Package ‘xcms’

August 7, 2022

Version 3.18.0

Title LC-MS and GC-MS Data Analysis

Description Framework for processing and visualization of chromatographically separated and single-spectra mass spectral data. Imports from AIA/ANDI NetCDF, mzXML, mzData and mzML files. Preprocesses data for high-throughput, untargeted analyte profiling.

Depends R (>= 4.0.0), BiocParallel (>= 1.8.0), MSnbase (>= 2.21.4)

Imports mzR (>= 2.25.3), methods, Biobase, BiocGenerics, ProtGenerics (>= 1.25.1), lattice, RColorBrewer, plyr, RANN, MassSpecWavelet (>= 1.5.2), S4Vectors, robustbase, IRanges, SummarizedExperiment, MsCoreUtils, MsFeatures

Suggests BiocStyle, caTools, knitr (>= 1.1.0), faahKO, msdata (>= 0.25.1), ncdf4, testthat, pander, magrittr, rmarkdown, multtest, MALDIquant, pheatmap, Spectra (>= 1.1.17), MsBackendMgf, progress, signal

Enhances Rgraphviz, rgl, XML

License GPL (>= 2) + file LICENSE

URL `https://github.com/sneumann/xcms`

BugReports `https://github.com/sneumann/xcms/issues/new`

VignetteBuilder knitr

biocViews ImmunoOncology, MassSpectrometry, Metabolomics

RoxygenNote 7.1.2

Collate 'AllGenerics.R' 'functions-XChromatograms.R' 'functions-XChromatogram.R' 'DataClasses.R' 'Deprecated.R' 'MPI.R' 'c.R' 'cwTools.R' 'databases.R' 'functions-MsFeatureData.R' 'do_adjustRtime-functions.R' 'functions-binning.R' 'do_findChromPeaks-functions.R' 'functions-Params.R' 'do_groupChromPeaks-functions.R' 'fastMatch.R' 'functions-Chromatogram.R' 'functions-utils.R' 'functions-IO.R' 'functions-OnDiskMSnExp.R' 'functions-ProcessHistory.R' 'functions-XCMSnExp.R'

'functions-imputation.R' 'functions-normalization.R'
'functions-xcmsEIC.R' 'functions-xcmsFragments.R'
'functions-xcmsRaw.R' 'functions-xcmsSet.R'
'functions-xcmsSwath.R' 'init.R' 'matchpeaks.R'
'methods-Chromatogram.R' 'methods-IO.R'
'methods-MChromatograms.R' 'methods-MsFeatureData.R'
'methods-OnDiskMSnExp.R' 'methods-Params.R'
'methods-ProcessHistory.R' 'methods-XCMSnExp.R'
'methods-XChromatogram.R' 'methods-XChromatograms.R'
'methods-group-features.R' 'methods-xcmsEIC.R'
'methods-xcmsFileSource.R' 'methods-xcmsFragments.R'
'methods-xcmsPeaks.R' 'methods-xcmsRaw.R' 'methods-xcmsSet.R'
'models.R' 'mzClust.R' 'plotQC.R' 'ramp.R' 'specDist.R'
'write.mzquantML.R' 'writemzdata.R' 'writemztab.R'
'xcmsSource.R' 'xdata.R' 'zzz.R'

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

git_branch RELEASE_3_15

git_last_commit f205ce1

git_last_commit_date 2022-04-26

Date/Publication 2022-08-07

Author Colin A. Smith [ctb],
Ralf Tautenhahn [ctb],
Steffen Neumann [aut, cre] (<https://orcid.org/0000-0002-7899-7192>),
Paul Benton [ctb],
Christopher Conley [ctb],
Johannes Rainer [ctb] (<https://orcid.org/0000-0002-6977-7147>),
Michael Witting [ctb],
William Kumler [ctb] (<https://orcid.org/0000-0002-5022-8009>)

Maintainer Steffen Neumann <`sneumann@ipb-halle.de`>

R topics documented:

`absent-methods` *Determine which peaks are absent / present in a sample class*

Description

Determine which peaks are absent / present in a sample class

Arguments

`object`	`xcmsSet-class` object
`class`	Name of a sample class from `sampclass`
`minfrac`	minimum fraction of samples necessary in the class to be absent/present

Details

Determine which peaks are absent / present in a sample class The functions treat peaks that are only present because of `fillPeaks` correctly, i.e. does not count them as present.

Value

An logical vector with the same length as `nrow(groups(object))`.

Methods

object = "xcmsSet" `absent(object, ...)` `present(object, ...)`

See Also

`group` `diffreport`

`adjustRtime` *Alignment: Retention time correction methods.*

Description

The `adjustRtime` method(s) perform retention time correction (alignment) between chromatograms of different samples. These methods are part of the modernized `xcms` user interface.

The implemented retention time adjustment methods are:

peakGroups retention time correction based on aligment of features (peak groups) present in most/all samples. See `adjustRtime-peakGroups` for more details.

obiwarp alignment based on the complete mz-rt data. This method does not require any identified peaks or defined features. See `adjustRtime-obiwarp` for more details.

Author(s)

Johannes Rainer

See Also

`retcor` for the *old* retention time correction methods. `plotAdjustedRtime` for visualization of alignment results.

Other retention time correction methods: `adjustRtime-obiwarp`, `adjustRtime-peakGroups`

`adjustRtime-obiwarp` *Align retention times across samples using Obiwarp*

Description

This method performs retention time adjustment using the Obiwarp method [Prince 2006]. It is based on the code at `http://obi-warp.sourceforge.net` but supports alignment of multiple samples by aligning each against a *center* sample. The alignment is performed directly on the `profile-matrix` and can hence be performed independently of the peak detection or peak grouping.

It is also possible to exclude certain samples within an experiment from the estimation of the alignment models. The parameter `subset` allows to define the indices of samples within `object` that should be aligned. Samples not part of this `subset` are left out in the estimation of the alignment models, but their retention times are subsequently adjusted based on the alignment results of the closest sample in `subset` (close in terms of position within the `object`). Alignment could thus be performed on only *real* samples leaving out e.g. blanks, which are then in turn adjusted based on the closest real sample. Here it is up to the user to ensure that the samples within `object` are ordered correctly (e.g. by injection index).

How the non-subset samples are adjusted bases also on the parameter `subsetAdjust`: with `subsetAdjust = "previous"`, each non-subset sample is adjusted based on the closest previous subset sample which results in most cases with adjusted retention times of the non-subset sample being identical to the subset sample on which the adjustment bases. The second, default, option is to use `subsetAdjust = "average"` in which case each non subset sample is adjusted based on the average retention time adjustment from the previous and following subset sample. For the average a weighted mean is used with weights being the inverse of the distance of the non-subset sample to the subset samples used for alignment.

See also section *Alignment of experiments including blanks* in the *xcms* vignette for an example.

The `ObiwarpParam` class allows to specify all settings for the retention time adjustment based on the *obiwarp* method. Class Instances should be created using the `ObiwarpParam` constructor.

`binSize`,`binSize<-`: getter and setter for the `binSize` slot of the object.

`centerSample`,`centerSample<-`: getter and setter for the `centerSample` slot of the object.

`response`,`response<-`: getter and setter for the `response` slot of the object.

`distFun`,`distFun<-`: getter and setter for the `distFun` slot of the object.

`gapInit`,`gapInit<-`: getter and setter for the `gapInit` slot of the object.

`gapExtend`,`gapExtend<-`: getter and setter for the `gapExtend` slot of the object.

`factorDiag`,`factorDiag<-`: getter and setter for the `factorDiag` slot of the object.

`factorGap`,`factorGap<-`: getter and setter for the `factorGap` slot of the object.

`localAlignment`,`localAlignment<-`: getter and setter for the `localAlignment` slot of the object.

`initPenalty`,`initPenalty<-`: getter and setter for the `initPenalty` slot of the object.

`subset`,`subset<-`: getter and setter for the `subset` slot of the object.

`subsetAdjust`,`subsetAdjust<-`: getter and setter for the `subsetAdjust` slot of the object.

`adjustRtime,XCMSnExp,ObiwarpParam`: performs retention time correction/alignment based on the total mz-rt data using the *obiwarp* method.

Usage

```
ObiwarpParam(
  binSize = 1,
  centerSample = integer(),
  response = 1L,
  distFun = "cor_opt",
  gapInit = numeric(),
  gapExtend = numeric(),
  factorDiag = 2,
  factorGap = 1,
  localAlignment = FALSE,
  initPenalty = 0,
  subset = integer(),
  subsetAdjust = c("average", "previous")
)

## S4 method for signature 'OnDiskMSnExp,ObiwarpParam'
adjustRtime(object, param, msLevel = 1L)
```

```
## S4 method for signature 'ObiwarpParam'
binSize(object)

## S4 replacement method for signature 'ObiwarpParam'
binSize(object) <- value

## S4 method for signature 'ObiwarpParam'
centerSample(object)

## S4 replacement method for signature 'ObiwarpParam'
centerSample(object) <- value

## S4 method for signature 'ObiwarpParam'
response(object)

## S4 replacement method for signature 'ObiwarpParam'
response(object) <- value

## S4 method for signature 'ObiwarpParam'
distFun(object)

## S4 replacement method for signature 'ObiwarpParam'
distFun(object) <- value

## S4 method for signature 'ObiwarpParam'
gapInit(object)

## S4 replacement method for signature 'ObiwarpParam'
gapInit(object) <- value

## S4 method for signature 'ObiwarpParam'
gapExtend(object)

## S4 replacement method for signature 'ObiwarpParam'
gapExtend(object) <- value

## S4 method for signature 'ObiwarpParam'
factorDiag(object)

## S4 replacement method for signature 'ObiwarpParam'
factorDiag(object) <- value

## S4 method for signature 'ObiwarpParam'
factorGap(object)

## S4 replacement method for signature 'ObiwarpParam'
factorGap(object) <- value
```

```
## S4 method for signature 'ObiwarpParam'
localAlignment(object)

## S4 replacement method for signature 'ObiwarpParam'
localAlignment(object) <- value

## S4 method for signature 'ObiwarpParam'
initPenalty(object)

## S4 replacement method for signature 'ObiwarpParam'
initPenalty(object) <- value

## S4 method for signature 'ObiwarpParam'
subset(x)

## S4 replacement method for signature 'ObiwarpParam'
subset(object) <- value

## S4 method for signature 'ObiwarpParam'
subsetAdjust(object)

## S4 replacement method for signature 'ObiwarpParam'
subsetAdjust(object) <- value

## S4 method for signature 'XCMSnExp,ObiwarpParam'
adjustRtime(object, param, msLevel = 1L)
```

