Package 'PSMatch'

November 12, 2026

Title Handling and Managing Peptide Spectrum Matches

Version 1.16.0

Description The PSMatch package helps proteomics practitioners to load, handle and manage Peptide Spectrum Matches. It provides functions to model peptide-protein relations as adjacency matrices and connected components, visualise these as graphs and make informed decision about shared peptide filtering. The package also provides functions to calculate and visualise MS2 fragment ions.

Depends S4Vectors, R (>= 4.1.0), PTMods (>= 0.99.4)

Imports utils, stats, igraph, methods, Spectra (>= 1.17.10), Matrix, BiocParallel, BiocGenerics, ProtGenerics (>= 1.27.1), QFeatures, MsCoreUtils

Suggests MsDataHub, rpx, mzID, mzR, SummarizedExperiment, BiocStyle, rmarkdown, knitr, factoextra, vdiffr (>= 1.0.0), testthat

License Artistic-2.0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

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

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

biocViews Infrastructure, Proteomics, MassSpectrometry

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

git_branch RELEASE_3_23

git_last_commit ba33712

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Laurent Gatto [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1520-2268>),
Johannes Rainer [aut] (ORCID: <https://orcid.org/0000-0002-6977-7147>),
Sebastian Gibb [aut] (ORCID: <https://orcid.org/0000-0001-7406-4443>),
Samuel Wieczorek [ctb],
Thomas Burger [ctb],
Guillaume Deflandre [ctb] (ORCID:
<https://orcid.org/0009-0008-1257-2416>)

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

Contents

adjacencyMatrix 2
calculateFragments 7
ConnectedComponents 10
describeProteins 14
filterPSMs 15
getAminoAcids 17
getAtomicMass 17
labelFragments 18
plotSpectraPTM 19
PSM 23
PSMatch 27

Index **30**

`adjacencyMatrix` *Convert to/from an adjacency matrix.*

Description

There are two ways that peptide/protein matches are commonly stored: either as a vector or an adjacency matrix. The functions described below convert between these two format.

Usage

```
makeAdjacencyMatrix(
  x,
  split = ";",
  peptide = psmVariables(x)["peptide"],
  protein = psmVariables(x)["protein"],
  score = psmVariables(x)["score"],
  binary = FALSE
)

makePeptideProteinVector(m, collapse = ";")
```

```
plotAdjacencyMatrix(
  m,
  protColors = 0,
  pepColors = NULL,
  layout = igraph::layout_nicely
)
```

Arguments

x	Either an instance of class PSM or a character. See example below for details.
split	character(1) defining how to split the string of protein identifiers (using strsplit()). Default is ";". If NULL, splitting is ignored.
peptide	character(1) indicating the name of the variable that defines peptides in the PSM object. Default is the peptide PSM variable as defined in psmVariables().
protein	character(1) indicating the name of the variable that defines proteins in the PSM object. Default is the peptide PSM variable as defined in psmVariables().
score	character(1) indicating the name of the variable that defines PSM scores in the PSM object. Default is the score PSM variable as defined in psmVariables(). Ignored when NA (which is the default value unless set by the user when constructing the PSM object).
binary	logical(1) indicates if the adjacency matrix should be strictly binary. In such a case, PSMs matching the same peptide but from different precursors (for example charge 2 and 3) or carrying different PTMs, are counted only once. Default if FALSE. This also overrides any score that would be set.
m	A peptide-by-protein adjacency matrix.
collapse	character(1) indicating how to collapse protein names for shared peptides. Default is ";".
protColors	Either a numeric(1) or a named character() of colour names. The numeric value indicates the protein colouring level to use. If 0 (default), all protein nodes are labelled in steelblue. For values > 0, approximate string distances (see adist()) between protein names are calculated and nodes of proteins that have names that differ will be coloured differently, with higher values leading to more colours. While no maximum to this value is defined in the code, it shouldn't be higher than the number of proteins. If a character is used, it should be a character of colour names named by protein identifiers. That vector should provide colours for at least all proteins in the adjacency matrix m, but more protein could be named. The latter is useful when generating a colour vector for all proteins in a dataset and use it for different adjacency matrix visualisations.
pepColors	Either NULL (default) for no peptide colouring (white nodes) or a named character() of colour names. It should be a character of colour names named by peptide identifiers. That vector should provide colours for at least all peptides in the adjacency matrix m, but more peptides could be named. The latter is useful when generating a colour vector for all peptides in a dataset and use it for different adjacency matrix visualisations.
layout	A graph layout, as defined in the ipgraph package. Default is igraph::layout_as_bipartite().

Details

The `makeAdjacencyMatrix()` function creates a peptide-by-protein adjacency matrix from a `character` or an instance of class `PSM()`.

The character is formatted as `x <- c("ProtA", "ProtB", "ProtA;ProtB", ...)`, as commonly encountered in proteomics data spreadsheets. It defines that the first peptide is mapped to protein "ProtA", the second one to protein "ProtB", the third one to "ProtA" and "ProtB", and so on. The resulting matrix contains as many rows as there are unique peptides and as many columns as there are unique protein identifiers in `x`. The columns are named after the protein identifiers and the peptide/protein vector names are used to name the matrix rows (retaining only the unique names).

The `makePeptideProteinVector()` function does the opposite operation, taking an adjacency matrix as input and returning a peptide/protein vector. The matrix colnames are used to populate the vector and the matrix rownames are used to name the vector elements.

Note that when creating an adjacency matrix from a PSM object, the matrix is not necessarily binary, as multiple PSMs can match the same peptide (sequence), such as for example precursors with different charge states. A binary matrix can either be generated with the `binary` argument (setting all non-0 values to 1) or by reducing the PSM object accordingly (see example below).

It is also possible to generate adjacency matrices populated with identification scores or probabilites by setting the "score" PSM variable upon construction of the PSM object (see `PSM()` for details). In case multiple PSMs occur, their respective scores get summed.

The `plotAdjacencyMatrix()` function is useful to visualise small adjacency matrices, such as those representing protein groups modelled as connected components, as described and illustrated in `ConnectedComponents()`. The function generates a graph modelling the relation between proteins (represented as squares) and peptides (represented as circes), which can further be coloured (see the `protColors` and `pepColors` arguments). The function invisibly returns the graph `igraph` object for additional tuning and/or interactive visualisation using, for example `igraph::tkplot()`.

Such as illustrated in the examples below, each row/peptide is expected to refer to protein groups or individual proteins (groups of size 1). These have to be split accordingly.

Value

A peptide-by-protein sparce adjacency matrix (or class `dgCMatrix` as defined in the `Matrix` package) or peptide/protein vector.

