

# Package ‘wrTopDownFrag’

July 22, 2025

**Title** Internal Fragment Identification from Top-Down Mass Spectrometry

**Version** 1.0.4

**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](https://doi.org/10.1021/acs.analchem.7b04747)>). The pattern of internal fragments (Haverland et al (2017) <[doi:10.1007/s13361-017-1635-x](https://doi.org/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](https://doi.org/10.1038/nchembio.2515)> and Li et al (2018) <[doi:10.1038/nchem.2908](https://doi.org/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)

**Imports** graphics, grDevices, stats, utils, wrMisc (>= 1.15.3), wrProteo

**Suggests** BiocParallel, knitr, parallel, preprocessCore, RColorBrewer, rmarkdown, wrGraph

**License** GPL-3

**Encoding** UTF-8

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2025-04-21 08:10:02 UTC

## Contents

.chargeCatchingAA . . . . .	3
.chColNa . . . . .	4
.checkModTy . . . . .	4
.countLET . . . . .	5
.countModif . . . . .	6
.CtermPepCut . . . . .	7
.evalIsoFra . . . . .	9
.exNamesTyDeList . . . . .	10
.multMatByColNa . . . . .	11
.NtermPepCut . . . . .	12
.parCombinateAllAndSum . . . . .	13
.prefFragPattern . . . . .	14
.singleSpecModif . . . . .	15
.termPepCut . . . . .	16
AAfragSettings . . . . .	17
addMassModif . . . . .	18
checkModTy . . . . .	20
combinateAllAndSum . . . . .	21
combinatIntTable . . . . .	22
corInDelShift . . . . .	23
corMutShift . . . . .	24
countChildrenParent . . . . .	25
countPotModifAAs . . . . .	26
evalIsoFragm . . . . .	27
fragmentSeq . . . . .	28
identifFixedModif . . . . .	30
identifVarModif . . . . .	32
identifyPepFragments . . . . .	34
makeFragments . . . . .	36
modifFragmTabOutput . . . . .	38
plotFragmLoc . . . . .	39
plotMgfLike . . . . .	41
plotNTheor . . . . .	42
plotPrefFragPat . . . . .	43
randMassByMut . . . . .	44
randMassByStochastic . . . . .	45
scoreChargeCatch . . . . .	46
scoreFragments . . . . .	48
scorePrefFrag . . . . .	49
scoreProteinFragments . . . . .	50

## Index

52

---

.chargeCatchingAA      *Cite Charge Catching Amino-Acids*

---

### Description

Return a matrix with charge-catching amino-acids and their assumed strength. So far, the strength shown/used is set rather empirically.

### Usage

```
.chargeCatchingAA(  
  chargeMode = "pos",  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL  
)
```

### Arguments

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
debug	(logical) additional messages and objects exportet to current session for debugging
callFrom	(character) allow easier tracking of messages produced

### Value

This function returns a matrix with charge-catching amino-acids and their assumed strength

### See Also

[fragmentSeq](#)

### Examples

```
.chargeCatchingAA()
```

---

`.chColNa` *Check Column Names from Matrix Or data.frame*

---

**Description**

Check matrix or data.frame for containing columns with specified name and optionally remove other columns (by their names)

**Usage**

```
.chColNa(x, colNa = "index", rmCol = NULL, callFrom = NULL)
```

**Arguments**

<code>x</code>	(matrix or data.frame)
<code>colNa</code>	(character)
<code>rmCol</code>	(character)
<code>callFrom</code>	(character) allow easier tracking of message(s) produced

**Value**

This function returns a matrix or data.frame with adjusted columns

**See Also**

[scoreFragments](#)

**Examples**

```
ma1 <- matrix(1:6, nrow=2, dimnames=list(NULL, c("index", "aa", "bb")))
.chColNa(ma1)
.chColNa(ma1, colNa="zz", rmCol="aa")
```

---

`.checkModTy` *Check Modification Type*

---

**Description**

Check Modification Type

### Usage

```
.checkModTy(  
  modTy,  
  knownMods,  
  phoDePho = c("p", "q"),  
  modTyGr = c("basMod", "varMod"),  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL  
)
```

### Arguments

modTy	(character) list of modification types to be considered
knownMods	(character) optional custom list of known modifications, default from AAfragSettings(outTy="all")\$knownMods
phoDePho	(character) names of modifications that may be de-phosphorylated
modTyGr	(character) groups of modifications to consider (defaults used both 'basMod' and 'varMod')
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

### Value

This function returns the 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, knownMods=c("a", "b", "h", "o", "p", "q", "y"))
```

---

.countLET

*Count Letters*

---

### Description

Count how many time a given letter occurs in each element of 'seq'

### Usage

```
.countLET(sequ, countCh = "K", silent = FALSE, debug = FALSE, callFrom = NULL)
```

**Arguments**

sequ	(character) eg peptide sequence(s)
countCh	(character)
silent	(logical) suppress messages
debug	(logical) for bug-tracking: more/enhanced messages and intermediate objects written in global name-space
callFrom	(character) allow easier tracking of message(s) produced

**Value**

This function returns a numeric vector of counts for 'countCh' (single element !) in each element of 'seq'

**See Also**

[AAfragSettings](#), [makeFragments](#)

**Examples**

```
protP2 <- c(mesp="MESPEPTIDES", pepe="PEPEPEP")
.countLET(protP2, "P")
```

---

`.countModif`

*Count For All Proteins The Occurance Of Modification Types*

---

**Description**

Count for all protein 'sequ' the occurrence of modification types defined in list 'modTyp' (only if in names(specAAMod)).

**Usage**

```
.countModif(  
  sequ,  
  modTyp,  
  specAAMod,  
  knownMods,  
  detailedCount = FALSE,  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL  
)
```

### Arguments

sequ	(character) peptide sequence(s)
modTyp	(list) modifications : \$basMod for character vector of fixed modifications and \$varMod for variable modifications. For one letter-code see AAfragSettings("modChem")
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)
detailedCount	(logical)
silent	(logical) suppress messages
debug	(logical) for bug-tracking: more/enhanced messages and intermediate objects written in global name-space
callFrom	(character) allow easier tracking of message(s) produced

### Value

This function returns a 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")
modTy2 <- list(basMod=c("b", "y", "h"), varMod=c("x", "p", "o", "q", "e", "j"))
.countModif(pepTab1[,2], modTyp=c("b", "y"), specAAMod=AAfragSettings(outTy="all")$specAAMod,
  knownMods=AAfragSettings(outTy="all")$knownMods)
```

---

.CtermPepCut

*Make Named Character Vector Of Sequential C-Terminal Fragments*

---

### Description

Make named character vector of sequential C-terminal fragments.