Arguments

`binSize`	`numeric(1)` defining the bin size (in mz dimension) to be used for the *profile matrix* generation. See `step` parameter in `profile-matrix` documentation for more details.
`centerSample`	`integer(1)` defining the index of the center sample in the experiment. It defaults to `floor(median(1:length(fileNames(object))))`. Note that if `subset` is used, the index passed with `centerSample` is within these subset samples.
`response`	`numeric(1)` defining the *responsiveness* of warping with `response = 0` giving linear warping on start and end points and `response = 100` warping using all bijective anchors.
`distFun`	character defining the distance function to be used. Allowed values are `"cor"` (Pearson's correlation), `"cor_opt"` (calculate only 10% diagonal band of distance matrix; better runtime), `"cov"` (covariance), `"prd"` (product) and `"euc"` (Euclidian distance). The default value is `distFun = "cor_opt"`.
`gapInit`	`numeric(1)` defining the penalty for gap opening. The default value for `gapInit` depends on the value of `distFun`: for `distFun = "cor"` and `distFun = "cor_opt"` it is `0.3`, for `distFun = "cov"` and `distFun = "prd"` `0.0` and for `distFun = "euc"` `0.9`.
`gapExtend`	`numeric(1)` defining the penalty for gap enlargement. The default value for `gapExtend` depends on the value of `distFun`, for `distFun = "cor"` and `distFun`

	`= "cor_opt"` it is `2.4`, for `distFun = "cov"` `11.7`, for `distFun = "euc"` `1.8` and for `distFun = "prd"` 7.8.
`factorDiag`	`numeric(1)` defining the local weight applied to diagonal moves in the alignment.
`factorGap`	`numeric(1)` defining the local weight for gap moves in the alignment.
`localAlignment`	`logical(1)` whether a local alignment should be performed instead of the default global alignment.
`initPenalty`	`numeric(1)` defining the penalty for initiating an alignment (for local alignment only).
`subset`	`integer` with the indices of samples within the experiment on which the alignment models should be estimated. Samples not part of the subset are adjusted based on the closest subset sample. See description above for more details.
`subsetAdjust`	`character` specifying the method with which non-subset samples should be adjusted. Supported options are `"previous"` and `"average"` (default). See description above for more information.
`object`	For `adjustRtime`: an `XCMSnExp` object. For all other methods: a `ObiwarpParam` object.
`param`	A `ObiwarpParam` object containing all settings for the alignment method.
`msLevel`	`integer` defining the MS level on which the retention time should be performed.
`value`	The value for the slot.
`x`	a `PeakGroupsParam` object.

Value

The `ObiwarpParam` function returns a `ObiwarpParam` class instance with all of the settings specified for obiwarp retention time adjustment and alignment.

For `adjustRtime,XCMSnExp,ObiwarpParam`: a `XCMSnExp` object with the results of the retention time adjustment step. These can be accessed with the `adjustedRtime` method. Retention time correction does also adjust the retention time of the identified chromatographic peaks (accessed *via* `chromPeaks`. Note that retention time correction drops all previous peak grouping results from the result object.

For `adjustRtime,OnDiskMSnExp,ObiwarpParam`: a `numeric` with the adjusted retention times per spectra (in the same order than `rtime`).

Slots

`binSize,centerSample,response,distFun,gapInit,gapExtend,factorDiag,factorGap,localAlignment,initPen`
See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `retcor` methods. All of the settings to the alignment algorithm can be passed with a `ObiwarpParam` object.

Alignment using obiwarp is performed on the retention time of spectra of on MS level. Retention times for spectra of other MS levels are subsequently adjusted based on the adjustment function defined on the retention times of the spectra of MS level `msLevel`.

Calling `adjustRtime` on an `XCMSnExp` object will cause all peak grouping (correspondence) results and any previous retention time adjustment results to be dropped.

Author(s)

Colin Smith, Johannes Rainer

References

John T. Prince and Edward M. Marcotte. "Chromatographic Alignment of ESI-LC-MS Proteomics Data Sets by Ordered Bijective Interpolated Warping" *Anal. Chem.* 2006, 78(17):6140-6152.

John T. Prince and Edward M. Marcotte. "Chromatographic Alignment of ESI-LC-MS Proteomic Data Sets by Ordered Bijective Interpolated Warping" *Anal. Chem.* 2006, 78 (17), 6140-6152.

See Also

`retcor.obiwarp` for the old user interface. `plotAdjustedRtime` for visualization of alignment results.

`XCMSnExp` for the object containing the results of the alignment.

Other retention time correction methods: `adjustRtime-peakGroups`, `adjustRtime()`

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Perform retention time correction:
res <- adjustRtime(faahko_sub, param = ObiwarpParam())

## As a result we get a numeric vector with the adjusted retention times for
## all spectra.
head(res)

## We can split this by file to get the adjusted retention times for each
## file
resL <- split(res, fromFile(res))
```

`adjustRtime-peakGroups`

Retention time correction based on alignment of house keeping peak groups

---

Description

This method performs retention time adjustment based on the alignment of chromatographic peak groups present in all/most samples (hence corresponding to house keeping compounds). First the retention time deviation of these peak groups is described by fitting either a polynomial (`smooth = "loess"`) or a linear (`smooth = "linear"`) model to the data points. These models are subsequently used to adjust the retention time of each spectrum in each sample.

It is also possible to exclude certain samples within an experiment from the estimation of the alignment models. The parameter `subset` allows to define the indices of samples within `object` that should be aligned. Samples not part of this `subset` are left out in the estimation of the alignment models, but their retention times are subsequently adjusted based on the alignment results of the closest sample in `subset` (close in terms of position within the `object`). Alignment could thus be performed on only *real* samples leaving out e.g. blanks, which are then in turn adjusted based on the closest real sample. Here it is up to the user to ensure that the samples within `object` are ordered correctly (e.g. by injection index).

How the non-subset samples are adjusted bases also on the parameter `subsetAdjust`: with `subsetAdjust = "previous"`, each non-subset sample is adjusted based on the closest previous subset sample which results in most cases with adjusted retention times of the non-subset sample being identical to the subset sample on which the adjustment bases. The second, default, option is to use `subsetAdjust = "average"` in which case each non subset sample is adjusted based on the average retention time adjustment from the previous and following subset sample. For the average a weighted mean is used with weights being the inverse of the distance of the non-subset sample to the subset samples used for alignment.

See also section *Alignment of experiments including blanks* in the *xcms* vignette for an example.

The `PeakGroupsParam` class allows to specify all settings for the retention time adjustment based on *house keeping* peak groups present in most samples. Instances should be created with the `PeakGroupsParam` constructor.

`adjustRtimePeakGroups` returns the features (peak groups) which would, depending on the provided `PeakGroupsParam`, be selected for alignment/retention time correction.

`minFraction`,`minFraction<-`: getter and setter for the `minFraction` slot of the object.

`extraPeaks`,`extraPeaks<-`: getter and setter for the `extraPeaks` slot of the object.

`smooth`,`smooth<-`: getter and setter for the `smooth` slot of the object.

`span`,`span<-`: getter and setter for the `span` slot of the object.

`family`,`family<-`: getter and setter for the `family` slot of the object.

`peakGroupsMatrix`,`peakGroupsMatrix<-`: getter and setter for the `peakGroupsMatrix` slot of the object.

`subset`,`subset<-`: getter and setter for the `subset` slot of the object.

`subsetAdjust`,`subsetAdjust<-`: getter and setter for the `subsetAdjust` slot of the object.

`adjustRtime,XCMSnExp,PeakGroupsParam`: performs retention time correction based on the alignment of peak groups (features) found in all/most samples. The correction function identified on these peak groups is applied to the retention time of all spectra in the object, i.e. retention times of all spectra, also MS level > 1 are adjusted.

Usage

```
PeakGroupsParam(
  minFraction = 0.9,
  extraPeaks = 1,
  smooth = "loess",
  span = 0.2,
  family = "gaussian",
  peakGroupsMatrix = matrix(nrow = 0, ncol = 0),
  subset = integer(),
  subsetAdjust = c("average", "previous")
)

adjustRtimePeakGroups(object, param = PeakGroupsParam(), msLevel = 1L)

## S4 method for signature 'PeakGroupsParam'
minFraction(object)

## S4 replacement method for signature 'PeakGroupsParam'
minFraction(object) <- value

## S4 method for signature 'PeakGroupsParam'
extraPeaks(object)

## S4 replacement method for signature 'PeakGroupsParam'
extraPeaks(object) <- value

## S4 method for signature 'PeakGroupsParam'
smooth(x)

## S4 replacement method for signature 'PeakGroupsParam'
smooth(object) <- value

## S4 method for signature 'PeakGroupsParam'
span(object)

## S4 replacement method for signature 'PeakGroupsParam'
span(object) <- value

## S4 method for signature 'PeakGroupsParam'
family(object)
```

```
## S4 replacement method for signature 'PeakGroupsParam'
family(object) <- value

