MetaCellFiltering      *Filter lines in the matrix of intensities w.r.t. some criteria*

---

**Description**

Filters the lines of Biobase::exprs() table with conditions on the number of missing values. The user chooses the minimum amount of intensities that is acceptable and the filter delete lines that do not respect this condition. The condition may be on the whole line or condition by condition.

**Usage**

```
MetaCellFiltering(obj, indices, cmd, processText = "")
```

**Arguments**

obj	An object of class MSnSet containing quantitative data.
indices	A vector of integers which are the indices of lines to keep.
cmd	xxxx. Available values are: 'delete', 'keep'.
processText	A string to be included in the MSnSet object for log.

**Details**

The different methods are : "WholeMatrix": given a threshold th, only the lines that contain at least th values are kept. "AllCond": given a threshold th, only the lines which contain at least th values for each of the conditions are kept. "AtLeastOneCond": given a threshold th, only the lines that contain at least th values, and for at least one condition, are kept.

**Value**

An instance of class MSnSet that have been filtered.



**Author(s)**

Florence Combes, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeLine(metacell.mask)
obj.filter <- MetaCellFiltering(obj, indices, 'delete')
```

---

MetacellFilteringScope

*Lists the metacell scopes for filtering*

---

**Description**

Lists the metacell scopes for filtering

**Usage**

```
MetacellFilteringScope()
```

---

metacellHisto\_HC

*Histogram of missing values*

---

**Description**

This method plots a histogram of missing values. Same as the function mvHisto but uses the package highcharter

**Usage**

```
metacellHisto_HC(  
  obj,  
  pattern,  
  indLegend = "auto",  
  showValues = FALSE,  
  pal = NULL  
)
```

**Arguments**

obj	xxx
pattern	xxx
indLegend	The indices of the column name's in Biobase::pData() tab
showValues	A logical that indicates wether numeric values should be drawn above the bars.
pal	xxx

**Value**

A histogram

**Author(s)**

Florence Combes, Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept
pattern <- 'missing POV'
pal <- ExtendPalette(2, 'Dark2')
metacellHisto_HC(obj, pattern, showValues=TRUE, pal=pal)
```

---

metacellPerLinesHistoPerCondition\_HC

*Bar plot of missing values per lines and per condition*

---

**Description**

This method plots a bar plot which represents the distribution of the number of missing values (NA) per lines (ie proteins) and per conditions.

**Usage**

```
metacellPerLinesHistoPerCondition_HC(  
  obj,  
  pattern,  
  indLegend = "auto",  
  showValues = FALSE,  
  pal = NULL  
)
```

**Arguments**

obj	xxx
pattern	xxx
indLegend	The indice of the column name's in Biobase::pData() tab
showValues	A logical that indicates wether numeric values should be drawn above the bars.
pal	xxx

**Value**

A bar plot

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept
pal <- ExtendPalette(length(unique( Biobase::pData(obj)$Condition)), 'Dark2')
metacellPerLinesHistoPerCondition_HC(obj, 'missing', pal=pal)
metacellPerLinesHistoPerCondition_HC(obj, 'quanti')
```

---

metacellPerLinesHisto\_HC

*Bar plot of missing values per lines using highcharter*

---

**Description**

This method plots a bar plot which represents the distribution of the number of missing values (NA) per lines (ie proteins).

**Usage**

```
metacellPerLinesHisto_HC(  
  obj,  
  pattern,  
  detailed = FALSE,  
  indLegend = "auto",  
  showValues = FALSE  
)
```

**Arguments**

obj	xxx.
pattern	xxx
detailed	'value' or 'percent'
indLegend	The indice of the column name's in Biobase::pData() tab
showValues	A logical that indicates whether numeric values should be drawn above the bars.

**Value**

A bar plot

**Author(s)**

Florence Combes, Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10,]
metacellPerLinesHisto_HC(obj, pattern = 'missing')
```

---

Metacell\_generic      *Sets the metacell dataframe*

---

**Description**

In the quantitative columns, a missing value is identified by no value rather than a value equal to 0.  
Conversion rules Quanti Tag NA or 0 NA

**Usage**

```
Metacell_generic(qdata, conds, level)
```

**Arguments**

qdata	An object of class MSnSet
conds	xxx
level	xxx

**Value**

xxxxx

**Author(s)**

Samuel Wiczorek

**Examples**

```

file <- system.file("extdata", "Exp1_R25_pept.txt", package="DAPARdata")
data <- read.table(file, header=TRUE, sep="\t", stringsAsFactors = FALSE)
metadataFile <- system.file("extdata", "samples_Exp1_R25.txt",
package="DAPARdata")
metadata <- read.table(metadataFile, header=TRUE, sep="\t", as.is=TRUE,
stringsAsFactors = FALSE)
conds <- metadata$Condition
qdata <- data[1:100,56:61]
df <- data[1:100 , 43:48]
df <- Metacell_generic(qdata, conds, level='peptide')

```

---

Metacell_maxquant	<i>Sets the metacell dataframe</i>
-------------------	------------------------------------

---

**Description**

Initial conversion rules for maxquant |-----|-----|-----| | Quanti | Identification  
| Tag | |-----|-----|-----| | == 0 | whatever | 2.0 | | > 0 | 'By MS/MS' | 1.1 | | > 0 |  
'By matching' | 1.2 | | > 0 | unknown col | 1.0 | |-----|-----|-----|

**Usage**

```
Metacell_maxquant(qdata, conds, df, level = NULL)
```

**Arguments**

qdata	An object of class MSnSet
conds	xxx
df	A list of integer xxxxxxxx
level	xxx

**Value**

xxxxx

**Author(s)**

Samuel Wieczorek

**Examples**

```

file <- system.file("extdata", "Exp1_R25_pept.txt", package="DAPARdata")
data <- read.table(file, header=TRUE, sep="\t", stringsAsFactors = FALSE)
metadataFile <- system.file("extdata", "samples_Exp1_R25.txt",
package="DAPARdata")
metadata <- read.table(metadataFile, header=TRUE, sep="\t", as.is=TRUE,
stringsAsFactors = FALSE)
conds <- metadata$Condition
qdata <- data[1:10,56:61]
df <- data[1:10 , 43:48]
df2 <- Metacell_maxquant(qdata, conds, df, level='peptide')

```

---

Metacell\_proline      *Sets the metacell dataframe*

---

**Description**

In the quantitative columns, a missing value is identified by no value rather than a value equal to 0. Conversion rules Initial conversion rules for maxquant |-----|-----|-----| Quantitative | PSM count | Tag |-----|-----|-----| == 0 | N.A. | whatever | 2.0 || > 0 | > 0 | 1.1 || > 0 | == 0 | 1.2 || > 0 | unknown col | 1.0 ||-----|-----|-----|

**Usage**

```
Metacell_proline(qdata, conds, df, level = NULL)
```

**Arguments**

qdata	An object of class MSnSet
conds	xxx
df	A list of integer xxxxxxxx
level	xxx

**Value**

xxxxx

**Author(s)**

Samuel Wieczorek

**Examples**

```
## Not run:
file <- system.file("extdata", "Exp1_R25_pept.txt", package="DAPARdata")
data <- read.table(file, header=TRUE, sep="\t", stringsAsFactors = FALSE)
metadataFile <- system.file("extdata", "samples_Exp1_R25.txt",
package="DAPARdata")
metadata <- read.table(metadataFile, header=TRUE, sep="\t", as.is=TRUE,
stringsAsFactors = FALSE)
conds <- metadata$Condition
qdata <- data[1:100,56:61]
df <- data[1:100 , 43:48]
df <- Metacell_proline(qdata, conds, df, level = 'peptide')

## End(Not run)
```

metacombine

*Combine peptide metadata to build protein metadata***Description**

Agregation rules for the cells metadata of peptides. Please refer to the metacell vocabulary in 'metacell.def()'

# Basic aggregation Agregation of non imputed values (2.X) with quantitative values (1.0, 1.X, 3.0, 3.X) |----- Not possible |-----

Agregation of different types of missing values (among 2.1, 2.2) |----- \* Agregation of 2.1 peptides between each other gives a missing value non imputed (2.0) \* Agregation of 2.2 peptides between each other gives a missing value non imputed (2.0) \* Agregation of a mix of 2.1 and 2.2 gives a missing value non imputed (2.0) |-----

Agregation of a mix of quantitative values (among 1.0, 1.1, 1.2, 3.0, 3.X) |----- \* if the type of all the peptides to aggregate is 1.0, 1.1 or 1.2, then the final metadata is set the this tag \* if the set of metacell to aggregate is a mix of 1.0, 1.1 or 1.2, then the final metadata is set to 1.0 \* if the set of metacell to aggregate is a mix of 3.X and 3.0, then the final metadata is set to 3.0 \* if the set of metacell to aggregate is a mix of 3.X and 3.0 and other (X.X), then the final metadata is set to 4.0 |-----

# Post processing Update metacell with POV/MEC status for the categories 2.0 and 3.0 TODO

**Usage**

```
metacombine(met, level)
```

**Arguments**

```
met          xxx
level       xxx
```

## Examples

```
## Not run:
ll <- metacell.def('peptide')$node
for (i in 1:length(ll))
test <- lapply(combn(ll, i, simplify = FALSE),
function(x) tag <- metacombine(x, 'peptide'))

## End(Not run)
```

---

mvImage

*Heatmap of missing values*

---

## Description

Plots a heatmap of the quantitative data. Each column represent one of the conditions in the object of class MSnSet and the color is proportional to the mean of intensity for each line of the dataset. The lines have been sorted in order to visualize easily the different number of missing values. A white square is plotted for missing values.