### Usage

```
.CtermPepCut(
  pe,
  mi,
  se1 = ".",
  se2 = "-",
```

```

    mainName = NULL,
    indexOffs = NULL,
    silent = FALSE,
    debug = FALSE,
    callFrom = NULL
  )

```

### Arguments

pe	(character, length=1) sequence to be cut in sequential way
mi	(integer) min number of AA residues for considering peptide fragments; should be <= length(pe) (otherwise the full length of 'pe' ALWAYS returned !)
se1	(character, length=1) separators for adding numbers to specify partial/fragment locations
se2	(character, length=1) separators for adding numbers to specify partial/fragment locations
mainName	(character, length=1)
indexOffs	(logical) offset to add for custom numbering in names (numeric, length=1), ie '1' will already increase by +1
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allow easier tracking of messages produced

### Value

This function returns a numeric vector with mass(es) and sequence in name(s)

### See Also

more flexible/sophisticated see [.termPepCut](#); [makeFragments](#); [convAASeq2mass](#)

### Examples

```

## Ubiquitin example
P0CG48 <- "MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLIFAGKQLEDGRTLSDYNIQKESTLHLVLRLLGG"
cut1 <- .CtermPepCut(P0CG48, mi=3, mainName="P0CG48")
head(cut1); tail(cut1)

```



---

.evalIsoFra                      *Evaluate Selected Lines Of PepTab*

---

**Description**

Evaluate selected lines of pepTab of SAME AA-length AND iso-mass for preferential cutting sites.

**Usage**

```
.evalIsoFra(  
  x,  
  prefFragPat = NULL,  
  seqCol = "seq",  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL  
)
```

**Arguments**

- x                      (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
- debug                (logical) additional messages for debugging
- callFrom             (character) allow easier tracking of messages produced

**Value**

This function returns line ID-numbers (pepTab[, "no"]) for those below median score (ie to remove from pepTab)

**See Also**

[makeFragments](#)

**Examples**

```
pepTab <- 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"))  
.evalIsoFra(pepTab)
```

---

.exNamesTyDeList      *Reorganize List Of Peptide Fragments To Matrix*

---

### Description

This function allows reorganizing a list (of lists) of peptide fragments into matrix

### Usage

```
.exNamesTyDeList(
  x,
  subLiNames = c("full", "Nter", "Cter", "inter"),
  inclNo = TRUE,
  fullSeq = NULL,
  outCol = c("seq", "orig", "origNa", "ty", "seqNa", "beg", "end", "precAA", "tailAA",
    "ambig", "mass"),
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)
```

### Arguments

x	(list) list of lists with character vectors of sequences with names that can be parsed eg 'x.1-7' to extract 'beg' & 'end' otherwise ALL output will be NA (+message form extractLast2numericParts())
subLiNames	(character)
inclNo	(logical) add 1st col with number
fullSeq	(character) to reinject full sequence which may not be used in names of 'x' and not be in x[[1]][["full"]]
outCol	(character) columns to create in output
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of message(s) produced
debug	(logical) for bug-tracking: more/enhanced messages

### Value

This function returns 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

```
prot1 <- c(protP="KEPTIDE", pro2="MPRATE")
## fragment all target proteins
pep3 <- lapply(prot1, fragmentSeq, minSize=3, maxSize=5, internFragments=FALSE,
  separTerm=TRUE, keepRedSeqs=TRUE)
pepTab <- .exNamesTyDeList(pep3, fullSeq=prot1)
```

---

<code>.multMatByColNa</code>	<i>Multiply Values Of Matrix By Its Colnames And Sum By Row</i>
------------------------------	---

---

### Description

This function allows multiplying values of 'mat' by its colnames and (optionally) summing along rows.

### Usage

```
.multMatByColNa(
  mat,
  sumByRow = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

### Arguments

<code>mat</code>	(matrix) main input
<code>sumByRow</code>	(logical)
<code>silent</code>	(logical) suppress messages
<code>debug</code>	(logical) additional messages for debugging
<code>callFrom</code>	(character) allows easier tracking of messages produced

### Value

This functions returns a numeric vector or a matrix if `sumByRow=FALSE`

### See Also

[convToNum](#)

### Examples

```
mat1 <- 3 + matrix(1:4, ncol=2, dimnames=list(letters[1:2], c("3","2")))
.multMatByColNa(mat1)
.multMatByColNa(mat1, sumByRow=FALSE)
```

---

*.NtermPepCut**Make Named Character Vector Of Sequential C-Terminal Fragments*

---

**Description**

Make named character vector of sequential C-terminal fragments.

**Usage**

```
.NtermPepCut(
  pe,
  mi,
  se1 = ".",
  se2 = "-",
  mainName = NULL,
  sepNC = FALSE,
  indexOffs = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

<code>pe</code>	(character, length=1) sequence to be cut in sequential way
<code>mi</code>	(integer) min number of AA residues for considering peptide fragments; should be $\leq$ length(pe) (otherwise the full length of 'pe' ALWAYS returned !)
<code>se1</code>	(character, length=1) separators for adding numbers to specify partial/fragment locations
<code>se2</code>	(character, length=1) separators for adding numbers to specify partial/fragment locations
<code>mainName</code>	(character, length=1)
<code>sepNC</code>	(logical) if TRUE, separate fragments from both ends as \$Nter & \$Cter in list
<code>indexOffs</code>	(logical) offset to add for custom numbering in names (numeric, length=1), ie '1' will already increase by +1
<code>silent</code>	(logical) suppress messages
<code>debug</code>	(logical) additional messages for debugging
<code>callFrom</code>	(character) allow easier tracking of messages produced

**Value**

This function returns a numeric vector with mass(es) and sequence in name(s)

### See Also

more flexible/sophisticated see [.termPepCut](#); [makeFragments](#); [convAASeq2mass](#)

### Examples

```
## Ubiquitin example
P0CG48 <- "MQIFVKLTGTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLIFAGKQLEDGRTLSDYNIQKESTLHLVLRLLGG"
cut1 <- .CtermPepCut(P0CG48, mi=3, mainName="P0CG48")
head(cut1); tail(cut1)
#' @export
```

---

.parCombinateAllAndSum

*Multiprocessor Version For Full Combinatorial And Cumulative Values*

---

### Description

This function combines all variants and sums them

### Usage

```
.parCombinateAllAndSum(
  uniqCo,
  massModV,
  nProc = NULL,
  firstOfRepeated = NULL,
  parRegDefault = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

### Arguments

uniqCo	(matrix) number of modifications to be considered for each peptide
massModV	(named numeric) mass modification values (names must match colnames of uniqCo)
nProc	(integer) number of processors to be used
firstOfRepeated	(character)
parRegDefault	(logical) - argument currently not in use
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

**Details**

This function requires the packages 'parallel' and 'BiocParallel' (from Bioconductor) Note : The function may work only on some Windows systems or may give warnings on Windows

**Value**

This functions returns a list with single and combined mass-modifications (PTMs) for each peptide

**See Also**

[convToNum](#)

**Examples**