## S4 method for signature 'PeakGroupsParam'
peakGroupsMatrix(object)

## S4 replacement method for signature 'PeakGroupsParam'
peakGroupsMatrix(object) <- value

## S4 method for signature 'PeakGroupsParam'
subset(x)

## S4 replacement method for signature 'PeakGroupsParam'
subset(object) <- value

## S4 method for signature 'PeakGroupsParam'
subsetAdjust(object)

## S4 replacement method for signature 'PeakGroupsParam'
subsetAdjust(object) <- value

## S4 method for signature 'XCMSnExp,PeakGroupsParam'
adjustRtime(object, param, msLevel = 1L)
```

Arguments

`minFraction`	`numeric(1)` between 0 and 1 defining the minimum required fraction of samples in which peaks for the peak group were identified. Peak groups passing this criteria will aligned across samples and retention times of individual spectra will be adjusted based on this alignment. For `minFraction = 1` the peak group has to contain peaks in all samples of the experiment. Note that if `subset` is provided, the specified fraction is relative to the defined subset of samples and not to the total number of samples within the experiment (i.e. a peak has to be present in the specified proportion of subset samples).
`extraPeaks`	`numeric(1)` defining the maximal number of additional peaks for all samples to be assigned to a peak group (i.e. feature) for retention time correction. For a data set with 6 samples, `extraPeaks = 1` uses all peak groups with a total peak count `<= 6 + 1`. The total peak count is the total number of peaks being assigned to a peak group and considers also multiple peaks within a sample being assigned to the group.
`smooth`	character defining the function to be used, to interpolate corrected retention times for all peak groups. Either `"loess"` or `"linear"`.
`span`	`numeric(1)` defining the degree of smoothing (if `smooth = "loess"`). This parameter is passed to the internal call to `loess`.
`family`	character defining the method to be used for loess smoothing. Allowed values are `"gaussian"` and `"symmetric"`.See `loess` for more information.

`peakGroupsMatrix`	optional `matrix` of (raw) retention times for the peak groups on which the alignment should be performed. Each column represents a sample, each row a feature/peak group. Such a matrix is for example returned by the `adjustRtimePeakGroups` method.
`subset`	`integer` with the indices of samples within the experiment on which the alignment models should be estimated. Samples not part of the subset are adjusted based on the closest subset sample. See description above for more details.
`subsetAdjust`	`character` specifying the method with which non-subset samples should be adjusted. Supported options are `"previous"` and `"average"` (default). See description above for more information.
`object`	For `adjustRtime`: an `XCMSnExp` object containing the results from a previous chromatographic peak detection (see `findChromPeaks`) and alignment analysis (see `groupChromPeaks`). For all other methods: a `PeakGroupsParam` object.
`param`	A `PeakGroupsParam` object containing all settings for the retention time correction method..
`msLevel`	`integer(1)` specifying the MS level. Currently only MS level 1 is supported.
`value`	The value for the slot.
`x`	a `PeakGroupsParam` object.

Value

The `PeakGroupsParam` function returns a `PeakGroupsParam` class instance with all of the settings specified for retention time adjustment based on *house keeping* features/peak groups.

For `adjustRtimePeakGroups`: a `matrix`, rows being features, columns samples, of retention times. The features are ordered by the median retention time across columns.

For `adjustRtime`: a `XCMSnExp` object with the results of the retention time adjustment step. These can be accessed with the `adjustedRtime` method. Retention time correction does also adjust the retention time of the identified chromatographic peaks (accessed *via* `chromPeaks`. Note that retention time correction drops all previous alignment results from the result object.

Slots

`minFraction,extraPeaks,smooth,span,family,peakGroupsMatrix,subset,subsetAdjust` See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `group` methods. All of the settings to the alignment algorithm can be passed with a `PeakGroupsParam` object.

The matrix with the (raw) retention times of the peak groups used in the alignment is added to the `peakGroupsMatrix` slot of the `PeakGroupsParam` object that is stored into the corresponding *process history step* (see `processHistory` for how to access the process history).

`adjustRtimePeakGroups` is supposed to be called *before* the sample alignment, but after a correspondence (peak grouping).

This method requires that a correspondence analysis has been performed on the data, i.e. that grouped chromatographic peaks/features are present (see `groupChromPeaks` for details).

Calling `adjustRtime` on an `XCMSnExp` object will cause all peak grouping (correspondence) results and any previous retention time adjustments to be dropped. In some instances, the `adjustRtime,XCMSnExp,PeakGroupsPara` re-adjusts adjusted retention times to ensure them being in the same order than the raw (original) retention times.

Author(s)

Colin Smith, Johannes Rainer

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" *Anal. Chem.* 2006, 78:779-787.

See Also

The `do_adjustRtime_peakGroups` core API function and `retcor.peakgroups` for the old user interface. `plotAdjustedRtime` for visualization of alignment results.

`XCMSnExp` for the object containing the results of the alignment.

Other retention time correction methods: `adjustRtime-obiwarp`, `adjustRtime()`

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")
res <- faahko_sub

## Disable parallel processing for this example
register(SerialParam())

head(chromPeaks(res))
## The number of peaks identified per sample:
table(chromPeaks(res)[, "sample"])

## Performing the peak grouping using the "peak density" method.
p <- PeakDensityParam(sampleGroups = c(1, 1, 1))
res <- groupChromPeaks(res, param = p)

## Perform the retention time adjustment using peak groups found in both
## files.
fgp <- PeakGroupsParam(minFraction = 1)

## Before running the alignment we can evaluate which features (peak groups)
```

```
## would be used based on the specified parameters.
pkGrps <- adjustRtimePeakGroups(res, param = fgp)

## We can also plot these to evaluate if the peak groups span a large portion
## of the retention time range.
plot(x = pkGrps[, 1], y = rep(1, nrow(pkGrps)), xlim = range(rtime(res)),
    ylim = c(1, 2), xlab = "rt", ylab = "", yaxt = "n")
points(x = pkGrps[, 2], y = rep(2, nrow(pkGrps)))
segments(x0 = pkGrps[, 1], x1 = pkGrps[, 2],
    y0 = rep(1, nrow(pkGrps)), y1 = rep(2, nrow(pkGrps)))
grid()
axis(side = 2, at = c(1, 2, 3), labels = colnames(pkGrps))

## Next we perform the alignment.
res <- adjustRtime(res, param = fgp)

## Any grouping information was dropped
hasFeatures(res)

## Plot the raw against the adjusted retention times.
plot(rtime(res, adjusted = FALSE),
    rtime(res), pch = 16, cex = 0.25, col = fromFile(res))

## Adjusterd retention times can be accessed using
## rtime(object, adjusted = TRUE) and adjustedRtime
all.equal(rtime(res), adjustedRtime(res))

## To extract the retention times grouped by sample/file:
rts <- rtime(res, bySample = TRUE)
```

`applyAdjustedRtime` *Replace raw with adjusted retention times*

Description

Replaces the raw retention times with the adjusted retention time or returns the object unchanged if none are present.

Usage

```
applyAdjustedRtime(object)
```

Arguments

`object`	An XCMSnExp object.

Details

Adjusted retention times are stored *in parallel* to the adjusted retention times in the `XCMSnExp`. The `applyAdjustedRtime` replaces the raw retention times (stored in the *feature data* (`fData` `data.frame`)) with the adjusted retention times.

Value

A `XCMSnExp` with the raw retention times being replaced with the adjusted retention time.

Note

Replacing the raw retention times with adjusted retention times disables the possibility to restore raw retention times using the `dropAdjustedRtime()` method. This function does **not** remove the retention time processing step with the settings of the alignment from the `processHistory()` of the `object` to ensure that the processing history is preserved.

Author(s)

Johannes Rainer

See Also

`adjustRtime()` for the function to perform the alignment (retention time correction).

```
[adjustedRtime()] for the method to extract adjusted retention times from
an [XCMSnExp] object.

[dropAdjustedRtime] for the method to delete alignment results and to
restore the raw retention times.
```

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

xod <- adjustRtime(faahko_sub, param = ObiwarpParam())

hasAdjustedRtime(xod)

## Replace raw retention times with adjusted retention times.
xod <- applyAdjustedRtime(xod)

## No adjusted retention times present
hasAdjustedRtime(xod)

## Raw retention times have been replaced with adjusted retention times
plot(split(rtime(faahko_sub), fromFile(faahko_sub))[[1]] -
    split(rtime(xod), fromFile(xod))[[1]], type = "l")

## And the process history still contains the settings for the alignment
processHistory(xod)
```

`AutoLockMass-methods` *Automatic parameter for Lock mass fixing* `AutoLockMass` ~~

Description

`AutoLockMass` - This function decides where the lock mass scans are in the xcmsRaw object. This is done by using the scan time differences.

Arguments

`object`	An `xcmsRaw-class` object

Value

`AutoLockMass` A numeric vector of scan locations corresponding to lock Mass scans

Methods

object = "xcmsRaw" `signature(object = "xcmsRaw")`

Author(s)

Paul Benton, `<hpaul.benton08@imperial.ac.uk>`

Examples

```
## Not run: library(xcms)
    library(faahKO)
    ## These files do not have this problem
    ## to correct for but just for an example
    cdfpath <- system.file("cdf", package = "faahKO")
    cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)
    xr<-xcmsRaw(cdffiles[1])
    xr
    ##Lets assume that the lockmass starts at 1 and is every 100 scans
    lockMass<-xcms:::makeacqNum(xr, freq=100, start=1)
    ## these are equalvent
    lockmass2<-AutoLockMass(xr)
    all((lockmass == lockmass2) == TRUE)

    ob<-stitch(xr, lockMass)

## End(Not run)
```

`bin,XCMSnExp-method` *XCMSnExp data manipulation methods inherited from MSnbase*

Description

The methods listed on this page are `XCMSnExp` methods inherited from its parent, the `OnDiskMSnExp` class from the `MSnbase` package, that alter the raw data or are related to data subsetting. Thus calling any of these methods causes all `xcms` pre-processing results to be removed from the `XCMSnExp` object to ensure its data integrity.

`bin`: allows to *bin* spectra. See `bin` documentation in the `MSnbase` package for more details and examples.

`clean`: removes unused `0` intensity data points. See `clean` documentation in the `MSnbase` package for details and examples.

`filterAcquisitionNum`: filters the `XCMSnExp` object keeping only spectra with the provided acquisition numbers. See `filterAcquisitionNum` for details and examples.

The `normalize` method performs basic normalization of spectra intensities. See `normalize` documentation in the `MSnbase` package for details and examples.

The `pickPeaks` method performs peak picking. See `pickPeaks` documentation for details and examples.

The `removePeaks` method removes mass peaks (intensities) lower than a threshold. Note that these peaks refer to *mass* peaks, which are different to the chromatographic peaks detected and analyzed in a metabolomics experiment! See `removePeaks` documentation for details and examples.

The `smooth` method smooths spectra. See `smooth` documentation in `MSnbase` for details and examples.

Usage

```
## S4 method for signature 'XCMSnExp'
bin(x, binSize = 1L, msLevel.)