Author(s)

Laurent Gatto

Examples

```
## ----------------------
## From a character
## ----------------------

## Protein vector without names
prots <- c("ProtA", "ProtB", "ProtA;ProtB")
makeAdjacencyMatrix(prots)
```

```
## Named protein vector
names(prots) <- c("pep1", "pep2", "pep3")
prots
m <- makeAdjacencyMatrix(prots)
m

## Back to vector
vec <- makePeptideProteinVector(m)
vec
identical(prots, vec)

## ----------------------------
## PSM object from a data.frame

## ----------------------------
## Case 1: Duplicate identifications

psmdf <- data.frame(psm = paste0("psm", 1:10),
                    peptide = paste0("pep", c(1, 1, 2, 2, 3, 4, 6, 7, 8, 8)),
                    protein = paste0("Prot", LETTERS[c(1, 1, 2, 2, 3, 4, 3, 5, 6, 6)]))
psmdf
psm <- PSM(psmdf, peptide = "peptide", protein = "protein")
psm
makeAdjacencyMatrix(psm)

## Reduce PSM object to peptides
rpsm <- reducePSMs(psm, k = psm$peptide)
rpsm
makeAdjacencyMatrix(rpsm)

## Or set binary to TRUE
makeAdjacencyMatrix(psm, binary = TRUE)

## ----------------------------
## Case 2: Protein groups are separated by a semicolon
psmdf <- data.frame(psm = paste0("psm", 1:5),
                    peptide = paste0("pep", c(1, 2, 3, 4, 5)),
                    protein = c("ProtA", "ProtB;ProtD", "ProtA;ProtC",
                                "ProtC", "ProtA;ProtC;ProtD"))
psmdf
psm <- PSM(psmdf, peptide = "peptide", protein = "protein")
psm
makeAdjacencyMatrix(psm, split = ";")

## ----------------------------
## PSM object from an mzid file
## ----------------------------

f <- MsDataHub::TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.20141210.mzid()

psm <- PSM(f) |>
       filterPsmDecoy() |>
       filterPsmRank()
```

```
psm
adj <- makeAdjacencyMatrix(psm)
dim(adj)
adj[1:10, 1:4]

## Binary adjacency matrix
adj <- makeAdjacencyMatrix(psm, binary = TRUE)
adj[1:10, 1:4]

## Peptides with rowSums > 1 match multiple proteins.
## Use filterPsmShared() to filter these out.
table(Matrix::rowSums(adj))

## ------------------------------------------------
## Binary, non-binary and score adjacency matrices
## ------------------------------------------------

## -------------------------------------
## Case 1: no scores, 1 PSM per peptides
psmdf <- data.frame(spectrum = c("sp1", "sp2", "sp3", "sp4", "sp5",
                                 "sp6", "sp7", "sp8", "sp9", "sp10"),
                    sequence = c("NKAVRTYHEQ", "IYNHSQGFCA", "YHWRLPVSEF",
                                 "YEHNGFPLKD", "WAQFDVYNLS", "EDHINCTQWP",
                                 "WSMKVDYEQT", "GWTSKMRYPL", "PMAYIWEKLC",
                                 "HWAEYFNDVT"),
                    protein = c("ProtB", "ProtB", "ProtA", "ProtD", "ProtD",
                                "ProtG", "ProtF", "ProtE", "ProtC", "ProtF"),
                    decoy = rep(FALSE, 10),
                    rank = rep(1, 10),
                    score = c(0.082, 0.310, 0.133, 0.174, 0.944, 0.0261,
                              0.375, 0.741, 0.254, 0.058))
psmdf

psm <- PSM(psmdf, spectrum = "spectrum", peptide = "sequence",
           protein = "protein", decoy = "decoy", rank = "rank")

## binary matrix
makeAdjacencyMatrix(psm)

## Case 2: sp1 and sp11 match the same peptide (NKAVRTYHEQ) as different PSMs
psmdf2 <- rbind(psmdf,
                data.frame(spectrum = "sp11",
                           sequence = psmdf$sequence[1],
                           protein = psmdf$protein[1],
                           decoy = FALSE,
                           rank = 1,
                           score = 0.011))
psmdf2
psm2 <- PSM(psmdf2, spectrum = "spectrum", peptide = "sequence",
            protein = "protein", decoy = "decoy", rank = "rank")

## Now NKAVRTYHEQ/ProtB counts 2 PSMs
makeAdjacencyMatrix(psm2)
```

```
## Force a binary matrix
makeAdjacencyMatrix(psm2, binary = TRUE)

## Case 3: Peptide (NKAVRTYHEQ) stems from multiple proteins (ProtB and
## ProtG). They are separated by a semicolon.
psmdf3 <- psmdf
psmdf3[psmdf3$sequence == "NKAVRTYHEQ","protein"] <- "ProtB;ProtG"
psmdf3
psm3 <- PSM(psmdf3, spectrum = "spectrum", peptide = "sequence",
            protein = "protein", decoy = "decoy", rank = "rank")

## Now ProtB & ProtG count 2 PSMs each: NKAVRTYHEQ and IYNHSQGFCA &
## EDHINCTQWP respectively
makeAdjacencyMatrix(psm3, split = ";")

## --------------------------------
## Case 4: set the score PSM values
psmVariables(psm) ## no score defined
psm4 <- PSM(psm, spectrum = "spectrum", peptide = "sequence",
            protein = "protein", decoy = "decoy", rank = "rank",
            score = "score")
psmVariables(psm4) ## score defined

## adjacency matrix with scores
makeAdjacencyMatrix(psm4)

## Force a binary matrix
makeAdjacencyMatrix(psm4, binary = TRUE)

## --------------------------------
## Case 5: scores with multiple PSMs

psm5 <- PSM(psm2, spectrum = "spectrum", peptide = "sequence",
            protein = "protein", decoy = "decoy", rank = "rank",
            score = "score")

## Now NKAVRTYHEQ/ProtB has a summed score of 0.093 computed as
## 0.082 (from sp1) + 0.011 (from sp11)
makeAdjacencyMatrix(psm5)
```

`calculateFragments` *Calculate ions produced by fragmentation with variable modifications*

Description

This method calculates a-, b-, c-, x-, y- and z-ions produced by fragmentation.

Available methods

- The default method with signature `sequence = "character"` and `object = "missing"` calculates the theoretical fragments for a peptide sequence. It returns a `data.frame` with the columns `mz`, `ion`, `type`, `pos`, `z`, `seq` and `peptide`.
- Additional method can be defined that will adapt their behaviour based on spectra defined in `object`. See for example the MSnbase package that implements a method for objects of class `Spectrum2`.

