Package ‘labelpepmatch’

November 23, 2015

Title Tools For Interpreting Mass Spectra With Labeled Peptides

Description Labelpepmatch is a package for the interpretation of nano-LC-MS spectra of peptides labeled with stable isotope tags. It takes as input a file with peaks with their m/z, retention time and signal quantity. Next to functions to detect peak pairs and mass match them to a database of known peptides, the package also contains appropriate statistics, inference and visualisation tools.

Version 0.0.99

Author Rik Verdonck <`rik.verdonck@bio.kuleuven.be`>

Maintainer Rik Verdonck <`rik.verdonck@bio.kuleuven.be`>

Depends R (>= 2.11.0),lme4

Enhances multcomp,limma

Imports bitops,brew,doParallel,foreach,influence.ME,lsmeans,plotrix,plyr,RCurl,reshape2,colorRamps,gplots

Suggests knitr,rmarkdown

VignetteBuilder knitr

License GNU General Public License version 2

LazyData true

R topics documented:

`add_id` *Manually add an identification from e.g. MS/MS*

Description

Manually add identifications to a pepmatched object. This can both be used as an extension to mass match, or without mass matching.

Usage

```
add_id(pepmatched, ID_vector, name_vector, pepseq_vector, pepmass_vector)
```

Arguments

`pepmatched`	Input object of class `pepmatched`, generated by the pepmatch() or pep.id() function of this package.
`ID_vector`	A character vector with the IDs of the features that get the identifications. This is the value of the first column of the "matchlist". See vignette for an example. Can also be a single character string.
`name_vector`	A character vector with the same length as the ID_vector, containing the names of the peptides.
`pepseq_vector`	Optional character vector with the peptide sequences of the identifications to be added.
`pepmass_vector`	Optional numeric vector with the precise masses of the identifications to be added.

Value

An object of class `pepmatched` with added identifications that can serve as input to downstream labelpepmatch functions.

Author(s)

Rik Verdonck

See Also

`pep.id`, `pep.massmatch`

`calculate_peptide_mass` *Calculate the mass of a peptide.*

Description

Takes a character string peptide sequence and calculates its mono-isotopic mass.

Usage

```
calculate_peptide_mass(peptide)
```

Arguments

`peptide`	Character string of amino acids. Amino acids are always one letter code and upper case. Non-standard symbols are "a" for C-terminal amidation, "p" for a pyroglutaminated N-terminal (should be "pE" or "pQ"), "$" for a sulfo-tyrosin, "J" for a leucine or an isoleucine and "&" for a glutamine (128.059) or a lysine (128.095). Note that in this last case, an average mass is used for the theoretical mass calculation.

Value

A single value for the mono-isotopic mass of the peptide.

Author(s)

Rik Verdonck

download_lpm_db *Download a peptide database.*

Description

Download a database of known peptides into your R-session.

Usage

```
download_lpm_db(db = c("desertlocust", "celegans"))
```

Arguments

db	Character. Database to be downloaded. Choice between "desertlocust", "celegans"

Value

A peptide database of standard format with columns `"name"`,`"MW"` and `"sequence"` and optional columns `"family"` and `"reference"`

Author(s)

Rik Verdonck

generate_random_db *Generate a random database of peptides.*

Description

This function generates a database of random sequences using a restricted randomization procedure that shuffels the amino acids of the input database over its peptide length distribution. It also respects the N-terminal pyroglutamination and C-terminal amidation frequencies. If we plot the sorted molecular weights of the mock database on the sorted molecular weights of the input database, we expect them to reside approximately on y=x. The generate_random_db function will be used in the false discovery estimation of `pep.id`.

Usage

```
generate_random_db(db, size = 1, plot = F, verbose = F)
```

Arguments

`db`	A database with the first 3 columns `"name"`,`"MW"` and `"sequence"` (as read in with the `download_lpm_db` function)
`size`	Numeric. The desired mock database size as a proportion of the original database size.
`plot`	Logical. If TRUE, mock database is plotted onto original database. Only works if mock and real database are of equal size (`size` paramter is 1).
`verbose`	Logical. If TRUE, some properties of the mock database are printed in the terminal.

Details

A mock database is generated based on the input database. This function works as follows: all peptide lengths (in number of amino acids) of the input database are stored in one vector, and all amino acids of the input are stored in a second vector. Next, samples with replacement are taken from the length distribution, and peptides with these lenghts are generated by sampling with replacement from the amino acid vectors. In a last step, amidations and pyroglutaminations are added with a chance equal to their proportion in the input database. As a result, all masses in the newly generated database are realistic peptide masses.