## Usage

```
mvImage(qData, conds)
```

## Arguments

qData	A dataframe that contains quantitative data.
conds	A vector of the conditions (one condition per sample).

## Value

A heatmap

## Author(s)

Samuel Wiczorek, Thomas Burger

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
conds <- Biobase::pData(Exp1_R25_pept)[,"Condition"]
mvImage(qData, conds)
```



---

my\_hc\_chart                      *Customised resetZoomButton of highcharts plots*

---

**Description**

Customise the resetZoomButton of highcharts plots.

**Usage**

```
my_hc_chart(hc, chartType, zoomType = "None")
```

**Arguments**

hc	A highcharter object
chartType	The type of the plot
zoomType	The type of the zoom (one of "x", "y", "xy", "None")

**Value**

A highchart plot

**Author(s)**

Samuel Wieczorek

**Examples**

```
library("highcharter")
hc <- highchart()
hc_chart(hc, type = "line")
hc_add_series(hc, data = c(29, 71, 40))
my_hc_ExportMenu(hc, filename='foo')
```

---

my\_hc\_ExportMenu                      *Customised contextual menu of highcharts plots*

---

**Description**

Customise the contextual menu of highcharts plots.

**Usage**

```
my_hc_ExportMenu(hc, filename)
```

**Arguments**

hc                    A highcharter object  
filename             The filename under which the plot has to be saved

**Value**

A contextual menu for highcharts plots

**Author(s)**

Samuel Wieczorek

**Examples**

```
library("highcharter")  
hc <- highchart()  
hc_chart(hc, type = "line")  
hc_add_series(hc, data = c(29, 71, 40))  
my_hc_ExportMenu(hc, filename='foo')
```

---

nonzero

*Retrieve the indices of non-zero elements in sparse matrices*

---

**Description**

This function retrieves the indices of non-zero elements in sparse matrices of class dgCMatrix from package Matrix. This function is largely inspired from the package RINGO

**Usage**

```
nonzero(x)
```

**Arguments**

x                    A sparse matrix of class dgCMatrix

**Value**

A two-column matrix

**Author(s)**

Samuel Wieczorek

**Examples**

```
library(Matrix)
mat <- Matrix(c(0,0,0,0,0,1,0,0,1,1,0,0,0,0,1),nrow=5, byrow=TRUE,
sparse=TRUE)
res <- nonzero(mat)
```

---

```
normalizeMethods.dapar
```

*List normalization methods with tracking option*

---

**Description**

List normalization methods with tracking option

**Usage**

```
normalizeMethods.dapar(withTracking = FALSE)
```

**Arguments**

```
withTracking    xxx
```

---

```
NumericalFiltering
```

*Removes lines in the dataset based on numerical conditions.*

---

**Description**

This function removes lines in the dataset based on numerical conditions.

**Usage**

```
NumericalFiltering(obj, name = NULL, value = NULL, operator = NULL)
```

**Arguments**

obj	An object of class MSnSet.
name	The name of the column that correspond to the line to filter
value	A number
operator	A string

**Value**

An list of 2 items : obj : an object of class MSnSet in which the lines have been deleted deleted : an object of class MSnSet which contains the deleted lines

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
NumericalFiltering(Exp1_R25_pept[1:100], 'A_Count', '6', '==')
```

---

NumericalgetIndicesOfLinesToRemove

*Get the indices of the lines to delete, based on a prefix string*

---

**Description**

This function returns the indice of the lines to delete, based on a prefix string

**Usage**

```
NumericalgetIndicesOfLinesToRemove(  
  obj,  
  name = NULL,  
  value = NULL,  
  operator = NULL  
)
```

**Arguments**

obj	An object of class MSnSet.
name	The name of the column that correspond to the data to filter
value	xxxx
operator	A xxxx

**Value**

A vector of integers.

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
NumericalgetIndicesOfLinesToRemove(Exp1_R25_pept[1:100], "A_Count", value="6",
operator=='=')
```

pepa.test

*PEptide based Protein differential Abundance test***Description**

This function is PEptide based Protein differential Abundance test

**Usage**

```
pepa.test(X, y, n1, n2, global = FALSE, use.lm = FALSE)
```

**Arguments**

X	Binary q x p design matrix for q peptides and p proteins. $X_{(ij)}=1$ if peptide i belongs to protein j, 0 otherwise.
y	q x n matrix representing the log intensities of q peptides among n MS samples.
n1	number of samples under condition 1. It is assumed that the first n1 columns of y correspond to observations under condition 1.
n2	number of samples under condition 2.
global	if TRUE, the test statistic for each protein uses all residues, including the ones for peptides in different connected components. Can be much faster as it does not require to compute connected components. However the p-values are not well calibrated in this case, as it amounts to adding a ridge to the test statistic. Calibrating the p-value would require knowing the amplitude of the ridge, which in turns would require computing the connected components.
use.lm	if TRUE (and if global=FALSE), use lm() rather than the result in Proposition 1 to compute the test statistic

**Value**

A list of the following elements: llr: log likelihood ratio statistic (maximum likelihood version). llr.map: log likelihood ratio statistic (maximum a posteriori version). llr.pv: p-value for llr. llr.map.pv: p-value for llr.map. mse.h0: Mean squared error under H0 mse.h1: Mean squared error under H1 s: selected regularization hyperparameter for llr.map. wchi2: weight used to make llr.map chi2-distributed under H0.

**Author(s)**

Thomas Burger, Laurent Jacob

plotJitter

*Jitter plot of CC*

---

**Description**

Jitter plot of CC

**Usage**

```
plotJitter(list.of.cc)
```

**Arguments**

```
list.of.cc      List of cc such as returned by the function get.pep.prot.cc
```

**Value**

A plot

**Author(s)**

Thomas Burger

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
X <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", TRUE)
ll <- get.pep.prot.cc(X)
plotJitter(ll)
```

---

plotJitter\_rCharts*Display a a jitter plot for CC*

---

**Description**

Display a jitter plot for CC

**Usage**

```
plotJitter_rCharts(df, clickFunction = NULL)
```

**Arguments**

```
df                xxxx
clickFunction     xxxx
```

**Value**

A plot

**Author(s)**

Thomas Burger, Samuel Wieczorek

---

plotPCA\_Eigen      *Plots the eigen values of PCA*

---

**Description**

Plots the eigen values of PCA

**Usage**

```
plotPCA_Eigen(res.pca)
```

**Arguments**

res.pca      xxx

**Value**

A histogram

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
res.pca <- wrapper.pca(Exp1_R25_pept, ncp=6)
plotPCA_Eigen(res.pca)
```

---

plotPCA\_Eigen\_hc      *Plots the eigen values of PCA with the highcharts library*

---

**Description**

Plots the eigen values of PCA with the highcharts library

**Usage**

```
plotPCA_Eigen_hc(res.pca)
```

**Arguments**

res.pca      xxx

**Value**

A histogram

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
res.pca <- wrapper.pca(Exp1_R25_pept, ncp=6)
plotPCA_Eigen_hc(res.pca)
```

---

plotPCA\_Ind      *Plots individuals of PCA*

---

**Description**

Plots individuals of PCA

**Usage**

```
plotPCA_Ind(res.pca, chosen.axes = c(1, 2))
```

**Arguments**

res.pca      xxx  
chosen.axes      The dimensions to plot



**Value**

A plot

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
res.pca <- wrapper.pca(Exp1_R25_pept)
plotPCA_Ind(res.pca)
```

---

plotPCA\_Var

*Plots variables of PCA*

---

**Description**

Plots the variables of PCA

**Usage**

```
plotPCA_Var(res.pca, chosen.axes = c(1, 2))
```

**Arguments**

res.pca	xxx
chosen.axes	The dimensions to plot

**Value**

A plot

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
res.pca <- wrapper.pca(Exp1_R25_pept)
plotPCA_Var(res.pca)
```

---

postHocTest	<i>Post-hoc tests for classic 1-way ANOVA</i>
-------------	---

---

### Description

This function allows to compute a post-hoc test after a 1-way ANOVA analysis. It expects as input an object obtained with the function `classic1wayAnova`. The second parameter allows to choose between 2 different post-hoc tests: the Tukey Honest Significant Differences (specified as "TukeyHSD") and the Dunnett test (specified as "Dunnett").