Usage

```
## S4 method for signature 'character,missing'
calculateFragments(
  sequence,
  type = c("b", "y"),
  z = 1,
  fixed_modifications = NULL,
  variable_modifications = NULL,
  addCarbamidomethyl = TRUE,
  max_mods = Inf,
  neutralLoss = defaultNeutralLoss(),
  verbose = TRUE
)
```

Arguments

`sequence`	`character()` providing a peptide sequence. If positional modifications are included in the sequence, variable modifications may not be used. See examples below for more detail.
`type`	`character` vector of target ions; possible values: `c("a", "b", "c", "x", "y", "z")`. Default is `type = c("b", "y")`.
`z`	numeric with a desired charge state; default is 1.
`fixed_modifications`	Deprecated parameter. Please use `PTMods::addFixedModifications()` to generate sequences with positional modifications instead. Named `numeric` or `character`. If a `character` is given, values must be in UniMod name or UniMod ID format (e.g. `"Phospho"`, `"UNIMOD:21"`). The annotation style of the values is preserved in the output. Specifies which fixed modifications are applied to which amino acids.
`variable_modifications`	Deprecated parameter. Please use `PTMods::addVariableModifications()` to generate sequences with positional modifications instead. Named `numeric` or `character`. If a `character` is given, values must be in UniMod name or UniMod ID format (e.g. `"Phospho"`, `"UNIMOD:21"`). The annotation style of the values is preserved in the output. Specifies which variable modifications are used on which amino acids.
`addCarbamidomethyl`	logical(1L) set to `TRUE` by default. Applies carbamidomethylation as a fixed modification unless `fixed_modifications` is not `NULL` or if a carbamidomethyl

is already present in the given sequences. It is strongly suggested to rely on `PTMods::addFixedModifications()` instead.

`max_mods` Deprecated parameter. Please use in combination with PTMods::addVariableModifications() instead. A numeric indicating the maximum number of variable modifications allowed on the sequence at once. Does not include fixed modifications. Default value is positive infinity.

`neutralLoss` `list`, it has to have two named elments, namely `water` and `ammonia` that contain a `character` vector which type of neutral loss should be calculated. Currently neutral loss on the C terminal `"Cterm"`, at the amino acids `c("D", "E", "S", "T")` for `"water"` (shown with an _) and `c("K", "N", "Q", "R")` for `"ammonia"` (shown with an `*`) are supported.

`There is a helper function `defaultNeutralLoss()` that returns the correct list. It has two arguments `disableWaterLoss` and `disableAmmoniaLoss` to remove single neutral loss options. See the example section for use cases.`

`verbose` `logical(1)`. Deprecated parameter. If `TRUE` (default) the used modifications are printed.

Value

A `data.frame` showing all the ions produced by fragmentation with all possible combinations of modifications. The used variable modifications are displayed in the `peptide` column through the use of amino acids followed by the modification within brackets. Fixed modifications are not displayed.

Author(s)

Sebastian Gibb mail@sebastiangibb.de

Guillaume Deflandre guillaume.deflandre@uclouvain.be

Examples

```
## No modifications
calculateFragments("ACE")

## Multiple ion types and charge states
calculateFragments("ACE",
                   type = c("a", "b", "c", "x", "y", "z"),
                   z = 1:2)

## Positional modification written directly in the sequence string
## The annotation style must be supported by PTMods::convertAnnotation
calculateFragments("T[+79.966]CE")
calculateFragments("T[Phospho]CE")
## Notice carbamidomethylation applied by default, but ignored if already
## present.
calculateFragments("T[UNIMOD:21]C[Carbamidomethyl]E")

## neutral loss
```

```
defaultNeutralLoss()

## disable water loss on the C terminal
defaultNeutralLoss(disableWaterLoss="Cterm")

## real example
calculateFragments("PQR")
calculateFragments("PQR",
                   neutralLoss=defaultNeutralLoss(disableWaterLoss="Cterm"))
calculateFragments("PQR",
                   neutralLoss=defaultNeutralLoss(disableAmmoniaLoss="Q"))

## disable neutral loss completely
calculateFragments("PQR", neutralLoss=NULL)

## Recommended workflow: use PTMods functions to produce positional sequences
## before calling calculateFragments.

## Fixed modification (Carbamidomethyl on C) using addFixedModifications
seq_fixed <- PTMods::addFixedModifications("ACE",
                                           fixedModifications = c(C = 57.02))
calculateFragments(seq_fixed)

## Fixed modification including N-terminus using addFixedModifications
seq_nterm <- PTMods::addFixedModifications(
    "ACE",
    fixedModifications = c(C = 57.02, Nterm = 229.16))
calculateFragments(seq_nterm)

## Variable modification (delta mass on A) using addVariableModifications
seq_var <- PTMods::addVariableModifications("ACE",
                                            variableModifications = c(A = 43.25))
calculateFragments(seq_var)

## Both fixed and variable modifications using addModifications
seq_mods <- PTMods::addModifications("ARGSHKATC",
                                     fixedModifications = c(C = 57),
                                     variableModifications = c(S = 79, T = 79),
                                     maxMods = 2)
calculateFragments(seq_mods)
```

`ConnectedComponents` *Connected components*

Description

Connected components are a useful representation when exploring identification data. They represent the relation between proteins (the connected components) and how they form groups of proteins as defined by shared peptides.

Connected components are stored as `ConnectedComponents` objects that can be generated using the `ConnectedComponents()` function.

Usage

```
ConnectedComponents(object, ...)

ccMatrix(x)

connectedComponents(x, i, simplify = TRUE)

## S4 method for signature 'ConnectedComponents'
length(x)

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

## S4 method for signature 'ConnectedComponents'
ncols(x)

## S4 method for signature 'ConnectedComponents'
nrows(x)

## S4 method for signature 'ConnectedComponents,integer,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'ConnectedComponents,logical,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'ConnectedComponents,numeric,ANY,ANY'
x[i, j, ..., drop = FALSE]

prioritiseConnectedComponents(x)

prioritizeConnectedComponents(x)

## S4 method for signature 'ConnectedComponents'
adjacencyMatrix(object)
```

Arguments

object	For the ConnectedComponents class constructor, either a sparse adjacency matrix of class Matrix or an instance of class PSM.
...	Additional arguments passed to makeAdjacencyMatrix() when object is of class PSM().
x	An object of class ConnectedComponents.
i	numeric(), integer() or logical() to subset the ConnectedComponents instance. If a logical(), it must be of same length as the object is subsets.
simplify	logical(1) if TRUE (default), the output is simplified to sparse matrix if i was of length 1, otherwise a List is returned. Always a List if FALSE.
j	ignored

drop ignore

Value

The ConnectedComponents() constructor returns an instance of class ConnectedComponents. The *Creating and manipulating objects* section describes the return values of the functions that manipulate ConnectedComponents objects.

Slots

adjMatrix The sparse adjacency matrix (class Matrix) of dimension *p* peptides by *m* proteins that was used to generate the object.

ccMatrix The sparse connected components matrix (class Matrix) of dimension *m* by *m* proteins.

adjMatrices A List containing adjacency matrices of each connected components.