Value

A database of random peptides.

Author(s)

Rik Verdonck

See Also

`pep.id`

`locustdata` *Dataset with TMAB labelled locust peptides.*

Description

Dataset with TMAB labelled locust peptides.

Usage

```
locustdata
```

Format

```
List of 2
 $ frame :'data.frame': 1425 obs. of  26 variables:
  ..$ id        : num [1:1425] 36286 36285 36044 36116 36274 ...
  ..$ z         : num [1:1425] 1 1 1 1 1 1 1 1 2 4 ...
  ..$ mz_1      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_2      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_3      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_4      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_5      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_6      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_7      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ mz_8      : num [1:1425] 1868 580 1160 934 361 ...
  ..$ Quantity_1: num [1:1425] 3252 68516 324316 279604 517488 ...
  ..$ Quantity_2: num [1:1425] 8164 67744 382124 382000 801112 ...
  ..$ Quantity_3: num [1:1425] 5696 140180 434388 428184 692692 ...
  ..$ Quantity_4: num [1:1425] 7056 115140 545744 478340 808132 ...
  ..$ Quantity_5: num [1:1425] 9080 168324 566776 564280 973372 ...
  ..$ Quantity_6: num [1:1425] 4444 65364 637096 554152 987188 ...
  ..$ Quantity_7: num [1:1425] 3212 87568 642764 461484 992424 ...
  ..$ Quantity_8: num [1:1425] 3572 103852 673296 467448 1030116 ...
  ..$ Ret_1     : num [1:1425] 33.6 30.7 32.7 33.6 32 ...
  ..$ Ret_2     : num [1:1425] 33.7 30.8 32.8 33.7 32.1 ...
  ..$ Ret_3     : num [1:1425] 33.5 30.7 32.5 33.5 31.9 ...
  ..$ Ret_4     : num [1:1425] 33.7 31 32.9 33.7 32.2 ...
  ..$ Ret_5     : num [1:1425] 33.8 31 32.9 33.8 32.3 ...
  ..$ Ret_6     : num [1:1425] 33.5 30.7 32.6 33.5 31.9 ...
  ..$ Ret_7     : num [1:1425] 33.5 30.8 32.7 33.5 32 ...
  ..$ Ret_8     : num [1:1425] 33.5 30.8 32.6 33.5 32 ...
 $ design:'data.frame': 8 obs. of  5 variables:
  ..$ RunName       : Factor w/ 8 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8
  .. ..- attr(*, "names")= chr [1:8] "1" "2" "3" "4" ...
  ..$ LightCondition: Factor w/ 2 levels "greg","sol": 2 2 1 1 2 2 1 1
  .. ..- attr(*, "names")= chr [1:8] "1" "2" "3" "4" ...
  ..$ HeavyCondition: Factor w/ 2 levels "greg","sol": 1 1 2 2 1 1 2 2
  .. ..- attr(*, "names")= chr [1:8] "1" "2" "3" "4" ...
  ..$ Direction     : Factor w/ 2 levels "F","R": 1 1 2 2 1 1 2 2
 ..$ FileName     : Factor w/ 1 level "exported_featuredata_schistoTMAB2011_2015.csv": 1 1 1 1 1 1 1
 - attr(*, "class")= chr "lpm_input"
```

Details

These are peptides from the corpora cardiaca of adult desert locusts (*Schistocerca gregaria*) labelled with light (D0) and heavy (D9) TMAB. The conditions are sol and greg: solitarious and gregarious animals. Check the design matrix `locustdata$design` to have an overview of the labeling. The quantity used here is the peak intensity as determined by Progenesis LC-MS.

Author(s)

Rik Verdonck

References

Will be published soon.

`lpm_heatmap` *Plot a heatmap.*

Description

Make a heatmap from an `lpm_linearmodel` object. Uses the residuals of the null model of the lpm_linearmodel. This is a linear model with label as a main effect, without accounting for treatment. The lpm_heatmap function can be applied both on the raw residuals or on the contrasts within one peak pair.

Usage

```
lpm_heatmap(x, contrasts = F, prawcutoff = 0.05, padjcutoff = 0.05,
  FCcutoff = 1.25, main = "heatmap")
```

Arguments

`x`	An object of class `lpm_linearmodel`
`contrasts`	Logical. Should the heatmap be on the contrasts, or on the residuals?
`prawcutoff`	Numeric. Cutoff for maximal raw p-value for features to be retained in the heatmap.
`padjcutoff`	Numeric. Cutoff for maximal adjusted p-value for features to be retained in the heatmap.
`FCcutoff`	Numeric. Cutoff for minimal log2 fold change.
`main`	Character. Title of the heatmap.