```

uniqCo <- matrix(c(1,1,1,0,1,1), nrow=2, dimnames=list(c("PTI","KPE"),c("d","p","h"))) )
massModV <- c(d=-18.01056, p=79.96633, h=-18.01056)
chPa <- c(requireNamespace("parallel", quietly=TRUE),
  requireNamespace("BiocParallel", quietly=TRUE), "windows" %in% .Platform$OS.type)
## Note : the function may work only on some windows systems
if(all(chPa)) if(parallel::detectCores() >1) {
  .parCombinateAllAndSum(uniqCo, massModV, nProc=2)}

```

---

*.prefFragPattern*      *Return data.frame with pattern of preferential fragmentation sites*

---

**Description**

Return data.frame with pattern of preferential fragmentation sites Here a simplified version (elaborate see Kelleher group: Haverland 2017, J Am Soc Mass Spectrom)

**Usage**

```
.prefFragPattern(silent = FALSE, debug = FALSE, callFrom = NULL)
```

**Arguments**

<code>silent</code>	(logical) suppress messages
<code>debug</code>	(logical) additional messages for debugging
<code>callFrom</code>	(character) allow easier tracking of messages produced

**Value**

This function returns a data.frame with pattern of preferential fragmentation sites

**See Also**

[makeFragments](#)

### Examples

```
pepTab <- 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")))
head(.prefFragPattern())
```

.singleSpecModif      *Add Single Specific Modifications*

### Description

Add single specific modification to peptide/protein fragments .

### Usage

```
.singleSpecModif(
  pepTab,
  specModif,
  nMaxMod = 1,
  massTy = "mono",
  callFrom = NULL,
  silent = FALSE,
  debug = FALSE
)
```

### Arguments

- pepTab            (matrix) matrix of fragments (cols 'no','seq','orig','ty','seqNa','beg','end','precAA','tailAA','ambig','ma
- specModif        (list) with elements 'modOrigin' (sequence), 'modPos' (position within sequence), 'modMass' (digits, ie mass to add), 'modName' (name of modif), 'modFixed' (fixed or , logical)
- nMaxMod          (numeric) max number a given modification may occur
- massTy            (character) 'mono' or 'average'
- callFrom         (character) allow easier tracking of message(s) produced
- silent            (logical) suppress messages
- debug            (logical) additional messages and objects exportet to current session for debugging

### Value

This function returns a list with \$massMatch (list of exerimental peptides matching to one or more predicted), \$preMa (predicted ions, including fixed modif), \$pepTab (predicted neutral peptides, wo modifications), \$expMa (experimental mass from input), \$recalibFact (recalibration factor as from input), \$docTi (time for calculations)

**See Also**

[makeFragments](#), [identifVarModif](#), [identifyPepFragments](#)

**Examples**

```
pep1 <- c(pe1="KPEPTI")
# The table of possible terminal fragments (for simplicity terminal only)
pepTab1 <- makeFragments(pep1, min=3, max=7, internFra=FALSE)
specModif1 <- list(modOrigin=pep1, modPos=1, modMass=579.9663, modName="p", modFixed=FALSE)
.singleSpecModif(pepTab1, specModif1 )

protP <- c(protP="PEPTIDEKR")
pep1 <- c("PTI", "KPE", "EPTI")
papTab1 <- cbind(no=c(7,2,6), seq=pep1, orig=rep("KPEPTI",3), origNa=rep("pe1",3),
  ty=paste0(c("C", "N", "C"), "ter"), beg=c(4,1,3), end=c(6,3,4),
  mass= wrProteo::convAASeq2mass(pep1, massTy="mono"), modSpec="")
```

---

.termPepCut

*Make Named Character Vector Of Sequential Terminal Fragments*

---

**Description**

Make named character vector of sequential terminal fragments

**Usage**

```
.termPepCut(
  pe,
  mi,
  ma = 1000,
  se1 = ".",
  se2 = "-",
  mainName = NULL,
  sepNC = FALSE,
  indexOffs = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

pe	(character, length=1) s
mi	(integer) min number of AA residues for considering peptide fragments; should be <= length(pe) (otherwise the full length of 'pe' ALWAYS returned !)
ma	(integer) max number of AA residues for considering peptide fragments



se1	(character, length=1) separators for adding numbers to specify partial/fragment locations
se2	(character, length=1) separators for adding numbers to specify partial/fragment locations
mainName	(character, length=1)
sepNC	(logical) if 'TRUE', separate N-terminal, C-terminal and internal fragments in list
indexOffs	(logical) offset to add for custom numbering in names (numeric, length=1), ie '1' will already increase by +1
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

This function returns a numeric vector with mass(es) and sequence in name(s)

**See Also**

[makeFragments](#); [convAASeq2mass](#)

**Examples**

```
## Ubiquitin example
P0CG48 <- "MQIFVKLTGKITLEVEPSDTIENVKAKIQDKEGIPPDQQLIFAGKQLEDGRTLSDYNIQKESTLHLVLRLLGG"
.termPepCut(P0CG48, mi=3, ma=12 ,sepNC=TRUE, mainName="P0CG48")
```

---

AAfragSettings

*Settings For AA Fragmentation*


---

**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: specific AA sites (if applicable), \$specAAMod: specific 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", silent = FALSE, debug = FALSE, callFrom = NULL)
```

**Arguments**

outTy (character) default "all" or any of the list-elements  
silent (logical) suppress messages  
debug (logical) additional messages for debugging  
callFrom (character) allows easier tracking of messages produced

**Value**

This function returns a 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 occurrences of modifications : Either fixed or variable modifications will be added to the mass of initial peptides from argument pepTab.

**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) 'mono' or 'average'
knownMods	(list) optional custom definition which modification is N-term, etc (see <a href="#">AAfragSettings</a> )
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) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

**Details**

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 called twice, it is recommended to always start with the fixed modifications, 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 typically no gain in performance.

**Value**

This functions returns a list containing \$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 peptides 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 modifications
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,
  debug = FALSE,
  callFrom = NULL
)

```

**Arguments**

modTy	(character) list of modification types to be considered
knownMods	(character) optional custom list of known modifications, default from AAfragSettings(outTy="all")\$kn
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

**Value**

This function returns the 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 this function for 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.

