Package 'QFeatures'

October 18, 2022

Title Quantitative features for mass spectrometry data

Version 1.6.0

Description The QFeatures infrastructure enables the management and processing of quantitative features for high-throughput mass spectrometry assays. It provides a familiar Bioconductor user experience to manages quantitative data across different assay levels (such as peptide spectrum matches, peptides and proteins) in a coherent and tractable format.

Depends R (>= 4.0), MultiAssayExperiment

Imports methods, stats, utils, S4Vectors, IRanges, SummarizedExperiment, BiocGenerics, ProtGenerics (>= 1.19.3), AnnotationFilter, lazyeval, Biobase, MsCoreUtils (>= 1.1.2), igraph, plotly

Suggests SingleCellExperiment, HDF5Array, msdata, ggplot2, gplots, dplyr, limma, magrittr, DT, shiny, shinydashboard, testthat, knitr, BiocStyle, rmarkdown, vsn, preprocessCore, matrixStats, imputeLCMD, pcaMethods, impute, norm, ComplexHeatmap

biocViews Infrastructure, MassSpectrometry, Proteomics, Metabolomics

License Artistic-2.0

Encoding UTF-8

LazyData true

VignetteBuilder knitr

BugReports https://github.com/RforMassSpectrometry/QFeatures/issues

URL https://github.com/RforMassSpectrometry/QFeatures

Collate 'AllGenerics.R' 'AssayLinks.R' 'QFeatures-class.R' 'QFeatures-constructors.R' 'QFeatures-functions.R' 'QFeatures-processing.R' 'QFeatures-filter.R' 'QFeatures-missing-data.R' 'QFeatures-aggregation.R' 'QFeatures-imputation.R' 'QFeatures-join.R' 'QFeatures-validity.R' 'SummarizedExperiment-methods.R' 'subsetBy-methods.R' 'data.R' 'reduce.R' 'utils.R' 'display.R'

Roxygen list(markdown=TRUE)

RoxygenNote 7.1.2

git_url https://git.bioconductor.org/packages/QFeatures

git_branch RELEASE_3_15

git_last_commit b79a83f

git_last_commit_date 2022-04-26

Date/Publication 2022-10-18

Author Laurent Gatto [aut, cre] (<https://orcid.org/0000-0002-1520-2268>),
Christophe Vanderaa [aut] (<https://orcid.org/0000-0001-7443-5427>)

Maintainer Laurent Gatto <laurent.gatto@uclouvain.be>

R topics documented:

`aggregateFeatures` *Aggregate an assay's quantitative features*

Description

This function aggregates the quantitative features of an assay, applying a summarisation function (`fun`) to sets of features as defined by the `fcol` feature variable. The new assay's features will be named based on the unique `fcol` values.

In addition to the results of the aggregation, the newly aggregated `SummarizedExperiment` assay also contains a new `aggcounts` assay containing the aggregation counts matrix, i.e. the number of features that were aggregated, which can be accessed with the `aggcounts()` accessor.

The rowData of the aggregated SummarizedExperiment assay contains a .n variable that provides the number of features that were aggregated. This .n value is always >= that the sample-level aggcounts.

Usage

```
## S4 method for signature 'QFeatures'
aggregateFeatures(
  object,
  i,
  fcol,
  name = "newAssay",
  fun = MsCoreUtils::robustSummary,
  ...
)

## S4 method for signature 'SummarizedExperiment'
aggregateFeatures(object, fcol, fun = MsCoreUtils::robustSummary, ...)

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

Arguments

object	An instance of class QFeatures or SummarizedExperiment.
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 either a character of 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.
...	Additional parameters passed the fun.

Details

Aggregation is performed by a function that takes a matrix as input and returns a vector of length equal to ncol(x). Examples thereof are

- MsCoreUtils::medianPolish() to fits an additive model (two way decomposition) using Tukey's median polish_ procedure using stats::medpolish();
- MsCoreUtils::robustSummary() to calculate a robust aggregation using MASS::rlm() (default);
- base::colMeans() to use the mean of each column;
- matrixStats::colMedians() to use the median of each column.
- base::colSums() to use the sum of each column;

Value

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

Missing quantitative values

Missing quantitative values have different effect based on the aggregation method employed:

- The aggregation functions should be able to deal with missing values by either ignoring them, and propagating them. This is often done with an na.rm argument, that can be passed with .... For example, rowSums, rowMeans, rowMedians, ... will ignore NA values with na.rm = TRUE, as illustrated below.
- Missing values will result in an error when using medpolish, unless na.rm = TRUE is used. Note that this option relies on implicit assumptions and/or performes an implicit imputation: when summing, the values are implicitly imputed by 0, assuming that the NA represent a trully absent features; when averaging, the assumption is that the NA represented a genuinely missing value.
- When using robust summarisation, individual missing values are excluded prior to fitting the linear model by robust regression. To remove all values in the feature containing the missing values, use filterNA().

More generally, missing values often need dedicated handling such as filtering (see filterNA()) or imputation (see impute()).

Missing values in the row data

Missing values in the row data of an assay will also impact the resulting (aggregated) assay row data, as illustrated in the example below. Any feature variables (a column in the row data) containing NA values will be dropped from the aggregated row data. The reasons underlying this drop are detailed in the reduceDataFrame() manual page: only invariant aggregated rows, i.e. rows resulting from the aggregation from identical variables, are preserved during aggregations.

The situation illustrated below should however only happen in rare cases and should often be imputable using the value of the other aggregation rows before aggregation to preserve the invariant nature of that column. In cases where an NA is present in an otherwise variant column, the column would be dropped anyway.

Using an adjacency matrix

When considering non-unique peptides, i.e. peptides that map to multiple proteins, it is necessary to encode this ambiguity explicitly using a peptide-by-proteins adjacency matrix. This matrix is typically stored in the rowdata, is conventionally named "adjacencyMatrix" and can be retrieved with adjacencyMatrix(). It can be created manually (as illustrated below) or using PSMatch::makeAdjacencyMatrix().

See Also

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

Examples

```
## ---------------------------------------
## An example QFeatures with PSM-level data
## ---------------------------------------
data(feat1)
feat1

