Package: wrTopDownFrag (via r-universe)

October 18, 2024

Title Internal Fragment Identification from Top-Down Mass Spectrometry

Version 1.0.2

Author Wolfgang Raffelsberger [aut, cre]

Maintainer Wolfgang Raffelsberger <w.raffelsberger@unistra.fr>

Description Top-Down mass spectrometry aims to identify entire proteins as well as their (post-translational) modifications or ions bound (eg Chen et al (2018) <doi:10.1021/acs.analchem.7b04747>). The pattern of internal fragments (Haverland et al (2017) <doi:10.1007/s13361-017-1635-x>) may reveal important information about the original structure of the proteins studied (Skinner et al (2018) <doi:10.1038/nchembio.2515> and Li et al (2018) <doi:10.1038/nchem.2908>). However, the number of possible internal fragments gets huge with longer proteins and subsequent identification of internal fragments remains challenging, in particular since the the accuracy of measurements with current mass spectrometers represents a limiting factor. This package attempts to deal with the complexity of internal fragments and allows identification of terminal and internal fragments from deconvoluted mass-spectrometry data.

Depends R (>= 3.1.0)

VignetteBuilder knitr, rmarkdown

Imports graphics, grDevices, stats, utils, wrMisc, wrProteo

Suggests BiocParallel, boot, data.tree, fdrtool, knitr, limma, parallel, preprocessCore, RColorBrewer, rmarkdown, wrGraph

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Date/Publication 2020-09-08 08:10:02 UTC

Repository https://wraff.r-universe.dev

RemoteUrl https://github.com/cran/wrTopDownFrag

RemoteRef HEAD

RemoteSha d6b00f3a649a7d820013a860b5b1f91f867815bd

Contents

`AAfragSettings` *Settings for AA fragments*

Description

This function provides basic settings for what types of fragments may accomodate which type of modifications : $knownMods: information about which modifications may be considered, $specAAMod: specifc AA sites (if applicable), $specAAMod: specifc AA sites (if applicable). For example, here 'p' codes for gain of mass for HPO3 only at S, T and Y residues. Note: $knownMods$Nterm and $knownMods$Cterm are treated as mutually exclusive

Usage

```
AAfragSettings(outTy = "all")
```

Arguments

`outTy`	(character) default "all" or any of the list-elements

Value

list ($knownMods, $knspecAAMods, $modChem, $neutralLossOrGain)

See Also

makeFragments, fragmentSeq, massDeFormula

Examples

```
AAfragSettings()
```

---

addMassModif *Add modifications to peptide mass*

Description

Adjust/add mass for modifications from 'modTy' to all peptides in 'pepTab' based on count 'cou' of occurances of modifications : Either fixed or variable modifications will be added to the mass of initial peptides from argument papTab. Terminal ionization (like 'b' or 'y' -fragments) is treated as fixed modification and the resulting masses will correspond to standard mono-protonated ions. Since variable and fixed modification types can't be run in a single instance, the function has to get calles twice, it is recommended to always start with the fixed modfications, In the case of fixed modifications (like defining 'b' or 'y' fragments) neutral peptide masses should be given to add the corresponding mass-shift (and to obtain mono-protonated ions). In case of variable modifications (like 'd' or 'p'), the corresponding ions from the fixed modifications should get furnished to add the corresponding mass-shift, the masses resulting from the initial fixed modifications run can be used. Note, that transforming a neutral precursor M into MH+ is also considered a modification. The results are also correct with obligatory fragments that can't occur the same time (eg x & y ions can't be same time, need to make add'l lines...). This function has a multiprocessor mode, with small data-sets (like the toy example below) there is typcally no gain in performance.

Usage

```
addMassModif(
  cou,
  pepTab,
  combTerm,
  modTy,
  lastIndex = NULL,
  modChem = NULL,
  basVarMod = "basMod",
  massTy = "mono",
  knownMods = NULL,
  nProc = 1,
  parallDefault = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

cou	(list) list of matrixes with counts for number of modifications per peptide
pepTab	(matrix) table with peptide properties
combTerm	(matrix) table with separate rows for $basMod that are exclusive (ie can't be accumulated, eg x & y ions)
modTy	(character) list of modification types to be considered
lastIndex	(integer) index-1 (ie last index from prev matrix) from which new peptide-variants should start from
modChem	(character) optional modifications
basVarMod	(character) toggle if fixed ('basMod') or variable ('varMod') modificatons should be calculated
massTy	(character) default 'mono'
knownMods	(list) optional custom definition whoch modification is N-term, etc (see AAfragSettings
nProc	(integer) number of processors in case of multi-processor use (requires Bioconductor package BiocParallel)
parallDefault	(logical) for use of other/previously set register(bpstart()) in case .parCombinateAllAndSum is called
silent	(logical) suppress messages
debug	(logical) for bug-tracking: more/enhanced messages and intermediate objects written in global name-space
callFrom	(character) allows easier tracking of message(s) produced

Value

list of $pepTab (table of peptide as single charge positive ions), $abc ('representative' list of all combinations to add). Main result in $pepTab

See Also

convToNum

Examples