Author(s)

Rik Verdonck

`lpm_linearmodel` *Linear models for labelpepmatch.*

Description

This function takes a `lpm_statlist` object and runs a linear model on it. In this version of the package, two models are available. See details.

Usage

```
lpm_linearmodel(statlist, method = "vanilla", p.adjust.method = "BH",
  cores = 1, logtransformed = T, verbose = F)
```

Arguments

`statlist`	An object of class `lpm_statlist`
`method`	Character. See details.
`p.adjust.method`	The method you want to use for correction for multiple testing. Choose between "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr" or "none". For more information, see `p.adjust`
`cores`	Interger. Number of cores that can be used on the computer for calculation. When >1, the packages foreach and doSNOW (windows) or doMC (linux) will be loaded.
`logtransformed`	Logical. Are your data already transformed on a log2 scale?
`verbose`	Logical. If TRUE, verbose output is generated during model estimation. Might be helpful when running computationally demanding models to monitor progress.

Details

The vanilla method runs a separate mixed model on each feature, using label effect as a covariate and run as a random effect. Hence, it corrects for a label bias within each feature separately. In the output you will find a p-value for the label effect for each separate feature. The complexmixed model is a mixed model ran on all features at once, with label effect nested in run as a covariate. This method is extremely powerful, but calculation times rise quickly, and hence it is only possible to use on a limited number of features (e.g. only mass matched features, only highest quantities etc.). The time complexity is estimated to be quasipolynomial nlog(n), and it is advised not to use this method for more than 50 features.

Author(s)

Rik Verdonck & Wouter De Haes

`lpm_make.RGList` *Bridge between labelpepmatch and limma.*

Description

Turn a labelpepmatch `pepmatched` or `lpm_statlist` object in a limma RGList object.

Usage

```
lpm_make.RGList(x)
```

Arguments

`x`	An object of class `lpm_statlist`.

Author(s)

Rik Verdonck & Tom Wenseleers

`lpm_MAplot` *Plot an MA plot.*

Description

Make MA plot of `lpm_statlist` object. An MA plot is a Bland-Altman plot where for every peak pair, the average (A) log2 quantity value of a feature is plotted in function of the difference (M) of log2 quantity. This is the ideal way of visualizing possible label bias over the entire continuum of quantities. Bias always needs some further investigation, but should, given a sufficient number of replicates, never be a big reason to worry. The bias can either be dealt with by accounting for label effect in a linear model (see `lpm_linearmodel`) or by normalizing the data within and between runs (see `limma`). The function also draws a loess fit.

Usage

```
lpm_MAplot(x, loess_span = 0.75, run = NULL)
```

Arguments

`x`	An object of class `lpm_statlist`
`loess_span`	Numeric. Span of the loess fit through the MA plot. See `loess`.
`run`	Integer. Choose the run that you want to visualize. If not specified, all runs are printed.

Author(s)

Rik Verdonck

lpm_mockdata *Generate mock data.*

Description

Generates mock data on the basis of an `lpm_input` object. This function is integrated in `pepmatch` FDR estimation with its default parameters. The use outside of the `pepmatch` function is only advised if you want to explore this function, or if you want to use non-default parameters. The way this function works, is by chopping up values of masses (or actually of m/z) and retention times, randomizing them, and pasting them back together. This is important since mass/charge values are non-random (since masses are often close to integer values) and also the coupling of retention time with mass/charge is non-random. For example, mass values 1155.2 and 2456.2 will be reshuffled to 1456.2 and 2115.2 with a masslevel of 1000.

Usage

```
lpm_mockdata(input, masslevel = 100, retlevel = 3,
  elutionunit = "minutes", graphics = F)
```

Arguments

`input`	An object of class `lpm_input`
`masslevel`	Numeric. Level at which the mass values should be shopped up.
`retlevel`	Numeric. Level at which the retention time values should be shopped up. Careful: it is best to either use small values (like up to maximum 10% of the range of retention times) for a tight coupling between m/z and retention time, or a value that is larger than the maximum retention time for a loose coupling between m/z and retention time. Intermediate values will chop up your data in chunks. Have a look at it with the graphics parameter on.
`elutionunit`	Character. Sketchy. Default is "minutes", alternative is "seconds".
`graphics`	Output a plot of the mock database versus the real database.

Value

An object of class `lpm_input`. The values for m/z and retention time are shuffeled. The quantity values are all set to 0.

