

# Package: DaparToolshed (via r-universe)

May 19, 2026

**Type** Package

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

**Version** 0.99.36

**Date** 2026-05-07

**Description** The package DaparToolshed is a Bioconductor distributed R package which provides all the necessary functions to analyze quantitative data from label-free proteomics experiments. It is an update of our previous package DAPAR and contains more functions to analyze the data and uses MultAssayExperiment and SummarizedExperiment data structures. Contrarily to most other similar R packages, it is endowed with rich and user-friendly graphical interfaces, so that no programming skill is required (see `Prostar` package).

**URL** <https://github.com/edyp-lab/DaparToolshed>,  
<https://edyp-lab.github.io/DaparToolshed/>

**BugReports** <https://github.com/edyp-lab/DaparToolshed/issues>

**License** Artistic-2.0

**Depends** R (>= 4.5.0)

**Imports** BiocGenerics, matrixStats, S4Vectors, SummarizedExperiment, openxlsx, QFeatures (>= 1.16), methods, MsCoreUtils, AnnotationFilter, RColorBrewer, plotly, stats, utils, tibble, Matrix, stringr, graph, igraph, preprocessCore, dplyr

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

**Suggests** BiocManager, visNetwork, htmlwidgets, rhandsontable, shinyalert, shinyjs, sos, knitr, shinyBS, MultiAssayExperiment, PSMATCH, lme4, DT, shinyWidgets, vsn, cp4p, limma, imp4p (>= 0.9), impute, apcluster, diptest, cluster, rmarkdown, BiocStyle, testthat, FactoMineR, factoextra, colourpicker, readxl, grDevices, forcats, multcomp, purrr, gplots, tidyr, Pirat, shinyjqui

**NeedsCompilation** no

**Collate** history\_utils.R plots\_volcano.R DiffAnalysis.R  
 adjacencyMatrix.R plots\_compare\_assays.R  
 missingValuesImputation\_ProteinLevel.R  
 missingValuesImputation\_PeptideLevel.R hypothesisTest\_plot.R  
 anova\_analysis.R compute.t.tests.R limmaAnalysis.R  
 plots\_normalization.R funcs\_FunctionFiltering.R plot\_metacell.R  
 funcs\_normalize.R doc-data.R QFeatures\_utils.R output2Excel.R  
 DaparToolshed\_Params\_Addons.R metacell.R  
 DaparToolshed\_filter\_Adjmat.R DaparToolshed\_QFeatures\_Addons.R  
 DaparToolshed\_filter\_qMetadata.R  
 DaparToolshed-filterFeaturesOneSE.R export\_qfeatures.R  
 import\_2\_qfeatures.R utils.R pepa.R global.R palette.R  
 shinyValue.R get\_pep\_prot\_cc.R aggregation.R heatmap.R  
 initialization.R

**RoxygenNote** 7.3.3

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**VignetteBuilder** knitr

**Config/pak/sysreqs** cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev zlib1g-dev

**Repository** <https://bioc.r-universe.dev>

**Date/Publication** 2026-05-07 08:57:44 UTC

**RemoteUrl** <https://github.com/bioc/DaparToolshed>

**RemoteRef** HEAD

**RemoteSha** dd69c315c69f443771cf70f41b7d57e3f11f259a

## Contents

Add_Item_to_Dataset . . . . .	5
Add2History . . . . .	5
adjacency-matrix-filter . . . . .	6
applyAnovasOnProteins . . . . .	8
BuildAdjacencyMatrix . . . . .	9
buildGraph . . . . .	9
BuildMetacell . . . . .	10
checkConditions . . . . .	11
checkDesign . . . . .	12
Children . . . . .	12
classic1wayAnova . . . . .	13
CleanRowData . . . . .	14
compareNormalizationD_HC . . . . .	15
compute_t_tests . . . . .	16
ConvertListToHtml . . . . .	17
createQFeatures . . . . .	18
DaparToolshed-aggregate . . . . .	19
DaparToolshed-aggregateRedistribution . . . . .	28

diffAnaComputeAdjustedPValues . . . . .	30
diffAnaComputeFDR . . . . .	31
diffAnaVolcanoplot_rCharts . . . . .	31
displayCCvisNet . . . . .	33
ExtendPalette . . . . .	34
formatHSDResults . . . . .	35
formatLimmaResult . . . . .	35
formatPHResults . . . . .	36
formatPHTRResults . . . . .	37
fudge2LRT . . . . .	38
GetColorsForConditions . . . . .	39
getDesignLevel . . . . .	40
GetHistory . . . . .	41
GetIndices_BasedOnConditions . . . . .	41
GetIndices_FunFiltering . . . . .	42
GetIndices_WholeLine . . . . .	44
GetIndices_WholeMatrix . . . . .	44
getListNbValuesInLines . . . . .	45
GetNbTags . . . . .	46
getNumberOfEmptyLines . . . . .	46
getPepProtCC . . . . .	47
GetSoftAvailables . . . . .	48
globalAdjPval . . . . .	48
hc_logFC_DensityPlot . . . . .	49
heatmapForMissingValues . . . . .	50
histPValue_HC . . . . .	51
imputePA2 . . . . .	52
InitializeHistory . . . . .	53
isDifferential . . . . .	53
isOfType . . . . .	54
isSubset . . . . .	55
Keep_Items_from_Dataset . . . . .	56
LH0 . . . . .	56
LH0_lm . . . . .	57
LH1 . . . . .	58
LH1_lm . . . . .	58
limmaCompleteTest . . . . .	59
makeContrast . . . . .	60
makeDesign . . . . .	61
makeDesign1 . . . . .	61
makeDesign2 . . . . .	62
makeDesign3 . . . . .	63
matchMetacell . . . . .	63
metacell-plots . . . . .	64
Metacell_DIA_NN . . . . .	67
Metacell_maxquant . . . . .	68
Metacell_proline . . . . .	69
MetacellFilteringScope . . . . .	70

mv_imputation_protein . . . . .	70
NAIsZero . . . . .	72
nEmptyLines . . . . .	73
nonzero . . . . .	73
normalization_methods . . . . .	74
normalizeFunction . . . . .	76
output_2_Excel . . . . .	77
OWAnova . . . . .	79
paramshistory . . . . .	80
Parent . . . . .	84
pepaTest . . . . .	85
Pipelines . . . . .	86
pkgsRequire . . . . .	86
plotCompareAssays . . . . .	87
plotJitter . . . . .	88
plotJitter_rCharts . . . . .	89
postHocTest . . . . .	89
pushpvalue . . . . .	91
q_metacell . . . . .	92
QFeatures-excel . . . . .	95
QFeatures-filtering-oneSE . . . . .	97
QFeatures-utils . . . . .	99
ReplaceSpecialChars . . . . .	100
samLRT . . . . .	101
separateAdjPval . . . . .	102
SetHistory . . . . .	103
splitAdjacencyMat . . . . .	103
subR25pept . . . . .	104
subR25prot . . . . .	105
SymFilteringOperators . . . . .	106
testAnovaModels . . . . .	108
testDesign . . . . .	109
thresholdpval4fdr . . . . .	109
translatedRandomBeta . . . . .	110
wrapperCalibrationPlot . . . . .	111
wrapperClassic1wayAnova . . . . .	112
wrapperDaparImputeMI . . . . .	113
wrapperImputeMLE . . . . .	115
wrapperImputePA2 . . . . .	116
wrapperPirat . . . . .	117

---

Add\_Item\_to\_Dataset     *Adds an instance of SummarizedExperiment to a QFeatures object*

---

**Description**

Adds one or more items to the dataset. This function is specific of the type of dataset.

**Usage**

```
Add_Item_to_Dataset(dataset, name)
```

**Arguments**

dataset	An instance of SummarizedExperiment class
name	A character() for the new assay

**Value**

The dataset minus some items

---

Add2History     *Add row to history*

---

**Description**

This function adds a row to the history.

**Usage**

```
Add2History(history, process, step.name, param.name, value)
```

**Arguments**

history	A data.frame corresponding to the current history.
process	A character(1) corresponding to the process name.
step.name	A character(1) corresponding to the step name.
param.name	A character(1) corresponding to the parameter name.
value	The value of the corresponding parameter.

**Value**

A data.frame with one added row

**Examples**

```
history <- InitializeHistory()
Add2History(history, "Process1", "Step1", "Parameter1", "Value1")
```

---

`adjacency-matrix-filter`*Filter a peptide assay on the basis of its adjacency matrix.*

---

### Description

These functions filters (delete) peptides of an assay, applying a function on peptides and proteins. They can be used alone but the usual usage is to create an instance of a class `FunctionFilter` and to pass it to the function `filterFeaturesOneSE` in order to create a new assay, embedded into the `QFeatures` object.

### Usage

```
AdjMatFilters()
allPeptides(object, ...)
specPeptides(object, ...)
subAdjMat_specificPeptides(X)
sharedPeptides(object, ...)
subAdjMat_sharedPeptides(X)
topnFunctions()
topnPeptides(object, fun, top)
subAdjMat_topnPeptides(X, qData, fun, top)
```

### Arguments

<code>object</code>	An object of class <code>SummarizedExperiment</code>
<code>...</code>	Additional arguments
<code>X</code>	A <code>Matrix()</code>
<code>fun</code>	A <code>list()</code> of additional parameters
<code>top</code>	A <code>integer(1)</code> which is the number of entities to use
<code>qData</code>	A <code>data.frame()</code> for the quantitative data

### Details

This function builds an intermediate matrix with scores for each peptide based on 'fun' parameter. Once this matrix is built, one select the 'n' peptides which have the higher score

The list of filter functions is given by `adjMatFilters()`:

- `specPeptides()`: returns a new assay of class `SummazedExperiment` with only specific peptides;
- `sharedpeptides()`: returns a new assay of class `SummazedExperiment` with only shared peptides;
- `opnPeptides()`: returns a new assay of class `SummazedExperiment` with only the 'n' peptides which best satisfies the condition. The condition is represented by functions which calculates a score for each peptide among all samples. The list of these functions is given by `topnFunctions()`;
- `rowMedians()`: returns the median of the entities in each selected row;
- `rowMeans()`: returns the median of the entities in each selected row;
- `rowSums()`: returns the sum of the entities in each selected row;

**Value**

NA

**Author(s)**

Samuel Wieczorek

**See Also**The `QFeatures-filtering-oneSE` man page for the class `FunctionFilter`.**Examples**

```
library(Matrix)
library(QFeatures)
#-----
# This function will keep only specific peptides
#-----

f1 <- FunctionFilter("specPeptides", list())

#-----
# This function will keep only shared peptides
#-----

f2 <- FunctionFilter("sharedPeptides", list())

#-----
# This function will keep only the 'n' best peptides
# w.r.t the quantitative sum of each peptides among
# all samples
#-----

f3 <- FunctionFilter("topnPeptides", fun = "rowSums", top = 2)

#-----
# IF several filters must be used, store them in a list
```

```
#-----  
  
data(subR25pept)  
lst.filters <- list()  
lst.filters <- append(lst.filters, f1)  
lst.filters <- append(lst.filters, f3)  
  
subR25prot <- filterFeaturesOneSE(  
  object = subR25pept,  
  i = 1,  
  name = "filtered",  
  filters = lst.filters  
)
```

---

applyAnovasOnProteins *iteratively applies OWAova() on the features of an MSnSet object*

---

### Description

iteratively applies OWAova() on the features of an MSnSet object

### Usage

```
applyAnovasOnProteins(obj, i)
```

### Arguments

obj            a QFeatures object  
i              An integer which is the index of the assay in the QFeatures object '

### Value

a list of linear models

### Author(s)

Thomas Burger

### Examples

```
data(subR25prot)  
applyAnovasOnProteins(subR25prot[seq_len(5),], 1)
```

---

BuildAdjacencyMatrix    *Function matrix of appartenance group*

---

**Description**

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

**Usage**

```
BuildAdjacencyMatrix(obj.pep)
```

**Arguments**

obj.pep            An object (peptides) of class SummarizedExperiment.

**Value**

A binary matrix

**Author(s)**

Florence Combes, Samuel Wiczorek, Alexia Dorffer

**Examples**

```
data(subR25pept)
BuildAdjacencyMatrix(subR25pept[[1]])
```

---

buildGraph            *Display a CC*

---

**Description**

Display a CC

**Usage**

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

**Arguments**

The.CC            A cc (a list)  
X                 An instance of the class Matrix

**Value**

A plot

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
data(subR25pept)
X <- QFeatures::adjacencyMatrix(subR25pept[[1]])
ll <- getPepProtCC(X)
g <- buildGraph(ll[[1]], X)
```

---

BuildMetacell

*Metacell function*

---

**Description**

Create metacell

**Usage**

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

**Arguments**

from	A string designing the software used, either "maxquant", "proline" or "DIA-NN"
level	A string designing the type of entity/pipeline. Available values are: peptide, protein
qdata	A matrix of quantitative data
conds	A 1-col dataframe with the condition associated to each sample.
df	A data.frame which contains the type of identification of the entities. It must have the same dimensions as qData.

**Value**

NA

**Author(s)**

Samuel Wiczorek

**Examples**

```
data(subR25pept)
conds <- design_qf(subR25pept)$Condition
qdata <- SummarizedExperiment::assay(subR25pept[[1]])
metacell1 <- BuildMetacell('maxquant', 'peptide', qdata, conds)
metacell2 <- BuildMetacell('proline', 'peptide', qdata, conds)
```

---

checkConditions	<i>Check if the design is valid</i>
-----------------	-------------------------------------

---

**Description**

Check if the design is valid

**Usage**

```
checkConditions(conds)
```

**Arguments**

conds            A vector containing the conditions.

**Value**

A list including : "valid" : Whether the conditions are valid or not. "warn" : A message describing the issue if the conditions are not valid.

**Author(s)**

Samuel Wiczorek

**Examples**

```
data(subR25pept)
checkConditions(design_qf(subR25pept)$Condition)
```

---

checkDesign	<i>Check if the design is valid</i>
-------------	-------------------------------------

---

**Description**

Check if the design is valid

**Usage**

```
checkDesign(sTab)
```

**Arguments**

sTab	The data.frame which correspond to the pData() function of package MSnbase.
------	---

**Value**

A boolean

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
data(subR25pept)
checkDesign(SummarizedExperiment::colData(subR25pept)[, -1])
```

---

Children	<i>Names of all children of a node</i>
----------	--

---

**Description**

Names of all children of a node

**Usage**

```
Children(level, parent = NULL)
```

**Arguments**

level	A string designing the type of entity/pipeline. Available values are: peptide, protein
parent	A vector of character()

**Value**

A vector

**Examples**

```
Children('protein', 'Missing')
Children('protein', 'Missing POV')
Children('protein', c('Missing POV', 'Missing MEC'))
Children('protein', c('Missing', 'Missing POV', 'Missing MEC'))
```

---

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**

```
library(SummarizedExperiment)
data(subR25prot)
obj <- subR25prot
filter <- FunctionFilter('qMetacellOnConditions',
  cmd = 'delete',
  mode = 'AtLeastOneCond',
  pattern = c("Missing POV", "Missing MEC"),
  conds = design_qf(obj)$Condition,
  percent = TRUE,
  th = 0.8,
  operator = '>')
obj <- filterFeaturesOneSE(obj, name = "Filtered", filters = list(filter))
```

```
qdata <- SummarizedExperiment::assay(obj[[2]])
conds <- design_qf(obj)$Condition
anova_tests <- apply(qdata, 1, classic1wayAnova, conditions = as.factor(conds))
anova_tests <- t(anova_tests)
```

---

CleanRowData

*Clean row data*

---

### Description

Clean row data

### Usage

```
CleanRowData(obj, i)
```

### Arguments

`obj` An instance of the class QFeatures  
`i` An integer which is the index of the assay in the QFeatures object

### Value

An instance of the SummarizedExperiment structure

### Author(s)

Samuel Wieczorek

### Examples

```
NULL
```

---

`compareNormalizationD_HC`

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

---

### Description

Plot to compare the quantitative proteomics data before and after normalization using the package `plotly`