```
pep1 <- c(pe1="KPEPTI")
# The table of possible terminal fragments (for simplicity terminal only)
pepTab1 <- makeFragments(pep1, min=3, max=7, internFra=FALSE)
# Which fragment may be subject to how many modification (including ionization by H+)
cou1 <- countPotModifAAs(pepTab=pepTab1, modTy=list(basMod=c("b","y")))
# Add modifications (here: ionize all pepptides by H+)
preMa1 <- addMassModif(cou=cou1$cou, pepTab=pepTab1, combTerm=cou1$combTerm,
  modTy=list(basMod=c("b","y")), basVarMod="basMod")
preMa1

## Example including variable modifications
modT3 <- list(basMod=c("b","y"),varMod=c("p","h","d"))
cou3 <- countPotModifAAs(pepTab=pepTab1, modTy=modT3)
```

```
## Now we re-use/inject the results for the fixed modificatons
preMa3 <- addMassModif(cou=cou3$cou, pepTab=preMa1$pepTab, combTerm=cou1$combTerm,
  modTy=modT3, basVarMod="varMod")
head(preMa3$pepTab,12)
```

---

checkModTy *Check & complete mixed of variable and fixed modifications*

Description

Check & complete settings for mixed of variable and fixed modifications. The final format is a list with $basMod, $varMod and $varMo2

Usage

```
checkModTy(modTy, knownMods = NULL, silent = TRUE, callFrom = NULL)
```

Arguments

modTy	(character) list of modification types to be considered
knownMods	(character) optonal custom list of known modifications, default from AAfragSettings(outTy="all")$kn
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of message(s) produced

Value

corrected list of mixed of variable and fixed modifications ($basMod, $varMod and $varMo2)

See Also

AAfragSettings

Examples

```
modTy1 <- list(basMod=c("b","y","h"),varMod=c("p","o","q"))
checkModTy(modTy1)
```

`combinateAllAndSum` *Full combinatorial and cumulative values*

Description

Use for all preparing all combinations of non-compulsatory, ie variable, mass modifications Variable modifications may or may not be present. Thus, for a given amino-acid with a variable modification two versions of the molecular weight need to be considered. Most (variable) modifications are linked to a type of amino acid, like serine-residues for phosphorlylation. Thus in this case, each instance of the amino acid in question may or may not be modified. So, for example if there are 2 serines, 0, 1 or 2 phosphorylation modifications may be present. For this reason the is the argument `nMax` to stay within biologically relevant ranges (external knowledge) and reduce complexity significantly. Some modifications are exclusive to others, argument `notSingle` : An (artificially occuring) de-phosphorylation event during fragmentation can only happen if the amino acid was already phosphorylated in the first place.

Usage

```
combinateAllAndSum(
  nMax,
  modVal,
  notSingle = NULL,
  silent = TRUE,
  callFrom = NULL
)
```

Arguments

`nMax`	(integer or data.frame with 1 line) maximum number of modifications
`modVal`	(numeric, has to have names !) the change of molecular mass introduced by given modifications (as specified by the name of the value)
`notSingle`	(character) names of 'modVal' where 1st element of 'notSingle' cannot happen/appear if 2nd element not present (eg de-phospho/phosphorylation)
`silent`	(logical) suppress messages
`callFrom`	(character) allow easier tracking of message(s) produced

Value

named (concatenated names of modVal) numeric vector

See Also

`convToNum`

Examples