**Usage**

```
combinateAllAndSum(  
  nMax,  
  modVal,  
  notSingle = NULL,  
  silent = TRUE,  
  debug = FALSE,  
  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
debug	(logical) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

**Details**

Most (variable) modifications are linked to a type of amino acid, like serine- or thyrrosine residues for phosphorylation. Thus in this case, each instance of the amino acids S or T may or may not be modified. So, for example if there are 2 serines on a given peptide/protein, 0, 1 or 2 phosphorylation modifications may be present. For this reason there is an argument called nMax to allow staying within biologically relevant ranges (external knowledge) and allowing to reduce complexity significantly. In the case of phosphorylations, the total number of actually phosphoylated amino-acids is typically way below the number of S and T residues in pthe initial sequence. Some modifications are exclusive to others, argument notSingle : An (artificially occurring) de-phosphorylation event during fragmentation can only happen if the amino acid was already phosphorylated in the first place.

**Value**

This functions returns a named (concatenated names of modVal) numeric vector

**See Also**

[convToNum](#)

**Examples**

```
uniqCo <- matrix(c(1,1,1,0,1,1), nrow=2, dimnames=list(c("PTI","KPE"),c("d","p","h")))
massModV <- c(d= -18.01056, p= 79.96633, h= -18.01056)
## for 1st peptide
combinateAllAndSum(uniqCo[1,], massModV, notSingle=c("q","p"))
## for all peptides
apply(uniqCo, 1, combineAllAndSum, massModV, notSingle=c("q","p"))
```

---

combinatIntTable

*Planing For Making All Multiplicative Combinations*

---

**Description**

Provide all combinations for each of n elements of vector 'nMax' (positive integer, eg number of max multiplicative value). Results allow to see possible total compositions and must be read vertically.

**Usage**

```
combinatIntTable(
  nMax,
  include0 = TRUE,
  asList = FALSE,
  callFrom = NULL,
  debug = FALSE,
  silent = TRUE
)
```

**Arguments**

nMax	(positiveinteger) atomic composition; could be max number of voting participants form different cities, eg Paris max 2 persons, Lyon max 1 person ...
include0	(logical) include 0 occurances, ie provide al combinations starting from 0 or from 1 up to nMax
asList	(logical) return result a
callFrom	(character) allows easier tracking of messages produced
debug	(logical) additional messages for debugging
silent	(logical) suppress messages

**Value**

This functions returns a list or array (as 2- or 3 dim) with possible number of occurances for each of the 3 elements in nMax. Read results vertical : out[[1]] or out[,1] .. (multiplicative) table for 1st element of nMax; out[,2] .. for 2nd

**See Also**

[combinateAllAndSum](#)

**Examples**

```
combinatIntTable(c(1,1,1,2), include0=TRUE, asList=FALSE, silent=TRUE)
nMa <- c(Paris=2,Lyon=1,Strasbourg=1)
combinatIntTable(nMa, include0=TRUE, asList=TRUE, silent=TRUE)
```

---

corInDelShift

*Corrective Values For Random Sequences For In/Dels*

---

**Description**

The protein sequence and composition of most proteomes cannot be well mimicked by pure random sequences. Thus, random sequences (for comparing the quality of fitting) shhould be corrected accordingly, this vector provides help to do so. This function loads corMutShift- or corInDelShift-values from RData. The vector contains possible mass alterations for random drawing either by mutating a given aminoacid (corMutShift), or by making or in/del changes (corInDelShift). These values are based on simulations in the human proteome (from UniProt).

**Usage**

```
corInDelShift(fi = NULL)
```

**Arguments**

fi	(character) file (and path) to RData to read as corMutShift or corInDelShift. The file when opened must contain a numeric vector either called 'corMutShift' or 'corInDelShift'.
----	--

**Value**

This functions returns a numeric vector (1907 possible mass alterations for random drawing)

**See Also**

[corMutShift](#), [randMassByStochastic](#)

**Examples**

```
corMutShift <- corMutShift()
str(corMutShift)

corInDelShift <- corInDelShift()
str(corInDelShift)
```

---

corMutShift

*Corrective Values For Random Sequences For Mutations*

---

**Description**

The protein sequence and composition of most proteomes cannot be well mimicked by pure random sequences. Thus, random sequences (for comparing the quality of fitting) should be corrected accordingly, this vector provides help to do so. This function loads corMutShift- or corInDelShift-values from RData. The vector contains possible mass alterations for random drawing either by mutating a given aminoacid (corMutShift), or by making or in/del changes (corInDelShift). These values are based on simulations in the human proteome (from UniProt).

**Usage**

```
corMutShift(fi = NULL)
```

**Arguments**

**fi** (character) file (and path) to RData to read as corMutShift or corInDelShift. The file when opened must contain a numeric vector either called 'corMutShift' or 'corInDelShift'.

**Value**

This functions returns a numeric vector (possible mass alterations for random drawing)

**See Also**

[corInDelShift](#), [randMassByStochastic](#)



**Examples**

```
corMutShift <- corMutShift()
str(corMutShift)

corInDelShift <- corInDelShift()
str(corInDelShift)
```

---

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 han-deled 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 re-dudant start- and end-sites accur with different row-names, repeated start- and end-sites will display NA.

**Usage**

```
countChildrenParent(
  fragments,
  output = "count",
  silent = FALSE,
  debug = 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
debug	(logical) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

**Value**

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

**See Also**

[simpleFragFig](#); for building longer consecutive trees (without identification of 'parent') [buildTree](#)

**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 mods, 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 <a href="#">makeFragments</a>
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 <a href="#">AAfragSettings</a> )
knownMods	(list) optional custom list showing which modification appears at what type of location, eg N-terminal, internal ... (default <a href="#">AAfragSettings</a> )
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")))
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,
  debug = 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
debug	(logical) additional messages for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

This function returns 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**

```
pepTab <- 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(pepTab)
```

---

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,
  debug = FALSE,
  callFrom = NULL
)

```

**Arguments**

sequ	(character, length=1) sequence used for fragmenting, as as mono-aminoacid letter code (so that cutting will be performed 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 may 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
debug	(logical) additional messages for debugging
callFrom	(character) allow easier tracking of messages produced

**Details**

Note: Thus function can handle only 1 sequence at each run ! Note: The mass values returned correspond to neutral peptides. If you are looking for ions, you need to adjust masses to the respective ion-characteristics (adding H+, etc).

**Value**

This function returns a numeric vector with the (neutral) mass of fragmented peptides

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

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

```

---

identifFixedModif      *Identify Fixed Modifications*

---

**Description**

Identify peptide/protein fragments 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,
  chargeCatchFilter = 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,
  massTy = "mono",
  prefFragPat = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)

```

**Arguments**

prot	(character) amino-acid sequene of peptide or protein
expMass	(numeric) experimental 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 consiered
chargeCatchFilter	(logical) by default remove all peptides not containing charge-catching (polar) AAs (K, R, H, defined via .chargeCatchingAA() )
maxMod	(integer) maximum number of residue modifications to be consiered in frag-ments (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
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 debug- ging
callFrom	(character) allow easier tracking of message(s) produced

**Details**

The main matching results are in `output$massMatch` : This list has one entry for each predicted mass where some matches were found. Thus, the names of the list-elements design the index from argument `expMass`. Each list-element contains a numeric vector giving the difference observed to predicted, the names design the unique predicted peptide index/number from `output$preMa[, "no"]`

The main element of the output is the `$massMatch` -list, which is in the format of `findCloseMatch`. Thus, the list-elements names represent the line-number of mass-predictions and the values the delta-mass and their names the position of the initial query.

**Value**

This function returns a list with \$massMatch (list of experimental peptides matching to one or more predicted), \$preMa (predicted ions, including fixed modif), \$pepTab (predicted neutral peptides, without modifications), \$expMa (experimental mass from input), \$recalibFact (recalibration factor as from input), \$docTi (time for calculations)

**See Also**

[makeFragments](#), [identifVarModif](#), [identifyPepFragments](#), [findCloseMatch](#)

**Examples**