### Usage

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

### Arguments

<code>qDataBefore</code>	A dataframe that contains quantitative data before normalization.
<code>qDataAfter</code>	A dataframe that contains quantitative data after normalization.
<code>keyId</code>	A character()
<code>conds</code>	A vector of the conditions (one condition per sample).
<code>pal</code>	A vector() of HEX color codes
<code>subset.view</code>	A vector() of integers
<code>n</code>	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
<code>type</code>	scatter or line

### Value

A plot

### Author(s)

Samuel Wieczorek

**Examples**

```

library(SummarizedExperiment)
data(subR25prot)
qDataBefore <- SummarizedExperiment::assay(subR25prot[[1]])
conds <- design_qf(subR25prot)$Condition
id <- rowData(subR25prot[[1]])[, idcol(subR25prot[[1]])]
# pal <- ExtendPalette(2)
qDataAfter <- LOESS(qDataBefore, conds, type = "overall")

n <- 1
compareNormalizationD_HC(
  qDataBefore = qDataBefore,
  qDataAfter = qDataAfter,
  keyId = id,
  pal = NULL,
  n = n,
  subset.view = seq_len(n),
  conds = conds)

```

---

compute\_t\_tests

*Compute a t-test*


---

**Description**

Compute a t-test

**Usage**

```

compute_t_tests(
  obj,
  i = 1,
  contrast = c("OnevsOne", "OnevsAll"),
  type = c("Student", "Welch")
)

```

**Arguments**

obj	A matrix of quantitative data, without any missing values.
i	An integer which is the index of the assay in the QFeatures object
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	Either "Student" or "Welch"

**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 Wieczorek

**Examples**

```
library(SummarizedExperiment)
data(subR25prot)
obj <- subR25prot
filter <- FunctionFilter('qMetacellOnConditions',
  cmd = 'delete',
  mode = 'AtLeastOneCond',
  pattern = c("Missing POV", "Missing MEC"),
  conds = design_qf(obj)$Condition,
  percent = TRUE,
  th = 0.8,
  operator = '>')
obj <- filterFeaturesOneSE(obj, name = "Filtered", filters = list(filter))
ttest <- compute_t_tests(obj, 2)
```

---

ConvertListToHtml

*Convert a list to unnumbered HTML list*

---

**Description**

Convert a list to unnumbered HTML list

**Usage**

```
ConvertListToHtml(l1)
```

**Arguments**

l1                    A list() of character()

**Value**

HTML

**Examples**

```
ConvertListToHtml(list('foo1', 'foo2', 'foo3'))
```

---

createQFeatures      *Creates an object of class QFeatures from text file.*

---

### Description

Creates an object of class QFeatures from a single tabulated-like file for quantitative and meta-data and a dataframe for the samples description.

### Usage

```
createQFeatures(
  data = NULL,
  file = "myDataset",
  sample,
  indQData,
  keyId = "AutoID",
  indexForMetacell = NULL,
  logData = FALSE,
  force.na = TRUE,
  typeDataset,
  parentProtId = NULL,
  analysis = "foo",
  description = NULL,
  processes = NULL,
  typePipeline = NULL,
  name.pipeline = NULL,
  software = NULL,
  name = "original"
)
```

### Arguments

data	The name of a tab-separated file that contains the data.
file	A character().
sample	A dataframe describing the samples (in lines).
indQData	A vector of string where each element is the name of a column in designTable that have to be integrated in the rowData() table of the QFeatures object.
keyId	A character(1) or numeric(1) which is the indice of the column containing the ID of entities (peptides or proteins)
indexForMetacell	They must be in the same order as the samples in the experimental design
logData	A boolean that indicates whether the data should be logged or not.
force.na	A boolean that indicates if the '0' and 'NaN' values of quantitative values must be replaced by 'NA' (Default is FALSE)
typeDataset	A string that indicates whether the dataset is about

parentProtId	A character(1) For peptide entities, a string which is the name of a column in rowData. It contains the id of parent proteins and is used to generate adjacency matrix and process to aggregation.
analysis	A character() which is the name of the MS study.
description	A text which describes the study.
processes	A vector of A character() which contains the name of processes which has already been run on the data. Default is 'original'.
typePipeline	A character(1) The type of pipeline used with this dataset. The list of predefined pipelines in DaparToolshed can be obtained with the function pipelines(). Default value is NULL
name.pipeline	A string
software	A character()
name	A character() which is the name of the assay in the QFeatures object. Default is 'original'

**Value**

An instance of class QFeatures.

**Author(s)**

Samuel Wiczorek, Manon Gaudin

**Examples**

NULL

---

DaparToolshed-aggregate

*Aggregate an assay's quantitative features*

---

**Description**

This function aggregates the quantitative features of an assay, applying a summarization function (fun) to sets of features. The fcol variable name points to a rowData column that defines how to group the features during aggregate. This variable has to be an adjacency matrix. This function uses QFeatures::aggregateFeatures() to aggregate quantitative data.

The list of aggregation methods can be obtained with the function aggregateMethods(). This function compiles both methods from the packages DaparToolshed and QFeatures.

Aggregate the quantitative metadata tag.

This function aggregate both quantitative and rowdata from the last assay contained in a QFeatures. Note that the function assumes that the intensities in the QFeatures are already log-transformed.

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

Aggregation of rowData of a QFeatures assay.

Aggregate the metadata

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

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

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

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

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

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

This function aggregate quantitative data using a method of redistribution of shared peptides. Intensity of shared peptides are redistributed proportionally to each protein. Note that the function assumes that the intensities are not log-transformed.

Aggregation using sum method.

Aggregation using mean method.

Aggregation using median method.

Aggregation using medianPolish method. Note that this method is parallelized to be more efficient.

Aggregation using robustSummary method.

## Usage

```
aggregateFeatures4Prostar(object, ...)
```

```
## S4 method for signature 'QFeatures'
```

```
aggregateFeatures4Prostar(
  object,
  i,
  fcol,
  name = "newAssay",
  fun = MsCoreUtils::robustSummary,
  shared = TRUE,
  n = NULL,
  ...
)
```

```
## S4 method for signature 'SummarizedExperiment'
```

```
aggregateFeatures4Prostar(
  object,
  fcol,
  fun = MsCoreUtils::robustSummary,
  conds,
  shared = TRUE,
  n = NULL,
)
```

```
    ...
)

aggQmetacell(qMeta, X, level, conds)

aggregateMethods()

RunAggregation(
  qf,
  includeSharedPeptides = "Yes_As_Specific",
  operator = "Mean",
  considerPeptides = "allPeptides",
  adjMatrix = "adjacencyMatrix",
  ponderation = "Global",
  n = NULL,
  aggregated_col = NULL,
  max_iter = 500
)

BuildColumnToProteinDataset(peptideData, matAdj, columnName, proteinNames)

Add_Aggregated_rowData(obj, col, i.agg)

metacell_agg(agggregatedSE, originalSE, adj_mat, conds, protname_order)

select_topn(pepData, X, n = 10, funpept = "Mean")

getProteinsStats(X)

CountPep(X)

GetNbPeptidesUsed(pepData, X)

GetDetailedNbPeptidesUsed(pepData, X)

GetDetailedNbPeptides(X)

GraphPepProt(mat)

ExtractUniquePeptides(X)

innerAggregateIter(
  pepData,
  X,
  init.method = "Mean",
  method = "Mean",
  n = NULL,
  uniqueiter = FALSE,
```

```

    topn_fun = "Mean",
    max_iter = 500
)

innerSum(pepData, X)

innerMean(pepData, X)

innerMedian(pepData, X)

innerMedianpolish(pepData, X)

innerRobustsummary(pepData, X)

```

### Arguments

object	An instance of class QFeatures or SummarizedExperiment
...	Additional parameters passed the fun.
i	The index or name of the assay which features will be aggregated the create the new assay.
fcol	A character(1) naming a rowdata variable (of assay i in case of a QFeatures) defining how to aggregate the features of the assay. This variable is a (possibly sparse) matrix. See below for details.
name	A character(1) naming the new assay. Default is newAssay. Note that the function will fail if there's already an assay with name.
fun	A function used for quantitative feature aggregation. See details for examples.
shared	A boolean indication if shared peptides should be considered. If TRUE, shared peptides
n	A numeric(1) specifying the number of peptides to use for each protein. If NULL, all peptides are considered.
conds	A character() vector which is the names of conditions for each sample in the dataset.
qMeta	A matrix with quantitative metadata tag.
X	A matrix acting as an adjacency matrix.
level	A character(1) which is the type of dataset
qf	An instance of class <a href="#">QFeatures::QFeatures</a> . The last assay contained in qf will be aggregated. Intensities are assumed to already be log-transformed.
includeSharedPeptides	How shared peptides are handled. Either Yes_As_Specific (default), Yes_Iterative_Redistribution, Yes_Simple_Redistribution or No. See below for details.
operator	A function used for quantitative feature aggregation. Available functions are Sum, Mean, Median, medianPolish or robustSummary. See below for details.
considerPeptides	A character(1) defining what peptide to consider. Available values are allPeptides (default) and topN.

adjMatrix	A character(1) naming a rowdata variable from the last assay of qf containing an adjacency matrix.
ponderation	A character(1) defining what to consider to create the coefficient for redistribution of shared peptides. Available values are Global (default), Condition or Sample.
aggregated_col	A character() of column names from rowdata to be aggregated.
max_iter	A numeric(1) setting the maximum number of iteration.
peptideData	A data.frame of meta data of peptides. It is the rowData of the SummarizedExperiment object.
matAdj	The adjacency matrix used to aggregate the peptides data.
columnName	The name(s) of the column in SummarizedExperiment::rowData(peptides_MSnsset) 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)
obj	An instance of class <a href="#">QFeatures::QFeatures</a> .
col	A character() of column names from rowdata to be aggregated.
i.agg	A numeric(1) indicating the index of the assay to which add the aggregated rowData, using the previous assay's rowData.
aggregatedSE	An instance of class <a href="#">SummarizedExperiment::SummarizedExperiment</a> containing the aggregated data.
originalSE	An instance of class <a href="#">SummarizedExperiment::SummarizedExperiment</a> containing the non-aggregated data.
adj_mat	An adjacency matrix.
protname_order	A character() vector with the protein name in order.
pepData	A matrix containing the peptide intensities. Note that the function assume that data is already log-transformed.
funpept	A function used for determining a peptide's value. Available functions are Sum, Mean or Median.
mat	An adjacency matrix.
init.method	A function used for initializing the aggregation. Available functions are Sum, Mean, Median, medianPolish or robustSummary. See below for details.
method	A function used for the aggregation. Available functions are Sum, Mean, Median, medianPolish or robustSummary. See below for details.
uniqueiter	A bole
topn_fun	A function used to determine how to choose the top n peptides. Available functions are Sum, Mean or Median. See below for details.

## Details

This function uses [QFeatures::aggregateFeatures\(\)](#) to aggregate quantitative data.

Aggregation of quantitative data is performed using [aggregateFeatures](#), or [innerAggregateIter](#) if [Yes\\_Iterative\\_Redistribution](#) or [Yes\\_Simple\\_Redistribution](#) is selected.

The handling of shared peptide is as follow :

- `Yes_As_Specific` : Shared peptides are used multiple times. Each peptide is duplicated as many times as the number of proteins in which they are present, and thus are considered as if they are specific to each protein.
- `Yes_Simple_Redistribution` : Intensity of shared peptides are redistributed proportionally to each protein. See `innerAggregateIter` for more information.
- `Yes_Iterative_Redistribution` : Intensity of shared peptides are redistributed proportionally to each protein. See `innerAggregateIter` for more information.
- `No` : No shared peptides are used. If a peptide contained only shared peptides, its intensity is set as 0 for every sample.

Available functions are :

- `Sum` : `base::colSums()` or `base::rowSums()` if `Yes_Iterative_Redistribution` or `Yes_Simple_Redistribution`.
- `Mean` : `base::colMeans()` or `base::rowMeans()` if `Yes_Iterative_Redistribution` or `Yes_Simple_Redistribution`.
- `Median` : `matrixStats::mcolMedians()` or `matrixStats::rowMedians()` if `Yes_Iterative_Redistribution` or `Yes_Simple_Redistribution`.
- `medianPolish` : `MsCoreUtils::medianPolish()`.
- `robustSummary` : `MsCoreUtils::robustSummary()`.

Available functions are :

- `Sum` : `base::rowSums()`
- `Mean` : `base::rowMeans()`
- `Median` : `matrixStats::rowMedians()`
- `medianPolish` : `MsCoreUtils::medianPolish()`, not available for `topn_fun`. Note that this method takes significantly more time than the others, and is parallelized to be more efficient.
- `robustSummary` : `MsCoreUtils::robustSummary()`, not available for `topn_fun`. Note that this method takes significantly more time than the others, and is parallelized to be more efficient.

## Value

A `QFeatures` object with an additional assay or a `SummarizedExperiment` object (or subclass thereof).

An instance of the `QFeatures` class

NA

A `QFeatures` with an aggregated assay added.

A vector

An instance of `QFeatures` class with aggregated `rowData` in specified assay.

A `SummarizedExperiment` containing the aggregated data.

An adjacency matrix with only the top n peptides selected.

A list

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

A data.frame  
 A list of two items  
 A data.frame  
 A histogram  
 A matrix containing the aggregated values.  
 A matrix containing the aggregated values.  
 A matrix containing the aggregated values.  
 A matrix containing the aggregated values.  
 A matrix containing the aggregated values.  
 A matrix containing the aggregated values.

### Quantitative metadata aggregation

The function to aggregate the quantitative metadata is `aggQmetadat()`.

### Author(s)

Samuel Wieczorek, Manon Gaudin  
 Samuel Wieczorek  
 Manon Gaudin  
 Alexia Dorffer  
 Alexia Dorffer, Samuel Wieczorek

### See Also

The *QFeatures* vignette provides an extended example and the *Aggregation* vignette, for a complete quantitative proteomics data processing pipeline.

### Examples

```
NULL
data(subR25pept)
qMeta <- qMetacell(subR25pept, 1)
X <- QFeatures::adjacencyMatrix(subR25pept[[1]])
level <- typeDataset(subR25pept[[1]])
conds <- SummarizedExperiment::colData(subR25pept)$Condition
aggQmeta <- aggQmetacell(qMeta, X, level, conds)

data(subR25pept)

# Remove empty lines
filter_emptyline <- FunctionFilter("qMetacellWholeLine",
  cmd = 'delete', pattern = 'Missing MEC')
subR25pept <- filterFeaturesOneSE(object = subR25pept,
  i = length(subR25pept), name = "Filtered",
  filters = list(filter_emptyline))
# Remove proteins with no peptide associated in adjacency matrix
```

```

indx <- which(Matrix::colSums(
SummarizedExperiment::rowData(
subR25pept[[length(subR25pept)]])$adjacencyMatrix) != 0)
SummarizedExperiment::rowData(
subR25pept[[length(subR25pept)]])$adjacencyMatrix <-
SummarizedExperiment::rowData(
subR25pept[[length(subR25pept)]])$adjacencyMatrix[, indx]

obj.agg <- RunAggregation(subR25pept, "Yes_As_Specific", "Sum", "allPeptides",
aggregated_col = c("Sequence", "Mass"))
obj.agg <- RunAggregation(subR25pept, "Yes_As_Specific", "Mean", "allPeptides",
aggregated_col = c("Sequence", "Mass"))
obj.agg <- RunAggregation(subR25pept, "Yes_As_Specific", "Sum", "topN", n = 4,
aggregated_col = c("Sequence", "Mass"))
obj.agg <- RunAggregation(subR25pept, "Yes_As_Specific", "Mean", "topN", n = 4,
aggregated_col = c("Sequence", "Mass"))

obj.agg <- RunAggregation(subR25pept, "No", "Sum", "allPeptides")
obj.agg <- RunAggregation(subR25pept, "No", "Sum", "topN", n = 4)

obj.agg <- RunAggregation(subR25pept, "Yes_Simple_Redistribution", "Sum", "allPeptides",
aggregated_col = c("Sequence", "Mass"))
obj.agg <- RunAggregation(subR25pept, "Yes_Iterative_Redistribution", "Sum", "topN", n = 4,
aggregated_col = c("Sequence", "Mass"))

library(QFeatures)

data(subR25pept)
protID <- parentProtId(subR25pept[[1]])
X <- QFeatures::adjacencyMatrix(subR25pept[[1]])

X.split <- DaparToolshed::splitAdjacencyMat(X)
X.shared <- X.split$Xshared
X.unique <- X.split$Xspec
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
X.topn <- select_topn(SummarizedExperiment::assay(subR25pept[[1]]), X, n = 3)

data(subR25pept)
obj.last <- subR25pept[[1]]
X <- BuildAdjacencyMatrix(subR25pept[[1]])
getProteinsStats(X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
CountPep(X)

library(QFeatures)
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
GetNbPeptidesUsed(SummarizedExperiment::assay(subR25pept), X)

```

```
library(SummarizedExperiment)
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
ll.n <- GetDetailedNbPeptidesUsed(SummarizedExperiment::assay(subR25pept), X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
n <- GetDetailedNbPeptides(X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
GraphPepProt(X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
ExtractUniquePeptides(X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
qdata.agg <- innerAggregateIter(SummarizedExperiment::assay(subR25pept[[1]]), X)

library(QFeatures)
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
i.sum <- innerSum(SummarizedExperiment::assay(subR25pept[[1]]), X)

library(QFeatures)
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
i.mean <- innerMean(SummarizedExperiment::assay(subR25pept), X)

library(QFeatures)
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
i.mean <- innerMedian(SummarizedExperiment::assay(subR25pept[[1]]), X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
i.mean <- innerMedianpolish(SummarizedExperiment::assay(subR25pept[[1]]), X)

data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
i.mean <- innerRobustsummary(SummarizedExperiment::assay(subR25pept[[1]]), X)
```

---

DaparToolshed-aggregateRedistribution

*Aggregate an assay's quantitative features with shared peptide redistribution*

---

## Description

This function aggregates the quantitative features of an assay, applying a summarization function (fun) to sets of features. The fcol variable name points to a rowData column that defines how to group the features during aggregate. This variable has to be an adjacency matrix. This function uses DaparToolshed::innerAggregateIter() to aggregate quantitative data.