## Aggregate PSMs into peptides
feat1 <- aggregateFeatures(feat1, "psms", "Sequence", name = "peptides")
feat1

## Aggregate peptides into proteins
feat1 <- aggregateFeatures(feat1, "peptides", "Protein", name = "proteins")
feat1

assay(feat1[[1]])
assay(feat1[[2]])
aggcounts(feat1[[2]])
assay(feat1[[3]])
aggcounts(feat1[[3]])

## -------------------------------------------
## Aggregation with missing quantitative values
## -------------------------------------------
data(ft_na)
ft_na

assay(ft_na[[1]])
rowData(ft_na[[1]])

## By default, missing values are propagated
ft2 <- aggregateFeatures(ft_na, 1, fcol = "X", fun = colSums)
assay(ft2[[2]])
aggcounts(ft2[[2]])

## The rowData .n variable tallies number of initial rows that
## were aggregated (irrespective of NAs) for all the samples.
rowData(ft2[[2]])

## Ignored when setting na.rm = TRUE
ft3 <- aggregateFeatures(ft_na, 1, fcol = "X", fun = colSums, na.rm = TRUE)
assay(ft3[[2]])
aggcounts(ft3[[2]])

## -----------------------------------------------
## Aggregation with missing values in the row data
## -----------------------------------------------
## Row data results without any NAs, which includes the
## Y variables
rowData(ft2[[2]])

## Missing value in the Y feature variable
```

```
rowData(ft_na[[1]])[1, "Y"] <- NA
rowData(ft_na[[1]])

ft3 <- aggregateFeatures(ft_na, 1, fcol = "X", fun = colSums)
## The Y feature variable has been dropped!
assay(ft3[[2]])
rowData(ft3[[2]])

## -----------------------------------------
## Using a peptide-by-proteins adjacency matrix
## -----------------------------------------

## Let's use assay peptides from object feat1 and
## define that peptide SYGFNAAR maps to proteins
## Prot A and B

se <- feat1[["peptides"]]
rowData(se)$Protein[3] <- c("ProtA;ProtB")
rowData(se)

## Manual encoding of the adjacency matrix
adj <- matrix(0, nrow = 3, ncol = 2,
              dimnames = list(rownames(se),
                              c("ProtA", "ProtB")))
adj[1, 1] <- adj[2, 2] <- adj[3, 1:2] <- 1
adj

rowData(se)$adjacencyMatrix <- adj
rowData(se)
adjacencyMatrix(se)
```

`AllGenerics` *Placeholder for generics functions documentation*

Description

Placeholder for generics functions documentation

`AssayLinks` *Links between Assays*

Description

Links between assays within a QFeatures object are handled by an `AssayLinks` object. It is composed by a list of `AssayLink` instances.

Usage

```
## S4 method for signature 'AssayLink'
show(object)

## S4 method for signature 'AssayLinks'
updateObject(object, ..., verbose = FALSE)

## S4 method for signature 'AssayLink'
updateObject(object, ..., verbose = FALSE)

AssayLink(name, from = NA_character_, fcol = NA_character_, hits = Hits())

AssayLinks(..., names = NULL)

assayLink(x, i)

assayLinks(x, i)

## S4 method for signature 'AssayLink,character,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'AssayLinks,list,ANY,ANY'
x[i, j, ..., drop = TRUE]

addAssayLink(object, from, to, varFrom, varTo)

addAssayLinkOneToOne(object, from, to)
```

Arguments

object	An AssayLink object to show.
...	A set of AssayLink objects or a list thereof.
verbose	logical (default FALSE) whether to print extra messages
name	A mandatory name of the assay(s).
from	A character() or integer() indicating which assay(s) to link from in object
fcol	The feature variable of the parent assay used to generate the current assay (used in aggregateFeatures). NA_character_, if not applicable.
hits	An object of class S4Vectors::Hits matching the features of two assays.
names	A character() of AssayLink names. If provided, ... are ignored, and names is used to create an AssayLinks object with AssayLink instances with names names.
x	An instance of class QFeatures.
i	The index or name of the assay whose AssayLink and parents AssayLink instances are to be returned. For [, the feature names to filter on.
j	ignored.

drop	ignored.
to	A character(1) or integer(1) indicating which assay to link to in object
varFrom	A character() indicating the feature variable(s) to use to match the from assay(s) to the to assay. varFrom must have the same length as from and is assumed to be ordered as from.
varTo	A character(1) indicating the feature variable to use to match the to assay to the from assay(s).

Value

assayLink returns an instance of class AssayLink.

assayLinks returns an instance of class AssayLinks.

Constructors

Object can be created with the AssayLink() and AssayLinks() constructors.

Methods and functions

- assayLink(x, i) accesses the AssayLink at position i or with name i in the QFeatures object x.
- parentAssayLinks(x, i, recursive = FALSE) accesses the parent(s) AssayLinks or assay with index or name i.

Creating links between assays

- addAssayLink takes a parent assay and a child assay contained in the QFeatures object and creates a link given a matching feature variable in each assay's rowData. addAssayLink also allows to link an assay from multiple parent assays (see Examples).
- addAssayLinkOneToOne links two assays contained in the QFeatures object. The parent assay and the child assay must have the same size and contain the same rownames (a different ordering is allowed). The matching is performed based on the row names of the assays, instead of a supplied variable name in rowData. Providing multiple parents is not supported.

Examples

```
##-----------------------------
## Creating an AssayLink object
##-----------------------------

al1 <- AssayLink(name = "assay1")
al1

##-----------------------------
## Creating an AssayLinks object
##-----------------------------

AssayLinks(al1)
```

```
al2 <- AssayLinks(names = c("Assay1", "Assay2"))
al2

##---------------------------------------
## Adding an AssayLink between two assays
##---------------------------------------

## create a QFeatures object with 2 (identical) assays
## see also '?QFeatures'
se <- SummarizedExperiment(matrix(runif(20), ncol = 2,
                                  dimnames = list(LETTERS[1:10],
                                                  letters[1:2])),
                           rowData = DataFrame(ID = 1:10))
ft <- QFeatures(list(assay1 = se, assay2 = se))