```
## to follow easily the results, hypothetical mass-modification values were chosen
mo1 <- c(a=10, b=1, c=0.1, d=0.01); nMa1 <- c(1,2,0,3)
combinateAllAndSum(nMa1, mo1)
## # like 'b' for phospho & 'd' for de-phospho (which can't happen without phospho event)
combinateAllAndSum(nMa1, mo1, notSingle=c("d","b"))
```

`countChildrenParent` *Identify Children/Parent settings as a+b=c*

Description

This functions helps identifying fragments ('parent') characterized by a start- and end-position, that got split into 2 'children' fragments. So, each one of the new 'children' conserves either the start- or end-site of the parent and the the remaining ends are on consecutive positions. For example if the sequence 'BCDEFG' (parent) gets split into 'BCD' (positions 1-3) and 'EFG' (positions 4-6), this will be identified as a children/parent 'family' which could be represented as 'a+b=c' case. Note : At this point only settings with 2 children are considered, for more complex scenarions one may build trees using `buildTree` (however, this function does not identify 'parents'). In proteomics-applications some start- and end-sites may occur multiple times, representing eg unmodified and modified versions of the same basal peptide-sequence. Such duplicated start- and end-cases are handeled as allowed, a 'child' (characterized by its start- and end-position) may occur multiple times, and the corresponding redundant rownames (eg peptide sequence like 'BCD') will be conserved. However, information reflecting eg different peptide modifications must be stored separately. If redudant start- and end-sites accur with different row-names, repeated start- and end-sites will display `NA`.

Usage

```
countChildrenParent(
  fragments,
  output = "count",
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

`fragments`	(matrix or data.frame) integer values in 1st column, for start site of fragment, and in 2nd column as end-sites of fragments, rownames as IDs
`output`	(character) choose simply returning results as counts or as list with `$counts` and `$detailIndex` (list with details showing each child1,child2 & parent)
`silent`	(logical) suppress messages
`callFrom`	(character) allows easier tracking of message(s) produced

Value

either numeric vector with cumulated counts (corresponding to rows of `fragments`) or list with $count and $detailIndex (list with indexes refering to non-redundant entries of all a+b=c settings identified)

See Also

`simpleFragFig` for graphical representation,`countSameStartEnd`; for building longer consecutive trees (without identification of 'parent') `buildTree` and `contribToContigPerFrag`

Examples

```
frag3 <- cbind(beg=c(4,2,3,7,13,13,15, 2,9,2,9), end=c(14,6,12,8,18,20,20, 8,12,12,18))
rownames(frag3) <- c("K","A","E","B","C","D","F", "H","G","I","J")
countChildrenParent(frag3)
## example with duplicate start- and end-position positions
frag3c <- cbind(beg=c(4,2,3,7, 7,13, 13,13,15, 2,9,2,9,9),
  end=c(14,6,12,8, 8,18, 18,20,20, 8,12,12,12,18))
rownames(frag3c) <- c("K","A","E", "B","B", "C","C","D","F", "H","G","I","G","J")
countChildrenParent(frag3c, out="det")
```

`countPotModifAAs` *Make table with counts of potential modification sites*

Description

Makes table 'cou' with counts of (potential) modification sites based on column 'seq' in matrix 'pepTab'. Note: if multiple N-or C-term modifs, then only the first is shown in resulting table 'cou'.

Usage

```
countPotModifAAs(
  pepTab,
  modTy,
  maxMod = c(p = 3, h = 1, k = 1, o = 1, m = 1, n = 1, u = 1, r = 1, s = 1),
  specAAMod = NULL,
  knownMods = NULL,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)
```

Arguments

pepTab	(matrix) peptide sequences, start and end sites, typically result from makeFragments
modTy	(list) modifications : $basMod for character vector of fixed modifications and $varMod for variable modifications. For one letter-code see AAfragSettings("modChem")
maxMod	(integer) maximal number variable modifications will be considered in given fragment (may increase complexity and RAM consumption)
specAAMod	(list) optional custom list showing which AA to be considered with which (one-letter) modification code (default AAfragSettings)
knownMods	(list) optional custom list showing which modification appears at what type of location, eg N-terminal, internal ... (default AAfragSettings)
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of message(s) produced
debug	(logical) for bug-tracking: more/enhanced messages and intermediate objects written in global name-space

Value

list of matrixes $cou and $combTerm, with number of modifications per peptides (line in 'pepTab') for basMod, varMod & varMo2