The list of aggregation methods can be obtained with the function aggregateMethods(). This function compiles both methods from the packages DaparToolshed and QFeatures.

## Usage

```
aggregateRedistribution(object, ...)

## S4 method for signature 'QFeatures'
aggregateRedistribution(
  object,
  i,
  name = "newAssay",
  fcol,
  init.method = "Mean",
  method = "Mean",
  ponderation = "Global",
  n = NULL,
  uniqueiter = FALSE,
  max_iter = 500
)

## S4 method for signature 'SummarizedExperiment'
aggregateRedistribution(
  object,
  fcol,
  init.method = "Mean",
  method = "Mean",
  ponderation = "Global",
  n = NULL,
  uniqueiter = FALSE,
  conds,
  max_iter = 500
)
```

**Arguments**

object	An instance of class QFeatures or SummarizedExperiment
...	Additional parameters.
i	The index or name of the assay which features will be aggregated the create the new assay.
name	A character(1) naming the new assay. Default is newAssay. Note that the function will fail if there's already an assay with name.
fcol	A character(1) naming a rowdata variable (of assay i in case of a QFeatures) defining how to aggregate the features of the assay. This variable is a (possibly sparse) matrix. See below for details.
init.method	A function used for initializing the aggregation. Available functions are Sum, Mean, Median, medianPolish or robustSummary. See DaparToolshed::innerAggregateIter() for details.
method	A function used for the aggregation. Available functions are Sum, Mean, Median or medianPolish. See DaparToolshed::innerAggregateIter() for details.
ponderation	A character(1) defining what to consider to create the coefficient for redistribution of shared peptides. Available values are Global (default), Condition or Sample.
n	A numeric(1) specifying the number of peptides to use for each protein. If NULL, all peptides are considered.
uniqueiter	A boolean indication if there should be only 1 iteration or not.
max_iter	A numeric(1) setting the maximum number of iteration.
conds	A character() vector which is the names of conditions.

**Details**

This function uses DaparToolshed::innerAggregateIter() to aggregate quantitative data.

**Value**

A QFeatures object with an additional assay or a SummarizedExperiment object (or subclass thereof).

**Iterative aggregation function**

NA

**Quantitative metadata aggregation**

The function to aggregate the quantitative metadata is aggQmetadat()

**See Also**

The *QFeatures* vignette provides an extended example and the *Aggregation* vignette, for a complete quantitative proteomics data processing pipeline.

**Examples**

NULL

---

diffAnaComputeAdjustedPValues

*Computes the adjusted p-values*

---

**Description**

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

**Usage**

```
diffAnaComputeAdjustedPValues(pval, pi0Method = 1)
```

**Arguments**

<code>pval</code>	The result (p-values) of the differential analysis
<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 adjusted p-values

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25prot)
obj <- subR25prot
# Simulate imputation
obj <- NAIsZero(obj, 1)
allComp <- limmaCompleteTest(
  SummarizedExperiment::assay(obj[[length(obj)]]), design_qf(obj),
  comp.type="0nevsOne")
diffAnaComputeAdjustedPValues(pval = allComp$P_Value[, 1])
```

---

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

---

**Description**

This function returns the FDR corresponding to the p-values of the differential analysis.

**Usage**

```
diffAnaComputeFDR(adj.pvals)
```

**Arguments**

adj.pvals      The adjusted p-values of the differential analysis

**Value**

The computed FDR value (floating number)

**Author(s)**

Samuel Wiczorek

**Examples**

```
NULL
```

---

diffAnaVolcanoplot_rCharts	<i>Volcanoplot of the differential analysis</i>
----------------------------	---

---

**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 th\_pval and the th\_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 plotly, a customizable tooltip appears when the user put the mouse's pointer over a point of the scatter plot.

**Usage**

```
diffAnaVolcanoplot_rCharts(
  df,
  th_pval = 1e-60,
  th_logfc = 0,
  conditions = 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 have 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.
th_pval	A floating number which represents the p-value that separates differential and non-differential data.
th_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.
pal	A list containing 2 color to use for the plot

**Value**

An interactive volcano plot

**Author(s)**

Samuel Wiczorek

**Examples**

```
data(subR25prot)
obj <- subR25prot
# Simulate imputation
obj <- NAIsZero(obj, 1)
allComp <- limmaCompleteTest(
  SummarizedExperiment::assay(obj[[length(obj)]]),
  design_qf(obj),
  comp.type="OnevsOne")
df <- data.frame(
  x = allComp$logFC[[1]],
  y = -log10(allComp$P_Value[[1]]),
  index = as.character(rownames(obj[[1]]))
)
tooltipSlot <- c("Fasta_headers", "Sequence_length")
```

```
df <- cbind(df, SummarizedExperiment::rowData(obj[[1]])[, tooltipSlot])
colnames(df) <- gsub(".", "_", colnames(df), fixed = TRUE)
if (ncol(df) > 3) {
  colnames(df)[4:ncol(df)] <- paste0("tooltip_",
                                     colnames(df)[4:ncol(df)])
}
cond <- c("25fmol", "10fmol")
diffAnaVolcanoplot_rCharts(
  df,
  th_pval = 2.5,
  th_logfc = 1,
  conditions = cond
)
```

---

displayCCvisNet

*Display a CC*

---

### Description

Display a CC

### Usage

```
displayCCvisNet(g)
```

### Arguments

g                    A cc (a list)

### Value

A plot

### Author(s)

Thomas Burger, Samuel Wiczorek

### Examples

```
data(subR25pept)
X <- QFeatures::adjacencyMatrix(subR25pept[[1]])
ll <- getPepProtCC(X)
g <- buildGraph(ll[[1]], X)
displayCCvisNet(g)
```

---

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 = 0, 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 seq_len(nPalette)) {
  pal <- ExtendPalette(n = i, base = "Dark2")
  barplot(seq_len(length(pal)), col = pal)
  print(pal)
}
```

---

formatHSDResults	<i>Extracts the FC and the FWER adjusted p-values and format them in a data.frame</i>
------------------	---

---

**Description**

Extracts the FC and the FWER adjusted p-values and format them in a data.frame

**Usage**

```
formatHSDResults(post_hoc_models_summaries)
```

**Arguments**

post\_hoc\_models\_summaries  
 resulting from applying lapply(summary((.)) to a multicomf function.

**Value**

A data.frame()

**Author(s)**

Thomas Burger

**Examples**

```
NULL
```

---

formatLimmaResult	<i>Format Limma results</i>
-------------------	-----------------------------

---

**Description**

Format Limma results

**Usage**

```
formatLimmaResult(fit, conds, contrast, design.level)
```

**Arguments**

fit	An object of class MArrayLM.
conds	A character() vector which is the names of conditions.
contrast	An integer(1) defining the type of contrast (1 for OnevsOne, or 2 for OnvsAll).
design.level	An integer(1) specifying the design level.

**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 Wieczorek

**Examples**

```
library(SummarizedExperiment)
data(subR25prot)
level <- 'protein'
metacell.mask <- matchMetacell(qMetacell(subR25prot[[1]]),
c("Missing POV", "Missing MEC"), level)
# Simulate imputation
assay(subR25prot[[1]][which(is.na(assay(subR25prot[[1]])))] <- 0
qData <- as.matrix(SummarizedExperiment::assay(subR25prot[[1]])
sTab <- SummarizedExperiment::colData(subR25prot)
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)**

Helene Borges

**Examples**

```

library(SummarizedExperiment)
data(subR25prot)
obj <- subR25prot
filter <- FunctionFilter('qMetacellOnConditions',
  cmd = 'delete',
  mode = 'AtLeastOneCond',
  pattern = c("Missing POV", "Missing MEC"),
  conds = design_qf(obj)$Condition,
  percent = TRUE,
  th = 0.8,
  operator = '>')
obj <- filterFeaturesOneSE(obj, name = "Filtered", filters = list(filter))
qdata <- SummarizedExperiment::assay(obj[[2]])
conds <- design_qf(obj)$Condition
anova_tests <- apply(qdata, 1, classic1wayAnova, conditions = as.factor(conds))
anova_tests <- t(anova_tests)

names(anova_tests) <- rownames(qdata)
tms <- lapply(
  anova_tests,
  function(x) {
    summary(multcomp::glht(x,
      linfct = multcomp::mcp(conditions = "Tukey")
    ),
    test = multcomp::adjusted("none")
  )
}
)
res <- formatPHResults(tms)

```

---

formatPHResults	<i>Extracts the FC and the raw p-values and format them in a data.frame</i>
-----------------	---

---

**Description**

Extracts the FC and the raw p-values and format them in a data.frame

**Usage**

```
formatPHResults(post_hoc_models_summaries)
```

**Arguments**

post\_hoc\_models\_summaries  
 resulting from applying lapply(summary(.)) to a multcomp function.

**Value**

A `data.frame()`

**Author(s)**

Thomas Burger

**Examples**

NULL

---

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.

<code>lmm.res.h1</code>	similar to <code>lmm.res.h0</code> , a vector of object containing the estimates (used to compute the statistic) under H1 for each protein.
<code>cc</code>	a list containing the indices of peptides and proteins belonging to each connected component.
<code>n</code>	the number of samples used in the test
<code>p</code>	the number of proteins in the experiment
<code>s</code>	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 <code>lmm.res.h1</code> . For other models (e.g. mixed models) it can be obtained from <code>samLRT</code> .
<code>alpha</code>	A vector of proportions used to build candidate values for the regularizer. We use quantiles of <code>s</code> with these proportions. Default to <code>seq(0, 1, 0.05)</code>
<code>include.zero</code>	logical value indicating if 0 should be included in the list of candidates. Default to <code>TRUE</code> .

**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

**Examples**

NULL

---

`GetColorsForConditions`

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

---

**Description**

Builds a complete color palette for the conditions given in argument

**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**

NA

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25pept)
GetColorsForConditions(design_qf(subR25pept)$Condition)
GetColorsForConditions(design_qf(subR25pept)$Condition, ExtendPalette(2))
```

---

<code>getDesignLevel</code>	<i>Get design level</i>
-----------------------------	-------------------------

---

**Description**

Get design level

**Usage**

```
getDesignLevel(sTab)
```

**Arguments**

`sTab` A `data.frame()` which contains the sample data

**Value**

An integer

**Examples**

```
data(subR25pept)
sTab <- SummarizedExperiment::colData(subR25pept)
getDesignLevel(sTab)
```

---

GetHistory	<i>Wrapper for the paramshistory function.</i>
------------	--

---

**Description**

Wrapper for the paramshistory function.

**Usage**

```
GetHistory(obj.se, history)
```

**Arguments**

obj.se	An instance of the class SummarizedExperiment
history	A data.frame()

**Value**

A data.frame()

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25prot)
GetHistory(subR25prot[[1]])
```

---

GetIndices_BasedOnConditions	<i>Search lines which respects request on one or more conditions.</i>
------------------------------	---

---

**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	A data.frame() of boolean values which indicates the presence (TRUE) or not (FALSE) of given tags in the quantitative cells metadata
type	Available values are: <ul style="list-style-type: none"> <li>• 'AllCond' (the query is valid in all the conditions),</li> <li>• 'AtLeastOneCond' (the query is valid in at least one condition).</li> </ul>
conds	A character() vector which is the names of conditions.
percent	A boolean to indicate whether the threshold represent an absolute value (percent = FALSE) or a percentage (percent=TRUE).
op	String for operator to use. List of operators is available with the function 'SymFilteringOperators()'.
th	The threshold to apply

**Value**

A vector of integer()

**Examples**

```
data(subR25pept)
level <- typeDataset(subR25pept[[1]])
pattern <- 'Missing'
metacell.mask <- matchMetacell(
  metadata=qMetacell(subR25pept[[1]]), pattern=pattern, level=level)
type <- 'AllCond'
conds <- design_qf(subR25pept)$Condition
op <- '>='
th <- 0.5
percent <- TRUE
ind <- GetIndices_BasedOnConditions(metacell.mask, type, conds, percent, op, th)
```

---

GetIndices\_FunFiltering

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

---

**Description**

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

**Usage**

```
GetIndices_FunFiltering(  
  obj,  
  conds,  
  level,  
  pattern = NULL,  
  type = NULL,  
  percent,  
  op,  
  th  
)
```

**Arguments**

obj	An object of class SummarizedExperiment containing quantitative data.
conds	A vector containing the names of the conditions from the sample.
level	A vector of integers which are the indices of lines to delete.
pattern	A string to be included in the SummarizedExperiment object for log.
type	Available values are: <ul style="list-style-type: none"><li>• 'AllCond' (the query is valid in all the conditions),</li><li>• 'AtLeatOneCond' (the query is valid in at least one condition).</li></ul>
percent	A boolean to indicate whether the threshold represent an absolute value (percent = FALSE) or a percentage (percent=TRUE).
op	String for operator to use. List of operators is available with 'SymFilteringOperators()'.
th	A floating number which is in the interval $[0, 1]$

**Value**

An instance of class SummarizedExperiment that have been filtered.