## S4 method for signature 'XCMSnExp'
clean(object, all = FALSE, verbose = FALSE, msLevel.)

## S4 method for signature 'XCMSnExp'
filterAcquisitionNum(object, n, file)

## S4 method for signature 'XCMSnExp'
normalize(object, method = c("max", "sum"), ...)

## S4 method for signature 'XCMSnExp'
pickPeaks(
  object,
  halfWindowSize = 3L,
  method = c("MAD", "SuperSmoother"),
```

```
  SNR = 0L,
  ...
)

## S4 method for signature 'XCMSnExp'
removePeaks(object, t = "min", verbose = FALSE, msLevel.)

## S4 method for signature 'XCMSnExp'
smooth(
  x,
  method = c("SavitzkyGolay", "MovingAverage"),
  halfWindowSize = 2L,
  verbose = FALSE,
  ...
)
```

Arguments

`x`	`XCMSnExp` or `OnDiskMSnExp` object.
`binSize`	`numeric(1)` defining the size of a bin (in Dalton).
`msLevel.`	For `bin`, `clean`, `filterMsLevel`, `removePeaks`: `numeric(1)` defining the MS level(s) to which operations should be applied or to which the object should be subsetted.
`object`	`XCMSnExp` or `OnDiskMSnExp` object.
`all`	For `clean`: `logical(1)`, if `TRUE` all zeros are removed.
`verbose`	`logical(1)` whether progress information should be displayed.
`n`	For `filterAcquisitionNum`: `integer` defining the acquisition numbers of the spectra to which the data set should be sub-setted.
`file`	For `filterAcquisitionNum`: `integer` defining the file index within the object to subset the object by file.
`method`	For `normalize`: `character(1)` specifying the normalization method. See `normalize` in the `MSnbase` package for details. For `pickPeaks`: `character(1)` defining the method. See `pickPeaks` for options. For `smooth`: `character(1)` defining the method. See `smooth` in the `MSnbase` package for options and details.
`...`	Optional additional arguments.
`halfWindowSize`	For `pickPeaks` and `smooth`: `integer(1)` defining the window size for the peak picking. See `pickPeaks` and `smooth` in the `MSnbase` package for details and options.
`SNR`	For `pickPeaks`: `numeric(1)` defining the signal to noise ratio to be considered. See `pickPeaks` documentation for details.
`t`	For `removePeaks`: either a `numeric(1)` or `"min"` defining the threshold (method) to be used. See `removePeaks` for details.

Value

For all methods: a `XCMSnExp` object.

Author(s)

Johannes Rainer

See Also

`XCMSnExp-filter` for methods to filter and subset `XCMSnExp` objects. `XCMSnExp` for base class documentation. `OnDiskMSnExp` for the documentation of the parent class.

`binYonX` *Aggregate values in y for bins defined on x*

Description

This functions takes two same-sized numeric vectors `x` and `y`, bins/cuts `x` into bins (either a predefined number of equal-sized bins or bins of a pre-defined size) and aggregates values in `y` corresponding to `x` values falling within each bin. By default (i.e. `method = "max"`) the maximal `y` value for the corresponding `x` values is identified. `x` is expected to be incrementally sorted and, if not, it will be internally sorted (in which case also `y` will be ordered according to the order of `x`).

Usage

```
binYonX(
  x,
  y,
  breaks,
  nBins,
  binSize,
  binFromX,
  binToX,
  fromIdx = 1L,
  toIdx = length(x),
  method = "max",
  baseValue,
  sortedX = !is.unsorted(x),
  shiftByHalfBinSize = FALSE,
  returnIndex = FALSE,
  returnX = TRUE
)
```

Arguments

`x`	Numeric vector to be used for binning.
`y`	Numeric vector (same length than `x`) from which the maximum values for each bin should be defined. If not provided, `x` will be used.
`breaks`	Numeric vector defining the breaks for the bins, i.e. the lower and upper values for each bin. See examples below.

`nBins`	integer(1) defining the number of desired bins.
`binSize`	numeric(1) defining the desired bin size.
`binFromX`	Optional numeric(1) allowing to manually specify the range of x-values to be used for binning. This will affect only the calculation of the breaks for the bins (i.e. if `nBins` or `binSize` is provided). If not provided the minimal value in the sub-set `fromIdx`-`toIdx` in input vector `x` will be used.
`binToX`	Same as `binFromX`, but defining the maximum x-value to be used for binning.
`fromIdx`	Integer vector defining the start position of one or multiple sub-sets of input vector `x` that should be used for binning.
`toIdx`	Same as `toIdx`, but defining the maximum index (or indices) in x to be used for binning.
`method`	A character string specifying the method that should be used to aggregate values in `y`. Allowed are `"max"`, `"min"`, `"sum"` and `"mean"` to identify the maximal or minimal value or to sum all values within a bin or calculate their mean value.
`baseValue`	The base value for empty bins (i.e. bins into which either no values in `x` did fall, or to which only `NA` values in `y` were assigned). By default (i.e. if not specified), `NA` is assigned to such bins.
`sortedX`	Whether `x` is sorted.
`shiftByHalfBinSize`	Logical specifying whether the bins should be shifted by half the bin size to the left. Thus, the first bin will have its center at `fromX` and its lower and upper boundary are `fromX - binSize/2` and `fromX + binSize/2`. This argument is ignored if `breaks` are provided.
`returnIndex`	Logical indicating whether the index of the max (if `method = "max"`) or min (if `method = "min"`) value within each bin in input vector `x` should also be reported. For methods other than `"max"` or `"min"` this argument is ignored.
`returnX`	`logical` allowing to avoid returning `$x`, i.e. the mid-points of the bins. `returnX = FALSE` might be useful in cases where `breaks` are pre-defined as it considerably reduces the memory demand.

Details

The breaks defining the boundary of each bin can be either passed directly to the function with the argument `breaks`, or are calculated on the data based on arguments `nBins` or `binSize` along with `fromIdx`, `toIdx` and optionally `binFromX` and `binToX`. Arguments `fromIdx` and `toIdx` allow to specify subset(s) of the input vector `x` on which bins should be calculated. The default the full `x` vector is considered. Also, if not specified otherwise with arguments `binFromX` and `binToX` , the range of the bins within each of the sub-sets will be from `x[fromIdx]` to `x[toIdx]`. Arguments `binFromX` and `binToX` allow to overwrite this by manually defining the a range on which the breaks should be calculated. See examples below for more details.

Calculation of breaks: for `nBins` the breaks correspond to `seq(min(x[fromIdx])), max(x[fromIdx], length.out = (nBins + 1))`. For `binSize` the breaks correspond to `seq(min(x[fromIdx]), max(x[toIdx]), by = binSize)` with the exception that the last break value is forced to be equal to `max(x[toIdx])`. This ensures that all values from the specified range are covered by the breaks defining the bins. The last bin could however in some instances be slightly larger than `binSize`. See `breaks_on_binSize` and `breaks_on_nBins` for more details.

Value

Returns a list of length 2, the first element (named `"x"`) contains the bin mid-points, the second element (named `"y"`) the aggregated values from input vector `y` within each bin. For `returnIndex = TRUE` the list contains an additional element `"index"` with the index of the max or min (depending on whether `method = "max"` or `method = "min"`) value within each bin in input vector `x`.

Note

The function ensures that all values within the range used to define the breaks are considered in the binning (and assigned to a bin). This means that for all bins except the last one values in `x` have to be >= `xlower` and < `xupper` (with `xlower` and `xupper` being the lower and upper boundary, respectively). For the last bin the condition is `x >= xlower & x <= xupper`. Note also that if `shiftByHalfBinSize` is `TRUE` the range of values that is used for binning is expanded by `binSize` (i.e. the lower boundary will be `fromX - binSize/2`, the upper `toX + binSize/2`). Setting this argument to `TRUE` resembles the binning that is/was used in `profBin` function from `xcms` < 1.51.

`NA` handling: by default the function ignores `NA` values in `y` (thus inherently assumes `na.rm = TRUE`). No `NA` values are allowed in `x`.

Author(s)

Johannes Rainer

See Also

`imputeLinInterpol`

Examples

```
########
## Simple example illustrating the breaks and the binning.
##
## Define breaks for 5 bins:
brks <- seq(2, 12, length.out = 6)
## The first bin is then [2,4), the second [4,6) and so on.
brks
## Get the max value falling within each bin.
binYonX(x = 1:16, y = 1:16, breaks = brks)
## Thus, the largest value in x = 1:16 falling into the bin [2,4) (i.e. being
## >= 2 and < 4) is 3, the largest one falling into [4,6) is 5 and so on.
## Note however the function ensures that the minimal and maximal x-value
## (in this example 1 and 12) fall within a bin, i.e. 12 is considered for
## the last bin.

#######
## Performing the binning ons sub-set of x
##
X <- 1:16
## Bin X from element 4 to 10 into 5 bins.
X[4:10]
binYonX(X, X, nBins = 5L, fromIdx = 4, toIdx = 10)
```

```
## This defines breaks for 5 bins on the values from 4 to 10 and bins
## the values into these 5 bins. Alternatively, we could manually specify
## the range for the binning, i.e. the minimal and maximal value for the
## breaks:
binYonX(X, X, nBins = 5L, fromIdx = 4, toIdx = 10, binFromX = 1, binToX = 16)
## In this case the breaks for 5 bins were defined from a value 1 to 16 and
## the values 4 to 10 were binned based on these breaks.