```
pro3 <- "HLVDEPQNLIK"
exp3 <- c( b4=465.2451, b5=594.2877, b6=691.3404, y7=841.4772, y6=712.4347, y5=615.3819)
ident3 <- identifFixedModif(prot=pro3, expMass=exp3, minFragSize=4,
  maxFragSize=60, modTy=list(basMod=c("b","y")))
ident3$massMatch
## as human readable table:
ident3$preMa[ ident3$preMa[, "no"] %in% (names(ident3$massMatch)),]
```

---

identifVarModif

*Identify Variable Modifications*

---

**Description**

Take result from identifFixedModif and search for variable modifications (only on identified fixed modif), ie 2nd step for identif of var modifs. To reduce the complexity of the search space, only peptide fragments identified with fixed identifications will be considered for possible variable modifications.

**Usage**

```
identifVarModif(
  zz,
  modTy,
  expMa,
  maxMod,
  identMeas = "ppm",
  knownMods = NULL,
  limitIdent = 5,
  filtAmbiguous = FALSE,
  indexStart = 1,
  recalibFact = NULL,
  suplPepTab = NULL,
  massTy = "mono",
  silent = FALSE,
  callFrom = NULL,
  debug = TRUE
)
```



**Arguments**

zz	(list) min input, result from <code>identifFixedModif</code> , must contain elements 'nmassMatch', 'preMa', 'pepTab', 'recalibFact', 'recalibData'
modTy	(character) type of fixed and variable modifications
expMa	(matrix) experimental m/z values
maxMod	(integer) maximum number of residue modifications to be considered in fragments (values >1 will increase complexity and RAM consumption)
identMeas	(character) comparison type (used in <code>findCloseMatch()</code> , default = "ppm"), used with limit 'limitIdent'
knownMods	(character) optional custom alternative to <code>AAfragSettings(ou="all")\$knownMods</code>
limitIdent	(integer) limit applied to 'identMeas'
filtAmbiguous	(logical) toggle to remove all ambiguous identifications
indexStart	(integer) for keeping correct index at iterative use
recalibFact	(numeric, length=1)
suplPepTab	(matrix) predicted fragments (incl fixed and var mods) to include to search (allowing to ensure overlap to include hits close to prev search)
massTy	(character) 'mono' or 'average'
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of messages produced
debug	(logical) additional messages for debugging

**Details**

The main matching results are in `output$massMatch` : This list has one entry for each predicted mass where some matches were found. Thus, the names of the list-elements design the index from argument `expMass`. Each list-element contains a numeric vector giving the difference observed to predicted, the names design the unique predicted peptide index/number from `output$preMa[, "no"]`

**Value**

list with `$massMatch` (list of experimental peptides matching to one or more predicted), `$preMa` (predicted ions, including fixed and variable modif), `$pepTab` (predicted neutral peptides, wo modifications), `$expMa` (experimental mass from input), `$recalibFact` (recalibration factor as from input), `$docTi` (time for calculations)

**See Also**

[makeFragments](#), [identifFixedModif](#), [identifyPepFragments](#)

**Examples**

```
protP <- c(protP="PEPTIDE")
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
identP10 <- identifFixedModif(prot=protP,expMass=as.numeric(obsMassX),minFragSize=2,
  maxFragSize=7,modTy=list(basMod=c("b","y")))    # looks ok
identP10v <- identifVarModif(identP10,list(varMod="h"), as.numeric(obsMassX),2)
identP10v$massMatch                               # list of matches

```

---

identifyPepFragments    *Identify terminal and internal protein/peptide-fragments as matches to experimental MS-peaks*

---

### Description

Function for predicting internal and terminal peptide-fragments and compare them with experimental monoisotopic masses. The accuracy of results is given in ppm and a false discovery rate (FDR) for the identification is estimated. The identified fragments are also checked for preferential break sites, a score including this and other parameters is given with the results.

### Usage

```

identifyPepFragments(
  expMass,
  pep,
  modTy = NULL,
  minFragSize = 6,
  maxFragSize = 75,
  identMeas = "ppm",
  limitIdent = 5,
  internFra = TRUE,
  specModif = NULL,
  massTy = "mono",
  chargeCatchFilter = TRUE,
  corMutShift = NULL,
  nProc = 1,
  parallDefault = TRUE,
  multParam = NULL,
  maxMod = c(p = 3, h = 1, k = 1, o = 1, m = 1, n = 1, u = 1, r = 1, s = 1),
  recalibrate = TRUE,
  filtAmbiguous = FALSE,
  prefFragmPat = NULL,
  sortOutputByMass = FALSE,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)

```

**Arguments**

expMass	(matrix or data.frame)
pep	(character) protein/peptide sequences to be used for fragmentation
modTy	(list) defining fixed and variable modifications
minFragSize	(integer) min length in AA of peptides to be considered (please see you spectrometers characteristics)
maxFragSize	(integer) max length in AA of peptides to be considered (please see you spectrometers characteristics)
identMeas	(character) comparison type (used in findCloseMatch(), default = "ppm"), used with limit 'limitIdent'
limitIdent	(integer) limit applied to 'identMeas'
internFra	(logical) switch from including all internal fragments to terminal fragments only (if F)
specModif	(list) optional custom single-site modifications (eg ions bound), will be processed using .singleSpecModif
massTy	(list) list of modifications/fragmentation-type(s) to consider, organipredMae as 'basMod' (any occurrence) and 'varMod' (optional occurrence), 'modPos' (position of modif, integer), 'modMass' (mass to be added), 'modName' (name), 'modFixed' (fixed or variable modif, logical)
chargeCatchFilter	(logical) filter (upfront) to consider only peptides containing AAs capable of catching extra charges (K, R, H, defined via .chargeCatchingAA())
corMutShift	(numeric) (numeric) vector of decoy-type possible mass shifts (eg from load("C:/E/projects/MassSpec/frag"))
nProc	(integer) number of preprocessors to use
parallDefault	(logical) if 'parallDefault'=F no multiprocessor parameters set for BiocParallel
multParam	(list)
maxMod	(integer) maximum number of residue modifications to be considered in fragments (values >1 will increase complexity and RAM consumption)
recalibrate	(logical) recalibrate based on region with highest density of experim values
filtAmbiguous	(logical)
prefFragmPat	(matrix) optional custom preferential fragmentation pattern (otherwise .prefFragPattern() will be used)
sortOutputByMass	(logical)
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of messages produced
debug	(logical) additional messages for debugging

**Value**

matrix of identified ions

**See Also**

[makeFragments](#), [identifVarModif](#), [identifFixedModif](#), [findCloseMatch](#), [scoreProteinFragments](#)

**Examples**