Author(s)

Rik Verdonck

See Also

`pepmatch`

lpm_refine *Refine* `pepmatched` *objects.*

Description

This function allows to make subselections of a `pepmatched` object with cutoffs for different parameters. In this way, the `pepmatch` function itself only has to be run once with not to strict parameters, and analysis can afterwards be refined. Notice that it is impossible to refine parameters to less strict values than the ones used to generate the pepmatched object.

Usage

```
lpm_refine(pepmatched_object, labelthresh, elutionthresh, MWmin, MWmax,
  quantmin, quantmax, labelcountmin, labelcountmax, zmin, zmax,
  remove.more.labels.than.charges = F, remove.run = NULL,
  only.identified = FALSE)
```

Arguments

`pepmatched_object`	Object of class `pepmatched` that you want to trim.
`labelthresh`	Numeric. Threshold for molecular weight difference (in Dalton) between to peaks to differ from theoretical mass difference between two labelled peptides. Regardless of number of labels.
`elutionthresh`	Numeric. Threshold for elution time difference between two peaks to be considered a peakpair.
`MWmin`	Numeric. Minimal molecular weight of the peptide without labels.
`MWmax`	Numeric. Maximal molecular weight of the peptide without labels.
`quantmin`	Numeric vector of one or two elements. Minimal untransformed quantity for a feature to be retained. The highest value of the vector is the cutoff for the most abundant peak in a pair, the lowest for the least abundant. If you want to make sure you pick up extreme up-or downregulations, the vector should contain one zero or very small number. If you want to discart a peak pair once one of the peaks is below a quantity threshold, you can set the threshold with one minimal value (a vector with one element). This is equivalent to a vector with two equal elements.
`quantmax`	Numeric. Maximal quantity (intensity or abundance). This is an AND function, so both peaks in a peak pair have to be above this quantity in order to be discarted.
`labelcountmin`	Integer. Minimal number of labels.
`labelcountmax`	Integer. Maximal number of labels.
`zmin`	Integer. Minimal number of charges.
`zmax`	Integer. Maximal number of charges.

`remove.more.labels.than.charges`	Logical. Remove features that have more labels than charges. This is usually relevant since most labels are charged, so finding a peptide that has more labels than charges is often impossible.
`remove.run`	Integer. A single value or a vector of runnumbers that should be discarted.
`only.identified`	Logical. Only features that have been identified with mass match are retained.

`lpm_summary` *Summary lpm objects.*

Description

This is a summary function that gives insightful summaries of lpm specific objects. These include objects of class `lpm_input`, `pepmatched` (both with or without mass matched peptides) and `lpm_statlist`.

Usage

```
lpm_summary(input, graphics = F, run = 1, printoutput = T)
```

Arguments

`input`	A labelpepmatch specific object of class `lpm_input`, `pepmatched` or `lpm_statlist`
`graphics`	Logical. Outputs a graphics window with several summaries for one run.
`run`	Integer. If graphics is TRUE, you can here choose the run that you want to visualize.
`printoutput`	Logical. If you just want to use the function for graphics, you can aks it not to print anything in your R-session.

Author(s)

Rik Verdonck

`lpm_volcanoplot` *Plot a volcano plot.*

Description

Make a volcanoplot from an `lpm_linearmodel` object. Here, every feature is plotted in the log2(fold change) * -log10(p-value) space. It is a very convenient way of visualizing multiple tests in one plot.

Usage

```
lpm_volcanoplot(x, adjusted = T, plotlocator = F)
```

Arguments

`x`	An object of class `lpm_linearmodel`
`adjusted`	Logical. If TRUE, adjusted p-values are used. If FALSE, raw p-values are used.
`plotlocator`	Logical. If TRUE, you can click on the plot and the identity of the nearest feature will be printed into the R-session. Click under the plot to return to your normal R-session.

Author(s)

Rik Verdonck

`make.statlist` *Reorganize a* `pepmatched` *object into matrix format.*

Description

This function takes an object of class `pepmatched` and transforms it to class `lpm_statlist`. This statlist is a format that is suited for statistical analysis. It also contains the same trimming parameters as the `lpm_refine` function.

Usage

```
make.statlist(pepmatched_object, cutoff = 1, logtransform = T, labelthresh,
  elutionthresh, MWmin, MWmax, quantmin, quantmax, labelcountmin, labelcountmax,
  zmin, zmax, remove.more.labels.than.charges = F, remove.run = NULL,
  only.identified = FALSE)
```

Arguments

`pepmatched_object`	Object of class `pepmatched` that you want to trim.
`cutoff`	Numeric. Proportion of runs in which a feature should be found to be retained in statlists. Default is 1
`logtransform`	Logical. Should the data be log2 transformed? Set to FALSE if data are already transformed in earlier step, or if you want to manually transform your data. Otherwise this is the best place to log2 transform your data.
`labelthresh`	Numeric. Threshold for molecular weight difference (in Dalton) between to peaks to differ from theoretical mass difference between two labelled peptides. Regardless of number of labels.
`elutionthresh`	Numeric. Threshold for elution time difference between two peaks to be considered a peakpair.
`MWmin`	Numeric. Minimal molecular weight of the peptide without labels.
`MWmax`	Numeric. Maximal molecular weight of the peptide without labels.