**Author(s)**

Samuel Wieczorek

**Examples**

NA

---

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` A `data.frame()` of boolean values which indictes the presence (TRUE) or not (FALSE) of given tags in the quantitative cells metadata

**Value**

A vector of `integer()`

**Examples**

```
data(subR25pept)
level <- 'peptide'
pattern <- "Missing POV"
metacell.mask <- matchMetacell(metadata = qMetacell(subR25pept[[1]]),
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	A data.frame() of boolean values which indicates the presence (TRUE) or not (FALSE) of given tags in the quantitative cells metadata
op	String for operator to use. List of operators is available with 'SymFilteringOperators()'.
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]

**Value**

A vector of integer()

**Examples**

```
data(subR25pept)
level <- 'peptide'
pattern <- "Missing"
metacell.mask <- matchMetacell(
  metadata = qMetacell(subR25pept[[1]]), pattern = pattern, level = level)
percent <- FALSE
th <- 3
op <- ">="
ind <- GetIndices_WholeMatrix(metacell.mask, op, percent, th)
```

---

getListNbValuesInLines

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

---

**Description**

Returns the possible number of values in lines in the data

**Usage**

```
getListNbValuesInLines(object, conds, type = "WholeMatrix")
```

**Arguments**

object	An object of class QFeatures
conds	A character() vector which is the names of conditions.
type	WholeMatrix, AllCond or AtLeastOneCond

**Value**

An integer

**Author(s)**

Samuel Wiczorek, Enora Fremy

**Examples**

```
data(subR25prot)
res <- getListNbValuesInLines(subR25prot[[1]])
```

---

GetNbTags

*Number of each metacell tags*

---

**Description**

Number of each metacell tags

**Usage**

```
GetNbTags(obj)
```

**Arguments**

obj                    A instance of the class SummarizedExperiment

**Value**

An integer

**Examples**

```
library(DaparToolshed)
data(subR25pept, package = 'DaparToolshed')
GetNbTags(subR25pept[[1]])
```

---

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 Wiczorek

**Examples**

```
library(QFeatures)
data(subR25prot)
qData <- SummarizedExperiment::assay(subR25prot[[1]])
getNumberOfEmptyLines(qData)
```

---

getPepProtCC            *Build the list of connex composant of the adjacency matrix*

---

**Description**

Build the list of connex composant of the adjacency matrix

**Usage**

```
getPepProtCC(X)
```

**Arguments**

X                An adjacency matrix

**Value**

A list of CC

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
data(subR25pept)
X <- QFeatures::adjacencyMatrix(subR25pept[[1]])
ll <- getPepProtCC(X)
```

---

GetSoftAvailables	<i>The set of available softwares to convert from</i>
-------------------	---

---

**Description**

The set of available softwares to convert from

**Usage**

```
GetSoftAvailables()
```

**Value**

A vector of character

**Examples**

```
GetSoftAvailables()
```

---

globalAdjPval	<i>Computes the adjusted p-values on all the stacked contrasts using CP4P</i>
---------------	---

---

**Description**

Computes the adjusted p-values on all the stacked contrasts using CP4P

**Usage**

```
globalAdjPval(x, pval.threshold = 1.05, method = 1, display = TRUE)
```

**Arguments**

x	a proteins x contrasts dataframe of (raw) p-values
pval.threshold	all the p-values above the threshold are not considered. Default is 1.05 (which is equivalent to have no threshold). Applying a threshold nearby 1 can be instrumental to improve the uniformity under the null, notably in case of upstream multiple contrast correction (for experienced users only)
method	method a method to estimate $\pi_0$ , see CP4P
display	if T, a calibration plot is displayed using CP4P

**Value**

a proteins x contrasts table of adjusted p-values

**Author(s)**

Thomas Burger

**Examples**

```
data(subR25prot)
obj <- subR25prot[seq_len(5),]
globalAdjPval(testAnovaModels(
  applyAnovasOnProteins(obj, 1), "TukeyHSD"))$P_Value)
```

---

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, th_logFC = 0, pal = NULL)
```

**Arguments**

df_logFC	A dataframe that contains the logFC values
th_logFC	The threshold on log(Fold Change) to distinguish between differential and non-differential data
pal	A vector of HEX codes for colors.

**Value**

A plotly density plot

**Author(s)**

Samuel Wiczorek

**Examples**

```
library(SummarizedExperiment)
data(subR25pept)
# Simulate missing value imputation
SummarizedExperiment::assay(subR25pept[[1]])[which(is.na(assay(subR25pept[[1]]))] <- 0

qData <- as.matrix(SummarizedExperiment::assay(subR25pept[[1]])
sTab <- SummarizedExperiment::colData(subR25pept)
limma <- limmaCompleteTest(qData, sTab)
```

```
pal <- ExtendPalette(2, "Dark2")
hc_logFC_DensityPlot(limma$logFC, th_logFC = 1, pal = pal)
```

---

heatmapForMissingValues

*Display a heatmap for data with missing values*

---

### Description

Display a heatmap for data with missing values

### Usage

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

### Arguments

x	A numeric matrix
col	See graphics::image()
srtCol	See graphics::text()
labCol	See graphics::text()
labRow	See graphics::axis()
key	See graphics::par()
key.title	See graphics::title()
main	See graphics::title()
ylab	See graphics::image()

### Value

A heatmap

---

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

---

**Description**

Plots a histogram of p-values

**Usage**

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

**Arguments**

pval_ll	A vector of the p-values.
bins	A integer indicating the number of cells for the histogram
pi0	A float between 0 and 1 corresponding to the proportion of true null hypotheses.

**Value**

A histogram of the p-values with pi0

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25prot)
obj <- subR25prot
# Simulate imputation
obj <- NAIsZero(obj, 1)
allComp <- limmaCompleteTest(
  SummarizedExperiment::assay(obj[[length(obj)]]),
  design_qf(obj),
  comp.type="OnevsOne")
histPValue_HC(allComp$P_Value[1])
```

---

`imputePA2`*Missing values imputation from a MSnSet object*

---

**Description**

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

**Usage**

```
imputePA2(  
  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> ).
<code>q.norm</code>	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 <code>sd</code> is the standard deviation of a row in a condition).
<code>eps</code>	A value 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.
<code>distribution</code>	The type of distribution used. Values are <code>unif</code> or <code>beta</code> .

**Value**

The object `obj` which has been imputed

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
library(QFeatures)
utils::data(subR25pept)
qdata <- SummarizedExperiment::assay(subR25pept[[1]])
conds <- design_qf(subR25pept)$Condition
obj.imp <- imputePA2(qdata, conds, distribution = "beta")
```

---

InitializeHistory	<i>Initialize the history</i>
-------------------	-------------------------------

---

**Description**

This function initializes the history.

**Usage**

```
InitializeHistory()
```

**Value**

An empty data.frame with 4 columns ('Process', 'Step', 'Parameter' and 'Value')

**Examples**

```
InitializeHistory()
```

---

isDifferential	<i>Identification of differentially abundant peptide/protein</i>
----------------	--

---

**Description**

This function allows to identify differentially abundant peptide/protein

**Usage**

```
isDifferential(pvalue, logFC, thpvalue, thlogFC)
```

**Arguments**

pvalue	A vector of p-values.
logFC	A vector of logFC.
thpvalue	A float indicating the p-value threshold.
thlogFC	A float indicating the logFC threshold.

**Value**

A vector indicating which peptide/protein is differentially abundant (1) or not (0).

**Author(s)**

Manon Gaudin

**Examples**

```
data(subR25prot)
obj <- subR25prot
# Simulate imputation
obj <- NAIsZero(obj, 1)
allComp <- limmaCompleteTest(
  SummarizedExperiment::assay(obj[[length(obj)]]),
  design_qf(obj),
  comp.type="OnevsOne")
isDifferential(allComp$P_Value[, 1], allComp$logFC[, 1], 0.05, 0.5)
```

---

isOfType

*Similar to the function is.na() but focused on the equality with the paramter 'type'.*

---

**Description**

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

**Usage**

```
isOfType(data, type)
```

**Arguments**

data	A data.frame
type	The value to search in the dataframe

**Value**

A boolean dataframe

**Author(s)**

Samuel Wieczorek

**Examples**

```
library(QFeatures)
data(subR25prot)
obj <- subR25prot[[1]]
data <- qMetacell(obj)
isOfType(as.data.frame(data), "MEC")
```

---

isSubset

*Check is a given set is a subset of another one.*

---

**Description**

Check is a given set is a subset of another one.

**Usage**

```
isSubset(set1, set2)
```

**Arguments**

set1	A vector of character()
set2	A vector of character()

**Value**

A boolean

**Examples**

```
isSubset('a', letters)
isSubset(c('a', 'c', 't'), letters)
isSubset(c('a', 3, 't'), letters)
isSubset(3, letters)
```

Keep\_Items\_from\_Dataset

*Removes assay from a QFeatures object*

---

### **Description**

Removes one or more items from the dataset. This function is specific of the type of dataset.

### **Usage**

```
Keep_Items_from_Dataset(dataset, range)
```

### **Arguments**

dataset            An instance of the QFeatures class

range             A vector of integers

### **Value**

The dataset minus some items

---

LH0

*Compute the likelihood of the null hypothesis for the global model*

---

### **Description**

Compute the likelihood of the null hypothesis for the global model

### **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**

A list of likelihoods

### **Author(s)**

Thomas Burger, Laurent Jacob

**Examples**

NULL

---

LH0_lm	<i>Compute the likelihoods of the null hypothesis for the "local" component-wise model</i>
--------	--

---

**Description**

Compute the likelihoods of the null hypothesis for the "local" component-wise model

**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**

A list of likelihoods with associated linear models

**Author(s)**

Thomas Burger, Laurent Jacob

**Examples**

NULL

---

LH1	<i>Compute the likelihood of the alternative hypothesis for the global model</i>
-----	--

---

**Description**

Compute the likelihood of the alternative hypothesis for the global model

**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**

A list of likelihoods

**Author(s)**

Thomas Burger, Laurent Jacob

**Examples**

NULL

---

LH1_lm	<i>Compute the likelihoods of the alternative hypothesis for the "local" component-wise model</i>
--------	---

---

**Description**

Compute the likelihoods of the alternative hypothesis for the "local" component-wise model

**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**

A list of likelihoods with associated linear models

**Author(s)**

Thomas Burger, Laurent Jacob

**Examples**

NULL

---

limmaCompleteTest	<i>Computes a hierarchical differential analysis</i>
-------------------	--

---

**Description**

Computes a hierarchical differential analysis

**Usage**

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

**Arguments**

qData	A matrix of quantitative data, without any missing values.
sTab	A dataframe of experimental design (design_qf()).
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 Gai-Gianetto, Samuel Wiczorek

**Examples**

```
data(subR25pept)
qData <- as.matrix(SummarizedExperiment::assay(subR25pept[[1]]))
sTab <- SummarizedExperiment::colData(subR25pept)
limma <- limmaCompleteTest(qData, sTab, comp.type = "anova1way")
```

---

makeContrast

*Builds the contrast matrix*


---

**Description**

Builds the contrast matrix

**Usage**

```
makeContrast(design, condition, contrast = 1, design.level = 1)
```

**Arguments**

design	The data.frame which correspond to the colData() function of package MultiAssayExperiment.
condition	A vector of conditions of the dataset
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).
design.level	An integer which specifies the level of the design

**Value**

A contrast matrix

**Author(s)**

Thomas Burger, Quentin Giai-Gianetto, Samuel Wiczorek

**Examples**

```
data(subR25pept)
design <- makeDesign(SummarizedExperiment::colData(subR25pept))
conds <- design_qf(subR25pept)$Condition
makeContrast(design, conds)
```

---

makeDesign	<i>Builds the design matrix</i>
------------	---------------------------------

---

**Description**

Builds the design matrix

**Usage**

```
makeDesign(sTab)
```

**Arguments**

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

**Value**

A design matrix

**Author(s)**

Thomas Burger, Quentin Giai-Gianetto, Samuel Wiczorek

**Examples**

```
data(subR25pept)
makeDesign(SummarizedExperiment::colData(subR25pept))
```

---

makeDesign1	<i>Builds the design matrix for designs of level 1</i>
-------------	--

---

**Description**

Builds the design matrix for designs of level 1

**Usage**

```
makeDesign1(sTab)
```

**Arguments**

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

**Value**

A design matrix

**Author(s)**

Thomas Burger, Quentin Giai-Gianetto, Samuel Wiczorek

**Examples**

```
data(subR25pept)
makeDesign1(SummarizedExperiment::colData(subR25pept))
```

---

makeDesign2

*Builds the design matrix for designs of level 2*

---

**Description**

Builds the design matrix for designs of level 2

**Usage**

```
makeDesign2(sTab)
```

**Arguments**

sTab            The data.frame which correspond to the colData() function of package MSnbase.

**Value**

A design matrix

**Author(s)**

Thomas Burger, Quentin Giai-Gianetto, Samuel Wiczorek

**Examples**

```
data(subR25pept)
makeDesign2(SummarizedExperiment::colData(subR25pept))
```

---

makeDesign3	<i>Builds the design matrix for designs of level 3</i>
-------------	--

---

**Description**

Builds the design matrix for designs of level 3

**Usage**

```
makeDesign3(sTab)
```

**Arguments**

sTab                   The data.frame which correspond to the colData() function of package MSnbase.

**Value**

A design matrix

**Author(s)**

Thomas Burger, Quentin Giai-Gianetto, Samuel Wiczorek

**Examples**

```
data(subR25pept)
sTab <- cbind(SummarizedExperiment::colData(subR25pept), Tech.Rep = seq_len(6))
makeDesign3(sTab)
```

---

matchMetacell	<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**

```
matchMetacell(metadata, pattern = NULL, level)
```

**Arguments**

metadata	A data.frame
pattern	The value to search in the dataframe
level	A string designing the type of entity/pipeline. Available values are: peptide, protein

**Value**

A boolean dataframe

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25pept)
metadata <- qMetacell(subR25pept[[1]])
m <- matchMetacell(metadata, pattern = "Missing", level = "peptide")
m <- matchMetacell(metadata, pattern = 'Missing POV', level = "peptide")
m <- matchMetacell(metadata, pattern = c('Missing', 'Missing POV'), level = "peptide")
```

---

metacell-plots

*Bar plot of missing values per lines using plotly*


---

**Description**

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

Plots a heatmap of the quantitative data. Each column represent one of the conditions in the object of class obj 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.

#' 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.

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**

```

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

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

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

wrapperMVImage(obj, group = NULL, pattern = "Missing MEC")

mvImage(obj, group)

hc_mvTypePlot2(obj, group, pal = NULL, pattern, title = NULL)

```

**Arguments**

obj	An instance of the class QFeatures
group	A vector
pattern	A character() indicating the tag pattern of interest.
detailed	'value' or 'percent'
indLegend	A vector() of integers
showValues	A logical that indicates whether numeric values should be drawn above the bars.
pal	The different colors for conditions
title	The title of the plot

**Value**

A bar plot  
 A heatmap  
 A heatmap  
 Density plots

**Author(s)**

Florence Combes, Samuel Wiczorek  
 Samuel Wiczorek, Alexia Dorffer  
 Samuel Wiczorek, Thomas Burger  
 Samuel Wiczorek

**Examples**

```

data(subR25prot)
grp <- design_qf(subR25prot)$Condition
metacellPerLinesHisto_HC(subR25prot[[1]], group = grp, pattern = "Missing POV")
metacellPerLinesHisto_HC(subR25prot[[1]])
metacellPerLinesHisto_HC(subR25prot[[1]], group = grp, pattern = "Quantified")
metacellPerLinesHisto_HC(subR25prot[[1]], group = grp, pattern = "Quant. by direct id")
metacellPerLinesHisto_HC(subR25prot[[1]], group = grp, pattern = "Quant. by recovery")
pattern <- c("Quantified", "Quant. by direct id", "Quant. by recovery")
metacellPerLinesHisto_HC(subR25prot[[1]], group = grp, pattern = pattern)

metacellPerLinesHistoPerCondition_HC(subR25prot[[1]], group = grp, pattern = "Missing POV")
metacellPerLinesHistoPerCondition_HC(subR25prot[[1]])
metacellPerLinesHistoPerCondition_HC(subR25prot[[1]], group = grp, pattern = "Quantified")
metacellPerLinesHistoPerCondition_HC(subR25prot[[1]], group = grp, pattern = "Quant. by direct id")
metacellPerLinesHistoPerCondition_HC(subR25prot[[1]], group = grp, pattern = "Quant. by recovery")
pattern <- c("Quantified", "Quant. by direct id", "Quant. by recovery")
metacellPerLinesHistoPerCondition_HC(subR25prot[[1]], group = grp, pattern = pattern)

metacellHisto_HC(subR25prot[[1]], group = grp, pattern = "Missing POV")
metacellHisto_HC(subR25prot[[1]])
metacellHisto_HC(subR25prot[[1]], group = grp, pattern = "Quantified")
metacellHisto_HC(subR25prot[[1]], group = grp, pattern = "Quant. by direct id")
metacellHisto_HC(subR25prot[[1]], group = grp, pattern = "Quant. by recovery")
pattern <- c("Quantified", "Quant. by direct id", "Quant. by recovery")
metacellHisto_HC(subR25prot[[1]], group = grp, pattern = pattern)

data(subR25pept)
pattern <- "Missing POV"
pal <- ExtendPalette(2, "Dark2")
metacellHisto_HC(subR25pept[[1]], pattern, showValues = TRUE, pal = pal)

```