```
protP <- c(protP="PEPTIDE")
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
modTy1 <- list(basMod=c("b","y"), varMod=c("p","o","q"))
frag1 <- identifyPepFragments(ex=as.numeric(obsMassX), pe=protP, modTy=modTy1,
  minFragSize=2, chargeCatchFilter=FALSE)
(frag1b <- if(length(unlist(frag1$identif)) >0) modifFragmTabOutput(frag1))
```

---

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 realistic values for the type of mass spectrometer used, since efficient filtering will reduce considerably the amount of memory (RAM) needed and will improve overall 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-sequences to fragment or or, matrix with 1st column 'ma' with mass and 2nd column 'se' with protein-/peptide-sequence (optionally sequence(s) also as rownames or 3rd column with 'trivial' names)
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 fragments 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 (alternative 'average')
specModif	(list) supplemental custom fixed or variable modifications (eg Zn++ at given residue)
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allow easier tracking of messages 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)

protP2 <- cbind(se="PEPTIDE", ma="1304.7088503626")
pepT2 <- makeFragments(protTab=protP2, minFragSize=2, maxFragSize=9, internFra=TRUE)
tail(pepT2)
```

---

modifFragmTabOutput     *Change fragment identification output format (for biologists)*

---

### Description

Change fragment identification output to format better adopted for biologists

### Usage

```
modifFragmTabOutput(
  datafr,
  addData = NULL,
  fuseC = c("precAA", "seq", "tailAA"),
  sep = ".",
  modifCol = "mod",
  replMod = cbind(old = "by", new = "i"),
  finCols = c("fraNa", "origNa", "beg", "end", "seq", "ty", "mod", "modSpec", "obsMass",
    "mass", "ppmToPred", "ambig", "runNo", "FDR", "sco4", "sc.prefFrag",
    "sc.chargeCatch", "sc.complemFra", "sc.sameSite", "logInt"),
  supFinCols = NULL,
  sortTable = "end",
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

### Arguments

datafr	(data.frame) initial output from identifyPepFragments()
addData	(matrix or data.frame) suppelemental data
fuseC	(character) columns to extract preceeding and tailing AA to fuse with separator 'sep' to main sequence
sep	(character) separator for concatenation
modifCol	(character) default 'modif'
replMod	(matrix) if names of modifications shoule be renamed : the columns 'old' and 'new' indicata how modifcations should be renamed
finCols	(character) columns to retain for final output
supFinCols	(character)
sortTable	(character) sort output 1st by name, then by 'beg' or 'end'
silent	(logical) suppress messages
debug	(logical) additional diagnostic messages
callFrom	(character) allow easier tracking of message produced

**Value**

data.frame of reorganized identification results

**See Also**

[identifyPepFragments](#)

**Examples**

```
protP <- c(protP="PEPTIDE")
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
modTy1 <- list(basMod=c("b","y"), varMod=c("p","o","q"))
frag1 <- identifyPepFragments(ex=as.numeric(obsMassX), pe=protP, modTy=modTy1,
  minFragSize=2, chargeCatchFilter=FALSE)
(frag1b <- if(length(unlist(frag1$identif)) >0) modifFragmTabOutput(frag1))
```

---

plotFragmLoc

*Plot Identified Fragments Relative To Their Location*

---

**Description**

Take result from fragIonMass and display identified fragments by their location, an additional parameter (default logIntensity) is used for coloring This function illustrates the distribution of identified peptides and thus common break-points.

**Usage**

```
plotFragmLoc(
  fraL,
  extrCol = NULL,
  useLog = FALSE,
  useCol = NULL,
  specLayout = NULL,
  useTi = NULL,
  subTit = NULL,
  footer = NULL,
  batchFig = FALSE,
  legCex = 0.5,
  legOffS = NULL,
  legBorder = NULL,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)
```

**Arguments**

fraL	(list) result from fragIonMass, must contain elements "
extrCol	(character) 1st should be aa seq of initial proteins (used for dimensioning graph and separating multiple input proteins), 2nd & 3rd start- and end-site for drawing;, 4th the column to use for coloring), 5th for protein name in title of figure
useLog	(logical) take values for coloring (4th element of 'extrCol') as log10
useCol	(character) custom colors
specLayout	(character) custom layout
useTi	(character) custom title
subTit	(character) custom sub-title
footer	(character) custom footer
batchFig	(logical) reduce text content for multiple figures on page
legCex	(numeric, length=1) expansion factor
legOffs	(numeric) legend-offset (passed to <a href="#">legendHist</a> )
legBorder	(logical) legend-border (passed to <a href="#">legendHist</a> )
silent	(logical) suppress messages
callFrom	(character) allow easier tracking of message(s) produced
debug	(logical) additional messages for debugging

**Value**

This function returns a figure

**See Also**

[identifFixedModif](#)

**Examples**

```
protP <- c(protP="PEPTIDE")
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"))) #
```



---

plotMgfLike	<i>Draw simplified (deconvoluted) spectrum of mgf type and highlight peaks with matches found to theoretical data</i>
-------------	---

---

### Description

Draw simplified (deconvoluted) spectrum of mgf type and highlight matches found

### Usage

```
plotMgfLike(
  lst,
  basInp = NULL,
  backgrCol = NULL,
  replNames = c("by", "i"),
  lwd = 1,
  col = NULL,
  tit = NULL,
  xLim = NULL,
  yLab = NULL,
  linPlot = FALSE,
  listNa = c("identif", "overview", "obsMass"),
  useCoIN = c("obsMass", "logInt", "origNa", "mod", "modSpec", "fraNa", "orig",
    "nIdentif", "minMassDec", "maxMass"),
  cex = 1,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

### Arguments

lst	(list) result of identification with element \$obsMass (the column 'obsMass' should be m/z values, the column 'sc.logInt' intensity values) and \$identif
basInp	(numeric) alternative/custom entry of observed masses
backgrCol	(character) color of background
replNames	(character)
lwd	(numeric) line width
col	(character) custom colors for different types of ions/modifications
tit	(character) custom title
xLim	(numeric length=2) custom x axis margins
yLab	(character) custom y axis label
linPlot	(logical) re-transform y-axis from log2 to linear scale
listNa	(character) list-elements of 'lst' to use/extract

useColN	(character) columns names tu use from input ie lst\$identif & lst\$overview
cex	(numeric) expansion factor for x- and y-label
silent	(logical) suppress messages
debug	(logical) additional messages for debugging
callFrom	(character) allows easier tracking of messages produced