## assay1 and assay2 are not linked
assayLink(ft, "assay2") ## 'from' is NA
assayLink(ft, "assay1") ## 'from' is NA

## Suppose assay2 was generated from assay1 and the feature variable
## 'ID' keeps track of the relationship between the two assays
ftLinked <- addAssayLink(ft, from = "assay1", to = "assay2",
                         varFrom = "ID", varTo = "ID")
assayLink(ftLinked, "assay2")

## For one-to-one relationships, you can also use
ftLinked <- addAssayLinkOneToOne(ft, from = "assay1", to = "assay2")
assayLink(ftLinked, "assay2")

##---------------------------------------
## Adding an AssayLink between more assays
##---------------------------------------

## An assay can also be linked to multiple parent assays
## Create a QFeatures object with 2 parent assays and 1 child assay
ft <- QFeatures(list(parent1 = se[1:6, ], parent2 = se[4:10, ], child = se))
ft <- addAssayLink(ft, from = c("parent1", "parent2"), to = "child",
                   varFrom = c("ID", "ID"), varTo = "ID")
assayLink(ft, "child")
```

`countUniqueFeatures` *Count Unique Features This function counts the number of unique features per sample. A grouping structure can be provided to count higher level features from assays, for example counting the number of unique proteins from PSM data.*

Description

Count Unique Features

This function counts the number of unique features per sample. A grouping structure can be provided to count higher level features from assays, for example counting the number of unique proteins from PSM data.

Usage

```
countUniqueFeatures(object, i, groupBy = NULL, colDataName = "count")
```

Arguments

object	An object of class QFeatures.
i	A numeric() or character() vector indicating from which assays the rowData should be taken.
groupBy	A character(1) indicating the variable name in the rowData that contains the grouping variable, for instance to count the unique number of peptides or proteins expressed in each samples (column). If groupBy is missing, the number of non zero elements per sample will be stored.
colDataName	A character(1) giving the name of the new variable in the colData where the number of unique features will be stored. The name cannot already exist in the colData.

Value

An object of class QFeatures.

Examples

```
data("ft_na")
## Count number of (non-missing) PSMs
ft_na <- countUniqueFeatures(ft_na,
                             i = "na",
                             colDataName = "counts")
ft_na$counts
## Count number of unique rowData feature
ft_na <- countUniqueFeatures(ft_na,
                             i = "na",
                             groupBy = "Y",
                             colDataName = "Y_counts")
ft_na$Y_counts
```

---

display *Interactive MultiAssayExperiment Explorer*

Description

A shiny app to browser and explore the assays in an `MultiAssayExperiment` object. Each assay can be selected from the dropdown meny in the side panel, and the quantitative data and row metadata are displayed in the respective *Assay* and *Row data* tabs. The *Heatmap* tab displays a heatmap of the assay. The selection of rows in the *Row data* table is used to subset the features displayed in the *Assay* table and the heatmap to those currectly selected. See QFeatures for an example.

Usage

```
display(object, n = 100, ...)
```

Arguments

`object`	An instance inheriting from `MultiAssayExperiment`.
`n`	A `numeric(1)` indicating the maximum number of features (rows) to consider before disabling row clustering and displaying feature names for speed purposes. Default is 100.
`...`	Additional parameters (other than `Rowv` and `labRow`, which are set internally based on the value of `n`) passed to heatmap.

Value

Used for its side effect.

Author(s)

Laurent Gatto

Examples

```
## Not run:
data(feat2)
display(feat2)

## End(Not run)
```

---

feat1 *Feature example data*

Description

`feat1` is a small test `QFeatures` object for testing and demonstration. `feat2` is used to demonstrate assay joins. `ft_na` is a tiny test set that contains missing values used to demonstrate and test the impact of missing values on data processing. `se_na2` is an `SummarizedExperiment` with missing values of mixed origin.

Usage

```
feat1
```

Format

An object of class `QFeatures` of length 1.

`feat3` *Example* `QFeatures` *object after processing*

Description

`feat3` is a small `QFeatures` object that contains 7 assays: `psms1`, `psms2`, `psmsall`, `peptides`, `proteins`, `normpeptides`, `normproteins`. The dataset contains example data that could be obtained after running a simple processing pipeline. You can use it to get your hands on manipulating `AssayLinks` since all 3 general cases are present:

- One parent to one child `AssayLink`: the relationship can either be one row to one row (e.g. "peptides" to "normpeptides") or multiple rows to one row (e.g. "peptides" to "proteins").
- One parent to multiple children `AssayLink`: for instance "peptides" to "normpeptides" and "proteins".
- Multiple parents to one child `AssayLink`: links the rows between multiple assays to a single assays where some rows in different parent assays may point to the same row in the child assay. E.g. "psms1" and "psms2" to "psmsall"

Usage

```
feat3
```

Format

An object of class `QFeatures` of length 7.

Source

`feat3` was built from `feat1`. The source code is available in `inst/scripts/test_data.R`

See Also

See `?feat1` for other example/test data sets.

Examples

```
data("feat3")
plot(feat3)
```

`hlpsms` *hyperLOPIT PSM-level expression data*

Description

A `data.frame` with PSM-level quantitation data by Christoforou *et al.* (2016). This is the first replicate of a spatial proteomics dataset from a hyperLOPIT experimental design on Mouse E14TG2a embryonic stem cells. Normalised intensities for proteins for TMT 10-plex labelled fractions are available for 3 replicates acquired in MS3 mode using an Orbitrap Fusion mass-spectrometer.

The variable names are

- X126, X127C, X127N, X128C, X128N, X129C, X129N, X130C, X130N and X131: the 10 TMT tags used to quantify the peptides along the density gradient.
- Sequence: the peptide sequence.
- ProteinDescriptions: the description of the protein this peptide was associated to.
- NbProteins: the number of proteins in the protein group.
- ProteinGroupAccessions: the main protein accession number in the protein group.
- Modifications: post-translational modifications identified in the peptide.
- qValue: the PSM identification q-value.
- PEP: the PSM posterior error probability.
- IonScore: the Mascot ion identification score.
- NbMissedCleavages: the number of missed cleavages in the peptide.
- IsolationInterference: the calculated precursor ion isolation interference.
- IonInjectTimems: the ions injection time in milli-seconds.
- Intensity: the precursor ion intensity.
- Charge: the peptide charge.
- mzDa: the peptide mass to charge ratio, in Daltons.
- MHDa: the peptide mass, in Daltons.
- DeltaMassPPM: the difference in measure and calculated mass, in parts per millions.
- RTmin: the peptide retention time, in minutes.
- markers: localisation for well known sub-cellular markers. QFeatures of unknown location are encode as `"unknown"`.

For further details, install the `pRolocdata` package and see `?hyperLOPIT2015`.

Usage