See Also

AAfragSettings, makeFragments

Examples

```
protP2 <- c(mesp="MESPEPTIDES", pepe="PEPEPEP")
pepTab1 <- makeFragments(protTab=protP2, minFra=6, internFr=TRUE, massTy="mono")
cou1 <- countPotModifAAs(pepTab=pepTab1, modTy=list(basMod=c("b","y"),
  varMod=c("p","h")), debug=FALSE)
modTy2 <- list(basMod=c("b","y","h"), varMod=c("x","p","o","q","e","j"))
cou2 <- countPotModifAAs(pepTab=pepTab1, modTy=modTy2)
```

---

evalIsoFragm *Evaluate selected lines of pepTab (iso-mass) for preferential cutting sites*

Description

Evaluate selected lines of pepTab (iso-mass) for preferential cutting sites. Such sites are taken by default from `.prefFragPattern()` simplified from a publication by the Kelleher group (Haverland 2017, J Am Soc Mass Spectrom) or can be furnished by the user.

Usage

```
evalIsoFragm(
  z,
  prefFragPat = NULL,
  seqCol = "seq",
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

z	(matrix) main input, must contain cols specified as seqCol and "no","tailAA","precAA"
prefFragPat	(matrix) specifies preferential fragmentation (which combination of AA to consider cols cTer,nTer,score), default made by `.prefFragPattern()`
seqCol	(character) column names for the column containing the sequence to search for preferential cutting sites
silent	(logical) suppress messages
callFrom	(character) allows easier tracking of message(s) produced

Value

line ID-numbers (pepTab[,"no"]) for those below median score (ie to remove from pepTab) or NULL if nothing to remove due to preferential fragmentation

See Also

makeFragments

Examples

```
peTab <- matrix(c("9","13","14","15", "LPVIAGHEAAG","PVIAGHEAAGI","EKKPFSI","KKPFSIE",
  "P","L","E","E", "I","V","E","E"),nr=4,dimnames=list(NULL,c("no","seq","precAA","tailAA")))
evalIsoFragm(peTab)
```

---

fragmentSeq *Fragment protein or peptide sequence*

Description

Makes internal/terminal fragments of a SINGLE peptide/protein input (as single letter amino-acid code) and returns list of all possible sequences ($full, $Nter, $Cter, $inter).

Usage

```
fragmentSeq(
  sequ,
  minSize = 3,
  maxSize = 300,
  internFragments = TRUE,
  separTerm = FALSE,
  keepRedSeqs = TRUE,
  prefName = NULL,
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

sequ	(character, length=1) sequence used for fragmenting, as as mono-aminoacid letter code (so that cuting will be perfomed between all the letters/characters)
minSize	(integer) min number of AA residues for considering peptide fragments
maxSize	(integer) max number of AA residues for considering peptide fragments
internFragments	(logical) logical (return only terminal fragments if 'FALSE')
separTerm	(logical) if 'TRUE', separate N-terminal, C-terminal and internal fragments in list
keepRedSeqs	(logical) if 'FALSE' remove fragments with redundant content (but my be from different origin in 'sequ'); remove redundant so far only when no separation of Nterm/Cterm/intern as list
prefName	(logical) alternative name for all fragments (default the sequence itself), avoid separators '.' and '-'
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of message(s) produced

Value

numeric vector with mass

See Also

makeFragments; convAASeq2mass

Examples

```
fragmentSeq("ABCDE")
fragmentSeq("ABCDE", minSize=3, internFragments=FALSE)
fragmentSeq("ABCDE", minSize=3, internFragments=TRUE)

## Run multiple peptides/proteins
twoPep <- cbind(c("a","ABCABCA"), c("e","EFGEFGEF"))
```

```
apply(twoPep, 2, function(x) fragmentSeq(x[2], mi=3, kee=FALSE, sep=TRUE, pre=x[1]))

## Ubiquitin example
P0CG48 <- "MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLIFAGKQLEDGRTLSDYNIQKESTLHLVLRLRGG"
system.time( fra1 <- (fragmentSeq(P0CG48, mi=5, kee=FALSE)))      # < 0.5 sec
```

`identifFixedModif` *Identify Fixed Modifications*

Description

Identify peptide/protein fragment based on experimental m/z values 'expMass' for given range of aa-length. Internally all possible fragments will be predicted and their mass compared to the experimental values (argument `expMass`).