#######
## Bin values within a sub-set of x, second example
##
## This example illustrates how the fromIdx and toIdx parameters can be used.
## x defines 3 times the sequence form 1 to 10, while y is the sequence from
## 1 to 30. In this very simple example x is supposed to represent M/Z values
## from 3 consecutive scans and y the intensities measured for each M/Z in
## each scan. We want to get the maximum intensities for M/Z value bins only
## for the second scan, and thus we use fromIdx = 11 and toIdx = 20. The breaks
## for the bins are defined with the nBins, binFromX and binToX.
X <- rep(1:10, 3)
Y <- 1:30
## Bin the M/Z values in the second scan into 5 bins and get the maximum
## intensity for each bin. Note that we have to specify sortedX = TRUE as
## the x and y vectors would be sorted otherwise.
binYonX(X, Y, nBins = 5L, sortedX = TRUE, fromIdx = 11, toIdx = 20)

#######
## Bin in overlapping sub-sets of X
##
## In this example we define overlapping sub-sets of X and perform the binning
## within these.
X <- 1:30
## Define the start and end indices of the sub-sets.
fIdx <- c(2, 8, 21)
tIdx <- c(10, 25, 30)
binYonX(X, nBins = 5L, fromIdx = fIdx, toIdx = tIdx)
## The same, but pre-defining also the desired range of the bins.
binYonX(X, nBins = 5L, fromIdx = fIdx, toIdx = tIdx, binFromX = 4, binToX = 28)
## The same bins are thus used for each sub-set.
```

`breaks_on_binSize` *Generate breaks for binning using a defined bin size.*

Description

Defines breaks for `binSize` sized bins for values ranging from `fromX` to `toX`.

Usage

```
breaks_on_binSize(fromX, toX, binSize)
```

Arguments

`fromX`	numeric(1) specifying the lowest value for the bins.
`toX`	numeric(1) specifying the largest value for the bins.
`binSize`	numeric(1) defining the size of a bin.

Details

This function creates breaks for bins of size `binSize`. The function ensures that the full data range is included in the bins, i.e. the last value (upper boundary of the last bin) is always equal `toX`. This however means that the size of the last bin will not always be equal to the desired bin size. See examples for more details and a comparisom to R's `seq` function.

Value

A numeric vector defining the lower and upper bounds of the bins.

Author(s)

Johannes Rainer

See Also

`binYonX` for a binning function.

Other functions to define bins: `breaks_on_nBins()`

Examples

```
## Define breaks with a size of 0.13 for a data range from 1 to 10:
breaks_on_binSize(1, 10, 0.13)
## The size of the last bin is however larger than 0.13:
diff(breaks_on_binSize(1, 10, 0.13))
## If we would use seq, the max value would not be included:
seq(1, 10, by = 0.13)

## In the next example we use binSize that leads to an additional last bin with
## a smaller binSize:
breaks_on_binSize(1, 10, 0.51)
## Again, the max value is included, but the size of the last bin is < 0.51.
diff(breaks_on_binSize(1, 10, 0.51))
## Using just seq would result in the following bin definition:
seq(1, 10, by = 0.51)
## Thus it defines one bin (break) less.
```

`breaks_on_nBins` *Generate breaks for binning*

Description

Calculate breaks for same-sized bins for data values from `fromX` to `toX`.

Usage

```
breaks_on_nBins(fromX, toX, nBins, shiftByHalfBinSize = FALSE)
```

Arguments

`fromX`	numeric(1) specifying the lowest value for the bins.
`toX`	numeric(1) specifying the largest value for the bins.
`nBins`	numeric(1) defining the number of bins.
`shiftByHalfBinSize`	Logical indicating whether the bins should be shifted left by half bin size. This results centered bins, i.e. the first bin being centered at `fromX` and the last around `toX`.

Details

This generates bins such as a call to `seq(fromX, toX, length.out = nBins)` would. The first and second element in the result vector thus defines the lower and upper boundary for the first bin, the second and third value for the second bin and so on.

Value

A numeric vector of length `nBins + 1` defining the lower and upper bounds of the bins.

Author(s)

Johannes Rainer

See Also

`binYonX` for a binning function.

Other functions to define bins: `breaks_on_binSize()`

Examples

```
## Create breaks to bin values from 3 to 20 into 20 bins
breaks_on_nBins(3, 20, nBins = 20)
## The same call but using shiftByHalfBinSize
breaks_on_nBins(3, 20, nBins = 20, shiftByHalfBinSize = TRUE)
```

`c-methods` *Combine xcmsSet objects*

Description

Combines the samples and peaks from multiple `xcmsSet` objects into a single object. Group and retention time correction data are discarded. The `profinfo` list is set to be equal to the first object.

Arguments

`xs1`	`xcmsSet` object
`...`	`xcmsSet` objects

Value

A `xcmsSet` object.

Methods

xs1 = "xcmsRaw" `c(xs1, ...)`

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`xcmsSet-class`

`CalibrantMassParam-class` *Calibrant mass based calibration of chromatgraphic peaks*

Description

Calibrate peaks using mz values of known masses/calibrants. mz values of identified peaks are adjusted based on peaks that are close to the provided mz values. See details below for more information.

The `isCalibrated` function returns `TRUE` if chromatographic peaks of the XCMSnExp object `x` were calibrated and `FALSE` otherwise.

Usage

```
CalibrantMassParam(
  mz = list(),
  mzabs = 1e-04,
  mzppm = 5,
  neighbors = 3,
  method = "linear"
)

isCalibrated(object)

## S4 method for signature 'XCMSnExp'
calibrate(object, param)
```

Arguments

`mz`	a `numeric` or `list` of `numeric` vectors with reference mz values. If a `numeric` vector is provided, this is used for each sample in the `XCMSnExp` object. If a `list` is provided, it's length has to be equal to the number of samples in the experiment.
`mzabs`	`numeric(1)` the absolute error/deviation for matching peaks to calibrants (in Da).
`mzppm`	`numeric(1)` the relative error for matching peaks to calibrants in ppm (parts per million).
`neighbors`	`integer(1)` with the maximal number of peaks within the permitted distance to the calibrants that are considered. Among these the mz value of the peak with the largest intensity is used in the calibration function estimation.
`method`	`character(1)` defining the method that should be used to estimate the calibration function. Can be `"shift"`, `"linear"` (default) or `"edgeshift"`.
`object`	An XCMSnExp object.
`param`	The `CalibrantMassParam` object with the calibration settings.

Details

The method does first identify peaks that are close to the provided mz values and, given that there difference to the calibrants is smaller than the user provided cut off (based on arguments `mzabs` and `mzppm`), their mz values are replaced with the provided mz values. The mz values of all other peaks are either globally shifted (for `method = "shift"` or estimated by a linear model through all calibrants. Peaks are considered close to a calibrant mz if the difference between the calibrant and its mz is `<= mzabs + mz * mzppm /1e6`.

Adjustment methods: adjustment function/factor is estimated using the difference between calibrant and peak mz values only for peaks that are close enough to the calibrants. The availabel methods are:

- `shift`: shifts the m/z of each peak by a global factor which corresponds to the average difference between peak mz and calibrant mz.

- `linear`: fits a linear model throught the differences between calibrant and peak mz values and adjusts the mz values of all peaks using this.
- `edgeshift`: performs same adjustment as `linear` for peaks that are within the mz range of the calibrants and shift outside of it.

For more information, details and examples refer to the *xcms-direct-injection* vignette.

Value

For `CalibrantMassParam`: a `CalibrantMassParam` instance. For `calibrate`: an XCMSnExp object with chromatographic peaks being calibrated. **Be aware** that the actual raw mz values are not (yet) calibrated, but **only** the identified chromatographic peaks.

The `CalibrantMassParam` function returns an instance of the `CalibrantMassParam` class with all settings and properties set.

The `calibrate` method returns an XCMSnExp object with the chromatographic peaks being calibrated. Note that **only** the detected peaks are calibrated, but not the individual mz values in each spectrum.

Note

`CalibrantMassParam` classes don't have exported getter or setter methods.

Author(s)

Joachim Bargsten, Johannes Rainer

`calibrate-methods` *Calibrate peaks for correcting unprecise m/z values*

Description

Calibrate peaks of a xcmsSet via a set of known masses

Arguments

`object`	a `xcmsSet` object with uncalibrated mz
`calibrants`	a vector or a list of vectors with reference m/z-values
`method`	the used calibrating-method, see below
`mzppm`	the relative error used for matching peaks in ppm (parts per million)
`mzabs`	the absolute error used for matching peaks in Da
`neighbours`	the number of neighbours from wich the one with the highest intensity is used (instead of the nearest)
`plotres`	can be set to TRUE if wanted a result-plot showing the found m/z with the distances and the regression

Value

`object`	a `xcmsSet` with one ore more samples
`calibrants`	for each sample different calibrants can be used, if a list of m/z-vectors is given. The length of the list must be the same as the number of samples, alternatively a single vector of masses can be given which is used for all samples.
`method`	"shift" for shifting each m/z, "linear" does a linear regression and adds a linear term to each m/z. "edgeshift" does a linear regression within the range of the mz-calibrants and a shift outside.

Methods

object = "xcmsSet" `calibrate(object, calibrants,method="linear", mzabs=0.0001, mzppm=5, neighbours=3, plotres=FALSE)`

See Also

`xcmsSet-class`,

`chromatogram,XCMSnExp-method`

Extracting chromatograms

Description

`chromatogram`: extract chromatographic data (such as an extracted ion chromatogram, a base peak chromatogram or total ion chromatogram) from an OnDiskMSnExp or XCMSnExp objects. See also the help page of the `chromatogram` function in the `MSnbase` package.

Usage