`quantmin`	Numeric. Minimal quantity (intensity or abundance). This is an AND function, so both peaks in a peak pair have to be below this quantity in order to be discarted.
`quantmax`	Numeric. Maximal quantity (intensity or abundance). This is an AND function, so both peaks in a peak pair have to be above this quantity in order to be discarted.
`labelcountmin`	Integer. Minimal number of labels.
`labelcountmax`	Integer. Maximal number of labels.
`zmin`	Integer. Minimal number of charges.
`zmax`	Integer. Maximal number of charges.
`remove.more.labels.than.charges`	Logical. Remove features that have more labels than charges. This is usually relevant since most labels are charged, so finding a peptide that has more labels than charges is often impossible.
`remove.run`	Integer. A single value or a vector of runnumbers that should be discarted.
`only.identified`	Logical. Only features that have been identified with mass match are retained.

Value

An object of class `lpm_statlist`

Author(s)

Rik Verdonck

See Also

`lpm_refine` `lpm_make.RGList`

`pep.id` *Mass match peak pairs from a pepmatched object.*

Description

Mass match peak pairs from a pepmatched object to known databases. Can call inbuilt databases, but you can also use your own local database.

Usage

```
pep.id(pepmatched, ID_thresh = 10, db, presetdb = NA, dbpath = NA,
  dbseparator = ",", dbheader = FALSE, masscorrection = FALSE,
  cores = 1, FDR = TRUE, iterations = 100, checkdb = F, graphics = F,
  verbose = FALSE)
```

Arguments

`pepmatched`	Input object of class `pepmatched`, generated by the pepmatch() function of this package.
`ID_thresh`	Numeric. Maximal allowed mass difference in ppm for identification.
`db`	In case no preset database is chosen: a database (table) with the first 3 columns `"name"`,`"MW"` and `"sequence"` (as read in with the `download_lpm_db` function) The amino acid sequences have to obey rules when you want to use them for mass recalculation or generation of a decoy database. Amino acids have to be capitalized one letter codes. For details, see `calculate_peptide_mass`
`presetdb`	A preset database. For the future (not ready yet)
`dbpath`	Character. In case a local database is used. Should be the filepath of a table, separated by dbseparator, with first tree columns: name, mass in Dalton and sequence.
`dbseparator`	Character. Column separator of database.
`dbheader`	Logical. Does the database have a header? Default FALSE
`masscorrection`	Logical. Should masses be corrected on the basis of identifications? Caution should be payed here, since this will only work with a sufficiently high number of real matches. This rather sketchy feature should be considered something interesting to have a look at, rather than a compulsory part of the pipeline. Always also run without mass correction and compare! Default is FALSE.
`cores`	Interger. Number of cores that can be used on the computer for calculation. When >1, the packages foreach and doParallel will be used for multithreading.
`FDR`	Logical. Test false discovery rate of peptide mass match identification by calculating a decoy database and look how many hits you get there. Uses `generate_random_db`.
`iterations`	How many iterations of FDR should be ran? This might take some time for larger datasets.
`checkdb`	Look if the masses and sequences in your database make sense. UNDER CONSTRUCTION!!!
`graphics`	Only applies when FDR is TRUE.
`verbose`	Logical. If TRUE, verbose output is generated during identifications.

Value

An object of class `pepmatched` with added mass matches that can be used for subsequent analysis using labelpepmatch functions. This pepmatched object will also contain a couple of extras like detailed pep.id parameters, details about the FDR estimation, a list of identified peptides with their counts, and a deltavector with the mass shifts in case the mass correction has been applied.

Author(s)

Rik Verdonck & Gerben Menschaert

See Also

`generate_random_db`, `pep.massmatch`

`pep.massmatch`	*Mass match a vector of masses to a database.*

Description

Mass match peak pairs from a pepmatched object to known databases. Can call inbuilt databases, but you can also use your own local database.