### Usage

```
postHocTest(aov_fits, post_hoc_test = "TukeyHSD")
```

### Arguments

`aov_fits` a list containing aov fitted model objects

`post_hoc_test` a character string indicating which post-hoc test to use. Possible values are "TukeyHSD" or "Dunnett". See details for what to choose according to your experimental design.

### Details

This is a function allowing to realise post-hoc tests for a set of proteins/peptides for which a classic 1-way anova has been performed with the function `classic1wayAnova`. Two types of tests are currently available: The Tukey HSD's test and the Dunnett's test. Default is Tukey's test. The Tukey HSD's test compares all possible pairs of means, and is based on a studentized range distribution. Here is used the `TukeyHSD()` function, which can be applied to balanced designs (same number of samples in each group), but also to midly unbalanced designs. The Dunnett's test compares a single control group to all other groups. Make sure the factor levels are properly ordered.

### Value

a list of 2 dataframes: first one called "LogFC" contains all pairwise comparisons logFC values (one column for one comparison) for each analysed feature; The second one named "P\_Value" contains the corresponding pvalues.

### Author(s)

Hélène Borges

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
```

```
obj <- MetaCellFiltering(obj, indices, cmd='delete')
anova_tests <- t(apply(Biobase::exprs(obj$new), 1, classic1wayAnova,
  conditions=as.factor(Biobase::pData(obj$new)$Condition)))
names(anova_tests) <- rownames(Biobase::exprs(obj$new))
pht <- postHocTest(aov_fits = anova_tests)
```

---

proportionConRev\_HC *Barplot of proportion of contaminants and reverse*

---

### Description

Plots a barplot of proportion of contaminants and reverse. Same as the function `proportionConRev` but uses the package `highcharter`

### Usage

```
proportionConRev_HC(nBoth = 0, nCont = 0, nRev = 0, lDataset = 0)
```

### Arguments

<code>nBoth</code>	The number of both contaminants and reverse identified in the dataset.
<code>nCont</code>	The number of contaminants identified in the dataset.
<code>nRev</code>	The number of reverse entities identified in the dataset.
<code>lDataset</code>	The total length (number of rows) of the dataset

### Value

A barplot

### Author(s)

Samuel Wiczorek

### Examples

```
proportionConRev_HC(10, 20, 100)
```

---

QuantileCentering      *Normalisation QuantileCentering*

---

## Description

Normalisation QuantileCentering

## Usage

```
QuantileCentering(  
  qData,  
  conds = NULL,  
  type = "overall",  
  subset.norm = NULL,  
  quantile = 0.15  
)
```

## Arguments

qData	xxx
conds	xxx
type	"overall" (shift all the sample distributions at once) or "within conditions" (shift the sample distributions within each condition at a time).
subset.norm	A vector of index indicating rows to be used for normalization
quantile	A float that corresponds to the quantile used to align the data.

## Value

A normalized numeric matrix

## Author(s)

Samuel Wiczorek, Thomas Burger, Helene Borges, Anais Courtier, Enora Fremy

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
obj <- Exp1_R25_pept  
conds <- Biobase::pData(Exp1_R25_pept)$Condition  
normalized <- QuantileCentering(Biobase::exprs(obj), conds,  
  type="within conditions", subset.norm=1:10)
```

---

rbindMSnset	<i>Similar to the function rbind but applies on two subsets of the same MSnSet object.</i>
-------------	--

---

**Description**

Similar to the function rbind but applies on two subsets of the same MSnSet object.

**Usage**

```
rbindMSnset(df1 = NULL, df2)
```

**Arguments**

df1	An object (or subset of) of class MSnSet. May be NULL
df2	A subset of the same object as df1

**Value**

An instance of class MSnSet.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
df1 <- Exp1_R25_pept[1:100]
df2 <- Exp1_R25_pept[200:250]
rbindMSnset(df1, df2)
```

---

readExcel	<i>This function reads a sheet of an Excel file and put the data into a data.frame.</i>
-----------	---

---

**Description**

This function reads a sheet of an Excel file and put the data into a data.frame.

**Usage**

```
readExcel(file, extension, sheet)
```

**Arguments**

file	The name of the Excel file.
extension	The extension of the file
sheet	The name of the sheet

**Value**

A data.frame

**Author(s)**

Samuel Wieczorek

---

reIntroduceMEC      *Put back LAPALA into a MSnSet object*

---

**Description**

Put back LAPALA into a MSnSet object

**Usage**

```
reIntroduceMEC(obj, MECIndex)
```

**Arguments**

obj	An object of class MSnSet.
MECIndex	A data.frame that contains index of MEC (see findMECBlock).

**Value**

The object obj where LAPALA have been reintroduced

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:100]
lapala <- findMECBlock(obj)
obj <- wrapper.impute.detQuant(obj, na.type='missing')
obj <- reIntroduceMEC(obj, lapala)
```

---

removeLines	<i>Removes lines in the dataset based on a prefix string.</i>
-------------	---

---

**Description**

This function removes lines in the dataset based on a prefix string.

**Usage**

```
removeLines(obj, idLine2Delete = NULL, prefix = NULL)
```

**Arguments**

obj	An object of class MSnSet.
idLine2Delete	The name of the column that correspond to the data to filter
prefix	A character string that is the prefix to find in the data

**Value**

An object of class MSnSet.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
removeLines(Exp1_R25_pept[1:100], "Potential_contaminant")
removeLines(Exp1_R25_pept[1:100], "Reverse")
```

---

samLRT	<i>xxxxxx</i>
--------	---------------

---

**Description**

This function computes a regularized version of the likelihood ratio statistic. The regularization adds a user-input fudge factor `s1` to the variance estimator. This is straightforward when using a fixed effect model (cases 'numeric' and 'lm') but requires some more care when using a mixed model.

**Usage**

```
samLRT(lmm.res.h0, lmm.res.h1, cc, n, p, s1)
```

**Arguments**

lmm.res.h0	a vector of object containing the estimates (used to compute the statistic) under H0 for each connected component. If the fast version of the estimator was used (as implemented in this package), lmm.res.h0 is a vector containing averages of squared residuals. If a fixed effect model was used, it is a vector of lm objects and if a mixed effect model was used it is a vector or lmer object.
lmm.res.h1	similar to lmm.res.h0, a vector of object containing the estimates (used to compute the statistic) under H1 for each protein.
cc	a list containing the indices of peptides and proteins belonging to each connected component.
n	the number of samples used in the test
p	the number of proteins in the experiment
s1	the fudge factor to be added to the variance estimate

**Value**

llr.sam: a vector of numeric containing the regularized log likelihood ratio statistic for each protein.  
s: a vector containing the maximum likelihood estimate of the variance for the chosen model. When using the fast version of the estimator implemented in this package, this is the same thing as the input lmm.res.h1. lh1.sam: a vector of numeric containing the regularized log likelihood under H1 for each protein. lh0.sam: a vector of numeric containing the regularized log likelihood under H0 for each connected component. sample.sizes: a vector of numeric containing the sample size (number of biological samples times number of peptides) for each protein. This number is the same for all proteins within each connected component.

**Author(s)**

Thomas Burger, Laurent Jacob

---

saveParameters *Saves the parameters of a tool in the pipeline of Prostar*

---

**Description**

Saves the parameters of a tool in the pipeline of Prostar

**Usage**

```
saveParameters(obj, name.dataset = NULL, name = NULL, l.params = NULL)
```

**Arguments**

obj	An object of class MSnSet
name.dataset	The name of the dataset
name	The name of the tool. Available values are: "Norm, Imputation, anaDiff, GO-Analysis,Aggregation"
l.params	A list that contains the parameters



**Value**

An instance of class MSnSet.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
l.params=list(method="Global quantile alignment", type="overall")
saveParameters(Exp1_R25_pept, "Filtered.peptide", "Imputation",l.params)
```

---

scatterplotEnrichGO\_HC

*A dotplot that shows the result of a GO enrichment, using the package highcharter*

---

**Description**

A scatter plot of GO enrichment analysis

**Usage**

```
scatterplotEnrichGO_HC(ego, maxRes = 10, title = NULL)
```

**Arguments**

ego	The result of the GO enrichment, provides either by the function enrichGO in DAPAR or the function enrichGO of the package 'clusterProfiler'
maxRes	The maximum number of categories to display in the plot
title	The title of the plot

**Value**

A dotplot

**Author(s)**

Samuel Wieczorek

---

`search.metacell.tags`    *Search pattern in metacell vocabulary*

---

**Description**

Gives all the tags of the metadata vocabulary containing the pattern (parent and all its children).