```

data(subR25pept)
mImage(subR25pept[[1]], design_qf(subR25pept)$Condition)

data(subR25pept)
pal <- ExtendPalette(length(unique(design_qf(subR25pept)$Condition)), "Dark2")
pattern <- "Missing MEC"
hc_mvTypePlot2(subR25pept[[1]],
group = design_qf(subR25pept)$Condition,
pattern = pattern, pal = pal)

```

---

Metacell_DIA_NN	<i>Sets the metacell dataframe for datasets which are from Dia-NN software</i>
-----------------	--

---

### Description

Actually, this function uses the generic function to generate metacell info

### Usage

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

### Arguments

qdata	An object of class MsnSet
conds	A 1-col dataframe with the condition associated to each sample.
df	A dataframe with the same dimension as qdata containing the metacell.
level	A string designing the type of entity/pipeline. Available values are: peptide, protein

### Value

NA

### Author(s)

Samuel Wiczorek

### Examples

```

data(subR25pept)
conds <- design_qf(subR25pept)$Condition
qdata <- SummarizedExperiment::assay(subR25pept[[1]])
df <- Metacell_DIA_NN(qdata, conds, df, level = "peptide")

```

---

Metacell\_maxquant      *Sets the metacell dataframe*

---

### 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 = NULL, level = NULL)
```

### Arguments

qdata	An object of class MsnSet
conds	A 1-col dataframe with the condition associated to each sample.
df	A dataframe with the same dimension as qdata containing the metacell.
level	A string designing the type of entity/pipeline. Available values are: peptide, protein

### Value

NA

### Author(s)

Samuel Wieczorek

### Examples

```
data(subR25pept)
conds <- design_qf(subR25pept)$Condition
qdata <- SummarizedExperiment::assay(subR25pept[[1]])
df2 <- Metacell_maxquant(qdata, conds, level = "peptide")
```

---

Metacell_proline	<i>Sets the metacell dataframe for datasets which are from Proline software</i>
------------------	---

---

## Description

In the quantitative columns, a missing value is identified by no value rather than a value equal to 0.

In these datasets, the metacell info is computed from the 'PSM count' columns.

Conversion rules Initial conversion rules for proline |—————|—————|———| | Quanti | 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 = NULL, level = NULL)
```

## Arguments

qdata	An object of class MsnSet
conds	A 1-col dataframe with the condition associated to each sample.
df	A dataframe with the same dimension as qdata containing the metacell.
level	A string designing the type of entity/pipeline. Available values are: peptide, protein

## Value

NA

## Author(s)

Samuel Wieczorek

## Examples

```
data(subR25pept)
conds <- design_qf(subR25pept)$Condition
qdata <- SummarizedExperiment::assay(subR25pept[[1]])
df <- Metacell_proline(qdata, conds, level = "peptide")
```

---

MetacellFilteringScope

*Lists the metacell scopes for filtering*

---

### Description

Lists the metacell scopes for filtering

### Usage

MetacellFilteringScope()

### Value

A vector of character()

### Examples

MetacellFilteringScope()

---

mv\_imputation\_protein *Finds the LAPALA*

---

### Description

Methods available are:

- wrapperImputeDetQuant(): This method is a wrapper of the function `impute.detQuant()` for objects of class `MSnSet`
- wrapperImputeKNN(): Can impute only POV missing values. This method is a wrapper for objects of class `QFeatures` and imputes missing values with a fixed value. This function imputes the missing values condition by condition.
- wrapperImputeSLSA(): Imputation of peptides having no values in a biological condition. This method is a wrapper to the function `impute.slsa()` of the package `imp4p` adapted to an object of class `MSnSet`.
- wrapperImputeFixedValue(): This method is a wrapper to objects of class `MSnSet` and imputes missing values with a fixed value.
- wrapperImputePA(): Imputation of peptides having no values in a biological condition. This method is a wrapper to the function `impute.pa` of the package `imp4p` adapted to an object of class `MSnSet`.

**Usage**

```

findMECBlock(obj, grp)

reIntroduceMEC(obj, grp, MECIndex)

wrapperImputeKNN(obj = NULL, grp, K)

wrapperImputeFixedValue(obj, grp, fixVal = 0, na.type)

wrapperImputePA(obj = NULL, grp, q.min = 0.025)

wrapperImputeDetQuant(obj, qval = 0.025, factor = 1, na.type)

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

wrapperImputeSLSA(obj = NULL, design = NULL)

```

**Arguments**

obj	An object of class QFeatures.
grp	A vector of conditions in the dataset
MECIndex	A data.frame that contains index of MEC (see findMECBlock)
K	the number of neighbors.
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.
q.min	Same as the function impute.pa() in the package imp4p
qval	An expression set containing quantitative values of various replicates
factor	A scaling factor to multiply the imputation value with
qdata	A data.frame() for the quantitative data
design	A data.frame() for the design of the dataset

**Value**

A data.frame containing the indexes of LAPALA

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

**Utilities functions:**

- findMECBlock()
- reIntroduceMEC()
- getQuantile4Imp(): Quantile imputation value definition. This method returns the q-th quantile of each column of an expression set, up to a s calling factor

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25prot)
obj <- subR25prot[[1]]
grp <- design_qf(subR25prot)$Condition
lapala <- findMECBlock(obj, grp)
na.type = c("Missing POV", "Missing MEC")
obj.imp.pov <- wrapperImputeDetQuant(obj, na.type = na.type)
obj.imp.pov <- reIntroduceMEC(obj, grp, lapala)

obj.imp.pov <- wrapperImputeKNN(obj, grp, 3)

obj.imp.pov <- wrapperImputeFixedValue(obj, grp, 0.001, na.type = "Missing POV")
obj.imp.mec <- wrapperImputeFixedValue(obj, grp, 0.001, na.type = "Missing MEC")
obj.imp.na <- wrapperImputeFixedValue(
  obj, grp, 0.001,
  na.type = c("Missing MEC", "Missing POV"))

obj.imp.pov <- wrapperImputePA(obj, grp)

qdata <- SummarizedExperiment::assay(obj)
quant <- getQuantile4Imp(qdata)
```

---

NAIsZero

*Set NA values to 0*

---

**Description**

Set NA values to 0

**Usage**

```
NAIsZero(obj, i)
```

**Arguments**

obj                   An instance of QFeatures class  
i                      An integer which is the index of the assay in the QFeatures object

**Value**

An instance of QFeatures class

---

nEmptyLines	<i>Number of empty lines in the data</i>
-------------	--

---

**Description**

This function counts the number of empty lines (all elements are equal to NA).

**Usage**

```
nEmptyLines(df)
```

**Arguments**

df                    A data.frame.

**Value**

A integer(1)

**Author(s)**

Samuel Wieczorek

**Examples**

```
library(QFeatures)
data(subR25prot)
nEmptyLines(SummarizedExperiment::assay(subR25prot, 1))
```

---

nonzero	<i>Retrieve the indices of non-zero elements in sparse matrices</i>
---------	---

---

**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)
```

---

normalization\_methods *Normalisation*

---

**Description**

Provides several methods to normalize quantitative data from a SummarizedExperiment 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 SummarizedExperiment::assay() data tab) is computed condition by condition.

**Usage**

```
normalizeMethods(target = "all")
```

```
GlobalQuantileAlignment(qData)
```

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

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

```
MeanCentering(
  qData,
  conds,
```

```

    type = "overall",
    subset.norm = NULL,
    scaling = FALSE
  )

  vsn(qData, conds, type = NULL)

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

```

### Arguments

target	Category of normalization method to show. Either "all", "withTracking" or "withoutTracking".
qData	A data.frame with quantitative data to normalize.
conds	A character() vector which is the names of conditions for each sample in the dataset.
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.
scaling	A boolean that indicates if the variance of the data have to be forced to unit (variance reduction) or not.
span	A float between 0 and 1 indicating the span of loess smoothing window.

### Value

A vector of character()  
 A normalized numeric matrix  
 A normalized numeric matrix  
 A normalized numeric matrix  
 A normalized numeric matrix  
 A normalized numeric matrix  
 A normalized numeric matrix

### Author(s)

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

### Examples

```

## Get the list of methods
normalizeMethods()

data(subR25pept)
qData <- SummarizedExperiment::assay(subR25pept[[1]])

```

```

conds <- design_qf(subR25pept)$Condition

#normalized <- GlobalQuantileAlignment(qData)

normalized <- SumByColumns(qData, conds,
  type = "within conditions",
  subset.norm = seq_len(10)
)

normalized <- QuantileCentering(
  SummarizedExperiment::assay(subR25pept), conds,
  type = "within conditions", subset.norm = seq_len(10)
)

normalized <- MeanCentering(qData, conds, type = "overall")

normalized <- LOESS(qData, conds, type = "overall")

```

---

normalizeFunction      *Normalisation for QFeatures*

---

## Description

This method is a wrapper that provides several methods to normalize quantitative data from objects of class `QFeatures` or `SummarizedExperiment`.

They are organized in six main families : `GlobalQuantileAlignment`, `sumByColumns`, `QuantileCentering`, `MeanCentering`, `LOESS`, `vs`n 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 `SummarizedExperiment::assay()` data tab) is computed condition by condition. The available methods are described in [normalizeMethods\(\)](#).

## Usage

```

normalizeFunction(
  obj,
  method,
  conditions = NULL,
  type = "overall",
  subset.norm = NULL,
  quantile = 0.15,
  scaling = FALSE,
  span = 0.7
)

```

**Arguments**

obj	An object of class QFeatures or SummarizedExperiment. If data is of class QFeatures, the last assay will be normalized.
method	Define the normalization method used : "GlobalQuantileAlignment", "QuantileCentering", "MeanCentering", "SumByColumns", "LOESS" or "vsn".
conditions	A vector of conditions in the dataset. If not provided, the vector "Condition" from the column metadata will be used.
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.
scaling	A boolean that indicates if the variance of the data have to be forced to unit (variance reduction) or not.
span	A floating number

**Value**

QFeatures including a new assay with normalized data or SummarizedExperiment with normalized data.

**Author(s)**

Manon Gaudin

**Examples**

```
data(subR25pept)
normalized <- normalizeFunction(subR25pept, method = 'GlobalQuantileAlignment')
```

---

output\_2\_Excel

*This function exports a data.frame 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**

```
readExcel(file, sheet = NULL)

listSheets(file)

write_Assay_To_Excel(wb, obj, i, n)

WriteHistory(wb, obj, n)

Write_SamplesData_to_Excel(wb, obj, n)

Write_RowData(wb, obj, i, n)

writeExcel(obj, filename)
```

**Arguments**

file	The name of the Excel file.
sheet	The worksheet to write to. Can be the worksheet index or name.
wb	A Workbook object containing a worksheet.
obj	An instance of the class QFeatures
i	An integer which is the index of the assay in the QFeatures object
n	The total number of sheets
filename	A character string for the name of the Excel file.

**Value**

A Excel file (.xlsx)

**Author(s)**

Samuel Wieczorek

**Examples**

```
library(QFeatures)
data(subR25prot)
df <- SummarizedExperiment::assay(subR25prot[[1]])
tags <- qMetacell(subR25prot[[1]])
colors <- list(
  "Missing POV" = "lightblue",
  "Missing MEC" = "orange",
  "Quant. by recovery" = "lightgrey",
  "Quant. by direct id" = "white",
  "Combined tags" = "red"
)
file <- tempfile('toto.xlsx')
writeExcel(subR25prot, filename = file)
```

```
unlink(file)

data(subR25pept)
file <- tempfile('foo.xlsx')
writeExcel(subR25pept, file)
unlink(file)
```

---

OWAnova

*Applies aov() on a vector of protein abundances using the design derived from the sample names (simple aov wrapper)*

---

### Description

Applies aov() on a vector of protein abundances using the design derived from the sample names (simple aov wrapper)

### Usage

```
OWAnova(current_protein, conditions)
```

### Arguments

current\_protein      a real vector

conditions          the list of groups the protein belongs to

### Value

See aov()

### Author(s)

Thomas Burger

### Examples

```
protein_abundance <- rep(rnorm(3, mean= 18, sd=2), each=3) + rnorm(9)
groups <- c(rep("group1",3),rep("group2",3),rep("group3",3))
OWAnova(protein_abundance,groups)
```

---

 paramshistory

*List of metacell tags*


---

## Description

This function gives the list of metacell tags available.

- **onlyPresent:** In this case, the function gives the tags found in a dataset. In addition, and w.r.t to the hierarchy of tags, if all leaves of a node are present, then the tag corresponding to this node is added.

These names are common to all assays contained in the object. This is why they are stored in the global metadata. This function is used whenever it is necessary to (re)detect MEC and POV (new dataset or when post processing protein qMetacell after aggregation)

## Usage

```

paramshistory(object, ...)

## S4 method for signature 'QFeatures'
paramshistory(object, i, slotName = "paramshistory")

## S4 method for signature 'SummarizedExperiment'
paramshistory(object, slotName = "paramshistory")

paramshistory(object, i, slotName = "paramshistory") <- value

GetMetacellTags(object, ...)

## S4 method for signature 'QFeatures'
GetMetacellTags(object, i, ...)

## S4 method for signature 'SummarizedExperiment'
GetMetacellTags(object, ...)

## S4 method for signature 'data.frame'
GetMetacellTags(object, ...)

qMetacell(object, ...)

## S4 method for signature 'QFeatures'
qMetacell(object, i)

## S4 method for signature 'SummarizedExperiment'
qMetacell(object)

qMetacell(object, i, slotName = "qMetacell") <- value

```

```
GetUniqueTags(object, ...)  
  
## S4 method for signature 'QFeatures'  
GetUniqueTags(object, i)  
  
## S4 method for signature 'SummarizedExperiment'  
GetUniqueTags(object)  
  
GetMetadataSlot(object, slotName = NULL)  
  
GetRowdataSlot(object, slotName = NULL)  
  
ConnectedComp(object, ...)  
  
## S4 method for signature 'QFeatures'  
ConnectedComp(object, i, slotName = "ConnectedComp")  
  
## S4 method for signature 'SummarizedExperiment'  
ConnectedComp(object, slotName = "ConnectedComp")  
  
ConnectedComp(object, i, slotName = "ConnectedComp") <- value  
  
typeDataset(object, ...)  
  
## S4 method for signature 'QFeatures'  
typeDataset(object, i, slotName = "typeDataset")  
  
## S4 method for signature 'SummarizedExperiment'  
typeDataset(object, slotName = "typeDataset")  
  
typeDataset(object, i, slotName = "typeDataset") <- value  
  
idcol(object, ...)  
  
## S4 method for signature 'QFeatures'  
idcol(object, i, slotName = "idcol")  
  
## S4 method for signature 'SummarizedExperiment'  
idcol(object, slotName = "idcol")  
  
idcol(object, i, slotName = "idcol") <- value  
  
parentProtId(object, ...)  
  
## S4 method for signature 'QFeatures'  
parentProtId(object, i, slotName = "parentProtId")
```

```
## S4 method for signature 'SummarizedExperiment'
parentProtId(object, slotName = "parentProtId")

parentProtId(object, i, slotName = "parentProtId") <- value

filename(object, ...)

## S4 method for signature 'QFeatures'
filename(object, slotName = "filename")

filename(object, slotName = "filename") <- value

analysis(object, ...)

## S4 method for signature 'QFeatures'
analysis(object, i, slotName = "analysis")

## S4 method for signature 'SummarizedExperiment'
analysis(object, slotName = "analysis")

analysis(object, i, slotName = "analysis") <- value

version(object, ...)

## S4 method for signature 'QFeatures'
version(object, slotName = "version")

version(object, slotName = "version") <- value

design_qf(object, ...)