Usage

```
identifFixedModif(
  prot,
  expMass,
  minFragSize = 5,
  maxFragSize = 60,
  indexStart = 1,
  suplPepTab = NULL,
  internFra = TRUE,
  filtChargeCatch = TRUE,
  maxMod = c(p = 3, h = 1, k = 1, o = 1, m = 1, n = 1, u = 1, r = 1, s = 1),
  modTy = NULL,
  specModif = NULL,
  knownMods = NULL,
  identMeas = "ppm",
  limitIdent = 5,
  filtAmbiguous = FALSE,
  recalibrate = FALSE,
  chargeCatchFilter = TRUE,
  massTy = "mono",
  prefFragPat = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

`prot`	(character) amino-acid sequene of peptide or protein
`expMass`	(numeric) erperimental masses to identify peptides from

minFragSize	(integer) min number of AA residues for considering peptide fragments
maxFragSize	(integer) max number of AA residues for considering peptide fragments
indexStart	(integer) for starting at correct index (if not 1)
suplPepTab	(matrix) additional peptides to be add to theoretical peptides
internFra	(logical) decide whether internal fragments should be cosiered
filtChargeCatch	(logical) by default removing of all fragments not containing a (polar) charge-cathing residue
maxMod	(integer) maximum number of residue modifications to be consiered in fragments (values >1 will increase complexity and RAM consumption)
modTy	(character) type of fixed and variable modifications
specModif	(list) supplemental custom fixed or variable modifications (eg Zn++ at given residue)
knownMods	(character) optional custom alternative to `AAfragSettings(ou="all")$knownMods`
identMeas	(character) default 'ppm'
limitIdent	(character) thershold for identification in 'identMeas' units
filtAmbiguous	(logical) allows filtering/removing ambiguous results (ie same mass peptides)
recalibrate	(logical or numeric) may be direct recalibration-factor (numeric,length=1), if 'TRUE' fresh determination of 'recalibFact' or 'FALSE' (no action); final recalibration-factor used exported in result as $recalibFact
chargeCatchFilter	(logical) optionally remove all peptides not containing charge-catch AAs (K, R, H, defined via `.chargeCatchingAA()`)
massTy	(character) 'mono' or 'average'
prefFragPat	(numeric) pattern for preferential fragmentation (see also Haverland 2017), if `NULL` default will be taken (in function `evalIsoFragm`) from `.prefFragPattern()`
silent	(logical) suppress messages
debug	(logical) additional messages and objects exportet to current session for debugging
callFrom	(character) allow easier tracking of message(s) produced

Value

list, ie result of massMatch() on 'pepTab' and 'expMass'

See Also

`makeFragments`

Examples

```
protP <- c(protP="PEPTIDEKR")
obsMassX <- cbind(a=c(199.1077,296.1605,397.2082,510.2922,625.3192),
  b=c(227.1026,324.1554,425.2031,538.2871,653.3141),
  x=c(729.2937,600.2511,503.1984,402.1507,289.0666),
  y=c(703.3145,574.2719,477.2191,376.1714,263.0874))
rownames(obsMassX) <- c("E","P","T","I","D")       # all 1 & 7 ions not included
identP1 <- identifFixedModif(prot=protP, expMass=as.numeric(obsMassX), minFragSize=2,
  maxFragSize=7,modTy=list(basMod=c("b","y")))     # looks ok
identP2 <- identifFixedModif(prot=protP, expMass=as.numeric(obsMassX), minFragSize=2,
 maxFragSize=7, modTy=list(basMod=c("a","x"), varMod=c("h","o","r","m")))
 head(identP1$preMa,n=17)     # predicted masses incl fixed modif
 head(identP2$preMa,n=17)     # predicted masses incl fixed modif
```

---

makeFragments *Make terminal and internal fragments from proteins*

Description

Makes terminal and internal fragments based on protein-sequence and present as matrix including heading and/or tailing amino-acid or theoretical molecular mass of all fragments. As the number of theoretically possible fragments increases with the size of the peptide/protein treated it is recommended to adopt arguments like `masFragSize` to realizstic values for the type of mass spectrometer used, since efficient filtering will reduce considerably the amount of memory (RAM) needed and will improve overal performance.

Usage

```
makeFragments(
  protTab,
  minFragSize = 6,
  maxFragSize = 300,
  internFra = TRUE,
  knownMods = NULL,
  redRedundSeq = FALSE,
  prefFragPat = NULL,
  remNonConfPrefFragm = TRUE,
  ambigLab = c(duplSequence = "duplSequence", isoMass = "isoMass"),
  massTy = "mono",
  specModif = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