```
## S4 method for signature 'XCMSnExp'
chromatogram(
  object,
  rt,
  mz,
  aggregationFun = "sum",
  missing = NA_real_,
  msLevel = 1L,
  BPPARAM = bpparam(),
  adjustedRtime = hasAdjustedRtime(object),
  filled = FALSE,
  include = c("apex_within", "any", "none")
)
```

Arguments

`object`	Either a OnDiskMSnExp or XCMSnExp object from which the chromatograms should be extracted.
`rt`	`numeric(2)` or two-column `matrix` defining the lower and upper boundary for the retention time range(s). If not specified, the full retention time range of the original data will be used.
`mz`	`numeric(2)` or two-column `matrix` defining the lower and upper mz value for the MS data slice(s). If not specified, the chromatograms will be calculated on the full mz range.
`aggregationFun`	`character(1)` specifying the function to be used to aggregate intensity values across the mz value range for the same retention time. Allowed values are `"sum"` (the default), `"max"`, `"mean"` and `"min"`.
`missing`	`numeric(1)` allowing to specify the intensity value to be used if for a given retention time no signal was measured within the mz range of the corresponding scan. Defaults to `NA_real_` (see also Details and Notes sections below). Use `missing = 0` to resemble the behaviour of the `getEIC` from the *old* user interface.
`msLevel`	`integer(1)` specifying the MS level from which the chromatogram should be extracted. Defaults to `msLevel = 1L`.
`BPPARAM`	Parallelisation backend to be used, which will depend on the architecture. Default is `BiocParallel::bparam()`.
`adjustedRtime`	For `chromatogram,XCMSnExp`: whether the adjusted (`adjustedRtime = TRUE`) or raw retention times (`adjustedRtime = FALSE`) should be used for filtering and returned in the resulting MChromatograms object. Adjusted retention times are used by default if available.
`filled`	`logical(1)` whether filled-in peaks should also be returned. Defaults to `filled = FALSE`, i.e. returns only detected chromatographic peaks in the result object.
`include`	`character(1)` defining which chromatographic peaks should be returned. Supported are `include = "apex_within"` (the default) which returns chromatographic peaks that have their apex within the `mz` `rt` range, `include = "any"` to return all chromatographic peaks which m/z and rt ranges overlap the `mz` and `rt` or `include = "none"` to not include any chromatographic peaks.

Details

Arguments `rt` and `mz` allow to specify the MS data slice (i.e. the m/z range and retention time window) from which the chromatogram should be extracted. These parameters can be either a `numeric` of length 2 with the lower and upper limit, or a `matrix` with two columns with the lower and upper limits to extract multiple EICs at once. The parameter `aggregationSum` allows to specify the function to be used to aggregate the intensities across the m/z range for the same retention time. Setting `aggregationFun = "sum"` would e.g. allow to calculate the **total ion chromatogram** (TIC), `aggregationFun = "max"` the **base peak chromatogram** (BPC).

If for a given retention time no intensity is measured in that spectrum a `NA` intensity value is returned by default. This can be changed with the parameter `missing`, setting `missing = 0` would result in a `0` intensity being returned in these cases.

Value

`chromatogram` returns a XChromatograms object with the number of columns corresponding to the number of files in `object` and number of rows the number of specified ranges (i.e. number of rows of matrices provided with arguments `mz` and/or `rt`). All chromatographic peaks with their apex position within the m/z and retention time range are also retained as well as all feature definitions for these peaks.

Note

For XCMSnExp objects, if adjusted retention times are available, the `chromatogram` method will by default report and use these (for the subsetting based on the provided parameter `rt`). This can be changed by setting `adjustedRtime = FALSE`.

Author(s)

Johannes Rainer

See Also

XCMSnExp for the data object. Chromatogram for the object representing chromatographic data.

```
[XChromatograms] for the object allowing to arrange
multiple [XChromatogram] objects.

[plot] to plot a [XChromatogram] or [MChromatograms] objects.

`as` (`as(x, "data.frame")`) in `MSnbase` for a method to extract
the MS data as `data.frame`.
```

Examples

```
## Load a test data set with identified chromatographic peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Extract the ion chromatogram for one chromatographic peak in the data.
chrs <- chromatogram(faahko_sub, rt = c(2700, 2900), mz = 335)

chrs

## Identified chromatographic peaks
chromPeaks(chrs)

## Plot the chromatogram
plot(chrs)

## Extract chromatograms for multiple ranges.
```

```
mzr <- matrix(c(335, 335, 344, 344), ncol = 2, byrow = TRUE)
rtr <- matrix(c(2700, 2900, 2600, 2750), ncol = 2, byrow = TRUE)
chrs <- chromatogram(faahko_sub, mz = mzr, rt = rtr)

chromPeaks(chrs)

plot(chrs)

## Get access to all chromatograms for the second mz/rt range
chrs[1, ]

## Plot just that one
plot(chrs[1, , drop = FALSE])
```

`chromatographic-peak-detection`
Chromatographic peak detection methods.

Description

The `findChromPeaks` methods perform the chromatographic peak detection on LC/GC-MS data and are part of the modernized `xcms` user interface.

The implemented peak detection methods in chromatographic space are:

centWave chromatographic peak detection using the *centWave* method. See `centWave` for more details.

centWave with predicted isotopes peak detection using a two-step centWave-based approach considering also feature isotopes. See `centWaveWithPredIsoROIs` for more details.

matchedFilter peak detection in chromatographic space. See `matchedFilter` for more details.

massifquant peak detection using the Kalman filter-based method. See `massifquant` for more details.

MSW single-spectrum non-chromatography MS data peak detection. See `MSW` for more details.

Author(s)

Johannes Rainer

See Also

`findPeaks` for the *old* peak detection methods.

`plotChromPeaks` to plot identified chromatographic peaks for one file.

`highlightChromPeaks` to highlight identified chromatographic peaks in an extracted ion chromatogram plot.

`refineChromPeaks` for methods to refine or clean identified chromatographic peaks.

`manualChromPeaks` to manually add/define chromatographic peaks.

Other peak detection methods: `findChromPeaks-centWaveWithPredIsoROIs`, `findChromPeaks-centWave`, `findChromPeaks-massifquant`, `findChromPeaks-matchedFilter`, `findPeaks-MSW`

`chromPeakSpectra` *Extract spectra associated with chromatographic peaks*

Description

Extract (MS1 or MS2) spectra from an XCMSnExp object for each identified chromatographic peak. The function returns by default spectra for chromatographic peaks of **all** MS levels, but parameter `peaks` allows to restrict the result to selected chromatographic peaks. For `msLevel = 1L` (only supported for `return.type = "Spectra"` or `return.type = "List"`) MS1 spectra within the retention time boundaries (in the file in which the peak was detected) are returned. For `msLevel = 2L` MS2 spectra are returned for a chromatographic peak if their precursor m/z is within the retention time and m/z range of the chromatographic peak. Parameter `method` allows to define whether all or a single spectrum should be returned:

- `method = "all"`: (default): return all spectra for each peak.
- `method = "closest_rt"`: return the spectrum with the retention time closest to the peak's retention time (at apex).
- `method = "closest_mz"`: return the spectrum with the precursor m/z closest to the peaks's m/z (at apex); only supported for `msLevel = 2L`.
- `method = "signal"`: return the spectrum with the sum of intensities most similar to the peak's apex signal (`"maxo"`); only supported for `msLevel = 2L`.
- `method = "largest_tic"`: return the spectrum with the largest total signal (sum of peaks intensities).
- `method = "largest_bpi"`: return the spectrum with the largest peak intensity (maximal peak intensity).

Parameter `return.type` allows to specify the *type* of the result object. Please use `return.type = "Spectra"` or `return.type = "List"`, `return.type = "list"` or the default `return.type = "MSpectra"` will be deprecated (also, they do not support extracting MS1 spectra).

See also the *LC-MS/MS data analysis* vignette for more details and examples.

Usage

```
chromPeakSpectra(
  x,
  msLevel = 2L,
  expandRt = 0,
  expandMz = 0,
  ppm = 0,
 method = c("all", "closest_rt", "closest_mz", "signal", "largest_tic", "largest_bpi"),
  skipFilled = FALSE,
  return.type = c("MSpectra", "Spectra", "list", "List"),
  peaks = character()
)
```

Arguments

`x`	XCMSnExp object with identified chromatographic peaks.
`msLevel`	`integer(1)` defining whether MS1 or MS2 spectra should be returned. `msLevel = 1` is currently only supported for `return.type` being `"Spectra"` or `"List"`.
`expandRt`	`numeric(1)` to expand the retention time range of each peak by a constant value on each side.
`expandMz`	`numeric(1)` to expand the m/z range of each peak by a constant value on each side.
`ppm`	`numeric(1)` to expand the m/z range of each peak (on each side) by a value dependent on the peak's m/z.
`method`	`character(1)` specifying which spectra to include in the result. Defaults to `method = "all"`. See function description for details.
`skipFilled`	`logical(1)` whether spectra for filled-in peaks should be reported or not.
`return.type`	`character(1)` defining the result type. Defaults to `return.type = "MSpectra"` but `return.type = "Spectra"` or `return.type = "List"` are preferred. See below for more information.
`peaks`	`character`, `logical` or `integer` allowing to specify a subset of chromatographic peaks in `chromPeaks` for which spectra should be returned (providing either their ID, a logical vector same length than `nrow(chromPeaks(x))` or their index in `chromPeaks(x)`). This parameter overrides `skipFilled` and is only supported for `return.type` being either `"Spectra"` or `"List"`.

Value

parameter `return.type` allow to specify the type of the returned object:

- `return.type = "MSpectra"`: a MSpectra object with elements being Spectrum objects. The result objects contains all spectra for all peaks. Metadata column `"peak_id"` provides the ID of the respective peak (i.e. its rowname in `chromPeaks()`).
- `return.type = "Spectra"`: a `Spectra` object (defined in the `Spectra` package). The result contains all spectra for all peaks. Metadata column `"peak_id"` provides the ID of the respective peak (i.e. its rowname in `chromPeaks()` and `"peak_index"` its index in the object's `chromPeaks` matrix.
- `return.type = "list"`: `list` of `list`s that are either of length 0 or contain Spectrum2 object(s) within the m/z-rt range. The length of the list matches the number of peaks.
- `return.type = "List"`: `List` of length equal to the number of chromatographic peaks is returned with elements being either `NULL` (no spectrum found) or a `Spectra` object.

Author(s)

Johannes Rainer

Examples

```
## Read a file with DDA LC-MS/MS data
fl <- system.file("TripleTOF-SWATH/PestMix1_DDA.mzML", package = "msdata")
dda <- readMSData(fl, mode = "onDisk")

## Subset the object to reduce runtime of the example
dda <- filterRt(dda, c(200, 400))

## Perform MS1 peak detection
dda <- findChromPeaks(dda, CentWaveParam(peakwidth = c(5, 15), prefilter = c(5, 1000)))

## Load the required Spectra package and return all MS2 spectro for each
## chromatographic peaks as a Spectra object
ms2_sps <- chromPeakSpectra(dda, return.type = "Spectra")
ms2_sps

## columns peak_id or peak_index assign spectra to the chromatographic peaks
ms2_sps$peak_id
ms2_sps$peak_index
chromPeaks(dda)

## Alternatively, return the result as a List of Spectra objects. This list
## is parallel to chromPeaks hence the mapping between chromatographic peaks
## and MS2 spectra is easier.
ms2_sps <- chromPeakSpectra(dda, return.type = "List")
ms2_sps[[1L]]
length(ms2_sps)

## In addition to MS2 spectra we could also return the MS1 spectrum for each
## chromatographic peak which is closest to the peak's apex position.
ms1_sps <- chromPeakSpectra(dda, msLevel = 1L, method = "closest_rt",
    return.type = "Spectra")
ms1_sps