### Value

This function returns a mgf-like figure

### See Also

[makeFragments](#), [identifVarModif](#), [identifFixedModif](#), [identifyPepFragments](#)

### Examples

```
set.seed(2025)
```

---

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) custom 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)
```

---

plotPrefFragPat	<i>plot preferential fragmentation pattern Plot preferential fragmentation pattern equivalent to Fig 1b of Haverland et al 2017 (J Am Soc Mass Spectrom)</i>
-----------------	--

---

**Description**

plot preferential fragmentation pattern

Plot preferential fragmentation pattern equivalent to Fig 1b of Haverland et al 2017 (J Am Soc Mass Spectrom)

**Usage**

```
plotPrefFragPat(
  prefPat,
  namesCex = 0.8,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

prefPat	(matrix)
namesCex	(numeric) expansion factor cex for display of AA-names
silent	(logical) suppress messages
debug	(logical) additional messages and objects exported to current session for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

This function returns a figure

**See Also**

[scoreFragments](#)

**Examples**

```
plotPrefFragPat(.prefFragPattern())
```

---

randMassByMut	<i>Make decoy mass by full randomization</i>
---------------	--

---

**Description**

Make full random decoy mass vector (mimick pepTab)

**Usage**

```
randMassByMut(  
  pepTab,  
  randCha,  
  useCol = "mass",  
  negAvoid = TRUE,  
  sepCol = FALSE,  
  inDel = FALSE,  
  nAlter = 1,  
  setSeed = NULL,  
  silent = FALSE,  
  debug = FALSE,  
  callFrom = NULL  
)
```

**Arguments**

pepTab	(matrix) typically table of peptides, one column should match 'useCol' for numeric mass values
randCha	(numeric) vector of possible mass alterations for random drawing
useCol	(character) column from 'pepTab' with mass values to make decoys
negAvoid	(logical) if TRUE try avoiding 0 or negative random mass in result
sepCol	(character) optional column from 'pepTab'
inDel	(logical) switch to make random mass by insertion/deletion of 1 AA
nAlter	(integer) number of alterations per peptide
setSeed	(character) seed for random number generation set.seed()
silent	(logical) suppress messages
debug	(logical) additional messages and objects exported to current session for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

This function returns a matrix with additional column 'decoyMass', or if 'sepCol'=FALSE as additional lines

**See Also**

[randMassByStochastic](#)

**Examples**

```
pepTab1 <- cbind(no=11:12, seq=c("YVVDTS", "YVVDTSK"), origNa="test.P000",
  ty=c("inter", "Cter"), mass=c(681.308997360991, 809.403960378691))
randMassByMut(pepTab1, corMutShift())
randMassByMut(pepTab1, corInDelShift(), inDel=TRUE)
```

---

randMassByStochastic *Make Decoy Mass By Full Randomization*

---

**Description**

Make full random decoy mass vector (mimick pepTab)

**Usage**

```
randMassByStochastic(
  xChar,
  nRepeat = 5,
  negAvoid = TRUE,
  setSeed = NULL,
  silent = FALSE,
  debug = TRUE,
  callFrom = NULL
)
```

**Arguments**

xChar	(numeric vector) characterize main data, must contain elements 'n', 'minV', 'maxV'
nRepeat	(integer) number of time whole randomization process should be repeated
negAvoid	(logical) if TRUE try avoiding 0 or negative random mass in result
setSeed	(integer) seed for random number generation set.seed()
silent	(logical) suppress messages
debug	(logical) additional messages and objects exported to current session for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

This function returns a matrix with additional column 'decoyMass', or if 'sepCol'=FALSE as additional lines

**See Also**

[Uniform](#), [randMassByMut](#), [convToNum](#)

**Examples**

```
rand <- randMassByStochastic(c(n=10, minV=2, maxV=7))
summary(rand)
```

---

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 privileged.

**Usage**

```
scoreChargeCatch(
  resTab,
  pepCol = "seq",
  scale01 = TRUE,
  chargeMode = "pos",
  silent = FALSE,
  debug = 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
debug	(logical) additional messages and objects exportet to current session for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

This function returns a 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)
```

---

scoreFragments                      *Scoring For Single Protein : Individual Components*

---

**Description**

Make scoring for single protein : individual components :sameSite,contiguous,prefFragSite,logPeakHeight + combined (sum of scales 0->1)

**Usage**

```
scoreFragments(
  resTab,
  fragmInp,
  supplTakeLog = TRUE,
  j = NULL,
  useResCol = c("orig", "seq", "precAA", "tailAA", "beg", "end", "ppmToPred", "obsInd",
    "predInd", "Abundance"),
  prefFragPat = NULL,
  contigTermFragWe = 0.5,
  figDraw = TRUE,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

resTab	(matrix or data.frame) matrix or data.frame of results for SINGLE protein (will use columns 'beg','end','orig','obsInd')
fragmInp	(matrix) experimental m/z values including suppl col(s) to be considered for score, its intensity column/value will be used for 'logInt' in output
supplTakeLog	(logical) if suppl info should be used as log2: if T all supplemental data columns ('fragmInp') will be taken as log2
j	(integer) which column of fragmInp has m/z values, the following column is assumed as peak-intensity
useResCol	(character) column-names from resTab to be used
prefFragPat	(matrix) for preferential fragmentation rules (see .prefFragPattern())
contigTermFragWe	(numeric, length=1) weight to add for terminal fragments at 'sc.complemFra' (since they cannot match other fragments beyond the protein limits)
figDraw	(logical) make additional figure
silent	(logical) suppress messages
debug	(logical) additional messages and objects exportet to current session for debugging
callFrom	(character) allow easier tracking of messages produced



**Value**

This function returns a list with matrix \$scaled (combined and indiv rescaled scores) and \$raw (matching lines of 'resTab'; 'index' refers to predictedIndex)

**See Also**

[identifyPepFragments](#), [scoreProteinFragments](#)

**Examples**

```
tab2 <- matrix(c("20", "2", "13", "11", "3", "10", "4", "PT", "PE", "EP", "DE", "PEP", "IDE", "PEPT",
  rep(c("PEPTIDE", "protP"), each=7), c("inter", "Nter", "Cter")[c(1, 2, 1, 3, 2, 3, 2)],
  c(3, 1, 2, 6, 1, 5, 1, 4, 2, 3, 7, 3, 7, 4), "E", NA, "P", "I", NA, "T", NA, "I", "P", "T", NA, "T", NA, "I",
  c(1, 6, 6, 20, 7, 19, 8), c(-0.094312, -0.14707, -0.14707, 0.08641, 0.0084762, -0.10965, 0.057087),
  rep(2, 7)), nrow=7, dimnames=list(NULL, c("predInd", "seq", "orig", "origNa", "ty", "beg", "end",
  "precAA", "tailAA", "obsInd", "ppmToPred", "mass")))
tab2 <- cbind(tab2, seqNa=paste0(tab2[, "origNa"], ".", tab2[, "beg"], "-", tab2[, "end"]), Abundance=1)
rownames(tab2) <- paste0(tab2[, "origNa"], ".", tab2[, "beg"], "", tab2[, "end"])
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))