Arguments

protTab	(character or matrix) named vector of protein-seqences to fragment or matrix (character) with lines for initial proteins/peptides, cols as name/sequence/mass
minFragSize	(integer) minimum number of amino-acids for being considered
maxFragSize	(integer) maximum number of amino-acids for being considered
internFra	(logical) toggle if internal framents will be produced or not
knownMods	(character) optional custom alternative to `AAfragSettings(ou="all")$knownMods`
redRedundSeq	(logical) reduce redundant sequences to 1st appearance in all further treatments
prefFragPat	(matrix) for preferential fragmentation rules (see also `.prefFragPattern`)
remNonConfPrefFragm	(logical) allows to remove (peptide-)fragments non conform with preferential fragmentation rules (using `evalIsoFragm`)
ambigLab	(character) text-labels for ambiguities (first for duplicated sequences second for iso-mass)
massTy	(character) default 'mono' for mono-isotopic masses (alterative 'average')
specModif	(list) supplemental custom fixed or variable modifications (eg Zn++ at given residue)
silent	(logical) suppress messages
debug	(logical) for bug-tracking: more/enhanced messages
callFrom	(character) allow easier tracking of message(s) produced

Value

matrix with fragment sequence, mass, start- and end-position, heading and tailing AA (or NA if terminal fragment)

See Also

`makeFragments`; `evalIsoFragm`, from package wrProteo `convAASeq2mass`, `AAmass`, `massDeFormula`

Examples

```
protP <- c(protP="PEPTIDE")
pepT1 <- makeFragments(protTab=protP, minFragSize=2, maxFragSize=9, internFra=TRUE)
tail(pepT1)
```

plotNTheor *Plot the number of theoretical random fragments*

Description

This simple function allows plotting the expected number of theoretical fragments from random fragmentation of peptides/proteins (in mass spectrometry). Here, only the pure fragmentation without any variable fragmentation is considered, all fragment-sizes are included (ie, no gating). For simplicity, possible (variable) modifications like loss of neutrals, etc, are not considered.

Usage

```
plotNTheor(
  x,
  tit = "Number of term and intern fragm",
  xlab = "Number of aa",
  ylab = "",
  col = 2:3,
  log = "",
  mark = NULL,
  cexMark = 0.75
)
```

Arguments

x	(integer) length (in amino-acids) of input peptides/proteins to be considered
tit	(character) custom title
xlab	(character) custom x-axis label
ylab	(character) custom y-axis label
col	(character or integer) cutsom colors
log	(character) define which axis should be log (use "xy" for drawing both x- and y-axis as log-scale)
mark	(matrix) first column for text and second column for where it should be stated along the top border of the figure (x-coordinate)
cexMark	(numeric) cex expansion-factor for text from argument mark

Value

figure only

See Also

AAfragSettings

Examples

```
marks <- data.frame(name=c("Ubiquitin\n76aa", "Glutamate dehydrogenase 1\n501aa"),
  length=c(76,501))
plotNTheor(x=20:750, log="", mark=marks)
```

`scoreChargeCatch` *Scoring of charge catching potential for peptides*

Description

Make score based on cumulative search for AA with given potential to catch charge (H+, or optionally any charge). Note : at current cumulative scoring large peptides may get priviliged.

Usage

```
scoreChargeCatch(
  resTab,
  pepCol = "seq",
  scale01 = TRUE,
  chargeMode = "pos",
  silent = FALSE,
  callFrom = NULL
)
```

Arguments

`resTab`	(matrix or data.frame) matrix or data.frame of results for SINGLE protein (here only the column specified with argument 'pepCol' will be used)
`pepCol`	(character) column name of 'resTab' containing the peptide sequence to be scored
`scale01`	(logical) linear rescale output to maximum 1.0
`chargeMode`	(character) this value may be 'pos' (default) for the positively charged amino-acids K,R and H or, if this argument has any other value, than all charged amino-acids (K,R,H, S,T,N,Q, D,E, W and Y) will be considered.
`silent`	(logical) suppress messages
`callFrom`	(character) allow easier tracking of message(s) produced

Value

numeric vector with score for each peptide of resTab (even if `scale01=TRUE` minimum may be >0 if all peptides do contain charge-catching AAs)

See Also

`fragmentSeq`

Examples

```
resTa <- matrix(c(1:4,"PEPTID","PEPTIK","PEPTRK","AGV"), ncol=2,
  dimnames=list(NULL,c("predInd","seq")))
scoreChargeCatch(resTa)
```

Index