**Usage**

```
search.metacell.tags(pattern, level, depth = "1")
```

**Arguments**

pattern	The string to search.
level	The available levels are : names()
depth	xxx

**Author(s)**

Samuel Wiczorek

**Examples**

```
search.metacell.tags('missing POV', 'peptide')
search.metacell.tags('quanti', 'peptide')
```

---

`SetCC`                    *Returns the connected components*

---

**Description**

Returns the connected components

**Usage**

```
SetCC(obj, cc)
```

**Arguments**

obj	An object (peptides) of class MSnSet.
cc	The connected components list

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
Xshared <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", FALSE)
Xunique <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", TRUE)
ll.X <- list(matWithSharedPeptides = Xshared, matWithUniquePeptides = Xunique)
Exp1_R25_pept <- SetMatAdj(Exp1_R25_pept, ll.X)
ll1 <- get.pep.prot.cc(GetMatAdj(Exp1_R25_pept)$matWithSharedPeptides)
ll2 <- DAPAR::get.pep.prot.cc(GetMatAdj(Exp1_R25_pept)$matWithUniquePeptides)
cc <- list(allPep = ll1, onlyUniquePep = ll2)
Exp1_R25_pept <- SetCC(Exp1_R25_pept, cc)
```

---

**SetMatAdj***Record the adjacency matrices in a slot of the dataset of class MSnSet*

---

**Description**

Record the adjacency matrices in a slot of the dataset of class MSnSet

**Usage**

```
SetMatAdj(obj, X)
```

**Arguments**

obj	An object (peptides) of class MSnSet.
X	A list of two adjacency matrices

**Value**

NA

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
Xshared <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", FALSE)
Xunique <- BuildAdjacencyMatrix(Exp1_R25_pept[1:1000], "Protein_group_IDs", TRUE)
ll.X <- list(matWithSharedPeptides = Xshared, matWithUniquePeptides = Xunique)
Exp1_R25_pept <- SetMatAdj(Exp1_R25_pept, ll.X)
```

---

Set\_POV\_MEC\_tags      *Sets the MEC tag in the metacell*

---

### Description

This function is based on the metacell dataframe to look for either missing values (used to update an initial dataset) or imputed values (used when post processing protein metacell after aggregation)

### Usage

```
Set_POV_MEC_tags(conds, df, level)
```

### Arguments

conds	xxx
df	An object of class MSnSet
level	Type of entity/pipeline

### Value

An instance of class MSnSet.

### Author(s)

Samuel Wieczorek

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10]
cols.for.ident <- obj@experimentData@other$names_metacell
conds <- Biobase::pData(obj)$Condition
df <- Biobase::fData(obj)[, cols.for.ident]
df <- Set_POV_MEC_tags(conds, df, level = 'peptide')
```

---

splitAdjacencyMat      *splits an adjacency matrix into specific and shared*

---

### Description

Method to split an adjacency matrix into specific and shared

### Usage

```
splitAdjacencyMat(X)
```

**Arguments**

X                    An adjacency matrix

**Value**

A list of two adjacency matrices

**Author(s)**

Samuel Wieczorek

---

StringBasedFiltering    *Removes lines in the dataset based on a prefix strings (contaminants, reverse or both).*

---

**Description**

This function removes lines in the dataset based on prefix strings (contaminants, reverse or both).

**Usage**

```
StringBasedFiltering(  
  obj,  
  idCont2Delete = NULL,  
  prefix_Cont = NULL,  
  idRev2Delete = NULL,  
  prefix_Rev = NULL  
)
```

**Arguments**

obj                    An object of class MSnSet.  
idCont2Delete        The name of the column that correspond to the contaminants to filter  
prefix\_Cont          A character string that is the prefix for the contaminants to find in the data  
idRev2Delete        The name of the column that correspond to the reverse data to filter  
prefix\_Rev            A character string that is the prefix for the reverse to find in the data

**Value**

An list of 4 items : obj : an object of class MSnSet in which the lines have been deleted  
deleted.both : an object of class MSnSet which contains the deleted lines corresponding to both contaminants and reverse,  
deleted.contaminants : n object of class MSnSet which contains the deleted lines corresponding to contaminants,  
deleted.reverse : an object of class MSnSet which contains the deleted lines corresponding to reverse,

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
StringBasedFiltering(Exp1_R25_pept[1:100], 'Potential_contaminant', '+',
'Reverse', '+')
```

---

StringBasedFiltering2 *Removes lines in the dataset based on a prefix strings.*

---

**Description**

This function removes lines in the dataset based on prefix strings.

**Usage**

```
StringBasedFiltering2(obj, cname = NULL, tag = NULL)
```

**Arguments**

obj	An object of class MSnSet.
cname	The name of the column that correspond to the line to filter
tag	A character string that is the prefix for the contaminants to find in the data

**Value**

An list of 4 items : obj : an object of class MSnSet in which the lines have been deleted deleted : an object of class MSnSet which contains the deleted lines

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.filter <- StringBasedFiltering2(Exp1_R25_pept[1:100], 'Potential_contaminant', '+')
```

---

SumByColumns	<i>Normalisation SumByColumns</i>
--------------	-----------------------------------

---

**Description**

Normalisation SumByColumns

**Usage**

```
SumByColumns(qData, conds = NULL, type = NULL, subset.norm = NULL)
```

**Arguments**

qData	xxxx
conds	xxx
type	Available values are "overall" (shift all the sample distributions at once) or "within conditions" (shift the sample distributions within each condition at a time).
subset.norm	A vector of index indicating rows to be used for normalization

**Value**

A normalized numeric matrix

**Author(s)**

Samuel Wiczorek, Thomas Burger, Helene Borges, Anais Courtier, Enora Fremy

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
conds <- Biobase::pData(Exp1_R25_pept)$Condition
normalized <- SumByColumns(qData, conds, type="within conditions",
subset.norm=1:10)
```

---

SymFilteringOperators	xxx
-----------------------	-----

---

**Description**

xxx

**Usage**

```
SymFilteringOperators()
```

---

test.design	<i>Check if xxxxxx</i>
-------------	------------------------

---

**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, Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
test.design(Biobase::pData(Exp1_R25_pept)[,1:3])
```

---

translatedRandomBeta	<i>Generator of simulated values</i>
----------------------	--------------------------------------

---

**Description**

Generator of simulated values

**Usage**

```
translatedRandomBeta(n, min, max, param1 = 3, param2 = 1)
```

**Arguments**

n	An integer which is the number of simulation (same as in rbeta)
min	An integer that corresponds to the lower bound of the interval
max	An integer that corresponds to the upper bound of the interval
param1	An integer that is the first parameter of rbeta function.
param2	An integer that is second parameter of rbeta function.



**Value**

A vector of n simulated values

**Author(s)**

Thomas Burger

**Examples**

```
translatedRandomBeta(1000, 5, 10, 1, 1)
```

---

univ_AnnotDbPkg	<i>Returns the totality of ENTREZ ID (gene id) of an OrgDb annotation package. Careful : org.Pf.plasmo.db : no ENTREZID but ORF</i>
-----------------	---

---

**Description**

Function to compute the "universe" argument for the `enrich_GO` function, in case this latter should be the entire organism. Returns all the ID of the OrgDb annotation package for the corresponding organism.

**Usage**

```
univ_AnnotDbPkg(orgdb)
```

**Arguments**

orgdb            a Bioconductor OrgDb annotation package

**Value**

A vector of ENTREZ ID

**Author(s)**

Florence Combes

UpdateMetacell      *Update metacell after imputation*

---

**Description**

Update the metacell information of missing values that were imputed

**Usage**