Creating and manipulating objects

- Instances of the class are created with the ConnectedComponent() constructor from a PSM() object or directly from a sparse adjacency matrix of class Matrix. Note that if using the latter, the rows and columns must be named.
- The sparse peptide-by-protein adjacency matrix is stored in the ConnectedComponent instance and can be accessed with the adjacencyMatrix() function.
- The protein-by-protein connected components sparse matrix of object x can be accessed with the ccMatrix(x) function.
- The number of connected components of object x can be retrieved with length(x).
- The size of the connected components of object x, i.e the number of proteins in each component, can be retrieved with ncols(x). The number of peptides defining the connected components can be retrieved with nrows(x). Both can be accessed with dims(x).
- The connectedComponents(x, i, simplify = TRUE) function returns the peptide-by-protein sparse adjacency matrix (or List of matrices, if length(i) > 1), i.e. the subset of the adjacency matrix defined by the proteins in connected component(s) i. i is the numeric index (between 1 and length(x)) of the connected connected. If simplify is TRUE (default), then a matrix is returned instead of a List of matrices of length 1. If set to FALSE, a List is always returned, irrespective of its length.
- To help with the exploration of individual connected Components, the prioritiseConnectedComponents() function will take an instance of ConnectedComponents and return a data.frame where the component indices are ordered based on their potential to clean up/flag some peptides and split protein groups in small groups or individual proteins, or simply explore them. The prioritisation is based on a set of metrics computed from the component's adjacency matrix, including its dimensions, row and col sums maxima and minima, its sparsity and the number of communities and their modularity that quantifies how well the communities separate (see igraph::modularity(). Note that trivial components, i.e. those composed of a single peptide and protein are excluded from the prioritised results. This data.frame is ideally suited for a principal component analysis (using for instance prcomp()) for further inspection for component visualisation with plotAdjacencyMatrix().

Examples

```
## ---------------------------------
## From an adjacency matrix
## ---------------------------------
library(Matrix)
adj <- sparseMatrix(i = c(1, 2, 3, 3, 4, 4, 5),
                    j = c(1, 2, 3, 4, 3, 4, 5),
                    x = 1,
                    dimnames = list(paste0("Pep", 1:5),
                                    paste0("Prot", 1:5)))
adj
cc <- ConnectedComponents(adj)
cc

length(cc)
ncols(cc)

adjacencyMatrix(cc) ## same as adj above
ccMatrix(cc)

connectedComponents(cc)
connectedComponents(cc, 3) ## a singel matrix
connectedComponents(cc, 1:2) ## a List

## ---------------------------------
## From an PSM object
## ---------------------------------
f <- MsDataHub::TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.20141210.mzid()

psm <- PSM(f) |>
       filterPsmDecoy() |>
       filterPsmRank()

cc <- ConnectedComponents(psm)
cc

length(cc)
table(ncols(cc))

(i <- which(ncols(cc) == 4))
ccomp <- connectedComponents(cc, i)

## A group of 4 proteins that all share peptide RTRYQAEVR
ccomp[[1]]

## Visualise the adjacency matrix - here, we see how the single
## peptides (white node) 'unites' the four proteins (blue nodes)
plotAdjacencyMatrix(ccomp[[1]])

## A group of 4 proteins formed by 7 peptides: THPAERKPRRRKKR is
## found in the two first proteins, KPTARRRKRK was found twice in
## ECA3389, VVPVGLRALVWVQR was found in all 4 proteins, KLKPRRR
```

```
## is specific to ECA3399, ...
ccomp[[3]]

## See how VVPVGLRALVWVQR is shared by ECA3406 ECA3415 ECA3389 and
## links the three other componennts, namely ECA3399, ECA3389 and
## (ECA3415, ECA3406). Filtering that peptide out would split that
## protein group in three.
plotAdjacencyMatrix(ccomp[[3]])

## Colour protein node based on protein names similarity
plotAdjacencyMatrix(ccomp[[3]], 1)

## To select non-trivial components of size > 1
cc2 <- cc[ncols(cc) > 1]
cc2

## Use components features to prioritise their exploration
pri_cc <- prioritiseConnectedComponents(cc)
pri_cc

plotAdjacencyMatrix(connectedComponents(cc, 1082), 1)
```

`describeProteins` *Describe protein and peptide compositions*

Description

It is important to explore PSM results prior to any further downstream analysies. Two functions, that work on `PSM()` and `ConnectedComponents()` objects can be used for this:

- The `describeProteins()` function describe protein composition in terms of unique and shared peptides.
- The `describePeptides()` function describe unique/shared peptide composition.

Usage

```
describeProteins(object, ...)

describePeptides(object, ...)
```

Arguments

`object`	Either an instance of class `Matrix`, `PSM()` or `ConnectedComponents()`.
`...`	Additional arguments passed to `makeAdjacencyMatrix()`.

Value

`describePeptides()` invisibly return the table of unique and shared peptides. `describeProteins()` invisibly returns a `data.frame` with logicals indicating the unique/shared peptide composition of proteins. Both functions are used for their side effects of printing a short descriptive output about peptides and proteins.

Examples

```
f <- MsDataHub::TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.20141210.mzid()

psm <- PSM(f) |>
       filterPsmDecoy() |>
       filterPsmRank()

describePeptides(psm)
describeProteins(psm)
```

`filterPSMs` *Filter out unreliable PSMs.*

Description

Functions to filter out PSMs matching. The PSMs should be stored in a `PSM` such as those produced by `PSM()`.

- `filterPsmDecoy()` filters out decoy PSMs, i.e. those annotated as `isDecoy`.
- `filterPsmRank()` filters out PSMs of rank > 1.
- `filterPsmShared()` filters out shared PSMs, i.e. those that match multiple proteins.
- `filterPsmFdr()` filters out PSMs based on their FDR.

Usage

```
filterPSMs(
  x,
  decoy = psmVariables(x)["decoy"],
  rank = psmVariables(x)["rank"],
  protein = psmVariables(x)["protein"],
  spectrum = psmVariables(x)["spectrum"],
  peptide = psmVariables(x)["peptide"],
  verbose = TRUE
)

filterPsmDecoy(x, decoy = psmVariables(x)["decoy"], verbose = TRUE)

filterPsmRank(x, rank = psmVariables(x)["rank"], verbose = TRUE)

filterPsmShared(
  x,
  protein = psmVariables(x)["protein"],
  peptide = psmVariables(x)["peptide"],
  verbose = TRUE
)

filterPsmFdr(x, FDR = 0.05, fdr = psmVariables(x)["fdr"], verbose = TRUE)
```

Arguments

x	An instance of class PSM.
decoy	character(1) with the column name specifying whether entries match the decoy database or not. Default is the decoy PSM variable as defined in psmVariables(). The column should be a logical and only PSMs holding a FALSE are retained. Filtering is ignored if set to NULL or NA.
rank	character(1) with the column name holding the rank of the PSM. Default is the rank PSM variable as defined in psmVariables(). This column should be a numeric and only PSMs having rank equal to 1 are retained. Filtering is ignored if set to NULL or NA.
protein	character(1) with the column name holding the protein (groups) protein. Default is the protein PSM variable as defined in psmVariables(). Filtering is ignored if set to NULL or NA.
spectrum	character(1) with the name of the spectrum identifier column. Default is the spectrum PSM variable as defined in psmVariables(). Filtering is ignored if set to NULL or NA.
peptide	character(1) with the name of the peptide identifier column. Default is the peptide PSM variable as defined in psmVariables(). Filtering is ignored if set to NULL or NA.
verbose	logical(1) setting the verbosity flag.
FDR	numeric(1) to be used to filter based on the fdr variable. Default is 0.05.
fdr	character(1) variable name that defines that defines the spectrum FDR (or any similar/relevant metric that can be used for filtering). This value isn't set by default as it depends on the search engine and application. Default is NA.