```
hlpsms
```

Format

An object of class `data.frame` with 3010 rows and 28 columns.

Source

The pRolocdata package: http://bioconductor.org/packages/pRolocdata/

References

A draft map of the mouse pluripotent stem cell spatial proteome Christoforou A, Mulvey CM, Breckels LM, Geladaki A, Hurrell T, Hayward PC, Naake T, Gatto L, Viner R, Martinez Arias A, Lilley KS. Nat Commun. 2016 Jan 12;7:8992. doi: 10.1038/ncomms9992. PubMed PMID: 26754106; PubMed Central PMCID: PMC4729960.

See Also

See QFeatures to import this data using the `readQFeatures()` function.

---

impute *Quantitative proteomics data imputation*

Description

The `impute` method performs data imputation on `QFeatures` and `SummarizedExperiment` instance using a variety of methods.

Users should proceed with care when imputing data and take precautions to assure that the imputation produce valid results, in particular with naive imputations such as replacing missing values with 0.

See `MsCoreUtils::impute_matrix()` for details on the different imputation methods available and strategies.

Usage

```
impute

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

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

Arguments

`object`	A `SummarizedExperiment` or `QFeatures` object with missing values to be imputed.
`method`	`character(1)` defining the imputation method. See `imputeMethods()` for available ones. See `MsCoreUtils::impute_matrix()` for details.
`...`	Additional parameters passed to the inner imputation function. See `MsCoreUtils::impute_matrix()` for details.
`i`	Defines which element of the `QFeatures` instance to impute. If missing, all assays will be imputed.

Format

An object of class `standardGeneric` of length 1.

Examples

```
MsCoreUtils::imputeMethods()

data(se_na2)
## table of missing values along the rows (proteins)
table(rowData(se_na2)$nNA)
## table of missing values along the columns (samples)
colData(se_na2)$nNA

## non-random missing values
notna <- which(!rowData(se_na2)$randna)
length(notna)
notna

impute(se_na2, method = "min")

if (require("imputeLCMD")) {
  impute(se_na2, method = "QRILC")
  impute(se_na2, method = "MinDet")
}

if (require("norm"))
  impute(se_na2, method = "MLE")

impute(se_na2, method = "mixed",
       randna = rowData(se_na2)$randna,
       mar = "knn", mnar = "QRILC")

## neighbour averaging
x <- se_na2[1:4, 1:6]
assay(x)[1, 1] <- NA ## min value
assay(x)[2, 3] <- NA ## average
assay(x)[3, 1:2] <- NA ## min value and average
## 4th row: no imputation
assay(x)

assay(impute(x, "nbavg"))
```

`joinAssays` *Join assays in a QFeatures object*

Description

This function applies a full-join type of operation on 2 or more assays in a `QFeatures` instance.

Usage

```
joinAssays(x, i, name = "joinedAssay")
```

Arguments

x	An instance of class QFeatures.
i	The indices or names of al least two assays to be joined.
name	A character(1) naming the new assay. Default is joinedAssay. Note that the function will fail if there's already an assay with name.

Details

The rows to be joined are chosen based on the rownames of the respective assays. It is the user's responsability to make sure these are meaningful, such as for example refering to unique peptide sequences or proteins.

The join operation acts along the rows and expects the samples (columns) of the assays to be disjoint, i.e. the assays mustn't share any samples. Rows that aren't present in an assay are set to NA when merged.

The rowData slots are also joined. However, only columns that are shared and that have the same values for matching columns/rows are retained. For example of a feature variable A in sample S1 contains value a1 and variable A in sample S2 in a different assay contains a2, then the feature variable A is dropped in the merged assay.

The joined assay is linked to its parent assays through an AssayLink object. The link between the child assay and the parent assays is based on the assay row names, just like the procedure for joining the parent assays.

Value

A QFeatures object with an additional assay.

Author(s)

Laurent Gatto

Examples

```
## ------------------------------------------------
## An example QFeatures with 3 assays to be joined
## ------------------------------------------------
data(feat2)
feat2

feat2 <- joinAssays(feat2, 1:3)

## Individual assays to be joined, each with 4 samples and a
## variable number of rows.
assay(feat2[[1]])
assay(feat2[[2]])
assay(feat2[[3]])
```

```
## The joined assay contains 14 rows (corresponding to the union
## of those in the initial assays) and 12 samples
assay(feat2[["joinedAssay"]])

## The individual rowData to be joined.
rowData(feat2[[1]])
rowData(feat2[[2]])
rowData(feat2[[3]])

## Only the 'Prot' variable is retained because it is shared among
## all assays and the values and coherent across samples (the
## value of 'Prot' for row 'j' is always 'Pj'). The variable 'y' is
## missing in 'assay1' and while variable 'x' is present is all
## assays, the values for the shared rows are different.
rowData(feat2[["joinedAssay"]])
```

`missing-data` *Managing missing data*

Description

This manual page describes the handling of missing values in QFeatures objects. In the following functions, if `object` is of class `QFeatures`, and optional assay index or name `i` can be specified to define the assay (by name of index) on which to operate.