```
UpdateMetacell(obj, method = "", na.type)
```

**Arguments**

obj	xxx
method	xxx
na.type	xxx

**Author(s)**

Samuel Wieczorek

---

violinPlotD      *Builds a violinplot from a dataframe*

---

**Description**

Builds a violinplot from a dataframe

**Usage**

```
violinPlotD(obj, conds, keyId, legend = NULL, pal = NULL, subset.view = NULL)
```

**Arguments**

obj	xxx
conds	xxx
keyId	xxx
legend	A vector of the conditions (one condition per sample).
pal	xxx
subset.view	xxx

**Value**

A violinplot

**Author(s)**

Samuel Wieczorek, Anais Courtier

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot
legend <- conds <- Biobase::pData(obj)$Condition
key <- "Protein_IDs"
violinPlotD(obj, conds, key, legend, subset.view=1:10)
```

---

visualizeClusters      *Visualize the clusters according to pvalue thresholds*

---

**Description**

Visualize the clusters according to pvalue thresholds

**Usage**

```
visualizeClusters(
  dat,
  clust_model,
  adjusted_pValues,
  FDR_th = NULL,
  ttl = "",
  subttl = ""
)
```

**Arguments**

dat	the standardize data returned by the function [checkClusterability()]
clust_model	the clustering model obtained with dat.
adjusted_pValues	vector of the adjusted pvalues obtained for each protein with a 1-way ANOVA (for example obtained with the function [wrapperClassic1wayAnova()]).
FDR_th	the thresholds of FDR pvalues for the coloring of the profiles. The default (NULL) creates 4 thresholds: 0.001, 0.005, 0.01, 0.05 For the sake of readability, a maximum of 4 values can be specified.
ttl	title for the plot.
subttl	subtitle for the plot.

**Value**

a ggplot object

**Author(s)**

Helene Borges

**Examples**

```

library(dplyr)
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
expR25_ttest <- compute_t_tests(obj$new)
averaged_means <- averageIntensities(obj$new)
only_means <- dplyr::select_if(averaged_means, is.numeric)
only_features <- dplyr::select_if(averaged_means, is.character)
means <- purrr::map(purrr::array_branch(as.matrix(only_means), 1), mean)
centered <- only_means - unlist(means)
centered_means <- dplyr::bind_cols(feature = dplyr::as_tibble(only_features),
dplyr::as_tibble(centered))
difference <- only_means[,1] - only_means[,2]
clusters <- as.data.frame(difference) %>%
  dplyr::mutate(cluster = dplyr::if_else(difference > 0, 1,2))
vizu <- visualizeClusters(dat = centered_means,
                          clust_model = as.factor(clusters$cluster),
                          adjusted_pValues = expR25_ttest$P_Value$`25fmol_vs_10fmol_pval`,
                          FDR_th = c(0.001,0.005,0.01,0.05),
                          ttl = "Clustering of protein profiles")

```

vsn

*Normalisation vsn***Description**

Normalisation vsn

**Usage**

vsn(qData, conds, type = NULL)

**Arguments**

qData	A numeric matrix.
conds	xxx
type	"overall" (shift all the sample distributions at once) or "within conditions" (shift the sample distributions within each condition at a time).

**Value**

A normalized numeric matrix

**Author(s)**

Thomas Burger, Helene Borges, Anais Courtier, Enora Fremy

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
qData <- Biobase::exprs(Exp1_R25_pept)
conds <- Biobase::pData(Exp1_R25_pept)$Condition
normalized <- vsn(qData, conds, type="overall")
```

---

wrapper.compareNormalizationD\_HC

*Builds a plot from a dataframe*

---

**Description**

Wrapper to the function that plot to compare the quantitative proteomics data before and after normalization.

**Usage**

```
wrapper.compareNormalizationD_HC(
  objBefore,
  objAfter,
  condsForLegend = NULL,
  ...
)
```

**Arguments**

objBefore      A dataframe that contains quantitative data before normalization.  
objAfter        A dataframe that contains quantitative data after normalization.  
condsForLegend A vector of the conditions (one condition per sample).  
...             arguments for palette

**Value**

A plot

**Author(s)**

Samuel Wiczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept
conds <- Biobase::pData(obj)[,"Condition"]
objAfter <- wrapper.normalizeD(obj = obj, method = "QuantileCentering",
conds=conds, type = "within conditions")
wrapper.compareNormalizationD_HC(obj, objAfter, conds)
```

---

```
wrapper.corrMatrixD_HC
```

*Displays a correlation matrix of the quantitative data of the Biobase::exprs() table*

---

**Description**

```
#' Builds a correlation matrix based on a MSnSet object. #' #' @title Displays a correlation matrix
of the quantitative data of the #' Biobase::exprs() table #' #' @param obj An object of class
MSnSet. #' #' @param rate A float that defines the gradient of colors. #' #' @return A colored cor-
relation matrix #' #' @author Alexia Dorffer #' #' @examples #' utils::data(Exp1_R25_pept, pack-
age='DAPARdata') #' wrapper.corrMatrixD(Exp1_R25_pept) #' #' #' @export #' wrapper.corrMatrixD
<- function(obj, rate=5) qData <- Biobase::exprs(obj) samplesData <- Biobase::pData(obj) corrMa-
trixD(qData, samplesData, rate)
```

Builds a correlation matrix based on a MSnSet object.

**Usage**

```
wrapper.corrMatrixD_HC(obj, rate = 0.5, showValues = TRUE)
```

**Arguments**

obj	An object of class MSnSet.
rate	A float that defines the gradient of colors.
showValues	xxx

**Value**

A colored correlation matrix

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
wrapper.corrMatrixD_HC(Exp1_R25_pept)
```

---

wrapper.CVDistD\_HC      *Distribution of CV of entities*

---

### Description

Builds a densityplot of the CV of entities in the Biobase::exprs() table. of an object MSnSet. The variance is calculated for each condition present in the dataset (see the slot 'Condition' in the Biobase::pData() table).

### Usage

```
wrapper.CVDistD_HC(obj, ...)
```

### Arguments

obj	An object of class MSnSet
...	arguments for palette.

### Value

A density plot

### Author(s)

Samuel Wiczorek

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
wrapper.CVDistD_HC(Exp1_R25_pept)
```

---

wrapper.dapar.impute.mi  
*Missing values imputation using the LSImpute algorithm.*

---

### Description

This method is a wrapper to the function impute.mi of the package imp4p adapted to an object of class MSnSet.

**Usage**

```

wrapper.dapar.impute.mi(
  obj,
  nb.iter = 3,
  nknn = 15,
  selec = 600,
  siz = 500,
  weight = 1,
  ind.comp = 1,
  progress.bar = FALSE,
  x.step.mod = 300,
  x.step.pi = 300,
  nb.rei = 100,
  method = 4,
  gridsize = 300,
  q = 0.95,
  q.min = 0,
  q.norm = 3,
  eps = 0,
  methodi = "slsa",
  lapala = TRUE,
  distribution = "unif"
)

```

**Arguments**

obj	An object of class MSnSet.
nb.iter	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
nknn	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
selec	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
siz	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
weight	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
ind.comp	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
progress.bar	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
x.step.mod	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
x.step.pi	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
nb.rei	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
method	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
gridsize	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
q	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
q.min	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>
q.norm	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>
eps	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>



methodi            Same as the function `mi.mix` in the package `imp4p`  
 lapala            xxxxxxxxxxxx  
 distribution      The type of distribution used. Values are `unif` (default) or `beta`.

**Value**

The `Biobase::exprs(obj)` matrix with imputed values instead of missing values.