Value

A new filtered PSM object with the same columns as the input x.

Author(s)

Laurent Gatto

Examples

```
f <- MsDataHub::TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.20141210.mzid()

id <- PSM(f)
filterPSMs(id)
```

getAminoAcids *Amino acids*

Description

Returns a data.frame of amino acid properties: AA, ResidueMass, Abbrev3, ImmoniumIonMass, Name, Hydrophobicity, Hydrophilicity, SideChainMass, pK1, pK2 and pI.

Usage

```
getAminoAcids()
```

Value

data.frame

Author(s)

Laurent Gatto

Examples

```
getAminoAcids()
```

getAtomicMass *Atomic mass.*

Description

Returns a double of used atomic mass.

Usage

```
getAtomicMass()
```

Value

A named double.

Author(s)

Sebastian Gibb

Examples

```
getAtomicMass()
```

`labelFragments` *labels MS2 Fragments*

Description

Creates a list of annotations based on `calculateFragments` results.

Usage

```
labelFragments(x, tolerance = 0, ppm = 20, what = c("ion", "mz"), ...)
```

Arguments

`x`	An instance of class `Spectra` of length 1, containing a spectra variable `"sequence"` with a `character(1)` representing a valid peptide sequence.
`tolerance`	absolute acceptable difference of m/z values for peaks to be considered matching (see `MsCoreUtils::closest()` for more details).
`ppm`	m/z relative acceptable difference (in ppm) for peaks to be considered matching (see `MsCoreUtils::closest()` for more details).
`what`	`character(1)`, one of `"ion"` (default) or `"mz"`, defining whether labels should be fragment ions, , or their m/z values. If the latter, then the m/z values are named with the ion labels.
`...`	additional parameters (except `verbose`) passed to `calculateFragments()` to calculate fragment m/z values to be added to the spectra in `x`.

Value

Return a `list()` of `character()` with fragment ion labels. The elements are named after the peptide they belong to (variable modifications included).

Author(s)

Johannes Rainer, Guillaume Deflandre, Sebastian Gibb, Laurent Gatto

Examples

```
library("Spectra")

sp <- DataFrame(msLevel = 2L, rtime = 2345, sequence = "SIGFEGDSIGR")
sp$mz <- list(c(100.048614501953, 110.069030761719, 112.085876464844,
                117.112571716309, 158.089569091797, 163.114898681641,
                175.117172241211, 177.098587036133, 214.127075195312,
                232.137542724609, 233.140335083008, 259.938415527344,
                260.084167480469, 277.111572265625, 282.680786132812,
                284.079437255859, 291.208282470703, 315.422576904297,
                317.22509765625, 327.2060546875, 362.211944580078,
                402.235290527344, 433.255004882812, 529.265991210938,
```

```
                549.305236816406, 593.217041015625, 594.595092773438,
                609.848327636719, 631.819702148438, 632.324035644531,
                632.804931640625, 640.8193359375, 641.309936523438,
                641.82568359375, 678.357238769531, 679.346252441406,
                688.291259765625, 735.358947753906, 851.384033203125,
                880.414001464844, 881.40185546875, 919.406433105469,
                938.445861816406, 1022.56658935547, 1050.50415039062,
                1059.82800292969, 1107.52734375, 1138.521484375,
                1147.51538085938, 1226.056640625))
sp$intensity <- list(c(83143.03, 65473.8, 192735.53, 3649178.5,
                       379537.81, 89117.58, 922802.69, 61190.44,
                       281353.22, 2984798.75, 111935.03, 42512.57,
                       117443.59, 60773.67, 39108.15, 55350.43,
                       209952.97, 37001.18, 439515.53, 139584.47,
                       46842.71, 1015457.44, 419382.31, 63378.77,
                       444406.66, 58426.91, 46007.71, 58711.72,
                       80675.59, 312799.97, 134451.72, 151969.72,
                       3215457.75, 1961975, 395735.62, 71002.98,
                       69405.73, 136619.47, 166158.69, 682329.75,
                       239964.69, 242025.44, 1338597.62, 50118.02,
                       1708093.12, 43119.03, 97048.02, 2668231.75,
                       83310.2, 40705.72))
sp <- Spectra(sp)

## The fragment ion labels
labelFragments(sp)

## The fragment mz labels
labelFragments(sp, what = "mz")

## Pass additional parameters to calculateFragments using a PTMods modified sequence
sp_mod <- sp
sp_mod$sequence <- PTMods::addFixedModifications("SIGFEGDSIGR",
                                                 fixedModifications = c(Nterm = 5))
labelFragments(sp_mod, type = c("a", "b", "x", "y"))

## Annotate the spectum with the fragment labels
plotSpectra(sp, labels = labelFragments, labelPos = 3)

## By default used in `plotSpectraPTM()`.
plotSpectraPTM(sp)
```

`plotSpectraPTM` *Function to plot MS/MS spectra with PTMs*

Description

`plotSpectraPTM()` creates annotated visualisations of MS/MS spectra, designed to explore fragment identifications and post-translational modifications (PTMs).

plotSpectraPTM() plots a spectrum's m/z values on the x-axis and corresponding intensities on the y-axis, labeling the peaks according to theoretical fragment ions (e.g., b, y, a, c, x, z) computed using labelFragments() and calculateFragments().