The following functions are currently available:

- `zeroIsNA(object, i)` replaces all 0 in `object` by `NA`. This is often necessary when third-party software assume that features that weren't quantified should be assigned an intensity of 0.
- `infIsNA(object, i)` replaces all infinite values in `object` by `NA`. This is necessary when third-party software divide expression data by zero values, for instance during custom normalization.
- `nNA(object, i)` return a list of missing value summaries. The first element `nNA` gives a `DataFrame` with the number and the percentage of missing values for the whole assay; the second element `nNArows` provides a `DataFrame` of the number and the percentage of missing values for the features (rows) of the assay(s); the third element `nNAcols` provides the number and the percentage of missing values in each sample of the assay(s). When `object` has class `QFeatures` and additional column with the assays is provided in each element's `DataFrame`.
- `filterNA(object, pNA, i)` removes features (rows) that contain `pNA` percentage or more missing values.

See the *Processing* vignette for examples.

Usage

```
## S4 method for signature 'SummarizedExperiment,missing'
zeroIsNA(object, i)

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

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

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

## S4 method for signature 'SummarizedExperiment,missing'
infIsNA(object, i)

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

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

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

## S4 method for signature 'SummarizedExperiment,missing'
nNA(object, i)

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

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

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

## S4 method for signature 'SummarizedExperiment'
filterNA(object, pNA = 0)

## S4 method for signature 'QFeatures'
filterNA(object, pNA = 0, i)
```

Arguments

object	An object of class QFeatures or SummarizedExperiment.
i	One or more indices or names of the assay(s) to be processed.

pNA numeric(1) providing the maximim percentage of missing values per feature (row) that is acceptable. Feature with higher percentages are removed. If 0 (default), features that contain any number of NA values are dropped.

Value

An instance of the same class as object.

See Also

The impute() for QFeautres instances.

Examples

```
se_na2

## Summary if missing values
nNA(ft_na, 1)

## Remove rows with missing values
assay(filterNA(ft_na, i = 1))

## Replace NAs by zero and back
ft_na <- impute(ft_na, i = 1, method = "zero")
assay(ft_na)
ft_na <- zeroIsNA(ft_na, 1)
assay(ft_na)
```

QFeatures *Quantitative MS QFeatures*

Description

Conceptually, a QFeatures object holds a set of *assays*, each composed of a matrix (or array) containing quantitative data and row annotations (meta-data). The number and the names of the columns (samples) must always be the same across the assays, but the number and the names of the rows (features) can vary. The assays are typically defined as SummarizedExperiment objects. In addition, a QFeatures object also uses a single DataFrame to annotate the samples (columns) represented in all the matrices.

The QFeatures class extends the MultiAssayExperiment::MultiAssayExperiment and inherits all the functionality of the MultiAssayExperiment::MultiAssayExperiment class.

A typical use case for such QFeatures object is to represent quantitative proteomics (or metabolomics) data, where different assays represent quantitation data at the PSM (the main assay), peptide and protein level, and where peptide values are computed from the PSM data, and the protein-level data is calculated based on the peptide-level values. The largest assay (the one with the highest number of features, PSMs in the example above) is considered the main assay.

The recommended way to create QFeatures objects is the use the readQFeatures() function, that creates an instance from tabular data. The QFeatures constructor can be used to create objects

from their bare parts. It is the user's responsability to make sure that these match the class validity requirements.

Usage

```
## S4 method for signature 'QFeatures'
show(object)

## S3 method for class 'QFeatures'
plot(x, interactive = FALSE, ...)

## S4 method for signature 'QFeatures,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'QFeatures'
dims(x)

## S4 method for signature 'QFeatures,character,ANY,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'QFeatures'
rowData(x, use.names = TRUE, ...)

## S4 replacement method for signature 'QFeatures,DataFrameList'
rowData(x) <- value

## S4 replacement method for signature 'QFeatures,ANY'
rowData(x) <- value

rbindRowData(object, i)

selectRowData(x, rowvars)

rowDataNames(x)

## S4 replacement method for signature 'QFeatures,character'
names(x) <- value

longFormat(object, colvars = NULL, rowvars = NULL, index = 1L)

addAssay(x, y, name = "newAssay", assayLinks = AssayLinks(names = name))

## S4 method for signature 'QFeatures'
updateObject(object, ..., verbose = FALSE)

QFeatures(..., assayLinks = NULL)
```

Arguments

object	An instance of class QFeatures.
x	An instance of class QFeatures.
interactive	A logical(1). If TRUE, an interactive graph is generated using plotly. Else, a static plot using igraph is generated. We recommend interactive exploration when the QFeatures object contains more than 50 assays.
...	See MultiAssayExperiment for details. For plot, further arguments passed to igraph::plot.igraph.
i	character(), integer(), logical() or GRanges() object for subsetting by rows.
j	character(), logical(), or numeric() vector for subsetting by colData rows.
drop	logical (default TRUE) whether to drop empty assay elements in the ExperimentList.
k	character(), logical(), or numeric() vector for subsetting by assays
use.names	A logical(1) indicating whether the rownames of each assay should be propagated to the corresponding rowData.
value	The values to use as a replacement. See the corresponding section in the documentation for more details.
rowvars	A character() with the names of the rowData variables (columns) to retain in any assay.
colvars	A character() that selects column(s) in the colData.
index	The assay indicator for SummarizedExperiment objects. A vector input is supported in the case that the SummarizedExperiment object(s) has more than one assay (default 1L)
y	A single assay or a *named* list of assays.
name	A character(1) naming the single assay (default is "newAssay"). Ignored if y is a list of assays.
assayLinks	An optional AssayLinks object.
verbose	logical (default FALSE) whether to print extra messages

Value

See individual method description for the return value.

Constructors

- QFeatures(..., assayLinks) allows the manual construction of objects. It is the user's responsability to make sure these comply. The arguments in ... are those documented in MultiAssayExperiment::MultiAssayExperiment(). For details about assayLinks, see AssayLinks. An example is shown below.
- The readQFeatures() function constructs a QFeatures object from text-based spreadsheet or a data.frame used to generate an assay. See the function manual page for details and an example.