## S4 method for signature 'QFeatures'
design_qf(object, slotName = "design")

design_qf(object, slotName = "design") <- value

mainAssay(object)

HypothesisTest(object, ...)

## S4 method for signature 'QFeatures'
HypothesisTest(object, i, slotName = "HypothesisTest")

## S4 method for signature 'SummarizedExperiment'
HypothesisTest(object, slotName = "HypothesisTest")

HypothesisTest(object, i) <- value
```

```

DifferentialAnalysis(object, ...)

## S4 method for signature 'QFeatures'
DifferentialAnalysis(object, i, slotName = "DifferentialAnalysis")

## S4 method for signature 'SummarizedExperiment'
DifferentialAnalysis(object, slotName = "DifferentialAnalysis")

DifferentialAnalysis(object, i) <- value

names_metacell(object, ...)

## S4 method for signature 'QFeatures'
names_metacell(object, i, slotName = "names_metacell")

## S4 method for signature 'SummarizedExperiment'
names_metacell(object, slotName = "names_metacell")

names_metacell(object, i, slotName = "names_metacell") <- value

```

### Arguments

object	n instance of class QFeatures or SummarizedExperiment
...	Additional parameters
i	The index or name of the assays to extract the quantitative metadata from. All must have a rowdata variable named as slotName
slotName	A character(0) which is the name of the slot in the metadata
value	The content of the slot in the metadata

### Details

Additional slots for Metadata for a SummarizedExperiment object:

- qMetacell: A data.frame()
- parentProtId: A character()
- idcol: A character()
- typeDataset: A character()

### Value

A vector of tags.

NA

NA

NA

NA

NA

NA

NA

NA

**Quantitative metadata**

Default slotName is "qMetacell". The value is an adjacency matrix with row and column names. The matrix will be coerced to compressed, column-oriented sparse matrix (class dgCMatrix) as defined in the Matrix package, as generated by the `Matrix::sparseMatrix()` constructor.

**Author(s)**

Samuel Wieczorek

**Examples**

```
data(subR25pept)
GetMetacellTags(subR25pept, 1, level="peptide")
GetMetacellTags(subR25pept, 1, level="peptide", onlyPresent=TRUE)
```

```
data(subR25pept)
design_qf(subR25pept)
```

---

 Parent

---

*Parent name of a node*


---

**Description**

Parent name of a node

**Usage**

```
Parent(level, node = NULL)
```

**Arguments**

level	A string designating the type of entity/pipeline. Available values are: peptide, protein
node	A character() #' @examples Parent('protein', 'Missing') Parent('protein', 'Missing POV') Parent('protein', c('Missing POV', 'Missing MEC')) Parent('protein', c('Missing', 'Missing POV', 'Missing MEC'))

**Value**

NA

---

pepaTest *PEptide based Protein differential Abundance test*

---

### Description

PEptide based Protein differential Abundance test

### Usage

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

### Arguments

X	Binary $q \times p$ design matrix for $q$ peptides and $p$ proteins. $X_{(ij)}=1$ if peptide $i$ belongs to protein $j$ , 0 otherwise.
y	$q \times n$ 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 <code>lm()</code> 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  $H_0$  mse.h1: Mean squared error under  $H_1$  s: selected regularization hyperparameter for llr.map. wchi2: weight used to make llr.map chi2-distributed under  $H_0$ .

### Author(s)

Thomas Burger, Laurent Jacob

### Examples

NA

Pipelines

*List of available pipelines in the package*

---

**Description**

Get the list of pipelines available in the package

**Usage**

```
Pipelines()
```

**Value**

NA

---

pkgsRequire

*Loads packages*

---

**Description**

Checks if a package is available to load it

**Usage**

```
pkgsRequire(ll.deps)
```

**Arguments**

ll.deps      A character() vector which contains packages names

**Value**

NA

**Author(s)**

Samuel Wieczorek

**Examples**

```
NULL
```

---

plotCompareAssays      *Compare two assays*

---

### Description

This plot compares the quantitative proteomics data between two assays. It can be used for example to compare the effect of the normalization process.

The comparison is made with the division operator.

### Usage

```
plotCompareAssays(  
  obj,  
  i,  
  j,  
  info = NULL,  
  pal.name = "Set1",  
  subset.view = NULL,  
  n = 100,  
  type = "scatter"  
)
```

### Arguments

obj	An instance of the class
i	A numeric matrix containing quantitative data after normalization.
j	A numeric matrix containing quantitative data after normalization
info	A vector
pal.name	The name of the palette to use. Default is 'Set1'
subset.view	A vector() of integers
n	The number of points to display
type	The type of plot. Available values are 'scatter' (default) or 'line'

### Value

A plot

### Author(s)

Samuel Wiczorek, Enora Fremy

**Examples**

```
data(subR25prot)
obj <- subR25prot
obj <- normalizeFunction(obj, method = "MeanCentering")
plotCompareAssays(obj, 1, 2, n = 5)
```

---

plotJitter

*Jitter plot of CC*

---

**Description**

Jitter plot of CC

**Usage**

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

**Arguments**

`list.of.cc` List of cc such as returned by the function `getPepProtCC`

**Value**

A plot

**Author(s)**

Thomas Burger

**Examples**

```
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
ll <- getPepProtCC(X)
plotJitter(ll)
```

---

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

---

**Description**

Display a a jitter plot for CC

**Usage**

```
plotJitter_rCharts(df)
```

**Arguments**

df                      A data.frame()

**Value**

A plot

**Author(s)**

Thomas Burger, Samuel Wiczorek

**Examples**

```
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
ll <- getPepProtCC(X)[seq_len(4)]
n.prot <- unlist(lapply(ll, function(x) {length(x$proteins)}))
n.pept <- unlist(lapply(ll, function(x) {length(x$peptides)}))
df <- tibble::tibble(
  x = jitter(n.pept),
  y = jitter(n.prot),
  index = seq_len(length(ll))
)
plotJitter_rCharts(df)
```

---

postHocTest              *Post-hoc tests for classic 1-way ANOVA*

---

**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**

```
library(SummarizedExperiment)
data(subR25prot)
obj <- subR25prot
filter <- FunctionFilter('qMetacellOnConditions',
  cmd = 'delete',
  mode = 'AtLeastOneCond',
  pattern = c("Missing POV", "Missing MEC"),
  conds = design_qf(obj)$Condition,
  percent = TRUE,
  th = 0.8,
  operator = '>')
obj <- filterFeaturesOneSE(obj, name = "Filtered", filters = list(filter))
qdata <- SummarizedExperiment::assay(obj[[2]])
conds <- design_qf(obj)$Condition
anova_tests <- apply(qdata, 1, classic1wayAnova, conditions = as.factor(conds))
anova_tests <- t(anova_tests)

names(anova_tests) <- rownames(qdata)
```

```
pht <- postHocTest(aov_fits = anova_tests)
```

---

pushpvalue

*Push p-values based on metacell tags*

---

### Description

This function allows to push p-values to 1 based on metacell tags.

### Usage

```
pushpvalue(
  obj,
  pvalue,
  scope = "WholeMatrix",
  pattern = "Imputed MEC",
  percent = TRUE,
  threshold = 1,
  conditions = NULL,
  operator = ">=",
  level = NULL,
  value = 1.00000000001
)
```

### Arguments

obj	An object of class QFeatures or SummarizedExperiment. If data is of class QFeatures, the last assay will be used.
pvalue	A vector of p-values.
scope	A string for scope to use. Available values are "WholeLine", "WholeMatrix", "AllCond" and "AtLeastOneCond".
pattern	A vector of tag to use.
percent	A boolean to indicate whether the threshold represent an absolute value (percent = FALSE) or a percentage (percent = TRUE).
threshold	A value that corresponds to the threshold value. Either an integer if percent = FALSE, or a float between 0 and 1 of percent = TRUE.
conditions	A vector of conditions in the dataset. If not provided, the vector "Condition" from the column metadata will be used.
operator	A string for operator to use. Available operators are "<=", "<", ">=", ">", "==" and "!=".
level	A string for dataset type. Either "peptide" or "protein" If not provided, the string obtained from typeDataset(obj) will be used.
value	A float, value to assign to the pushed p-value. By default, the value is set slightly above 1 to be able to differentiate the pushed value.

**Value**

A vector with pushed p-values.

**Author(s)**

Manon Gaudin

**Examples**

```
data(subR25prot)
obj <- subR25prot
# Simulate imputation
obj <- NAIsZero(obj, 1)
allComp <- limmaCompleteTest(SummarizedExperiment::assay(
obj[[length(obj)]]),
design_qf(obj),
comp.type="OnevsOne")
pushpvalue(obj,
allComp$P_Value[, 1],
scope = "WholeMatrix",
pattern = c("Missing MEC", "Missing POV"),
percent = TRUE,
threshold = 0.5,
operator = ">=",)
```

---

q\_metacell

*Quantitative metadata vocabulary for entities*

---

**Description**

This function gives the vocabulary used for the quantitative metadata of each entity in each condition.

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

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

Update the quantitative metadata information of missing values that were imputed

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

Aggregation rules for the cells quantitative metadata of peptides. Please refer to the qmetacellDef vocabulary in qmetacellDef()

**Usage**

```

metacellDef(level)

custom_metacell_colors()

Set_POV_MEC_tags(obj, conds)

Set_POV_MEC_tags2(conds, df, level)

Metacell_generic(qdata, conds, level)

UpdateMetacellAfterImputation(object, ...)

## S4 method for signature 'SummarizedExperiment'
UpdateMetacellAfterImputation(object)

searchMetacellTags(pattern, level, depth = "1")

metacombine(met, level)

```

**Arguments**

level	A string designing the type of entity/pipeline. Available values are: peptide, protein
obj	An object of class SummarizedExperiment
conds	A 1-col dataframe with the condition associated to each sample.
df	An object of class SummarizedExperiment
qdata	A matrix of quantitative data
object	An object of class SummarizedExperiment
...	Additional parameters
pattern	The string to search.
depth	Either "0", "1" or "*".
met	Metacells

**Value**

A data.frame containing the different tags and corresponding colors for the level given in parameter

A list

An instance of class QFeatures.

NA

NA

NA

## Glossary

### Peptide-level vocabulary

└ 'Any' ||||└ 1.0 'Quantified' |||||└ 1.1 "Quant. by direct id" (color 4, white) |||||└ 1.2 "Quant. by recovery" (color 3, lightgrey) ||||└ 2.0 "Missing" (no color) |||||└ 2.1 "Missing POV" (color 1) |||||└ 2.2 'Missing MEC' (color 2) ||||└ 3.0 'Imputed' |||||└ 3.1 'Imputed POV' (color 1) ||||└ 3.2 'Imputed MEC' (color 2)

Protein-level vocabulary: └ 'Any' ||||└ 1.0 'Quantified' |||||└ 1.1 "Quant. by direct id" (color 4, white) |||||└ 1.2 "Quant. by recovery" (color 3, lightgrey) ||||└ 2.0 "Missing" |||||└ 2.1 "Missing POV" (color 1) |||||└ 2.2 'Missing MEC' (color 2) ||||└ 3.0 'Imputed' |||||└ 3.1 'Imputed POV' (color 1) |||||└ 3.2 'Imputed MEC' (color 2) ||||└ 4.0 'Combined tags' (color 3bis, lightgrey)

## Conversion to the glossary

A generic conversion

Conversion for Proline datasets

Conversion from Maxquant datasets

## Basic aggregation

Aggregation of non imputed values (2.X) with quantitative values

(1.0, 1.X, 3.0, 3.X)

Not possible

---

Aggregation of different types of missing values (among 2.1, 2.2)

- Aggregation of 2.1 peptides between each other gives a missing value non imputed (2.0)
- Aggregation of 2.2 peptides between each other gives a missing value non imputed (2.0)
- Aggregation of a mix of 2.1 and 2.2 gives a missing value non imputed (2.0) |—————  
—————

Aggregation 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

**Author(s)**

Thomas Burger, Samuel Wiczorek

Samuel Wiczorek

**Examples**

```
metacellDef('protein')
metacellDef('peptide')

library(QFeatures)
data(subR25prot)
conds <- design_qf(subR25prot)$Condition
df <- Set_POV_MEC_tags(subR25prot[[1]], conds)

library(SummarizedExperiment)
data(subR25pept)
conds <- design_qf(subR25pept)$Condition
qdata <- SummarizedExperiment::assay(subR25pept[[1]])
df <- Metacell_generic(qdata, conds, 'peptide')

data(subR25prot)
subR25prot[[1]] <- UpdateMetacellAfterImputation(subR25prot[[1]])

searchMetacellTags('Missing POV', 'peptide')
searchMetacellTags('Quantified', 'peptide')

ll <- metacellDef('peptide')$node
for (i in seq_along(ll))
  test <- lapply(combn(ll, i, simplify = FALSE),
    function(x) tag <- metacombine(x, 'peptide'))
```

---

QFeatures-excel

*Exports a QFeatures object to a Excel file.*

---

**Description**

This function exports an instance of the class QFeatures to a Excel file. The resulting file is composed of four sheets:

- quantitative data which contains the content of assay() object with a color code for each cell w.r.t. to cell quantitative metadata.

- metadata which is the content of `rowData()` with only one-dimensional data (i.e. the adjacencyMatrix and the `qMetacell` slots are not part of the sheet),
- `exp.design` which is the content of `colData()`. Each condition in the table is colored with a different color,
- quantitative metadata which is the content of `qMetacell()`. There is a color code for the different tags.

### Usage

```
write2excel(object, ...)

## S4 method for signature 'QFeatures'
write2excel(object, i = NULL, filename = "newFile", writeColdData = TRUE, ...)

## S4 method for signature 'SummarizedExperiment'
write2excel(object, filename, exp.design, writeColData = TRUE, ...)

write2excelSE(object, filename, exp.design, writeColData = TRUE)

addColors(wb, n, tags, colors)
```

### Arguments

<code>object</code>	An instance of the class <code>QFeatures</code>
<code>...</code>	Additional parameters.
<code>i</code>	An integer which is the index of the assay in the <code>QFeatures</code> object
<code>filename</code>	A <code>character()</code> for the name of the Excel file
<code>writeColdData</code>	A boolean that indicates whether to include coldata or not in the Excel file
<code>exp.design</code>	A <code>data.frame()</code> for the sample design of the dataset
<code>writeColData</code>	A boolean that indicates whether to include col data in the final file
<code>wb</code>	A workbook
<code>n</code>	A <code>integer(1)</code> which is the number of sheet in the workbook.
<code>tags</code>	A <code>data.frame()</code>
<code>colors</code>	A <code>character()</code> which contains the HEX code for colors. The size of this vector must be the same as the number of tags.

### Value

A Excel file.

### Author(s)

Samuel Wieczorek

**Examples**

```

data(subR25prot)

#-----
# Export the whole dataset
#-----

write2excel(subR25prot, filename = "foo")
unlink('foo.xls')
write2excel(subR25prot, 1, "foo")
unlink('foo.xls')

```

---

QFeatures-filtering-oneSE

*Filter features of one SE based on their rowData*

---

**Description**

The filterFeaturesOneSE methods enables users to filter features based on a variable in their rowData. It is directly inspired of the function filterFeature of the package QFeatures. The first difference is that the filter only applies to one SummarizedExperiment contained in the object rather than applying on all the SE. This method generates a new SummarizedExperiment object which is added to the QFeatures object. If the SE on which the filter applies is the last one of the object, then a new SE is created. If it is not the last one, the new SE is added and all the further SE are deleted. The features matching the. The filters can be provided as instances of class AnnotationFilter (see the package QFeatures) or of class FunctionFilter (see below).

**Usage**

```

FunctionFilter(name, ...)

filterFeaturesOneSE(object, ...)