Usage

```
plotSpectraPTM(
  x,
  deltaMz = TRUE,
  ppm = 20,
  xlab = "m/z",
  ylab = "intensity [%]",
  xlim = numeric(),
  ylim = numeric(),
  main = character(),
  col = c(y = "darkred", b = "darkblue", acxy = "darkgreen", other = "grey40"),
  labelCex = 1,
  labelSrt = 0,
  labelAdj = NULL,
  labelPos = 3,
  labelOffset = 0.5,
  asp = 1,
  minorTicks = TRUE,
  USI = TRUE,
  fixedModifications = NULL,
  variableModifications = NULL,
  addCarbamidomethyl = TRUE,
  ...
)
```

Arguments

x	a Spectra() object.
deltaMz	logical(1L) If TRUE, adds an additional plot showing the difference of mass over charge between matched oberved and theoretical fragments in parts per million. Does not yet support modifications. The matching is based on calculateFragments() and needs a 'sequence' variable in spectraVariables(x). Default is set to TRUE.
ppm	integer(1L) Sets the limits of the delta m/z plot and is passed to labelFragments().
xlab	character(1) with the label for the x-axis (by default xlab = "m/z").
ylab	character(1) with the label for the y-axis (by default ylab = "intensity").
xlim	numeric(2) defining the x-axis limits. The range of m/z values are used by default.
ylim	numeric(2) defining the y-axis limits. The range of intensity values are used by default.
main	character(1) with the title for the plot. By default the spectrum's MS level and retention time (in seconds) is used.

col — Named character(4L). Colors for the labels, the character names need to be "b", "y", "acxz" and "other", respectively for the b-ions, y-ions, a,c,x,z-ions and the unidentified fragments.

labelCex — numeric(1) giving the amount by which the text should be magnified relative to the default. See parameter cex in par().

labelSrt — numeric(1) defining the rotation of the label. See parameter srt in text().

labelAdj — see parameter adj in text().

labelPos — see parameter pos in text().

labelOffset — see parameter offset in text().

asp — for plotSpectraPTM(), the target ratio (columns / rows) when plotting mutliple spectra (e.g. for 20 spectra use asp = 4/5 for 4 columns and 5 rows or asp = 5/4 for 5 columns and 4 rows; see grDevices::n2mfrow() for details). If deltaMz is TRUE, asp is ignored.

minorTicks — logical(1L). If TRUE, minor ticks are added to the plots. Default is set to TRUE.

USI — logical(1L). If TRUE, the universal spectrum identifier is displayed.

fixedModifications — Named numeric or character, passed to PTMods::addFixedModifications(). Applied to all sequences before plotting. NULL by default (no fixed modifications applied).

variableModifications — Named numeric or character, passed to PTMods::addVariableModifications(). Each unique combination of variable modifications generates a separate copy of the corresponding spectrum. NULL by default (no variable modifications applied).

addCarbamidomethyl — logical(1L) set to TRUE by default. Applies carbamidomethylation as a fixed modification unless carbamidomethyl is already present in the given sequences. If carbamidomethylation should be applied as a variable modification, do set addCarbamidomethylation = FALSE. For more details on this, see the appropriate vignette by running 'vignette("Fragments", package = "PSMatch")

... — additional parameters to be passed to the labelFragments() and calculateFragments() functions.

Value

Creates a plot depicting an MS/MS-MS spectrum.

Author(s)

Johannes Rainer, Sebastian Gibb, Guillaume Deflandre, Laurent Gatto

See Also

Spectra::plotSpectra()

Examples

```
library("Spectra")

sp <- DataFrame(msLevel = 2L, rtime = 2345, sequence = "SIGFEGDSIGR")
sp$mz <- list(c(75.048614501953, 81.069030761719, 86.085876464844,
                88.039, 158.089569091797, 163.114898681641,
                173.128, 177.098587036133, 214.127075195312,
                232.137542724609, 233.140335083008, 259.938415527344,
                260.084167480469, 277.111572265625, 282.680786132812,
                284.079437255859, 291.208282470703, 315.422576904297,
                317.22509765625, 327.2060546875, 362.211944580078,
                402.235290527344, 433.255004882812, 534.258783,
                549.305236816406, 593.217041015625, 594.595092773438,
                609.848327636719, 631.819702148438, 632.324035644531,
                632.804931640625, 640.8193359375, 641.309936523438,
                641.82568359375, 678.357238769531, 679.346252441406,
                706.309623, 735.358947753906, 851.384033203125,
                880.414001464844, 881.40185546875, 906.396433105469,
                938.445861816406, 1022.56658935547, 1050.50415039062,
                1059.82800292969, 1107.52734375, 1138.521484375,
                1147.51538085938, 1226.056640625))
sp$intensity <- list(c(83143.03, 65473.8, 192735.53, 3649178.5,
                       379537.81, 89117.58, 922802.69, 61190.44,
                       281353.22, 2984798.75, 111935.03, 42512.57,
                       117443.59, 60773.67, 39108.15, 55350.43,
                       209952.97, 37001.18, 439515.53, 139584.47,
                       46842.71, 1015457.44, 419382.31, 63378.77,
                       444406.66, 58426.91, 46007.71, 58711.72,
                       80675.59, 312799.97, 134451.72, 151969.72,
                       1961975, 69405.76, 395735.62, 71002.98,
                       3215457.75, 136619.47, 166158.69, 682329.75,
                       239964.69, 242025.44, 1338597.62, 50118.02,
                       1708093.12, 43119.03, 97048.02, 2668231.75,
                       83310.2, 40705.72))
sp <- Spectra(sp)

## Annotate the spectum with the fragment labels
plotSpectraPTM(sp, main = "An example of an annotated plot")

## Annotate the spectrum without the delta m/z plot
plotSpectraPTM(sp, deltaMz = FALSE)

## Annotate the spectrum with different ion types
plotSpectraPTM(sp, type = c("a", "b", "x", "y"))

## Annotate the spectrum with modifications using PTMods
sp_mod <- sp
sp_mod$sequence <- PTMods::addFixedModifications("SIGFEGDSIGR", fixedModifications = c(Nterm = "Acetyl"))
plotSpectraPTM(sp_mod)

## Or call them within the function directly:
plotSpectraPTM(sp, fixedModifications = NULL,
```

```
variableModifications = c(R = "Methyl"))

## Annotate multiple spectra at a time
plotSpectraPTM(c(sp, sp))

## Color the peaks with different colors
plotSpectraPTM(sp, col = c(y = "red", b = "blue", acxy = "chartreuse3", other = "black"))
```

PSM *A class for peptide-spectrum matches*

Description

The PSM class is a simple class to store and manipulate peptide-spectrum matches. The class encapsulates PSM data as a DataFrame (or more specifically a DFrame) with additional lightweight metadata annotation.

There are two types of PSM objects:

- Objects with duplicated spectrum identifiers. This holds for multiple matches to the same spectrum, be it different peptide sequences or the same sequence with or without a post-translational modification. Such objects are typically created with the PSM() constructor starting from mzIdentML files.
- Reduced objects where the spectrum identifiers (or any equivalent column) are unique keys within the PSM table. Matches to the same scan/spectrum are merged into a single PSM data row. Reduced PSM object are created with the reducePSMs() function. See examples below.

Objects can be checked for their reduced state with the reduced() function which returns TRUE for reduced instances, FALSE when the spectrum identifiers are duplicated, or NA when unknown. The flag can also be set explicitly with the reduced()<- setter.