Usage

```
pep.massmatch(input, db, presetdb = NA, dbpath = NA, dbseparator = ",",
  dbheader = FALSE, ID_thresh = 10, masscorrection = F, FDR = F,
  iterations = 10, checkdb = F, graphics = F, verbose = F)
```

Arguments

`input`	Numeric. Either a single value, a vector of values, or a dataframe or matrix with one column with name MW
`db`	In case no preset database is chosen: a database (table) with the first 3 columns `name, MW` and sequence (as read in with the `download_lpm_db` function) The amino acid sequences have to obey rules when you want to use them for mass recalculation or generation of a decoy database. Amino acids have to be capitalized one letter codes. For details, see `calculate_peptide_mass`
`presetdb`	A preset database. For the future (not ready yet)
`dbpath`	Character. In case a local database is used. Should be the filepath of a table, separated by dbseparator, with first tree columns: name, mass in Dalton and sequence.
`dbseparator`	Character. Column separator of database.
`dbheader`	Logical. Does the database have a header? Default FALSE
`ID_thresh`	Numeric. Maximal allowed mass difference in ppm for identification.
`masscorrection`	Logical. Should masses be corrected on the basis of identifications? Caution should be payed here, since this will only work with a sufficiently high number of real matches. This rather sketchy feature should be considered something interesting to have a look at, rather than a compulsory part of the pipeline. Always also run without mass correction and compare!!! Default is FALSE.
`FDR`	Logical. Test false discovery rate of peptide mass match identification by calculating a decoy database and look how many hits you get there. Uses `generate_random_db`.
`iterations`	How many iterations of FDR should be ran? This might take some time for larger datasets.
`checkdb`	Look if the masses and sequences in your database make sense. UNDER CONSTRUCTION!!!
`graphics`	Only applies when FDR is TRUE.
`verbose`	Logical. If TRUE verbose output is generated during identifications.

Value

An object of class `pep_massmatched` that can be used for subsequent analysis using labelpepmatch functions.

Author(s)

Rik Verdonck & Gerben Menschaert

See Also

`pep.id`, `pep.massmatch`

`pepmatch` *Identify peak pairs.*

Description

Identifies pairs of labelled peptides in an `lpm_input` object.

Usage

```
pepmatch(lpm_input, elutionthresh = 0.2, elutionunit = "minutes",
  labelthresh = 0.1, labelcountmax = 6, labellightmass, labelheavymass,
  label = "unspecified", minmolweight = 132, quantmin = 0, FDR = T,
  iterations = 10, cores = 1, verbose = FALSE)
```

Arguments

`lpm_input`	Input object of class "LPM_input", generated by the read functions of labelpepmatch, or example data object present in package.
`elutionthresh`	Numeric. Threshold for elution time difference between two peaks to be considered a peakpair. Default is 0.2 (minutes)
`elutionunit`	Character. Default is "minutes", alternative is "seconds". This is sketchy, only just for FDR estimation.
`labelthresh`	Numeric. Threshold for molecular weight difference (in Dalton) between to peaks to differ from theoretical mass difference between two labelled peptides. Regardless of number of labels. Default is 0.1.
`labelcountmax`	Integer. Maximal number of labels allowed. Default is 6.
`labellightmass`	Numeric. Mass of light label. Default is 128.1177, the mass of light TMAB
`labelheavymass`	Numeric. Mass of heavy label. Default is 137.1728, the mass of heavy TMAB
`label`	Character. Optional argument. If it is "TMAB", automatically the right values for light and heavy TMAB are used. Can be extended in the future with newer labels.
`minmolweight`	Numeric. Minimal molecular weight for a peptide to be retained. Default is 132, the smallest possible peptide.

`quantmin`	Numeric vector of one or two elements. Minimal untransformed quantity for a feature to be retained. The highest value of the vector is the cutoff for the most abundant peak in a pair, the lowest for the least abundant. If you want to make sure you pick up extreme up-or downregulations, the vector should contain one zero or very small number. If you want to discart a peak pair once one of the peaks is below a quantity threshold, you can set the threshold with one minimal value (a vector with one element). This is equivalent to a vector with two equal elements.
`FDR`	Logical. Generates mock data according to a restricted randomization procedure that produces mock data with a very comparable structure to the original data. Structure elements that are retained are the dispersion of features in the retention time - m/z space and the structure of decimals for m/z. This is important because decimals are non random in m/z data. For more information see `lpm_mockdata`.
`iterations`	Integer: the number of iterations to estimate the FDR. Careful: for large datasets this can take a long time! A modest number (3 to 5) of iterations will already give a good indication of FDR.
`cores`	Interger. Number of cores that can be used on the computer for calculation. When >1, the packages foreach and doParallel will be used for multithreading. Optimally, the number of cores is a multiple of the number of runs. Since every run makes up a single thread, it makes no sense to use more cores than the number of runs.
`verbose`	Logical. Gives verbose output.