**Author(s)**

Samuel Wiczorek

**Examples**

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:100]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj.imp.na <- wrapper.dapar.impute.mi(obj, nb.iter=1, lapala = TRUE)
obj.imp.pov <- wrapper.dapar.impute.mi(obj, nb.iter=1, lapala = FALSE)

## End(Not run)
```

---

wrapper.heatmapD	<i>This function is a wrapper to <a href="#">heatmap.2</a> that displays quantitative data in the <code>Biobase::exprs()</code> table of an object of class <code>MSnSet</code></i>
------------------	---

---

**Description**

Builds a heatmap of the quantitative proteomic data of a `MSnSet` object.

**Usage**

```
wrapper.heatmapD(
  obj,
  distance = "euclidean",
  cluster = "complete",
  dendro = FALSE
)
```

**Arguments**

obj	An object of class MSnSet.
distance	The distance used by the clustering algorithm to compute the dendrogram. See <code>help(heatmap.2)</code> .
cluster	the clustering algorithm used to build the dendrogram. See <code>help(heatmap.2)</code>
dendro	A boolean to indicate if the dendrogram has to be displayed

**Value**

A heatmap

**Author(s)**

Alexia Dorffer

**Examples**

```
## Not run:
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeLine(metacell.mask)
wrapper.heatmapD(obj)

## End(Not run)
```

---

wrapper.impute.detQuant

*Wrapper of the function 'impute.detQuant' for objects of class MSnSet*

---

**Description**

This method is a wrapper of the function 'impute.detQuant' for objects of class MSnSet

**Usage**

```
wrapper.impute.detQuant(obj, qual = 0.025, factor = 1, na.type)
```

**Arguments**

obj	An instance of class MSnSet
qual	An expression set containing quantitative values of various replicates
factor	A scaling factor to multiply the imputation value with
na.type	A string which indicates the type of missing values to impute. Available values are: 'NA' (for both POV and MEC), 'POV', 'MEC'.

**Value**

An imputed instance of class MSnSet

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10]
obj.imp.pov <- wrapper.impute.detQuant(obj, na.type='missing POV')
obj.imp.mec <- wrapper.impute.detQuant(obj, na.type='missing MEC')
obj.imp.na <- wrapper.impute.detQuant(obj, na.type='missing')
```

---

wrapper.impute.fixedValue

*Missing values imputation from a MSnSet object*

---

**Description**

This method is a wrapper to objects of class MSnSet and imputes missing values with a fixed value.

**Usage**

```
wrapper.impute.fixedValue(obj, fixVal = 0, na.type)
```

**Arguments**

obj	An object of class MSnSet.
fixVal	A float.
na.type	A string which indicates the type of missing values to impute. Available values are: 'NA' (for both POV and MEC), 'POV', 'MEC'.

**Value**

The object obj which has been imputed

**Author(s)**

Samuel Wieczorek

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10,]
obj.imp.pov <- wrapper.impute.fixedValue(obj, 0.001, na.type = 'missing POV')
obj.imp.mec <- wrapper.impute.fixedValue(obj, 0.001, na.type = 'missing MEC')
obj.imp.na <- wrapper.impute.fixedValue(obj, 0.001, na.type = 'missing')
```

---

wrapper.impute.KNN      *KNN missing values imputation from a MSnSet object*

---

## Description

Can impute only POV missing values

## Usage

```
wrapper.impute.KNN(obj = NULL, K, na.type)
```

## Arguments

obj	An object of class MSnSet.
K	the number of neighbors.
na.type	A string which indicates the type of missing values to impute. Available values are: 'missing POV'.

## Details

This method is a wrapper for objects of class MSnSet and imputes missing values with a fixed value. This function imputes the missing values condition by condition.

## Value

The object obj which has been imputed

## Author(s)

Samuel Wieczorek

## Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.imp.pov <- wrapper.impute.KNN(obj = Exp1_R25_pept[1:10], K=3,
na.type = 'missing POV')
```

---

wrapper.impute.mle      *Imputation of peptides having no values in a biological condition.*

---

**Description**

This method is a wrapper to the function `impute.mle` of the package `imp4p` adapted to an object of class `MSnSet`.

**Usage**

```
wrapper.impute.mle(obj, na.type)
```

**Arguments**

`obj`                      An object of class `MSnSet`.  
`na.type`                  A string which indicates the type of missing values to impute. Available values are: 'NA' (for both POV and MEC).

**Value**

The `Biobase::exprs(obj)` matrix with imputed values instead of missing values.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10,]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj.imp.na <- wrapper.impute.mle(obj, na.type = 'missing')
```

---

wrapper.impute.pa      *Imputation of peptides having no values in a biological condition.*

---

**Description**

This method is a wrapper to the function `impute.pa` of the package `imp4p` adapted to an object of class `MSnSet`.

**Usage**

```
wrapper.impute.pa(obj = NULL, q.min = 0.025, na.type)
```

**Arguments**

obj	An object of class MSnSet.
q.min	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>
na.type	A string which indicates the type of missing values to impute. Available values are: 'NA' (for both POV and MEC).

**Value**

The `Biobase::exprs(obj)` matrix with imputed values instead of missing values.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj.imp.pov <- wrapper.impute.pa(obj, na.type = 'missing POV')
```

---

`wrapper.impute.pa2`      *Missing values imputation from a MSnSet object*

---

**Description**

This method is a wrapper to the function `impute.pa2` adapted to objects of class `MSnSet`.

**Usage**

```
wrapper.impute.pa2(obj, q.min = 0, q.norm = 3, eps = 0, distribution = "unif")
```

**Arguments**

obj	An object of class <code>MSnSet</code> .
q.min	A quantile value of the observed values allowing defining the maximal value which can be generated. This maximal value is defined by the quantile <code>q.min</code> of the observed values distribution minus <code>eps</code> . Default is 0 (the maximal value is the minimum of observed values minus <code>eps</code> ).
q.norm	A quantile value of a normal distribution allowing defining the minimal value which can be generated. Default is 3 (the minimal value is the maximal value minus <code>qn*median(sd(observed values))</code> where <code>sd</code> is the standard deviation of a row in a condition).

eps                    A value allowing defining the maximal value which can be generated. This maximal value is defined by the quantile `q.min` of the observed values distribution minus `eps`. Default is 0.

distribution        The type of distribution used. Values are `unif` (default) or `beta`.

**Value**

The object `obj` which has been imputed

**Author(s)**

Thomas Burger, Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj.imp.pa2 <- wrapper.impute.pa2(Exp1_R25_pept[1:100], distribution="beta")
```

---

wrapper.impute.slsa     *Imputation of peptides having no values in a biological condition.*

---

**Description**

```
# This method replaces each missing value by a given value #' #' @title Deterministic imputation
#' #' @param qdata An expression set containing quantitative or missing values #' #' @param meta-
data xxx #' #' @param values A vector with as many elements as the number of colums #' of qdata
#' #' @param na.type xxx #' #' @return An imputed dataset #' #' @author Thomas Burger, Samuel
Wieczorek #' #' @examples #' utils::data(Exp1_R25_pept, package='DAPARdata') #' qdata <-
Biobase::exprs(Exp1_R25_pept) #' meta <- #' values <- getQuantile4Imp(qdata)$shiftedImpVal #'
obj.imp.mec <- impute.detQuant(qdata, values, na.type = 'missing POV') #' obj.imp.mec <- im-
pute.detQuant(qdata, values, na.type = 'missing MEC') #' obj.imp.na <- impute.detQuant(qdata,
values, na.type = 'missing') #' #' @export #' impute.detQuant <- function(qdata, metadata, values,
na.type=NULL) availablePatterns <- unname(search.metacell.tags(pattern=na.type, level=obj@experimentData@other$typeOfData))
if (is.null(na.type)) stop(paste0("'na.type' is required. Available values are:", paste0(metacell.def(),
collapse=' '))) else if (!(na.type level=obj@experimentData@other$typeOfData)) stop(paste0("Available
values for na.type are: ", paste0(availablePatterns, collapse=' ')))
if(missing(metadata)) stop("'metadata' is missing.")
#browser() for(i in 1:ncol(qdata)) col <- qdata[,i] ind.na.type <- match.metacell(Biobase::fData(obj)[,
obj@experimentData@other$names_metacell[i]], pattern = na.type, level = obj@experimentData@other$typeOfData)
col[which(is.na(col) & ind.na.type)] <- values[i] qdata[,i] <- col
return(qdata)
```

**Usage**

```
wrapper.impute.slsa(obj = NULL, na.type = NULL)
```

**Arguments**

obj	An object of class MSnSet.
na.type	A string which indicates the type of missing values to impute. Available values are: 'NA' (for both POV and MEC), 'POV', 'MEC'.

**Details**

This method is a wrapper to the function `impute.slsa` of the package `imp4p` adapted to an object of class `MSnSet`.

**Value**

The `Biobase::exprs(obj)` matrix with imputed values instead of missing values.

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:100]
obj.slsa.pov <- wrapper.impute.slsa(obj, na.type = 'missing POV')
```

---

wrapper.mvImage

*Heatmap of missing values from a MSnSet object*

---

**Description**

Plots a heatmap of the quantitative data. Each column represent one of the conditions in the object of class `MSnSet` and the color is proportional to the mean of intensity for each line of the dataset. The lines have been sorted in order to visualize easily the different number of missing values. A white square is plotted for missing values.