## Parameter peaks would allow to extract spectra for specific peaks only
chromPeakSpectra(dda, msLevel = 1L, method = "closest_rt", peaks = c(3, 5))
```

`CleanPeaksParam` *Remove chromatographic peaks with too large rt width*

Description

Remove chromatographic peaks with a retention time range larger than the provided maximal acceptable width (`maxPeakwidth`).

Usage

```
CleanPeaksParam(maxPeakwidth = 10)

## S4 method for signature 'XCMSnExp,CleanPeaksParam'
refineChromPeaks(object, param = CleanPeaksParam(), msLevel = 1L)
```

Arguments

`maxPeakwidth`	for `CleanPeaksParam`: `numeric(1)` defining the maximal allowed peak width (in retention time).
`object`	XCMSnExp object with identified chromatographic peaks.
`param`	`CleanPeaksParam` object defining the settings for the method.
`msLevel`	`integer` defining for which MS level(s) the chromatographic peaks should be cleaned.

Value

`XCMSnExp` object with chromatographic peaks exceeding the specified maximal retention time width being removed.

Note

`refineChromPeaks` methods will always remove feature definitions, because a call to this method can change or remove identified chromatographic peaks, which may be part of features.

Author(s)

Johannes Rainer

See Also

Other chromatographic peak refinement methods: `FilterIntensityParam`, `MergeNeighboringPeaksParam`

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Distribution of chromatographic peak widths
quantile(chromPeaks(faahko_sub)[, "rtmax"] - chromPeaks(faahko_sub)[, "rtmin"])

## Remove all chromatographic peaks with a width larger 60 seconds
data <- refineChromPeaks(faahko_sub, param = CleanPeaksParam(60))

quantile(chromPeaks(data)[, "rtmax"] - chromPeaks(data)[, "rtmin"])
```

`collect-methods` *Collect MS^n peaks into xcmsFragments*

Description

Collecting Peaks into `xcmsFragments`s from several MS-runs using `xcmsSet` and `xcmsRaw`.

Arguments

`object`	(empty) `xcmsFragments-class` object
`xs`	A `xcmsSet-class` object which contains picked ms1-peaks from several experiments
`compMethod`	("floor", "round", "none"): compare-method which is used to find the parent peak of a MSnpeak through comparing the MZ-values of the MS1peaks with the MSnParentPeaks.
`snthresh, mzgap, uniq`	these are the parameters for the getspec-peakpicker included in xcmsRaw.

Details

After running collect(xFragments,xSet) The peak table of the xcmsFragments includes the ms1Peaks from all experiments stored in a xcmsSet-object. Further it contains the relevant msN-peaks from the xcmsRaw-objects, which were created temporarily with the paths in xcmsSet.

Value

A matrix with columns:

`peakID`	unique identifier of every peak
`MSnParentPeakID`	PeakID of the parent peak of a msLevel>1 - peak, it is 0 if the peak is msLevel 1.
`msLevel`	The msLevel of the peak.
`rt`	retention time of the peak midpoint
`mz`	the mz-Value of the peak
`intensity`	the intensity of the peak
`sample`	the number of the sample from the xcmsSet
`GroupPeakMSn`	Used for grouped xcmsSet groups
`CollisionEnergy`	The collision energy of the fragment

Methods

object = "xcmsFragments" `collect(object, ...)`

`correlate,Chromatogram,Chromatogram-method`
Correlate chromatograms

Description

For `xcms` **>= 3.15.3 please use** `compareChromatograms()` **instead of** `correlate`

Correlate intensities of two chromatograms with each other. If the two `Chromatogram` objects have different retention times they are first *aligned* to match data points in the first to data points in the second chromatogram. See help on `alignRt` in `MSnbase::Chromatogram()` for more details.

If `correlate` is called on a single `MChromatograms()` object a pairwise correlation of each chromatogram with each other is performed and a `matrix` with the correlation coefficients is returned.

Note that the correlation of two chromatograms depends also on their order, e.g. `correlate(chr1, chr2)` might not be identical to `correlate(chr2, chr1)`. The lower and upper triangular part of the correlation matrix might thus be different.

Usage

```
## S4 method for signature 'Chromatogram,Chromatogram'
correlate(
  x,
  y,
  use = "pairwise.complete.obs",
  method = c("pearson", "kendall", "spearman"),
  align = c("closest", "approx"),
  ...
)

## S4 method for signature 'MChromatograms,missing'
correlate(
  x,
  y = NULL,
  use = "pairwise.complete.obs",
  method = c("pearson", "kendall", "spearman"),
  align = c("closest", "approx"),
  ...
)

## S4 method for signature 'MChromatograms,MChromatograms'
correlate(
  x,
  y = NULL,
  use = "pairwise.complete.obs",
  method = c("pearson", "kendall", "spearman"),
  align = c("closest", "approx"),
  ...
```

```
)
```

Arguments

`x`	`Chromatogram()` or `MChromatograms()` object.
`y`	`Chromatogram()` or `MChromatograms()` object.
`use`	`character(1)` passed to the `cor` function. See `cor()` for details.
`method`	`character(1)` passed to the `cor` function. See `cor()` for details.
`align`	`character(1)` defining the alignment method to be used. See help on `alignRt` in `MSnbase::Chromatogram()` for details. The value of this parameter is passed to the `method` parameter of `alignRt`.
`...`	optional parameters passed along to the `alignRt` method such as `tolerance` that, if set to `0` requires the retention times to be identical.

Value

`numeric(1)` or `matrix` (if called on `MChromatograms` objects) with the correlation coefficient. If a `matrix` is returned, the rows represent the chromatograms in `x` and the columns the chromatograms in `y`.

Author(s)

Michael Witting, Johannes Rainer

Examples

```
chr1 <- Chromatogram(rtime = 1:10 + rnorm(n = 10, sd = 0.3),
    intensity = c(5, 29, 50, NA, 100, 12, 3, 4, 1, 3))
chr2 <- Chromatogram(rtime = 1:10 + rnorm(n = 10, sd = 0.3),
    intensity = c(80, 50, 20, 10, 9, 4, 3, 4, 1, 3))
chr3 <- Chromatogram(rtime = 3:9 + rnorm(7, sd = 0.3),
    intensity = c(53, 80, 130, 15, 5, 3, 2))

chrs <- MChromatograms(list(chr1, chr2, chr3))

## Using `compareChromatograms` instead of `correlate`.
compareChromatograms(chr1, chr2)
compareChromatograms(chr2, chr1)

compareChromatograms(chrs, chrs)
```

`diffreport-methods` *Create report of analyte differences*

Description

Create a report showing the most significant differences between two sets of samples. Optionally create extracted ion chromatograms for the most significant differences.

Arguments

`object`	the `xcmsSet` object
`class1`	character vector with the first set of sample classes to be compared
`class2`	character vector with the second set of sample classes to be compared
`filebase`	base file name to save report, `.tsv` file and `_eic` will be appended to this name for the tabular report and EIC directory, respectively. if blank nothing will be saved
`eicmax`	number of the most significantly different analytes to create EICs for
`eicwidth`	width (in seconds) of EICs produced
`sortpval`	logical indicating whether the reports should be sorted by p-value
`classeic`	character vector with the sample classes to include in the EICs
`value`	intensity values to be used for the diffreport. If `value="into"`, integrated peak intensities are used. If `value="maxo"`, maximum peak intensities are used. If `value="intb"`, baseline corrected integrated peak intensities are used (only available if peak detection was done by `findPeaks.centWave`).
`metlin`	mass uncertainty to use for generating link to Metlin metabolite database. the sign of the uncertainty indicates negative or positive mode data for M+H or M-H calculation. a value of FALSE or 0 removes the column
`h`	Numeric variable for the height of the eic and boxplots that are printed out.
`w`	Numeric variable for the width of the eic and boxplots print out made.
`mzdec`	Number of decimal places of title m/z values in the eic plot.
`missing`	`numeric(1)` defining an optional value for missing values. `missing = 0` would e.g. replace all `NA` values in the feature matrix with `0`. Note that also a call to `fillPeaks` results in a feature matrix in which `NA` values are replaced by `0`.
`...`	optional arguments to be passed to `mt.teststat` from the `multtest` package.

Details

This method handles creation of summary reports with statistics about which analytes were most significantly different between two sets of samples. It computes Welch's two-sample t-statistic for each analyte and ranks them by p-value. It returns a summary report that can optionally be written out to a tab-separated file.

Additionally, it does all the heavy lifting involved in creating superimposed extracted ion chromatograms for a given number of analytes. It does so by reading the raw data files associated with the samples of interest one at a time. As it does so, it prints the name of the sample it is currently reading. Depending on the number and size of the samples, this process can take a long time.

If a base file name is provided, the report (see Value section) will be saved to a tab separated file. If EICs are generated, they will be saved as 640x480 PNG files in a newly created subdirectory. However this parameter can be changed with the commands arguments. The numbered file names correspond to the rows in the report.

Chromatographic traces in the EICs are colored and labeled by their sample class. Sample classes take their color from the current palette. The color a sample class is assigned is dependent its order in the `xcmsSet` object, not the order given in the class arguments. Thus `levels(sampclass(object))[1]` would use color `palette()[1]` and so on. In that way, sample classes maintain the same color across any number of different generated reports.

When there are multiple sample classes, xcms will produce boxplots of the different classes and will generate a single anova p-value statistic. Like the eic's the plot number corresponds to the row number in the report.

Value

A data frame with the following columns:

`fold`	mean fold change (always greater than 1, see `tstat` for which set of sample classes was higher)
`tstat`	Welch's two sample t-statistic, positive for analytes having greater intensity in `class2`, negative for analytes having greater intensity in `class1`
`pvalue`	p-value of t-statistic
`anova`	p-value of the anova statistic if there are multiple classes
`mzmed`	median m/z of peaks in the group
`mzmin`	minimum m/z of peaks in the group
`mzmax`	maximum m/z of peaks in the group
`rtmed`	median retention time of peaks in the group
`rtmin`	minimum retention time of peaks in the group
`rtmax`	maximum retention time of peaks in the group
`npeaks`	number of peaks assigned to the group
`Sample Classes`	number samples from each sample class represented in the group
`metlin`	A URL to metlin for that mass
`...`	one column for every sample class
`Sample Names`	integrated intensity value for every sample
`...`	one column for every sample

Methods

object = "xcmsSet" `diffreport(object, class1 = levels(sampclass(object))[1], class2 = levels(sampclass(object))[2], filebase = character(), eicmax = 0, eicwidth = 200, sortpval = TRUE, classeic = c(class1,class2), value=c("into","maxo","intb"), metlin = FALSE, h=480,w=640, mzdec=2, missing = numeric(), ...)`

See Also

`xcmsSet-class`, `palette`

`dirname` *Change the file path of an* `OnDiskMSnExp` *object*

Description

`dirname` allows to get and set the path to the directory containing the source files of the OnDiskMSnExp (or XCMSnExp) object.