Details

This is the central function of the labelpepmatch package. Candidate matching features are those features that have the same charge, that co-elute within a given time interval and that have a mass difference of an integer multiple (the number of labels) of the mass difference between two labels. However, only m/z values are given, and masses have to be calculated first (deconvolution). Note that in the case one or more labels are present, the deconvoluted mass will not be correct because an unknown number of charges originate from labels rather than from protons. However, this error is systematic and would by definition be the same for two features within a peak pair. In other words, the mass difference within a peak pair is still calculated correctly. The function follows a simple algorithm where features are first sorted based on their preliminarily deconvoluted masses and retention time. Every feature is then matched to a set of equally charged features that fall within the same retention time window. Once a peak pair is detected, the number of labels is inferred from the mass difference and the correct mass of the peptide is recalculated.

Value

An object of class `pepmatched` without mass matchings. This object can serve as an imput to the `pep.id` function, or in case of no mass matching, can go directly in `make.statlist`. If FDR is TRUE, it also contains a summary of the false discovery rate: the mean, median, minimal and maximal number of false positive hits per run over all iterations, along with their proportion to the presumed real positives. Finally, if FDR is TRUE, the pepmatched object contains all the mass match precisions of the false positive hits along with the actual mockdata used for FDR estimation.

Author(s)

Rik Verdonck & Gerben Menschaert

See Also

`lpm_mockdata`

`QC_database` *Quality check a database.*

Description

Verify if masses and sequences in database match.

Usage

```
QC_database(db)
```

Arguments

`db`	A database with the first 3 columns `"name"`,`"MW"` and `"sequence"` (as read in with the `download_lpm_db` function)

Value

A vector with differences between observed and theoretical masses. Plots the observed and theoretical masses onto each other.

Author(s)

Rik Verdonck

See Also

`calculate_peptide_mass`

`read.labelpepmatch` *Read labelpepmatch input.*

Description

Read in data in a standard format for labelpepmatch.

Usage

```
read.labelpepmatch(indata = file.choose(), designvector, sep = "\t",
  dec = ".", samplenames = NA, verbose = FALSE)
```

Arguments

`indata`	Character. A table with standard columns: id, z, mz * n, quantity * n and retention time * n
`designvector`	Character. A vector with the design of the labelling. This is a vector that contains in this exact order: name of first condition, name of second condition, labelling order of first sample, labelling order of second sample, ... labelling order of last sample. The labelling order can either be `F` (forward, first condition is light), or `R` (reverse, first condition is heavy).
`sep`	Character. The field separator of the data you read in. Default is `\t`
`dec`	Character. The decimal separator of the data you read in. Default is "."
`samplenames`	Character. A vector with all the names of the samples.
`verbose`	Logical. Gives verbose output.

Value

An object of class `lpm_input` that can serve as an imput to the `pepmatch` function.

Author(s)

Rik Verdonck

`read.progenesis` *Read progenesis data.*

Description

Read in data generated with progenesis LC-MS. Use the standard output function in progenesis and write your file to a table or csv. Do not bother about extra statistics columns. Normalizing the data is not neccesary.

Usage

```
read.progenesis(indata = file.choose(), designvector, samplenames = NA,
  sep = ",", dec = ".", intORabu = "int", chargename = "Charge",
  abundname = "Raw abundance", intensname = "Intensity",
  rettimename = "Sample retention time (min)", verbose = FALSE)
```

Arguments

`indata`	Character. A table exported from progenesis LC-MS. Either reads a filepath, or reads the file directly if you are in the right folder. If no indata is specified, a prompt is presented.
`designvector`	Character. A Character vector of length N + 2 with N = number of runs. A vector with the design of the labelling. This is a vector that contains in this exact order: name of first condition, name of second condition, labelling order of first sample, labelling order of second sample, ... labelling order of last sample. The labelling order can either be `F` (forward, first condition is light), or `R` (reverse, first condition is heavy).
`samplenames`	Character. A character vector with the names of your samples. The length should be the number of runs.
`sep`	Character. The field separator of the data you read in. Default is ","
`dec`	Character. The decimal separator of the data you read in. Default is "."
`intORabu`	Character. Can be "int" or "abu", default "int". Will you work with the intensity (height of the peaks) or the abundance (surface under the peaks) of the data?
`chargename`	Character. Name of the charge columns. Default is `Charge`
`abundname`	Character. Name of abundance columns header. Default is `Raw abundance`
`intensname`	Character. Name of intensity columns header. Default is `Intensity`
`rettimename`	Character. Name of retention time columns header. Default is `Sample retention time (min)`
`verbose`	Logical. Gives verbose output.