**Usage**

```
wrapper.mvImage(obj, pattern = "missing MEC")
```

**Arguments**

obj	An object of class MSnSet.
pattern	xxx

**Value**

A heatmap



**Author(s)**

Alexia Dorffer

**Examples**

```

utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
wrapper.mvImage(obj$new)

```

---

 wrapper.normalized      *Normalisation*


---

**Description**

Provides several methods to normalize quantitative data from a MSnSet object. They are organized in six main families : GlobalQuantileAlignment, sumByColumns, QuantileCentering, MeanCentering, LOESS, vsn For the first family, there is no type. For the five other families, two type categories are available : "Overall" which means that the value for each protein (ie line in the expression data tab) is computed over all the samples ; "within conditions" which means that the value for each protein (ie line in the Biobase::exprs() data tab) is computed condition by condition.

**Usage**

```
wrapper.normalized(obj, method, withTracking = FALSE, ...)
```

**Arguments**

obj	An object of class MSnSet.
method	One of the following : "GlobalQuantileAlignment" (for normalizations of important magnitude), "SumByColumns", "QuantileCentering", "Mean Centering", "LOESS" and "vsn".
withTracking	xxx
...	xxx

**Author(s)**

Samuel Wieczorek, Thomas Burger, Helene Borges

**Examples**

```
utils::data(Exp1_R25_pept, package='DAPARdata')
conds <- Biobase::pData(Exp1_R25_pept)$Condition
obj <- wrapper.normalized(obj = Exp1_R25_pept, method = "QuantileCentering",
conds=conds, type = "within conditions")
```

---

 wrapper.pca

*Compute the PCA*


---

**Description**

Compute the PCA

**Usage**

```
wrapper.pca(obj, var.scaling = TRUE, ncp = NULL)
```

**Arguments**

obj	xxx
var.scaling	The dimensions to plot
ncp	xxxx

**Value**

A xxxxxx

**Author(s)**

Samuel Wieczorek

**Examples**

```
utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
res.pca <- wrapper.pca(obj$new)
```

---

wrapperCalibrationPlot

*Performs a calibration plot on an MSnSet object, calling the cp4p package functions.*

---

### Description

This function is a wrapper to the calibration.plot method of the cp4p package for use with MSnSet objects.

### Usage

```
wrapperCalibrationPlot(vPVal, pi0Method = "pounds")
```

### Arguments

vPVal            A dataframe that contains quantitative data.  
pi0Method        A vector of the conditions (one condition per sample).

### Value

A plot

### Author(s)

Samuel Wieczorek

### Examples

```
utils::data(Exp1_R25_prot, package='DAPARdata')  
obj <- Exp1_R25_prot[1:1000]  
level <- obj@experimentData@other$typeOfData  
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)  
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)  
obj <- MetaCellFiltering(obj, indices, cmd='delete')  
qData <- Biobase::exprs(obj$new)  
sTab <- Biobase::pData(obj$new)  
limma <- limmaCompleteTest(qData,sTab)  
wrapperCalibrationPlot(limma$P_Value[,1])
```

---

`wrapperClassic1wayAnova`*Wrapper for One-way Anova statistical test*

---

**Description**

Wrapper for One-way Anova statistical test

**Usage**

```
wrapperClassic1wayAnova(obj, with_post_hoc = "No", post_hoc_test = "No")
```

**Arguments**

<code>obj</code>	An object of class MSnSet.
<code>with_post_hoc</code>	a character string with 2 possible values: "Yes" and "No" (default) saying if function must perform a Post-Hoc test or not.
<code>post_hoc_test</code>	character string, possible values are "No" (for no test; default value) or "TukeyHSD" or "Dunnnett". See details of <code>postHocTest()</code> function to choose the appropriate one.

**Details**

This function allows to perform a 1-way Analysis of Variance. Also computes the post-hoc tests if the `with_post_hoc` parameter is set to yes. There are two possible post-hoc tests: the Tukey Honest Significant Differences (specified as "TukeyHSD") and the Dunnnett test (specified as "Dunnnett").

**Value**

A list of two dataframes. First one called "logFC" contains all pairwise comparisons logFC values (one column for one comparison) for each analysed feature (Except in the case without post-hoc testing, for which NAs are returned.); The second one named "P\_Value" contains the corresponding p-values.

**Author(s)**

Hélène Borges

**See Also**

[`postHocTest()`]

**Examples**

```

utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
anovatest <- wrapperClassic1wayAnova(obj$new)

```

---

wrapperRunClustering    *Run a clustering pipeline of protein/peptide abundance profiles.*

---

**Description**

This function does all of the steps necessary to obtain a clustering model and its graph from average abundances of proteins/peptides. It is possible to carry out either a kmeans model or an affinity propagation model. See details for exact steps.

**Usage**

```

wrapperRunClustering(
  obj,
  clustering_method,
  conditions_order = NULL,
  k_clusters = NULL,
  adjusted_pvals,
  ttl = "",
  subttl = "",
  FDR_thresholds = NULL
)

```

**Arguments**

obj	ExpressionSet or MSnSet object.
clustering_method	character string. Three possible values are "kmeans", "affinityProp" and "affinityPropReduced". See the details section for more explanation.
conditions_order	vector specifying the order of the Condition factor levels in the phenotype data. Default value is NULL, which means that it is the order of the condition present in the phenotype data of "obj" which is taken to create the profiles.
k_clusters	integer or NULL. Number of clusters to run the kmeans algorithm. If 'clustering_method' is set to "kmeans" and this parameter is set to NULL, then a kmeans model will be realized with an optimal number of clusters 'k' estimated by the Gap statistic method. Ignored for the Affinity propagation model.

adjusted\_pvals vector of adjusted pvalues returned by the [wrapperClassicIwayAnova()]

ttl the title for the final plot

subttl the subtitle for the final plot

FDR\_thresholds vector containing the different threshold values to be used to color the profiles according to their adjusted value. The default value (NULL) generates 4 thresholds: [0.001, 0.005, 0.01, 0.05]. Thus, there will be 5 intervals therefore 5 colors: the pvalues <0.001, those between 0.001 and 0.005, those between 0.005 and 0.01, those between 0.01 and 0.05, and those > 0.05. The highest given value will be considered as the threshold of insignificance, the profiles having a pvalue > this threshold value will then be colored in gray.

## Details

The first step consists in averaging the abundances of proteins/peptides according to the different conditions defined in the phenotype data of the expressionSet / MSnSet. Then we standardize the data if there are more than 2 conditions. If the user asks to realize a kmeans model without specifying the desired number of clusters ('clustering\_method = "kmeans"' and 'k\_clusters = NULL'), the function checks data's clusterability and estimates a number of clusters k using the gap statistic method. It is advised however to specify a k for the kmeans, because the gap stat gives the smallest possible k, whereas in biology a small number of clusters can turn out to be uninformative. If you want to run a kmeans but you don't know what number of clusters to give, you can let the pipeline run the first time without specifying 'k\_clusters', in order to view the profiles the first time and choose by the following is a more appropriate value of k. If it is assumed that the data can be structured with a large number of clusters, it is recommended to use the affinity propagation model instead. This method simultaneously considers all the data as exemplary potentials, unlike hard clustering (kmeans) which initializes with a number k of points taken at random. The "affinityProp" model will use a q parameter set to NA, meaning that exemplar preferences are set to the median of non-Inf values in the similarity matrix (set q to 0.5 will be the same). The "affinityPropReduced" model will use a q set to 0, meaning that exemplar preferences are set to the sample quantile with threshold 0 of non-Inf values. This should lead to a smaller number of final clusters.

## Value

a list of 2 elements: "model" is the clustering model, "ggplot" is the ggplot of profiles clustering.

## Author(s)

Helene Borges

## References

Tibshirani, R., Walther, G. and Hastie, T. (2001). Estimating the number of data clusters via the Gap statistic. *Journal of the Royal Statistical Society* B, 63, 411–423.

Frey, B. J. and Dueck, D. (2007) Clustering by passing messages between data points. *Science* 315, 972-976. DOI: [10.1126/science.1136800](https://doi.org/10.1126/science.1136800)

**Examples**

```

utils::data(Exp1_R25_prot, package='DAPARdata')
obj <- Exp1_R25_prot[1:1000]
level <- obj@experimentData@other$typeOfData
metacell.mask <- match.metacell(GetMetacell(obj), 'missing', level)
indices <- GetIndices_WholeMatrix(metacell.mask, op='>=', th=1)
obj <- MetaCellFiltering(obj, indices, cmd='delete')
expR25_ttest <- compute_t_tests(obj$new)
wrapperRunClustering(obj = obj$new,
adjusted_pvals = expR25_ttest$P_Value$`25fmol_vs_10fmol_pval`)

```

---

write.excel

*This function exports a data.frame to a Excel file.*


---

**Description**

This function exports a data.frame to a Excel file.