Usage

```
PSM(
  x,
  spectrum = NA,
  peptide = NA,
  protein = NA,
  decoy = NA,
  rank = NA,
  score = NA,
  fdr = NA,
  parser = c("mzR", "mzID"),
  BPPARAM = SerialParam()
)

reduced(object, spectrum = psmVariables(object)["spectrum"])
```

```
reduced(object) <- value

psmVariables(object, which = "all")

reducePSMs(object, k = object[[psmVariables(object)["spectrum"]]])

## S4 method for signature 'PSM'
adjacencyMatrix(object)
```

Arguments

x	character() of mzid file names, an instance of class PSM, or a data.frame.
spectrum	character(1) variable name that defines a spectrum in the PSM data. Default are "spectrumID" (mzR parser) or "spectrumid" (mzID parser). It is also used to calculate the reduced state.
peptide	character(1) variable name that defines a peptide in the PSM data. Detaults are "sequence" (mzR parser) or "pepSeq" (mzID parser).
protein	character(1) variable name that defines a protein in the PSM data. Detaults are "DatabaseAccess" (mzR parser) or "accession" (mzID parser).
decoy	character(1) variable name that defines a decoy hit in the PSM data. Detaults are "isDecoy" (mzR parser) or "isdecoy" (mzID parser).
rank	character(1) variable name that defines the rank of the peptide spectrum match in the PSM data. Default is "rank".
score	character(1) variable name that defines the PSM score. This value isn't set by default as it depends on the search engine and application. Default is NA.
fdr	character(1) variable name that defines that defines the spectrum FDR (or any similar/relevant metric that can be used for filtering). This value isn't set by default as it depends on the search engine and application. Default is NA.
parser	character(1) defining the parser to be used to read the mzIdentML files. One of "mzR" (default) or "mzID".
BPPARAM	an object from the BiocParallel package to control parallel processing. The default value is SerialParam()‘ to read files in series.
object	An instance of class PSM.
value	new value to be passed to setter.
which	character() with the PSM variable name to retrieve. If "all" (default), all named variables are returned. See PSM() for valid PSM variables.
k	A vector or factor of length equal to nrow(x) that defines the primary key used to reduce x. This typically corresponds to the spectrum identifier. The defauls is to use the spectrum PSM variable.

Value

PSM() returns a PSM object.

reducePSMs() returns a reduced version of the x input.

Creating and using PSM objects

- The PSM() constructor uses parsers provided by the mzR or mzID packages to read the mzIdentML data. The vignette describes some apparent differences in their outputs. The constructor input is a character of one more multiple file names.
- PSM objects can also be created from a data.frame object (or any variable that can be coerced into a S4Vectors::DataFrame.
- Finally, PSM() can also take a PSM object as input, which leaves the PSM data as is and is used to set/update the PSM variables.
- The constructor can also initialise variables (called *PSM variables*) needed for downstream processing, notably filtering (see filterPSMs()) and to generate a peptide-by-protein adjacency matrix (see makeAdjacencyMatrix()). These variables can be extracted with the psmVariables() function. They represent the columns in the PSM table that identify spectra, peptides, proteins, decoy peptides hit ranks and, optionally, a PSM score. The value of these variables will depend on the backend used to create the object, or left blank (i.e. encoded as NA) when building an object by hand from a data.frame. In such situation, they need to be passed explicitly by the user as arguments to PSM().
- The adjacencyMatrix() accessor can be used to retrieve the binary sparse peptide-by-protein adjacency matrix from the PSM object. It also relies on PSM variables which thus need to be set beforehand. For more flexibility in the generation of the adjacency matrix (for non-binary matrices), use makeAdjacencyMatrix().

Examples

```
## ---------------------------------
## Example with a single mzid file
## ---------------------------------

f <- MsDataHub::TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.20141210.mzid()

## mzR parser (default)
psm <- PSM(f)
psm

## PSM variables
psmVariables(psm)

## mzID parser
psm_mzid <- PSM(f, parser = "mzID")
psm_mzid

## different PSM variables
psmVariables(psm_mzid)

## Reducing the PSM data
(i <- which(duplicated(psm$spectrumID))[1:2])
(i <- which(psm$spectrumID %in% psm$spectrumID[i]))
psm2 <- psm[i, ]
reduced(psm2)
```

```
## Peptide sequence CIDRARHVEVQIFGDGKGRVVALGERDCSLQRR with
## Carbamidomethyl modifications at positions 1 and 28.
DataFrame(psm2[, c("sequence", "spectrumID", "modName", "modLocation")])
reduced(psm2) <- FALSE
reduced(psm2)

## uses by default the spectrum PSM variable, as defined during
## the construction - see psmVariables()
rpsm2 <- reducePSMs(psm2)
rpsm2
DataFrame(rpsm2[, c("sequence", "spectrumID", "modName", "modLocation")])
reduced(rpsm2)

## ---------------------------------
## Multiple mzid files
## ---------------------------------

library(rpx)
PXD022816 <- PXDataset("PXD022816")
PXD022816

(mzids <- pxget(PXD022816, grep("mzID", pxfiles(PXD022816))[1:2]))
psm <- PSM(mzids)
psm
psmVariables(psm)

## Here, spectrum identifiers are repeated accross files
psm[grep("scan=20000", psm$spectrumID), "spectrumFile"]

## Let's create a new primary identifier composed of the scan
## number and the file name
psm$pkey <- paste(sub("^.+Task\\\\", "", psm$spectrumFile),
                  sub("^.+scan=", "", psm$spectrumID),
                  sep = "::")
head(psm$pkey)

## the PSM is not reduced
reduced(psm, "pkey")
DataFrame(psm[6:7, ])

## same sequence, same spectrumID, same file
psm$sequence[6:7]
psm$pkey[6:7]

## different modification locations
psm$modLocation[6:7]

## here, we need to *explicitly* set pkey to reduce
rpsm <- reducePSMs(psm, psm$pkey)
rpsm
reduced(rpsm, "pkey")

## the two rows are now merged into a single one; the distinct
```

```
## modification locations are preserved.
(i <- which(rpsm$pkey == "QEP2LC6_HeLa_50ng_251120_01-calib.mzML::12894"))
DataFrame(rpsm[i, c("sequence", "pkey", "modName", "modLocation")])

## ---------------------------------
## PSM from a data.frame
## ---------------------------------

psmdf <- data.frame(spectrum = paste0("sp", 1:10),
                    sequence = replicate(10,
                                         paste(sample(getAminoAcids()[-1, "AA"], 10),
                                               collapse = "")),
                    protein = sample(paste0("Prot", LETTERS[1:7]), 10,
                                     replace = TRUE),
                    decoy = rep(FALSE, 10),
                    rank = rep(1, 10),
                    score = runif(10))
psmdf

psm <- PSM(psmdf)
psm
psmVariables(psm)

## no PSM variables set
try(adjacencyMatrix(psm))

## set PSM variables
psm <- PSM(psm, spectrum = "spectrum", peptide = "sequence",
           protein = "protein", decoy = "decoy", rank = "rank")
psm
psmVariables(psm)

adjacencyMatrix(psm)
```

PSMatch *PSMatch: Handling and Managing Peptide Spectrum Matches*

Description

The PSMatch package offers functionality to load, manage and analyse Peptide Spectrum Matches as generated in mass spectrometry-based proteomics. The four main objects and concepts that are proposed in this package are described below, and are aimed to proteomics practitioners to explore and understand their identification data better.