Accessors

- The QFeatures class extends the MultiAssayExperiment::MultiAssayExperiment class and inherits all its accessors and replacement methods.
- The rowData method returns a DataFrameList containing the rowData for each assay of the QFeatures object. On the other hand, rowData can be modified using rowData(x) <- value, where value is a list of tables that can be coerced to DFrame tables. The names of value point to the assays for which the rowData must be replaced. The column names of each table are used to replace the data in the existing rowData. If the column name does not exist, a new column is added to the rowData.
- The rbindRowData functions returns a DFrame table that contains the row binded rowData tables from the selected assays. Only rowData variables that are common to all assays are kept.
- The rowDataNames accessor returns a list with the rowData variable names.
- The longFormat accessor takes a QFeatures object and returns it in a long format DataFrame. Each quantitative value is reported on a separate line. colData and rowData data can also be added. This function is an extension of the longFormat function in the MultiAssayExperiment::MultiAssayExperiment.

Adding assays

- The aggregateFeatures() function creates a new assay by aggregating features of an existing assay.
- addAssay(x, y, name, assayLinks): Adds a new assay (or list of assays) y to the QFeatures instance x. name is a character(1) naming the single assay (default is "newAssay"), and is ignored if y is a list of assays. assayLinks is an optional AssayLinks.

Subsetting

- QFeatures object can be subset using the x[i, j, k, drop = TRUE] paradigm. See the argument descriptions for details.
- The subsetByFeature() function can be used to subset a QFeatures object using one or multiple feature names that will be matched across different assays, taking the aggregation relation between assays.
- The selectRowData(x, rowvars) function can be used to select a limited number of rowData columns of interest named in rowvars in the x instance of class QFeatures. All other variables than rowvars will be dropped. In case an element in rowvars isn't found in any rowData variable, a message is printed.

Author(s)

Laurent Gatto

See Also

- The readQFeatures() constructor and the aggregateFeatures() function. The *QFeatures* vignette provides an extended example.

- The QFeatures-filtering manual page demonstrates how to filter features based on their rowData.
- The missing-data manual page to manage missing values in `QFeatures` objects.
- The QFeatures-processing and `aggregateFeatures()` manual pages and *Processing* vignette describe common quantitative data processing methods using in quantitative proteomics.

Examples

```
## ------------------------
## An empty QFeatures object
## ------------------------

QFeatures()

## -----------------------------------
## Creating a QFeatures object manually
## -----------------------------------

## two assays (matrices) with matching column names
m1 <- matrix(1:40, ncol = 4)
m2 <- matrix(1:16, ncol = 4)
sample_names <- paste0("S", 1:4)
colnames(m1) <- colnames(m2) <- sample_names
rownames(m1) <- letters[1:10]
rownames(m2) <- letters[1:4]

## two corresponding feature metadata with appropriate row names
df1 <- DataFrame(Fa = 1:10, Fb = letters[1:10],
                 row.names = rownames(m1))
df2 <- DataFrame(row.names = rownames(m2))

(se1 <- SummarizedExperiment(m1, df1))
(se2 <- SummarizedExperiment(m2, df2))

## Sample annotation (colData)
cd <- DataFrame(Var1 = rnorm(4),
                Var2 = LETTERS[1:4],
                row.names = sample_names)

el <- list(assay1 = se1, assay2 = se2)
fts1 <- QFeatures(el, colData = cd)
fts1
fts1[[1]]
fts1[["assay1"]]

## Rename assay
names(fts1) <- c("se1", "se2")

## Add an assay
fts1 <- addAssay(fts1, se1[1:2, ], name = "se3")

## Get the assays feature metadata
```

```
rowData(fts1)

## Keep only the Fa variable
selectRowData(fts1, rowvars = "Fa")

## ----------------------------------
## See ?readQFeatures to create a
## QFeatures object from a data.frame
## or spreadsheet.
## ----------------------------------
```

`QFeatures-filtering` *Filter features based on their rowData*

Description

The `filterFeatures` methods enables users to filter features based on a variable in their `rowData`. The features matching the filter will be returned as a new object of class `QFeatures`. The filters can be provided as instances of class `AnnotationFilter` (see below) or as formulas.

Usage

```
VariableFilter(field, value, condition = "==", not = FALSE)

## S4 method for signature 'QFeatures,AnnotationFilter'
filterFeatures(object, filter, na.rm = FALSE, ...)

## S4 method for signature 'QFeatures,formula'
filterFeatures(object, filter, na.rm = FALSE, ...)
```

Arguments

`field`	`character(1)` refering to the name of the variable to apply the filter on.
`value`	`character()` or `integer()` value for the `CharacterVariableFilter` and `NumericVariableFilter` filters respectively.
`condition`	`character(1)` defining the condition to be used in the filter. For `NumericVariableFilter`, one of `"=="`, `"!="`, `">"`, `"<"`, `">="` or `"<="`. For `CharacterVariableFilter`, one of `"=="`, `"!="`, `"startsWith"`, `"endsWith"` or `"contains"`. Default condition is `"=="`.
`not`	`logical(1)` indicating whether the filtering should be negated or not. `TRUE` indicates is negated (!). `FALSE` indicates not negated. Default `not` is `FALSE`, so no negation.
`object`	An instance of class QFeatures.
`filter`	Either an instance of class AnnotationFilter or a formula.
`na.rm`	`logical(1)` indicating whether missing values should be removed. Default is `FALSE`.
`...`	Additional parameters. Currently ignored.

Value

An filtered QFeature object.

Variable filters

The variable filters are filters as defined in the AnnotationFilter package. In addition to the predefined filter, users can arbitrarily set a field on which to operate. These arbitrary filters operate either on a character variables (as CharacterVariableFilter objects) or numerics (as NumericVariableFilters objects), which can be created with the VariableFilter constructor.

Author(s)

Laurent Gatto

See Also

The QFeatures man page for subsetting and the QFeatures vignette provides an extended example.