## S4 method for signature 'QFeatures'
filterFeaturesOneSE(object, i, name = "newAssay", filters)

```

**Arguments**

name	A character(1) naming the new assay. Default is newAssay. Note that the function will fail if there's already an assay with name.
...	Additional arguments
object	An instance of class QFeatures or SummarizedExperiment.
i	The index or name of the assay which features will be filtered the create the new assay.
filters	A list() containing instances of class AnnotationFilter or FunctionFilter

**Value**

A filtered QFeature object

**Function filters**

The function filters are filters as defined in the `DaparToolshed` package. Each filter is defined by a name (which is the name of a function) and a list which contains the parameters passed to the function. Those filters can be created with the `FunctionFilter` constructor.

Those functions are divided into two main categories:

- the one that filter on one `rowData` feature,
- the one based on a two-dimensional information such as the adjacency matrix

for the first category, all filters of class `AnnotationFilter::AnnotationFilter` can be used as they are used in `QFeatures`

For the second category, the package `DaparToolshed` provides filter functions based either on the adjacency matrix:

- `topnPeptides()`
- `sharedPeptides()`
- `specPeptides()`

Or based on the quantitative metadata (identification):

- `qMetacellWholeMatrix()`
- `qMetacellWholeLine()`
- `qMetacellOnConditions()`

**Author(s)**

Samuel Wiczorek

**Examples**

```
data("subR25prot")
data("subR25pept")
## -----
## Creating function filters
## -----

#FunctionFilter('FUN',
#               param1 = 'value_of_param1',
#               param2 = 'value_of_param2')

FunctionFilter('qMetacellWholeLine',
               cmd = 'delete',
               pattern = 'Missing MEC')

## -----
```

```

## Filter the last assay to keep only specific peptides. This filter
## only applies on peptide dataset.
## -----

spec.filter <- FunctionFilter('specPeptides', list())
## using a user-defined character filter
filterFeaturesOneSE(subR25pept, filters = list(FunctionFilter('specPeptides', list())))

## -----
## Filter the last assay to keep only specific peptides and topn
## peptides. The two filters are run sequentially.
## -----

lst.filters <- list(FunctionFilter('specPeptides', list()))
lst.filters <- append(lst.filters,
FunctionFilter('topnPeptides',
fun = 'rowSums',
top = 2))
filterFeaturesOneSE(subR25pept, filters = lst.filters)

## -----
## Filter the last assay to delete peptides where, in at least one
## condition, there is less than 80% of samples marked as 'imputed POV'
## -----

filter <- FunctionFilter('qMetacellOnConditions',
cmd = 'delete',
mode = 'AtLeastOneCond',
pattern = 'Missing POV',
conds = SummarizedExperiment::colData(subR25prot)$Condition,
percent = TRUE,
th = 0.8,
operator = '<')

filterFeaturesOneSE(subR25prot, filters = list(filter))

```

## Description

Utility functions to dela with QFeatures objects.

## Usage

```
last_assay(object)
```

```
n_assays_in_qf(object)

QFeaturesFromSE(
  obj.se,
  colData = data.frame(),
  metadata.qf = data.frame(),
  name = "myname"
)
```

**Arguments**

object	An instance of the class QFeatures
obj.se	An instance of the class QFeatures
colData	A data.frame() which contains data for the future object,
metadata.qf	A data.frame() which contains the metadata for the future object,
name	The name of the assay in the QFeatures object

**Value**

NA  
An instance of QFeatures class

**Examples**

```
NULL

# example code
```

---

ReplaceSpecialChars    *Standardize names*

---

**Description**

Replace ".", ' ', '-' in character() by '\_' to be compliant with functions of Shinyjs, Shiny

**Usage**

```
ReplaceSpecialChars(x)
```

**Arguments**

x	A character() to be processed
---	-------------------------------

**Value**

A character() of the same length as 'x' with modified names.

**Author(s)**

Samuel Wieczorek

**Examples**

```
ReplaceSpecialChars(c("foo.1", "foo-2", "foo 3"))
```

---

samLRT

*Computes a regularized version of the likelihood ratio statistic*

---

**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**

<code>lmm.res.h0</code>	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), <code>lmm.res.h0</code> 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.
<code>lmm.res.h1</code>	similar to <code>lmm.res.h0</code> , a vector of object containing the estimates (used to compute the statistic) under H1 for each protein.
<code>cc</code>	a list containing the indices of peptides and proteins belonging to each connected component.
<code>n</code>	the number of samples used in the test
<code>p</code>	the number of proteins in the experiment
<code>s1</code>	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

**Examples**

NULL

---

separateAdjPval	<i>Computes the adjusted p-values separately on contrast using CP4P</i>
-----------------	---

---

**Description**

Computes the adjusted p-values separately on contrast using CP4P

**Usage**

```
separateAdjPval(x, pval.threshold = 1.05, method = 1)
```

**Arguments**

x	a proteins x contrasts dataframe of (raw) p-values
pval.threshold	all the p-values above the threshold are not considered. Default is 1.05 (which is equivalent to have no threshold). Applying a threshold nearby 1 can be instrumental to improve the uniformity under the null, notably in case of upstream multiple contrast correction (for experienced users only)
method	a method to estimate pi_0, see CP4P

**Value**

a proteins x contrasts table of adjusted p-values

**Author(s)**

Thomas Burger

**Examples**

```
data(subR25prot)
obj <- subR25prot[seq_len(5),]
separateAdjPval(
  testAnovaModels(
    applyAnovasOnProteins(obj, 1), "TukeyHSD")$P_Value)
```

---

SetHistory	<i>Standardize names</i>
------------	--------------------------

---

**Description**

Standardize names

**Usage**

```
SetHistory(obj.se, history)
```

**Arguments**

obj.se	An instance of the class SummarizedExperiment
history	A data.frame()

**Value**

A data.frame()

**Author(s)**

Samuel Wiczorek

**Examples**

```
data(subR25prot)
history <- GetHistory(subR25prot[[1]])
history <- rbind(history, c('Example', 'Step Ex', 'ex_param', 'Ex'))
subR25prot[[1]] <- SetHistory(subR25prot[[1]], history)
```

---

splitAdjacencyMat	<i>splits an adjacency matrix into specific and shared</i>
-------------------	--

---

**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

**Examples**

```
data(subR25pept)
X <- BuildAdjacencyMatrix(subR25pept[[1]])
ll <- splitAdjacencyMat(X)
```

---

subR25pept

*subR25pept dataset*

---

**Description**

This dataset is a subset of the final outcome of a quantitative mass spectrometry-based proteomic analysis of two samples containing different concentrations of 48 human proteins (UPS1 standard from Sigma-Aldrich) within a constant yeast background (see Gai Gianetto et al. (2016) for details). It contains the abundance values of the different human and yeast proteins identified and quantified in these two conditions. The two conditions represent the measured abundances of peptides when respectively 5 fmol and 10 fmol of UPS1 human proteins were mixed with the yeast extract before mass spectrometry analyses. This results in a concentration ratio of 2. Three technical replicates were acquired for each condition.

The original dataset is available as a CSV file (see `inst/extdata/Exp1_R25_pept_100.txt`). In the latter case, the quantitative data are those of the raw intensities.

This dataset is a subset containing the first 500 peptides from the original dataset, which comes from: <https://doi.org/10.1002/pmic.201500189>

**Usage**

```
data(subR25pept)
```

**Format**

An object of class `QFeatures` related to proteins quantification. It contains 6 samples divided into two conditions (10fmol and 5fmol) and 500 peptides

The data frame `assay(subR25pept)` contains six columns that are the quantitation of peptides for the six replicates.

The data frame `fData(subR25pept)` contains the meta data about the peptides

The data frame `pData(subR25pept)` contains the experimental design and gives few information about the samples.

**Value**

An object of class QFeatures related to proteins quantification.

**References**

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

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

---

subR25prot

*subR25prot dataset*

---

**Description**

This dataset is a subset of the final outcome of a quantitative mass spectrometry-based proteomic analysis of two samples containing different concentrations of 48 human proteins (UPS1 standard from Sigma-Aldrich) within a constant yeast background (see Giai Gianetto et al. (2016) for details). It contains the abundance values of the different human and yeast proteins identified and quantified in these two conditions. The two conditions represent the measured abundances of proteins when respectively 5 fmol and 10 fmol of UPS1 human proteins were mixed with the yeast extract before mass spectrometry analyses. This results in a concentration ratio of 2. Three technical replicates were acquired for each condition.

The original dataset is available as a CSV file (see `inst/extdata/Exp1_R25_prot_100.txt`). In the latter case, the quantitative data are those of the raw intensities.

This dataset is a subset containing the first 100 proteins from the original dataset, which comes from: <https://doi.org/10.1002/pmic.201500189>

**Usage**

```
data(subR25prot)
```

**Format**

An object of class QFeatures related to proteins quantification. It contains 6 samples divided into two conditions (10fmol and 5fmol) and 100 proteins.

The data frame `assay(subR25prot)` contains six columns that are the quantitation of proteins for the six replicates.

The data frame `fData(subR25prot)` contains the meta data about the proteins.

The data frame `pData(subR25prot)` contains the experimental design and gives few informations about the samples.

**Value**

An object of class QFeatures related to proteins quantification.

**References**

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

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

---

SymFilteringOperators *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**

```
SymFilteringOperators()

qMetacellFilteringScope()

qMetacellWholeMatrix(
  object,
  cmd,
  pattern,
  percent = "Percentage",
  th,
  operator
)

qMetacellWholeLine(object, cmd, pattern)

qMetacellOnConditions(
  object,
  cmd,
  mode,
  pattern,
  conds,
  percent = "Percentage",
  operator,
  th
)
```



```
th <- 0.5
percent <- "Percentage"
mode <- "AllCond"
ind <- qMetacellOnConditions(obj, cmd, mode, pattern, conds, percent, op, th)

qMetacellFilteringScope()

data(subR25prot)
obj <- subR25prot[[1]]
```

---

testAnovaModels      *Applies a statistical test on each element of a list of linear models*

---

### Description

Applies a statistical test on each element of a list of linear models

### Usage

```
testAnovaModels(aov_fits, test = "Omnibus")
```

### Arguments

aov_fits	a list of linear models, such as those outputted by <code>applyAnovasOnProteins</code>
test	a character string among "Omnibus", "TukeyHSD", "TukeySinglestep", "TukeyStepwise", "TukeyNoMTC", "DunnettSinglestep", "DunnettStepwise" and "DunnettNoMTC". "Omnibus" tests the all-mean equality, the Tukey tests compares all pairs of means and the Dunnett tests compare all the means to the first one. For multiple tests (Dunnett's or Tukey's) it is possible to correct for multiplicity (either with single-step or step-wise FWER) or not. All the Tukey's and Dunnett's tests use the multcomp package expect for "TukeyHSD" which relies on the stats package. "TukeyHSD" and "TukeyStepwise" gives similar results.

### Value

a list of 2 tables (p-values and fold-changes, respectively)

### Author(s)

Thomas Burger

### Examples

```
data(subR25prot)
obj <- subR25prot[seq_len(5),]
testAnovaModels(applyAnovasOnProteins(obj, 1))
```

---

testDesign	<i>Check the validity of the design</i>
------------	---

---

**Description**

Check the validity of the design

**Usage**

```
testDesign(tab)
```

**Arguments**

tab                    A data.frame which correspond to the design

**Value**

A list of two items

**Author(s)**

Thomas Burger, Samuel Wieczorek

**Examples**

```
data(subR25pept)
testDesign(SummarizedExperiment::colData(subR25pept)[, -1])
```

---

thresholdpval4fdr	<i>Applies p-value adjustment using cp4p on p-values below a given threshold</i>
-------------------	--

---

**Description**

Applies p-value adjustment using cp4p on p-values below a given threshold

**Usage**

```
thresholdpval4fdr(x, pval.T, M)
```

**Arguments**

x                    vector of p-values  
pval.T                p-value threshold  
M                    a pi0.method from cp4p package

**Value**

adjusted p-values

**Author(s)**

Thomas Burger

**Examples**

NULL

---

translatedRandomBeta *Generator of simulated values*

---

**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)
```

---

wrapperCalibrationPlot

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

---

## Description

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

```
data(subR25prot)
obj <- subR25prot
# Simulate imputation
obj <- NAIsZero(obj, 1)
allComp <- limmaCompleteTest(
  SummarizedExperiment::assay(obj[[length(obj)]]),
  design_qf(obj),
  comp.type="OnevsOne")
wrapperCalibrationPlot(allComp$P_Value[, 1])
```

---

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

---

**Description**

Wrapper for One-way Anova statistical test

**Usage**

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

**Arguments**

<code>obj</code>	An object of class QFeatures.
<code>i</code>	An integer which is the index of the assay in the QFeatures object
<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

```
library(SummarizedExperiment)
data(subR25prot)
obj <- subR25prot
filter <- FunctionFilter('qMetacellOnConditions',
  cmd = 'delete',
  mode = 'AtLeastOneCond',
  pattern = c("Missing POV", "Missing MEC"),
  conds = design_qf(obj)$Condition,
  percent = TRUE,
  th = 0.8,
  operator = '>')
obj <- filterFeaturesOneSE(obj, name = "Filtered", filters = list(filter))
anovatest <- wrapperClassic1wayAnova(obj, 2)
```

---

wrapperDaparImputeMI *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 `SummarizedExperiment`.

## Usage

```
wrapperDaparImputeMI(
  obj,
  design,
  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

<code>obj</code>	An object of class <code>SummarizedExperiment</code> .
<code>design</code>	A <code>data.frame()</code> for the design of the dataset
<code>nb.iter</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>nknn</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>selec</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>siz</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>weight</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>ind.comp</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>progress.bar</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>x.step.mod</code>	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
<code>x.step.pi</code>	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
<code>nb.rei</code>	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
<code>method</code>	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
<code>gridsize</code>	Same as the function <code>estim.mix</code> in the package <code>imp4p</code>
<code>q</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>q.min</code>	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>
<code>q.norm</code>	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>
<code>eps</code>	Same as the function <code>impute.pa</code> in the package <code>imp4p</code>
<code>methodi</code>	Same as the function <code>mi.mix</code> in the package <code>imp4p</code>
<code>lapala</code>	A boolean
<code>distribution</code>	The type of distribution used. Values are <code>unif</code> (default) or <code>beta</code> .

### Value

A matrix with imputed values instead of missing values.

### Author(s)

Samuel Wieczorek

### Examples

```

utils::data(subR25pept)
design <- design_qf(subR25pept)
obj.imp.na <- wrapperDaparImputeMI(subR25pept[[1]], design, nb.iter = 1, lapala = TRUE)
obj.imp.pov <- wrapperDaparImputeMI(subR25pept[[1]], design, nb.iter = 1, lapala = FALSE)

```

---

wrapperImputeMLE	<i>Imputation of peptides having no values in a biological condition.</i>
------------------	---

---

## Description

This method is a wrapper to the function `impute.mle()` of the package `imp4p` adapted to an object of class `SummarizedExperiment`. It does not impute MEC missing values.

## Usage

```
wrapperImputeMLE(obj, grp)
```

## Arguments

<code>obj</code>	An object of class <code>SummarizedExperiment</code> .
<code>grp</code>	A vector of conditions in the dataset.

## Value

The `SummarizedExperiment::assay(obj)` matrix with imputed values instead of missing values.

## Author(s)

Samuel Wieczorek

## Examples

```
utils::data(subR25pept)
level <- 'peptide'
# Delete whole empty lines
metacell.mask <- DaparToolshed::matchMetacell(
  qMetacell(subR25pept[[1]]),
  c("Missing POV", "Missing MEC"), level)
indices <- GetIndices_WholeMatrix(metacell.mask, op = ">=", th = 1)
grp <- design_qf(subR25pept)$Condition
subR25pept <- wrapperImputeMLE(subR25pept[[1]], grp)
```

---

wrapperImputePA2      *Missing values imputation from a SummarizedExperiment object*

---

### Description

This method is a wrapper to the function `imputePA2()` adapted to objects of class `SummarizedExperiment`.

### Usage