PSM objects

As mentioned in the PSM() manual page, The PSM class is a simple class to store and manipulate peptide-spectrum matches. The class encapsulates PSM data as a DataFrame (or more specifically a DFrame) with additional lightweight metadata annotation. PSM objects are typically creatd from XML-based mzID files or data.frames imported from spreadsheets. It is then possible to

apply widely used filters (such as removal of decoy hits, PSMs of rank > 1, ...) as described in `filterPSMs()`.

Adjacency matrices

PSM data, as produced by all proteomics search engines, is exported as a table-like structure where PSM are documented along the rows by variables such as identification scores, peptides sequences, modifications and the protein which the peptides originate from. There is always a level of ambiguity in such data, as peptides can be mapped to mutliple proteins; they are then called shared peptides, as opposed to unique peptides.

One convenient way to store the relation between peptides and proteins is as a peptide-by-protein adjacency matrix. Such matrices can be generated from PSM object or vectors using the `makeAdjacencyMatrix()` function.

The `describePeptides()` and `describeProteins()` functions are also helpful to tally the number of unique and shared peptides and the number of proteins composed of unique or shared peptides, or a combination thereof.

Connected Components

Once we model the peptide-to-protein relations explicitly using an adjacency matrix, it becomes possible to perform computations on the proteins that are grouped by the peptides they share. These groups are mathematically defined as connected components, which are implemented as `ConnectedComponents()` objects.

Fragment ions

The package also provides functionality to calculate ions produced by the fragmentation of a peptides (see `calculateFragments()`) and annotated MS2 `Spectra::Spectra()` objects (see `labelFragments()`).

Vignettes

A couple of vignette describe how to several of these concepts through illustrative use-cases. Use `vignette(package = "PSMatch")` to get a list and open them directly in `R` or read them online on the package's webpage.

Author(s)

Maintainer: Laurent Gatto <laurent.gatto@uclouvain.be> (ORCID)

Authors:

- Johannes Rainer <Johannes.Rainer@eurac.edu> (ORCID)
- Sebastian Gibb <mail@sebastiangibb.de> (ORCID)

Other contributors:

- Samuel Wieczorek <samuel.wieczorek@cea.fr> [contributor]
- Thomas Burger <thomas.burger@cea.fr> [contributor]
- Guillaume Deflandre <guillaume.deflandre@uclouvain.be> (ORCID) [contributor]

See Also

Useful links:

- https://github.com/RforMassSpectrometry/PSM
- Report bugs at https://github.com/RforMassSpectrometry/PSM/issues

Index

[,ConnectedComponents,integer,ANY,ANY
(ConnectedComponents), 10
[,ConnectedComponents,integer,ANY,ANY-method
(ConnectedComponents), 10
[,ConnectedComponents,logical,ANY,ANY
(ConnectedComponents), 10
[,ConnectedComponents,logical,ANY,ANY-method
(ConnectedComponents), 10
[,ConnectedComponents,numeric,ANY,ANY
(ConnectedComponents), 10
[,ConnectedComponents,numeric,ANY,ANY-method
(ConnectedComponents), 10

adist(), *3*
adjacencyMatrix, 2
adjacencyMatrix,ConnectedComponents
(ConnectedComponents), 10
adjacencyMatrix,ConnectedComponents-method
(ConnectedComponents), 10
adjacencyMatrix,PSM-method (PSM), 23

calculateFragments, 7
calculateFragments(), *18*, *28*
calculateFragments,character,missing-method
(calculateFragments), 7
ccMatrix (ConnectedComponents), 10
ConnectedComponents, 10
connectedComponents
(ConnectedComponents), 10
ConnectedComponents(), *4*, *14*, *28*
ConnectedComponents-class
(ConnectedComponents), 10

defaultNeutralLoss
(calculateFragments), 7
describePeptides (describeProteins), 14
describePeptides(), *28*
describeProteins, 14
describeProteins(), *28*
dims,ConnectedComponents
(ConnectedComponents), 10
dims,ConnectedComponents-method
(ConnectedComponents), 10

filterPsmDecoy (filterPSMs), 15
filterPsmFdr (filterPSMs), 15
filterPsmRank (filterPSMs), 15
filterPSMs, 15
filterPSMs(), *25*, *28*
filterPsmShared (filterPSMs), 15

getAminoAcids, 17
getAtomicMass, 17

igraph::layout_as_bipartite(), *3*
igraph::modularity(), *12*
igraph::tkplot(), *4*

labelFragments, 18
labelFragments(), *28*
length,ConnectedComponents
(ConnectedComponents), 10
length,ConnectedComponents-method
(ConnectedComponents), 10

makeAdjacencyMatrix (adjacencyMatrix), 2
makeAdjacencyMatrix(), *4*, *11*, *14*, *25*, *28*
makePeptideProteinVector
(adjacencyMatrix), 2
makePeptideProteinVector(), *4*
modificationPositions
(calculateFragments), 7
MsCoreUtils::closest(), *18*

ncols,ConnectedComponents
(ConnectedComponents), 10
ncols,ConnectedComponents-method
(ConnectedComponents), 10
nrows,ConnectedComponents
(ConnectedComponents), 10

nrows,ConnectedComponents-method
(ConnectedComponents), 10

plotAdjacencyMatrix (adjacencyMatrix), 2
plotAdjacencyMatrix(), *12*
plotSpectraPTM, 19
prcomp(), *12*
prioritiseConnectedComponents
(ConnectedComponents), 10
prioritizeConnectedComponents
(ConnectedComponents), 10
PSM, 23
PSM(), *4*, *11*, *12*, *14*, *15*, *24*, *25*, *27*
PSM,character (PSM), 23
PSM,data.frame (PSM), 23
PSM,PSM (PSM), 23
PSM-class (PSM), 23
PSMatch, 27
PSMatch-package (PSMatch), 27
psmVariables (PSM), 23
psmVariables(), *3*, *16*, *25*
PTMods::addFixedModifications(), *8*
PTMods::addVariableModifications(), *8*

readPSMs (PSM), 23
reduced (PSM), 23
reduced<- (PSM), 23
reducePSMs (PSM), 23

S4Vectors::DataFrame, *25*
show,ConnectedComponents
(ConnectedComponents), 10
Spectra::plotSpectra(), *21*
Spectra::Spectra(), *28*
strsplit(), *3*