Examples

```
## ----------------------------------------
## Creating character and numberic
## variable filters
## ----------------------------------------

VariableFilter(field = "my_var",
               value = "value_to_keep",
               condition = "==")

VariableFilter(field = "my_num_var",
               value = 0.05,
               condition = "<=")

example(aggregateFeatures)

## ------------------------------------------------------------------
## Filter all features that are associated to the Mitochondrion in
## the location feature variable. This variable is present in all
## assays.
## ------------------------------------------------------------------

## using the forumla interface, exact mathc
filterFeatures(feat1, ~  location == "Mitochondrion")

## using the forumula intefrace, martial match
filterFeatures(feat1, ~startsWith(location, "Mito"))

## using a user-defined character filter
filterFeatures(feat1, VariableFilter("location", "Mitochondrion"))

## using a user-defined character filter with partial match
```

```
filterFeatures(feat1, VariableFilter("location", "Mito", "startsWith"))
filterFeatures(feat1, VariableFilter("location", "itochon", "contains"))

## ------------------------------------------------------------------
## Filter all features that aren't marked as unknown (sub-cellular
## location) in the feature variable
## ------------------------------------------------------------------

## using a user-defined character filter
filterFeatures(feat1, VariableFilter("location", "unknown", condition = "!="))

## using the forumula interface
filterFeatures(feat1, ~ location != "unknown")

## ------------------------------------------------------------------
## Filter features that have a p-values lower or equal to 0.03
## ------------------------------------------------------------------

## using a user-defined numeric filter
filterFeatures(feat1, VariableFilter("pval", 0.03, "<="))

## using the formula interface
filterFeatures(feat1, ~ pval <= 0.03)

## you can also remove all p-values that are NA (if any)
filterFeatures(feat1, ~ !is.na(pval))

## ------------------------------------------------------------------
## Negative control - filtering for an non-existing markers value
## or a missing feature variable, returning empty results
## ------------------------------------------------------------------

filterFeatures(feat1, VariableFilter("location", "not"))

filterFeatures(feat1, ~ location == "not")

filterFeatures(feat1, VariableFilter("foo", "bar"))

filterFeatures(feat1, ~ foo == "bar")

## ------------------------------------------------------------------
## Example with missing values
## ------------------------------------------------------------------

data(feat1)
rowData(feat1[[1]])[1, "location"] <- NA
rowData(feat1[[1]])

## The row with the NA is not removed
rowData(filterFeatures(feat1, ~ location == "Mitochondrion")[[1]])
rowData(filterFeatures(feat1, ~ location == "Mitochondrion", na.rm = FALSE)[[1]])

## The row with the NA is removed
```

```
rowData(filterFeatures(feat1, ~ location == "Mitochondrion", na.rm = TRUE)[[1]])

## Note that in situations with missing values, it is possible to
## use the `%in%` operator or filter missing values out
## explicitly.

rowData(filterFeatures(feat1, ~ location %in% "Mitochondrion")[[1]])
rowData(filterFeatures(feat1, ~ location %in% c(NA, "Mitochondrion"))[[1]])

## Explicit handling
filterFeatures(feat1, ~ !is.na(location) & location == "Mitochondrion")

## Using the pipe operator
library("magrittr")
feat1 %>%
   filterFeatures( ~ !is.na(location)) %>%
   filterFeatures( ~ location == "Mitochondrion")
```

QFeatures-processing *QFeatures processing*

Description

This manual page describes common quantitative proteomics data processing methods using QFeatures objects. In the following functions, if `object` is of class `QFeatures`, and optional assay index or name `i` can be specified to define the assay (by name of index) on which to operate.

The following functions are currently available:

- `logTransform(object, base = 2, i, pc = 0)` log-transforms (with an optional pseudocount offset) the assay(s).
- `normalize(object, method, i)` normalises the assay(s) according to `method` (see Details).
- `scaleTransform(object, center = TRUE, scale = TRUE, i)` applies `base::scale()` to `SummarizedExperiment` and `QFeatures` objects.
- `sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)` sweeps out array summaries from `SummarizedExperiment` and `QFeatures` objects. See `base::sweep()` for details.

See the *Processing* vignette for examples.

Usage

```
## S4 method for signature 'SummarizedExperiment'
logTransform(object, base = 2, pc = 0)

## S4 method for signature 'QFeatures'
logTransform(object, i, name = "logAssay", base = 2, pc = 0)

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

```
scaleTransform(object, center = TRUE, scale = TRUE)

## S4 method for signature 'QFeatures'
scaleTransform(object, i, name = "scaledAssay", center = TRUE, scale = TRUE)

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

## S4 method for signature 'QFeatures'
normalize(object, i, name = "normAssay", method, ...)

## S4 method for signature 'SummarizedExperiment'
sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)

## S4 method for signature 'QFeatures'
sweep(
  x,
  MARGIN,
  STATS,
  FUN = "-",
  check.margin = TRUE,
  ...,
  i,
  name = "sweptAssay"
)
```

Arguments

object	An object of class QFeatures or SummarizedExperiment.
base	numeric(1) providing the base with respect to which logarithms are computed. Defaults is 2.
pc	numeric(1) with a pseudocount to add to the quantitative data. Useful when (true) 0 are present in the data. Default is 0 (no effect).
i	A numeric vector or a character vector giving the index or the name, respectively, of the assay(s) to be processed.
name	A character(1) naming the new assay name. Defaults are logAssay for logTransform, scaledAssay for scaleTranform and normAssay for normalize.
center	logical(1) (default is TRUE) value or numeric-alike vector of length equal to the number of columns of object. See base::scale() for details.
scale	logical(1) (default is TRUE) or a numeric-alike vector of length equal to the number of columns of object. See base::scale() for details.
method	character(1) defining the normalisation method to apply. See Details.
...	Additional parameters passed to inner functions.
x	An object of class QFeatures or SummarizedExperiment in sweep.
MARGIN	As in base::sweep(), a vector of indices giving the extent(s) of x which correspond to STATS.

STATS	As in base::sweep(), the summary statistic which is to be swept out.
FUN	As in base::sweep(), the function to be used to carry out the sweep.
check.margin	As in base::sweep(), a logical. If TRUE (the default), warn if the length or dimensions of STATS do not match the specified dimensions of x. Set to FALSE for a small speed gain when you know that dimensions match.

Details

The method parameter in normalize can be one of "sum", "max", "center.mean", "center.median", "div.mean", "div.median", "diff.meda", "quantiles", "quantiles.robust" or "vsn". The MsCoreUtils::normalizeMethods() function returns a vector of available normalisation methods.

- For "sum" and "max", each feature's intensity is divided by the maximum or the sum of the feature respectively. These two methods are applied along the features (rows).
- "center.mean" and "center.median" center the respective sample (column) intensities by subtracting the respective column means or medians. "div.mean" and "div.median" divide by the column means or medians. These are equivalent to sweeping the column means (medians) along MARGIN = 2 with FUN = "-" (for "center.*") or FUN = "/" (for "div.*").
- "diff.median" centers all samples (columns) so that they all match the grand median by subtracting the respective columns medians differences to the grand median.
- Using "quantiles" or "quantiles.robust" applies (robust) quantile normalisation, as implemented in preprocessCore::normalize.quantiles() and preprocessCore::normalize.quantiles.robust(). "vsn" uses the vsn::vsn2() function. Note that the latter also glog-transforms the intensities. See respective manuals for more details and function arguments.

For further details and examples about normalisation, see MsCoreUtils::normalize_matrix().

Value

An processed object of the same class as x or object.

Examples