(outF <- scoreFragments(tab2, fragmInp=cbind(as.numeric(obsMassX), Abundance=1)))
```

---

scorePrefFrag

*Identification and scoring of preferential cutting sites*

---

**Description**

Search for preferential fragmentation sites from 'pepTab' among the 3 columns specified via 'useCol' (for full AA sequence, preceding AA, tailing AA) and return sum of scores (from 3rd column of prefFragPat) for both ends. Note : proteins must be written as single letter code.

**Usage**

```
scorePrefFrag(
  pepTab,
  useCol = c("seq", "precAA", "tailAA"),
  prefFragPat = NULL,
  silent = FALSE,
  debug = FALSE,
  callFrom = NULL
)
```

**Arguments**

pepTab	(matrix) peptide-fragments with lines for peptides, cols as sequence/preceedingAA/tailingAA
useCol	(character) column names for peptide-sequence,preceeding and tailing AA
prefFragPat	(matrix) for preferential fragmentation rules (see .prefFragPattern())
silent	(logical) suppress messages
debug	(logical) additional messages and objects exportet to current session for debugging
callFrom	(character) allow easier tracking of messages produced

**Value**

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

**See Also**

[makeFragments](#)

**Examples**

```
pepT <- cbind(precAA=c("A","D","D","A","D"),seq=c("AKA","PKA","AKA","PKD","PKD"),
  tailAA=c("A","A","D","P","P"))
scorePrefFrag(pepT)
```

---

scoreProteinFragments *Scoring Of Identifications (For Multi-Protein Queries)*

---

**Description**

Make scoring for multiple protein queries: individual components :sameSite,contiguous,prefFragSite,logPeakHeight + combined (sum of scales 0->1)

**Usage**

```
scoreProteinFragments(
  resTab,
  fragmInp = NULL,
  j = 2,
  useCol = c("orig", "precAA", "tailAA", "beg", "end", "ppmToPred", "obsInd", "predInd"),
  prefFragPat = NULL,
  contigTermFragWe = 0.5,
  returnCombined = TRUE,
  figDraw = TRUE,
  silent = FALSE,
  callFrom = NULL,
  debug = FALSE
)
```

**Arguments**

resTab	(matrix or data.frame) identification results (will use columns 'beg', 'end', 'orig', 'obsInd')
fragmInp	(numeric vector or matrix) experimental m/z values, may include suppl col(s) to be considered for score (se argument 'j')
j	(integer) which column of fragmInp has m/z values, the following column is assumed as peak-intensity
useCol	(character) colnames from resTab to be used, 1st position must be present as column of 'resTab' and must represent name of original, used for splitting by proteins (eg original input protein sequence), will be passed to scoreFragments
prefFragPat	(matrix) for preferential fragmentation rules (see .prefFragPattern())
contigTermFragWe	(numeric, length=1) weight to add for terminal fragments (since they cannot match other fragments beyond the protein limits)
returnCombined	(logical)
figDraw	(logical) make additional figure
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**

This function returns a list with matrix \$scaled (combined and indiv rescaled scores) and \$raw (matching lines of 'resTab')

**See Also**

[scoreFragments](#), [identifyPepFragments](#)

**Examples**

```

tab2 <- matrix(c("20", "2", "13", "11", "3", "10", "4", "PT", "PE", "EP", "DE", "PEP", "IDE", "PEPT",
  rep(c("PEPTIDE", "protP"), each=7), c("inter", "Nter", "Cter")[c(1,2,1,3,2,3,2)],
  c(3,1,2,6,1,5,1, 4,2,3,7,3,7,4), "E", NA, "P", "I", NA, "T", NA, "I", "P", "T", NA, "T", NA, "I",
  c(1,6,6,20,7,19,8), c(-0.094312, -0.14707, -0.14707, 0.08641, 0.0084762, -0.10965, 0.057087),
  rep(2,7)), nrow=7, dimnames=list(NULL, c("predInd", "seq", "orig", "origNa", "ty", "beg", "end",
  "precAA", "tailAA", "obsInd", "ppmToPred", "mass")))
tab2 <- cbind(tab2, seqNa=paste0(tab2[, "origNa"], ".", tab2[, "beg"], "-", tab2[, "end"]), Abundance=1)
rownames(tab2) <- paste0(tab2[, "origNa"], ".", tab2[, "beg"], "", tab2[, "end"])
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))

(outF <- scoreFragments(tab2, fragmInp=cbind(as.numeric(obsMassX), Abundance=1)))
(out <- scoreProteinFragments(tab2, fragmInp=cbind(as.numeric(obsMassX), Abundance=1)))

```

# Index

.CtermPepCut, 7  
.NtermPepCut, 12  
.chColNa, 4  
.chargeCatchingAA, 3  
.checkModTy, 4  
.countLET, 5  
.countModif, 6  
.evalIsoFra, 9  
.exNamesTyDeList, 10  
.multMatByColNa, 11  
.parCombinateAllAndSum, 13  
.prefFragPattern, 14  
.singleSpecModif, 15  
.termPepCut, 8, 13, 16

AAfragSettings, 5–7, 17, 19, 21, 27, 43  
AAmass, 10, 37  
addMassModif, 18

buildTree, 25, 26

checkModTy, 20  
combinateAllAndSum, 21, 23  
combinatIntTable, 22  
convAASeq2mass, 8, 10, 13, 17, 29, 37  
convToNum, 11, 14, 19, 22, 46  
corInDelShift, 23, 24  
corMutShift, 24, 24  
countChildrenParent, 25  
countPotModifAAs, 26

evalIsoFragm, 10, 27, 37

findCloseMatch, 31, 32, 36  
fragmentSeq, 3, 18, 28, 47

identifFixedModif, 30, 33, 36, 40, 42  
identifVarModif, 16, 32, 32, 36, 42  
identifyPepFragments, 16, 32, 33, 34, 39, 42, 49, 51

legendHist, 40

makeFragments, 6–10, 13, 14, 16–18, 27–29, 32, 33, 36, 36, 37, 42, 50  
massDeFormula, 10, 18, 37  
modifFragmTabOutput, 38

plotFragmLoc, 39  
plotMgfLike, 41  
plotNTheor, 42  
plotPrefFragPat, 43

randMassByMut, 44, 46  
randMassByStochastic, 24, 45, 45

scoreChargeCatch, 46  
scoreFragments, 4, 44, 48, 51  
scorePrefFrag, 49  
scoreProteinFragments, 49, 50  
simpleFragFig, 26

Uniform, 46