**Usage**

```
write.excel(df, tags = NULL, colors = NULL, tabname = "foo", filename = NULL)
```

**Arguments**

df	An data.frame
tags	xxx
colors	xxx
tabname	xxx
filename	A character string for the name of the Excel file.

**Value**

A Excel file (.xlsx)

**Author(s)**

Samuel Wieczorek

**Examples**

```

utils::data(Exp1_R25_pept, package='DAPARdata')
df <- Biobase::exprs(Exp1_R25_pept[1:100])
tags <- GetMetacell(Exp1_R25_pept[1:100])
colors <- list('missing POV' = "lightblue",
              'missing MEC' = "orange",
              'recovered' = "lightgrey",

```

```
        'identified' = "white",  
        'combined' = "red")  
write.excel(df, tags, colors, filename = 'toto')
```

---

writeMSnsetToCSV	<i>Exports a MSnset dataset into a zip archive containing three zipped CSV files.</i>
------------------	---

---

### Description

Exports a MSnset dataset into a zip archive containing three zipped CSV files.

### Usage

```
writeMSnsetToCSV(obj, fname)
```

### Arguments

obj	An object of class MSnSet.
fname	The name of the archive file.

### Value

A compressed file

### Author(s)

Samuel Wieczorek

### Examples

```
utils::data(Exp1_R25_pept, package='DAPARdata')  
obj <- Exp1_R2_pept[1:1000]  
writeMSnsetToCSV(obj, "foo")
```



---

writeMSnsetToExcel      *This function exports a MSnSet object to a Excel file.*

---

### Description

This function exports a MSnSet data object to a Excel file. Each of the three data.frames in the MSnSet object (ie experimental data, phenoData and metaData are respectively integrated into separate sheets in the Excel file). The colored cells in the experimental data correspond to the original missing values which have been imputed.

### Usage

```
writeMSnsetToExcel(obj, filename)
```

### Arguments

obj	An object of class MSnSet.
filename	A character string for the name of the Excel file.

### Value

A Excel file (.xlsx)

### Author(s)

Samuel Wieczorek

### Examples

```
Sys.setenv("R_ZIPCMD"= Sys.which("zip"))
utils::data(Exp1_R25_pept, package='DAPARdata')
obj <- Exp1_R25_pept[1:10]
writeMSnsetToExcel(obj, "foo")
```

# Index

aggregateIter, 6  
aggregateIterParallel, 7  
aggregateMean, 8  
AggregateMetacell, 8  
aggregateSum, 9  
aggregateTopn, 10  
averageIntensities, 11

barplotEnrichGO\_HC, 12  
barplotGroupGO\_HC, 12  
boxPlotD\_HC, 13  
BuildAdjacencyMatrix, 14  
BuildColumnToProteinDataset, 15  
BuildColumnToProteinDataset\_par, 16  
buildGraph, 17  
BuildMetaCell, 17

check.conditions, 18  
check.design, 19  
checkClusterability, 20  
classic1wayAnova, 21  
compareNormalizationD\_HC, 21  
compute\_t\_tests, 23  
corrMatrixD\_HC, 24  
CountPep, 25  
createMSnset, 25  
CVDistD\_HC, 27

dapar\_hc\_chart, 28  
dapar\_hc\_ExportMenu, 28  
deleteLinesFromIndices, 29  
densityPlotD\_HC, 30  
diffAnaComputeFDR, 31  
diffAnaGetSignificant, 32  
diffAnaSave, 33  
diffAnaVolcanoplot, 34  
diffAnaVolcanoplot\_rCharts, 35  
display.CC.visNet, 37

enrich\_GO, 38

ExtendPalette, 39

finalizeAggregation, 40  
findMECBlock, 40  
formatLimmaResult, 41  
formatPHResults, 42  
fudge2LRT, 43

get.pep.prot.cc, 44  
Get\_AllComparisons, 65  
GetCC, 45  
GetColorsForConditions, 45  
GetDetailedNbPeptides, 46  
GetDetailedNbPeptidesUsed, 47  
GetIndices\_BasedOnConditions, 49  
GetIndices\_MetacellFiltering, 50  
GetIndices\_WholeLine, 51  
GetIndices\_WholeMatrix, 51  
getIndicesConditions, 48  
getIndicesOfLinesToRemove, 48  
GetKeyId, 52  
getListNbValuesInLines, 53  
GetMatAdj, 53  
GetMetacell, 54  
GetNbPeptidesUsed, 54  
getNumberOf, 55  
getNumberOfEmptyLines, 56  
getPourcentageOfMV, 56  
getProcessingInfo, 57  
getProteinsStats, 58  
getQuantile4Imp, 58  
getTextForAggregation, 59  
getTextForAnaDiff, 60  
getTextForFiltering, 60  
getTextForGOAnalysis, 61  
getTextForHypothesisTest, 61  
getTextForNewDataset, 62  
getTextForNormalization, 63  
getTextForpeptideImputation, 63  
getTextForproteinImputation, 64

- GetTypeofData, 65
- GlobalQuantileAlignment, 66
- GOAnalysisSave, 67
- GraphPepProt, 68
- group\_GO, 68
  
- hc\_logFC\_DensityPlot, 69
- hc\_mvTypePlot2, 70
- heatmap.2, 71, 72, 129
- heatmapD, 71
- heatmapForMissingValues, 72
- histPValue\_HC, 73
  
- impute.pa2, 74
- inner.aggregate.iter, 75
- inner.aggregate.topn, 76
- inner.mean, 77
- inner.sum, 77
  
- LH0, 78
- LH0.lm, 78
- LH1, 79
- LH1.lm, 80
- limma, 33
- limmaCompleteTest, 31, 33, 80
- listSheets, 81
- LOESS, 82
  
- make.contrast, 82
- make.design, 83
- make.design.1, 84
- make.design.2, 84
- make.design.3, 85
- match.metacell, 86
- MeanCentering, 86
- metacell.def, 87
- Metacell\_generic, 92
- Metacell\_maxquant, 93
- Metacell\_proline, 94
- MetaCellFiltering, 88
- MetacellFilteringScope, 89
- metacellHisto\_HC, 89
- metacellPerLinesHisto\_HC, 91
- metacellPerLinesHistoPerCondition\_HC, 90
- metacombine, 95
- mvImage, 96
- my\_hc\_chart, 97
- my\_hc\_ExportMenu, 97
  
- nonzero, 98
- normalizeMethods.dapar, 99
- NumericalFiltering, 99
- NumericalgetIndicesOfLinesToRemove, 100
  
- pepa.test, 101
- plotJitter, 102
- plotJitter\_rCharts, 102
- plotPCA\_Eigen, 103
- plotPCA\_Eigen\_hc, 104
- plotPCA\_Ind, 104
- plotPCA\_Var, 105
- postHocTest, 106
- proportionConRev\_HC, 107
  
- QuantileCentering, 108
  
- rbindMSnset, 109
- readExcel, 109
- reIntroduceMEC, 110
- removeLines, 111
  
- samLRT, 111
- saveParameters, 112
- scatterplotEnrichGO\_HC, 113
- search.metacell.tags, 114
- Set\_POV\_MEC\_tags, 116
- SetCC, 114
- SetMatAdj, 115
- splitAdjacencyMat, 116
- StringBasedFiltering, 117
- StringBasedFiltering2, 118
- SumByColumns, 119
- SymFilteringOperators, 119
  
- test.design, 120
- translatedRandomBeta, 120
  
- univ\_AnnotDbPkg, 121
- UpdateMetacell, 122
  
- violinPlotD, 122
- visualizeClusters, 123
- vsn, 124
  
- wrapper.compareNormalizationD\_HC, 125
- wrapper.corrMatrixD\_HC, 126
- wrapper.CVDistD\_HC, 127
- wrapper.dapar.impute.mi, 127

- wrapper.heatmapD, [129](#)
- wrapper.impute.detQuant, [130](#)
- wrapper.impute.fixedValue, [131](#)
- wrapper.impute.KNN, [132](#)
- wrapper.impute.mle, [133](#)
- wrapper.impute.pa, [133](#)
- wrapper.impute.pa2, [134](#)
- wrapper.impute.slsa, [135](#)
- wrapper.mvImage, [136](#)
- wrapper.normalizedD, [137](#)
- wrapper.pca, [138](#)
- wrapperCalibrationPlot, [139](#)
- wrapperClassic1wayAnova, [140](#)
- wrapperRunClustering, [141](#)
- write.excel, [143](#)
- writeMSnsetToCSV, [144](#)
- writeMSnsetToExcel, [145](#)