```
MsCoreUtils::normalizeMethods()
```

---

readQFeatures *QFeatures from tabular data*

Description

Convert tabular data from a spreadsheet or a data.frame into a QFeatures object.

Usage

```
readQFeatures(table, ecol, fnames, ..., name = NULL)

readSummarizedExperiment(table, ecol, fnames, ...)
```

Arguments

table	File or object holding the quantitative data. Can be either a character(1) with the path to a text-based spreadsheet (comma-separated values by default, but see ...) or an object that can be coerced to a data.frame. It is advised not to encode characters as factors.
ecol	A numeric indicating the indices of the columns to be used as expression values. Can also be a character indicating the names of the columns. Caution must be taken if the column names are composed of special characters like (or - that will be converted to a . by the read.csv function. If ecol does not match, the error message will display the column names as seen by the read.csv function.
fnames	An optional character(1) or numeric(1) indicating the column to be used as feature names.
...	Further arguments that can be passed on to read.csv except stringsAsFactors, which is always FALSE.
name	An character(1) to name assay in the QFeatures object. If not set, features is used.

Value

An instance of class QFeatures or SummarizedExperiment.

Functions

- readQFeatures: See description.
- readSummarizedExperiment: Convert tabular data from a spreadsheet or a data.frame into a SummarizedExperiment object.

Author(s)

Laurent Gatto

See Also

The QFeatures class for an example on how to use readQFeatures and how to further manipulate the resulting data.

Examples

```
## Load a data.frame with PSM-level data
data(hlpsms)

## Create the QFeatures object
fts2 <- readQFeatures(hlpsms, ecol = 1:10, name = "psms")
fts2
```

reduceDataFrame *Reduces and expands a* DataFrame

Description

A long dataframe can be *reduced* by mergeing certain rows into a single one. These new variables are constructed as a SimpleList containing all the original values. Invariant columns, i.e columns that have the same value along all the rows that need to be merged, can be shrunk into a new variables containing that invariant value (rather than in list columns). The grouping of rows, i.e. the rows that need to be shrunk together as one, is defined by a vector.

The opposite operation is *expand*. But note that for a DataFrame to be expanded back, it must not to be simplified.

Usage

```
reduceDataFrame(x, k, count = FALSE, simplify = TRUE, drop = FALSE)

expandDataFrame(x, k = NULL)
```

Arguments

x	The DataFrame to be reduced or expanded.
k	A 'vector' of length nrow(x) defining the grouping based on which the DataFrame will be shrunk.
count	logical(1) specifying of an additional column (called by default .n) with the tally of rows shrunk into on new row should be added. Note that if already existing, .n will be silently overwritten.
simplify	A logical(1) defining if invariant columns should be converted to simple lists. Default is TRUE.
drop	A logical(1) specifying whether the non-invariant columns should be dropped altogether. Default is FALSE.

Value

An expanded (reduced) DataFrame.

Missing values

Missing values do have an important effect on reduce. Unless all values to be reduces are missing, they will result in an non-invariant column, and will be dropped with drop = TRUE. See the example below.

The presence of missing values can have side effects in higher level functions that rely on reduction of DataFrame objects.

Author(s)

Laurent Gatto

Examples

```
library("IRanges")

k <- sample(100, 1e3, replace = TRUE)
df <- DataFrame(k = k,
                x = round(rnorm(length(k)), 2),
                y = seq_len(length(k)),
                z = sample(LETTERS, length(k), replace = TRUE),
                ir = IRanges(seq_along(k), width = 10),
                r = Rle(sample(5, length(k), replace = TRUE)),
                invar = k + 1)
df

## Shinks the DataFrame
df2 <- reduceDataFrame(df, df$k)
df2

## With a tally of the number of members in each group
reduceDataFrame(df, df$k, count = TRUE)

## Much faster, but more crowded result
df3 <- reduceDataFrame(df, df$k, simplify = FALSE)
df3

## Drop all non-invariant columns
reduceDataFrame(df, df$k, drop = TRUE)

## Missing values
d <- DataFrame(k = rep(1:3, each = 3),
               x = letters[1:9],
               y = rep(letters[1:3], each = 3),
               y2 = rep(letters[1:3], each = 3))
d

## y is invariant and can be simplified
reduceDataFrame(d, d$k)
## y isn't not dropped
reduceDataFrame(d, d$k, drop = TRUE)

## BUT with a missing value
d[1, "y"] <- NA
d

## y isn't invariant/simplified anymore
reduceDataFrame(d, d$k)
## y now gets dropped
reduceDataFrame(d, d$k, drop = TRUE)
```

subsetByFeature *Subset by feature name*

Description

This function will find the assays and features that match directly (by name) or indirectly (through aggregation) the feature name.

The `subsetByFeature` function will first identify the assay that contains the feature(s) `i` and filter the rows matching these feature names exactly. It will then find, in the other assays, the features that produces `i` through aggregation with the `aggregateQFeatures` function.

See QFeatures for an example.

Arguments

`x`	An instance of class QFeatures.
`y`	A `character` of feature names present in an assay in `x`.
`...`	Additional parameters. Ignored.

Value

An new instance of class QFeatures containing relevant assays and features.

Examples

```
example(aggregateFeatures)

## Retrieve protein 'ProtA' and its 2 peptides and 6 PSMs
feat1["ProtA", , ]
```

Index