Value

An object of class `lpm_input` that that can be used for subsequent analysis using labelpepmatch functions. Feeds straigth in the `pepmatch` function.

Author(s)

Rik Verdonck & Gerben Menschaert

`view_spectra` *Plot spectra in m/z - retention time space.*

Description

This is a plot function that plots "top views" of LC-MS spectra.

Usage

```
view_spectra(lpm_input, matched = NULL, run = NULL, pch = 20)
```

Arguments

`lpm_input`	An `lpm_input` object
`matched`	Optional: A `pepmatched` object that corresponds to the `lpm_input` object. Will color-code the location (mass matched) peak pairs in the plot.
`run`	Integer. Choose the run that you want to visualize. If not specified, all runs are printed.
`pch`	Either an integer specifying a symbol or a single character to be used as the default in plotting points.

Author(s)

Rik Verdonck

`wormdata` *Dataset with TMAB labelled worm peptides.*

Description

Dataset with TMAB labelled worm peptides.

Usage

```
wormdata
```

Format

```
List of 2
 $ frame :'data.frame': 5112 obs. of  26 variables:
  ..$ id        : num [1:5112] 52 77 98 109 328 336 392 406 408 412 ...
  ..$ z         : num [1:5112] 2 2 2 2 2 2 2 2 2 2 ...
  ..$ mz_1      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_2      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_3      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_4      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_5      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_6      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_7      : num [1:5112] 300 357 316 385 437 ...
  ..$ mz_8      : num [1:5112] 300 357 316 385 437 ...
  ..$ Quantity_1: num [1:5112] 8337600 2784898 12070 4357060 784775 ...
  ..$ Quantity_2: num [1:5112] 8493706 3092316 7645 2888636 693567 ...
  ..$ Quantity_3: num [1:5112] 4930026 2139606 1238075 3368120 672805 ...
  ..$ Quantity_4: num [1:5112] 3728062 3711008 4214 1648094 1156652 ...
  ..$ Quantity_5: num [1:5112] 960269 1872255 4620346 355810 904497 ...
  ..$ Quantity_6: num [1:5112] 1199628 2023318 1203373 576177 806132 ...
  ..$ Quantity_7: num [1:5112] 617110 1618121 3585914 239157 581701 ...
  ..$ Quantity_8: num [1:5112] 547073 1077971 3928426 289456 585546 ...
  ..$ Ret_1     : num [1:5112] 23.4 29 35.8 19.8 36.7 ...
  ..$ Ret_2     : num [1:5112] 23.2 29.1 35.7 19.8 36.5 ...
  ..$ Ret_3     : num [1:5112] 23.4 29.1 35.7 19.9 36.6 ...
  ..$ Ret_4     : num [1:5112] 23.5 29.4 35.9 20.1 36.8 ...
  ..$ Ret_5     : num [1:5112] 23.6 29.4 36 20.1 36.9 ...
  ..$ Ret_6     : num [1:5112] 23.5 29.4 35.9 20.1 36.8 ...
  ..$ Ret_7     : num [1:5112] 23.6 29.4 35.9 20.1 36.8 ...
  ..$ Ret_8     : num [1:5112] 23.5 29.4 35.9 20.1 36.8 ...
 $ design:'data.frame': 8 obs. of  5 variables:
  ..$ RunName       : Factor w/ 8 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8
  .. ..- attr(*, "names")= chr [1:8] "1" "2" "3" "4" ...
  ..$ LightCondition: Factor w/ 2 levels "AEX5","N2": 2 2 2 2 1 1 1 1
  .. ..- attr(*, "names")= chr [1:8] "1" "2" "3" "4" ...
  ..$ HeavyCondition: Factor w/ 2 levels "AEX5","N2": 1 1 1 1 2 2 2 2
  .. ..- attr(*, "names")= chr [1:8] "1" "2" "3" "4" ...
  ..$ Direction     : Factor w/ 2 levels "F","R": 1 1 1 1 2 2 2 2
  ..$ FileName     : Factor w/ 1 level "AEX5_20150429_editedfeatures.csv": 1 1 1 1 1 1 1 1
 - attr(*, "class")= chr "lpm_input"
```

Details

These are peptides from entire body homogenates of adult (*Caenorhabditis elegans*) worms labelled with light (D0) and heavy (D9) TMAB. The conditions are N2 and AEX5: Bristol N2 wild type, and aex5 mutants. Check the design matrix `wormdata$design` to have an overview of the labeling.

Author(s)

Wouter Dehaes

References

Will be published soon.

Index