```
wrapperImputePA2(  
  obj,  
  design,  
  q.min = 0,  
  q.norm = 3,  
  eps = 0,  
  distribution = "unif"  
)
```

### Arguments

<code>obj</code>	An object of class <code>SummarizedExperiment</code> .
<code>design</code>	A <code>data.frame</code> containing the columns "quantCols" corresponding to the samples name and "Condition" to the condition of each sample.
<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> ).
<code>q.norm</code>	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 <code>sd</code> is the standard deviation of a row in a condition).
<code>eps</code>	A value 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.
<code>distribution</code>	The type of distribution used. Values are <code>unif</code> (default) or <code>beta</code> .

### Value

The object `obj` which has been imputed

### Author(s)

Thomas Burger, Samuel Wieczorek

**Examples**

```
NULL
```

---

wrapperPirat	<i>Missing values imputation using Pirat</i>
--------------	--

---

**Description**

This method is a wrapper to the function `pipeline_llkimpute()` of the package `Pirat` adapted to an object of class `QFeatures` of `SummarizedExperiment`.

**Usage**

```
wrapperPirat(data, adjmat, rnas_ab = NULL, adj_rna_pg = NULL, ...)
```

**Arguments**

<code>data</code>	An object of class <code>QFeatures</code> or <code>SummarizedExperiment</code> . If <code>data</code> is of class <code>QFeatures</code> , the last assay will be imputed.
<code>adjmat</code>	Adjacency matrix corresponding to the <code>SummarizedExperiment</code> or the last assay of <code>QFeatures</code> .
<code>rnas_ab</code>	Transcriptomic data with sample as row, used only if <code>extension = 'T'</code> .
<code>adj_rna_pg</code>	Adjacency matrix of rna (rows) and peptides or precursors (columns), used only if <code>extension = 'T'</code> .
<code>...</code>	Additional arguments to pass to <code>my_pipeline_llkimpute()</code>

**Value**

`QFeatures` including a new assay with imputed data or `SummarizedExperiment` with imputed data.

**Author(s)**

Manon Gaudin

**Examples**

```
data(subR25pept)

# Delete whole empty lines
filter_emptyline <- FunctionFilter("qMetacellWholeLine", cmd = 'delete', pattern = 'Missing MEC')
subR25pept <- filterFeaturesOneSE(object = subR25pept, i = length(subR25pept), name = "Filtered",
  filters = list(filter_emptyline))

subR25pept <- wrapperPirat(data = subR25pept,
  adjmat = SummarizedExperiment::rowData(subR25pept[[length(subR25pept)]])$adjacencyMatrix,
  extension = "base")
```

# Index

- \* **datasets**
  - subR25pept, [104](#)
  - subR25prot, [105](#)
- \* **data**
  - subR25pept, [104](#)
  - subR25prot, [105](#)
- Add2History, [5](#)
- Add\_Aggregated\_rowData
  - (DaparToolshed-aggregate), [19](#)
- Add\_Item\_to\_Dataset, [5](#)
- addColors (QFeatures-excel), [95](#)
- adjacency-matrix-filter, [6](#)
- AdjMatFilters
  - (adjacency-matrix-filter), [6](#)
- aggQmetacell (DaparToolshed-aggregate), [19](#)
- aggregateFeatures4Prostar
  - (DaparToolshed-aggregate), [19](#)
- aggregateFeatures4Prostar, QFeatures-method
  - (DaparToolshed-aggregate), [19](#)
- aggregateFeatures4Prostar, SummarizedExperiment-method (paramshistory), [80](#)
  - (DaparToolshed-aggregate), [19](#)
- aggregateMethods
  - (DaparToolshed-aggregate), [19](#)
- aggregateRedistribution
  - (DaparToolshed-aggregateRedistribution), [28](#)
- aggregateRedistribution, QFeatures-method
  - (DaparToolshed-aggregateRedistribution), [28](#)
- aggregateRedistribution, SummarizedExperiment-method
  - (DaparToolshed-aggregateRedistribution), [28](#)
- allPeptides (adjacency-matrix-filter), [6](#)
- analysis (paramshistory), [80](#)
- analysis, QFeatures-method
  - (paramshistory), [80](#)
- analysis, SummarizedExperiment-method
  - (paramshistory), [80](#)
- analysis<- (paramshistory), [80](#)
- AnnotationFilter::AnnotationFilter, [98](#)
- applyAnovasOnProteins, [8](#)
- BuildAdjacencyMatrix, [9](#)
- BuildColumnToProteinDataset
  - (DaparToolshed-aggregate), [19](#)
- buildGraph, [9](#)
- BuildMetacell, [10](#)
- checkConditions, [11](#)
- checkDesign, [12](#)
- Children, [12](#)
- classic1wayAnova, [13](#)
- CleanRowData, [14](#)
- compareNormalizationD\_HC, [15](#)
- compute\_t\_tests, [16](#)
- ConnectedComp (paramshistory), [80](#)
- ConnectedComp, QFeatures-method
  - (paramshistory), [80](#)
- ConnectedComp, SummarizedExperiment-method
  - (paramshistory), [80](#)
- ConnectedComp<- (paramshistory), [80](#)
- ConvertListToHtml, [17](#)
- CountPep (DaparToolshed-aggregate), [19](#)
- createQFeatures, [18](#)
- custom\_metacell\_colors (q\_metacell), [92](#)
- DaparToolshed,
  - (QFeatures-filtering-oneSE), [97](#)
  - DaparToolshed-aggregate, [19](#)
  - DaparToolshed-aggregateRedistribution, [28](#)
  - design\_qf (paramshistory), [80](#)
  - design\_qf, QFeatures-method
    - (paramshistory), [80](#)
  - design\_qf<- (paramshistory), [80](#)
  - diffAnaComputeAdjustedPValues, [30](#)
  - diffAnaComputeFDR, [31](#)
  - diffAnaVolcanoplot\_rCharts, [31](#)

- DifferentialAnalysis (paramshistory), 80
- DifferentialAnalysis, QFeatures-method (paramshistory), 80
- DifferentialAnalysis, SummarizedExperiment-method (paramshistory), 80
- DifferentialAnalysis<- (paramshistory), 80
- displayCCvisNet, 33
- ExtendPalette, 34
- ExtractUniquePeptides (DaparToolshed-aggregate), 19
- filename (paramshistory), 80
- filename, QFeatures-method (paramshistory), 80
- filename<- (paramshistory), 80
- filterFeaturesOneSE (QFeatures-filtering-oneSE), 97
- filterFeaturesOneSE, (QFeatures-filtering-oneSE), 97
- filterFeaturesOneSE, QFeatures-method (QFeatures-filtering-oneSE), 97
- findMECBlock (mv\_imputation\_protein), 70
- formatHSDResults, 35
- formatLimmaResult, 35
- formatPHResults, 36
- formatPHTResults, 37
- fudge2LRT, 38
- FunctionFilter (QFeatures-filtering-oneSE), 97
- FunctionFilter, (QFeatures-filtering-oneSE), 97
- FunctionFilter-class (QFeatures-filtering-oneSE), 97
- GetColorsForConditions, 39
- getDesignLevel, 40
- GetDetailedNbPeptides (DaparToolshed-aggregate), 19
- GetDetailedNbPeptidesUsed (DaparToolshed-aggregate), 19
- GetHistory, 41
- GetIndices\_BasedOnConditions, 41
- GetIndices\_FunFiltering, 42
- GetIndices\_WholeLine, 44
- GetIndices\_WholeMatrix, 44
- getListNbValuesInLines, 45
- GetMetacellTags (paramshistory), 80
- GetMetacellTags, data.frame-method (paramshistory), 80
- GetMetacellTags, QFeatures-method (paramshistory), 80
- GetMetacellTags, SummarizedExperiment-method (paramshistory), 80
- GetMetadataSlot (paramshistory), 80
- GetNbPeptidesUsed (DaparToolshed-aggregate), 19
- GetNbTags, 46
- getNumberOfEmptyLines, 46
- getPepProtCC, 47
- getProteinsStats (DaparToolshed-aggregate), 19
- getQuantile4Imp (mv\_imputation\_protein), 70
- GetRowdataSlot (paramshistory), 80
- GetSoftAvailables, 48
- GetUniqueTags (paramshistory), 80
- GetUniqueTags, QFeatures-method (paramshistory), 80
- GetUniqueTags, SummarizedExperiment-method (paramshistory), 80
- globalAdjPval, 48
- GlobalQuantileAlignment (normalization\_methods), 74
- GraphPepProt (DaparToolshed-aggregate), 19
- hc\_logFC\_DensityPlot, 49
- hc\_mvTypePlot2 (metacell-plots), 64
- heatmapForMissingValues, 50
- histPValue\_HC, 51
- HypothesisTest (paramshistory), 80
- HypothesisTest, QFeatures-method (paramshistory), 80
- HypothesisTest, SummarizedExperiment-method (paramshistory), 80
- HypothesisTest<- (paramshistory), 80
- idcol (paramshistory), 80
- idcol, QFeatures-method (paramshistory), 80
- idcol, SummarizedExperiment-method (paramshistory), 80
- idcol<- (paramshistory), 80
- imputePA2, 52
- InitializeHistory, 53

- innerAggregateIter
  - (DaparToolshed-aggregate), 19
- innerMean (DaparToolshed-aggregate), 19
- innerMedian (DaparToolshed-aggregate), 19
- innerMedianpolish
  - (DaparToolshed-aggregate), 19
- innerRobustsummary
  - (DaparToolshed-aggregate), 19
- innerSum (DaparToolshed-aggregate), 19
- isDifferential, 53
- isOfType, 54
- isSubset, 55
  
- Keep\_Items\_from\_Dataset, 56
  
- last\_assay (QFeatures-utils), 99
- LH0, 56
- LH0\_lm, 57
- LH1, 58
- LH1\_lm, 58
- limmaCompleteTest, 59
- listSheets (output\_2\_Excel), 77
- LOESS (normalization\_methods), 74
  
- mainAssay (paramshistory), 80
- makeContrast, 60
- makeDesign, 61
- makeDesign1, 61
- makeDesign2, 62
- makeDesign3, 63
- matchMetacell, 63
- Matrix::sparseMatrix(), 84
- MeanCentering (normalization\_methods), 74
- metacell-plots, 64
- metacell\_agg (DaparToolshed-aggregate), 19
- Metacell\_DIA\_NN, 67
- Metacell\_generic (q\_metacell), 92
- Metacell\_maxquant, 68
- Metacell\_proline, 69
- metacellDef (q\_metacell), 92
- MetacellFilteringScope, 70
- metacellHisto\_HC (metacell-plots), 64
- metacellPerLinesHisto\_HC
  - (metacell-plots), 64
- metacellPerLinesHistoPerCondition\_HC
  - (metacell-plots), 64
  
- metacombine (q\_metacell), 92
- mv\_imputation\_protein, 70
- mvImage (metacell-plots), 64
  
- n\_assays\_in\_qf (QFeatures-utils), 99
- NAIsZero, 72
- names\_metacell (paramshistory), 80
- names\_metacell, QFeatures-method
  - (paramshistory), 80
- names\_metacell, SummarizedExperiment-method
  - (paramshistory), 80
- names\_metacell<- (paramshistory), 80
- nEmptyLines, 73
- nonzero, 73
- normalization\_methods, 74
- normalizeFunction, 76
- normalizeMethods
  - (normalization\_methods), 74
- normalizeMethods(), 76
  
- output\_2\_Excel, 77
- OWAnova, 79
  
- paramshistory, 80
- paramshistory, QFeatures-method
  - (paramshistory), 80
- paramshistory, SummarizedExperiment-method
  - (paramshistory), 80
- paramshistory<- (paramshistory), 80
- Parent, 84
- parentProtId (paramshistory), 80
- parentProtId, QFeatures-method
  - (paramshistory), 80
- parentProtId, SummarizedExperiment-method
  - (paramshistory), 80
- parentProtId<- (paramshistory), 80
- pepaTest, 85
- Pipelines, 86
- pkgsRequire, 86
- plotCompareAssays, 87
- plotJitter, 88
- plotJitter\_rCharts, 89
- postHocTest, 89
- pushpvalue, 91
  
- q\_metacell, 92
- QFeatures-accessors (paramshistory), 80
- QFeatures-excel, 95
- QFeatures-filtering-oneSE, 97

- QFeatures-utils, 99
- QFeatures::aggregateFeatures(), 23
- QFeatures::QFeatures, 22, 23
- QFeaturesFromSE (QFeatures-utils), 99
- qMetacell (paramshistory), 80
- qMetacell, QFeatures-method  
(paramshistory), 80
- qMetacell, SummarizedExperiment-method  
(paramshistory), 80
- qMetacell-filter  
(SymFilteringOperators), 106
- qMetacell<- (paramshistory), 80
- qMetacellFilteringScope  
(SymFilteringOperators), 106
- qMetacellOnConditions  
(SymFilteringOperators), 106
- qMetacellOnConditions(), 98
- qMetacellWholeLine  
(SymFilteringOperators), 106
- qMetacellWholeLine(), 98
- qMetacellWholeMatrix  
(SymFilteringOperators), 106
- qMetacellWholeMatrix(), 98
- QuantileCentering  
(normalization\_methods), 74
  
- readExcel (output\_2\_Excel), 77
- reIntroduceMEC (mv\_imputation\_protein),  
70
- ReplaceSpecialChars, 100
- RunAggregation  
(DaparToolshed-aggregate), 19
  
- samLRT, 101
- searchMetacellTags (q\_metacell), 92
- select\_topn (DaparToolshed-aggregate),  
19
- separateAdjPval, 102
- Set\_POV\_MEC\_tags (q\_metacell), 92
- Set\_POV\_MEC\_tags2 (q\_metacell), 92
- SetHistory, 103
- sharedPeptides  
(adjacency-matrix-filter), 6
- sharedPeptides(), 98
- specPeptides (adjacency-matrix-filter),  
6
- specPeptides(), 98
- splitAdjacencyMat, 103
  
- subAdjMat\_sharedPeptides  
(adjacency-matrix-filter), 6
- subAdjMat\_specificPeptides  
(adjacency-matrix-filter), 6
- subAdjMat\_topnPeptides  
(adjacency-matrix-filter), 6
- subR25pept, 104
- subR25prot, 105
- SumByColumns (normalization\_methods), 74
- SummarizedExperiment::SummarizedExperiment,  
23
- SymFilteringOperators, 106
  
- testAnovaModels, 108
- testDesign, 109
- thresholdpval4fdr, 109
- topnFunctions  
(adjacency-matrix-filter), 6
- topnPeptides (adjacency-matrix-filter),  
6
- topnPeptides(), 98
- translatedRandomBeta, 110
- typeDataset (paramshistory), 80
- typeDataset, QFeatures-method  
(paramshistory), 80
- typeDataset, SummarizedExperiment-method  
(paramshistory), 80
- typeDataset<- (paramshistory), 80
  
- UpdateMetacellAfterImputation  
(q\_metacell), 92
- UpdateMetacellAfterImputation, SummarizedExperiment-method  
(q\_metacell), 92
  
- VariableFilter  
(QFeatures-filtering-oneSE), 97
- version (paramshistory), 80
- version, QFeatures-method  
(paramshistory), 80
- version<- (paramshistory), 80
- vsn (normalization\_methods), 74
  
- wrapperCalibrationPlot, 111
- wrapperClassic1wayAnova, 112
- wrapperDaparImputeMI, 113
- wrapperImputeDetQuant  
(mv\_imputation\_protein), 70
- wrapperImputeFixedValue  
(mv\_imputation\_protein), 70

wrapperImputeKNN  
    (mv\_imputation\_protein), 70

wrapperImputeMLE, 115

wrapperImputePA  
    (mv\_imputation\_protein), 70

wrapperImputePA2, 116

wrapperImputeSLSA  
    (mv\_imputation\_protein), 70

wrapperMImage (metacell-plots), 64

wrapperPirat, 117

write2excel (QFeatures-excel), 95

write2excel, QFeatures-method  
    (QFeatures-excel), 95

write2excel, SummarizedExperiment-method  
    (QFeatures-excel), 95

write2excelSE (QFeatures-excel), 95

write\_Assay\_To\_Excel (output\_2\_Excel),  
    77

Write\_RowData (output\_2\_Excel), 77

Write\_SamplesData\_to\_Excel  
    (output\_2\_Excel), 77

writeExcel (output\_2\_Excel), 77

WriteHistory (output\_2\_Excel), 77