Usage

```
## S4 method for signature 'OnDiskMSnExp'
dirname(path)

## S4 replacement method for signature 'OnDiskMSnExp'
dirname(path) <- value
```

Arguments

`path`	OnDiskMSnExp.
`value`	`character` of length 1 or length equal to the number of files defining the new path to the files.

Author(s)

Johannes Rainer

`do_adjustRtime_peakGroups` *Align spectrum retention times across samples using peak groups found in most samples*

Description

The function performs retention time correction by assessing the retention time deviation across all samples using peak groups (features) containg chromatographic peaks present in most/all samples. The retention time deviation for these features in each sample is described by fitting either a polynomial (`smooth = "loess"`) or a linear (`smooth = "linear"`) model to the data points. The models are subsequently used to adjust the retention time for each spectrum in each sample.

Usage

```
do_adjustRtime_peakGroups(
  peaks,
  peakIndex,
  rtime,
  minFraction = 0.9,
  extraPeaks = 1,
  smooth = c("loess", "linear"),
  span = 0.2,
  family = c("gaussian", "symmetric"),
  peakGroupsMatrix = matrix(ncol = 0, nrow = 0),
  subset = integer(),
  subsetAdjust = c("average", "previous")
)
```

Arguments

`peaks`	a `matrix` or `data.frame` with the identified chromatographic peaks in the samples.
`peakIndex`	a `list` of indices that provides the grouping information of the chromatographic peaks (across and within samples).
`rtime`	a `list` of `numeric` vectors with the retention times per file/sample.
`minFraction`	`numeric(1)` between 0 and 1 defining the minimum required fraction of samples in which peaks for the peak group were identified. Peak groups passing this criteria will aligned across samples and retention times of individual spectra will be adjusted based on this alignment. For `minFraction = 1` the peak group has to contain peaks in all samples of the experiment. Note that if `subset` is provided, the specified fraction is relative to the defined subset of samples and not to the total number of samples within the experiment (i.e. a peak has to be present in the specified proportion of subset samples).
`extraPeaks`	`numeric(1)` defining the maximal number of additional peaks for all samples to be assigned to a peak group (i.e. feature) for retention time correction. For a data set with 6 samples, `extraPeaks = 1` uses all peak groups with a total peak count `<= 6 + 1`. The total peak count is the total number of peaks being assigned to a peak group and considers also multiple peaks within a sample being assigned to the group.
`smooth`	character defining the function to be used, to interpolate corrected retention times for all peak groups. Either `"loess"` or `"linear"`.
`span`	`numeric(1)` defining the degree of smoothing (if `smooth = "loess"`). This parameter is passed to the internal call to `loess`.
`family`	character defining the method to be used for loess smoothing. Allowed values are `"gaussian"` and `"symmetric"`.See `loess` for more information.
`peakGroupsMatrix`	optional `matrix` of (raw) retention times for peak groups on which the alignment should be performed. Each column represents a sample, each row a feature/peak group. If not provided, this matrix will be determined depending on parameters

	`minFraction` and `extraPeaks`. If provided, `minFraction` and `extraPeaks` will be ignored.
`subset`	`integer` with the indices of samples within the experiment on which the alignment models should be estimated. Samples not part of the subset are adjusted based on the closest subset sample. See description above for more details.
`subsetAdjust`	`character` specifying the method with which non-subset samples should be adjusted. Supported options are `"previous"` and `"average"` (default). See description above for more information.

Details

The alignment bases on the presence of compounds that can be found in all/most samples of an experiment. The retention times of individual spectra are then adjusted based on the alignment of the features corresponding to these *house keeping compounds*. The paraneters `minFraction` and `extraPeaks` can be used to fine tune which features should be used for the alignment (i.e. which features most likely correspond to the above mentioned house keeping compounds).

Parameter `subset` allows to define a subset of samples within the experiment that should be aligned. All samples not being part of the subset will be aligned based on the adjustment of the closest sample within the subset. This allows to e.g. exclude blank samples from the alignment process with their retention times being still adjusted based on the alignment results of the *real* samples.

Value

A `list` with `numeric` vectors with the adjusted retention times grouped by sample.

Note

The method ensures that returned adjusted retention times are increasingly ordered, just as the raw retention times.

Author(s)

Colin Smith, Johannes Rainer

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" *Anal. Chem.* 2006, 78:779-787.

`do_findChromPeaks_centWave`

Core API function for centWave peak detection

Description

This function performs peak density and wavelet based chromatographic peak detection for high resolution LC/MS data in centroid mode [Tautenhahn 2008].

Usage

```
do_findChromPeaks_centWave(
  mz,
  int,
  scantime,
  valsPerSpect,
  ppm = 25,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
  integrate = 1,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  roiList = list(),
  firstBaselineCheck = TRUE,
  roiScales = NULL,
  sleep = 0,
  extendLengthMSW = FALSE
)
```

Arguments

`mz`	Numeric vector with the individual m/z values from all scans/ spectra of one file/sample.
`int`	Numeric vector with the individual intensity values from all scans/spectra of one file/sample.
`scantime`	Numeric vector of length equal to the number of spectra/scans of the data representing the retention time of each scan.
`valsPerSpect`	Numeric vector with the number of values for each spectrum.
`ppm`	`numeric(1)` defining the maximal tolerated m/z deviation in consecutive scans in parts per million (ppm) for the initial ROI definition.
`peakwidth`	`numeric(2)` with the expected approximate peak width in chromatographic space. Given as a range (min, max) in seconds.

`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`prefilter`	`numeric(2)`: `c(k, I)` specifying the prefilter step for the first analysis step (ROI detection). Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Name of the function to calculate the m/z center of the chromatographic peak. Allowed are: `"wMean"`: intensity weighted mean of the peak's m/z values, `"mean"`: mean of the peak's m/z values, `"apex"`: use the m/z value at the peak apex, `"wMeanApex3"`: intensity weighted mean of the m/z value at the peak apex and the m/z values left and right of it and `"meanApex3"`: mean of the m/z value of the peak apex and the m/z values left and right of it.
`integrate`	Integration method. For `integrate = 1` peak limits are found through descent on the mexican hat filtered data, for `integrate = 2` the descent is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak. This affects mostly the retention time position of the peak.
`noise`	`numeric(1)` allowing to set a minimum intensity required for centroids to be considered in the first analysis step (centroids with intensity `< noise` are omitted from ROI detection).
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`roiList`	An optional list of regions-of-interest (ROI) representing detected mass traces. If ROIs are submitted the first analysis step is omitted and chromatographic peak detection is performed on the submitted ROIs. Each ROI is expected to have the following elements specified: `scmin` (start scan index), `scmax` (end scan index), `mzmin` (minimum m/z), `mzmax` (maximum m/z), `length` (number of scans), `intensity` (summed intensity). Each ROI should be represented by a `list` of elements or a single row `data.frame`.
`firstBaselineCheck`	`logical(1)`. If `TRUE` continuous data within regions of interest is checked to be above the first baseline.
`roiScales`	Optional numeric vector with length equal to `roiList` defining the scale for each region of interest in `roiList` that should be used for the centWave-wavelets.
`sleep`	`numeric(1)` defining the number of seconds to wait between iterations. Defaults to `sleep = 0`. If `> 0` a plot is generated visualizing the identified chromatographic peak. Note: this argument is for backward compatibility only and will be removed in future.
`extendLengthMSW`	Option to force centWave to use all scales when running centWave rather than truncating with the EIC length. Uses the "open" method to extend the EIC to a integer base-2 length prior to being passed to `convolve` rather than the default "reflect" method. See https://github.com/sneumann/xcms/issues/445 for more information.

Details

This algorithm is most suitable for high resolution LC/{TOF,OrbiTrap,FTICR}-MS data in centroid mode. In the first phase the method identifies *regions of interest* (ROIs) representing mass traces that are characterized as regions with less than `ppm` m/z deviation in consecutive scans in the LC/MS map. In detail, starting with a single m/z, a ROI is extended if a m/z can be found in the next scan (spectrum) for which the difference to the mean m/z of the ROI is smaller than the user defined `ppm` of the m/z. The mean m/z of the ROI is then updated considering also the newly included m/z value.

These ROIs are then, after some cleanup, analyzed using continuous wavelet transform (CWT) to locate chromatographic peaks on different scales. The first analysis step is skipped, if regions of interest are passed with the `roiList` parameter.

Value

A matrix, each row representing an identified chromatographic peak, with columns:

mz Intensity weighted mean of m/z values of the peak across scans.

mzmin Minimum m/z of the peak.

mzmax Maximum m/z of the peak.

rt Retention time of the peak's midpoint.

rtmin Minimum retention time of the peak.

rtmax Maximum retention time of the peak.

into Integrated (original) intensity of the peak.

intb Per-peak baseline corrected integrated peak intensity.

maxo Maximum intensity of the peak.

sn Signal to noise ratio, defined as `(maxo - baseline)/sd`, `sd` being the standard deviation of local chromatographic noise.

egauss RMSE of Gaussian fit.

Additional columns for `verboseColumns = TRUE`:

mu Gaussian parameter mu.

sigma Gaussian parameter sigma.

h Gaussian parameter h.

f Region number of the m/z ROI where the peak was localized.

dppm m/z deviation of mass trace across scans in ppm.

scale Scale on which the peak was localized.

scpos Peak position found by wavelet analysis (scan number).

scmin Left peak limit found by wavelet analysis (scan number).

scmax Right peak limit found by wavelet analysis (scan numer).

Note

The *centWave* was designed to work on centroided mode, thus it is expected that such data is presented to the function.

This function exposes core chromatographic peak detection functionality of the *centWave* method. While this function can be called directly, users will generally call the corresponding method for the data object instead.

Author(s)

Ralf Tautenhahn, Johannes Rainer

References

Ralf Tautenhahn, Christoph B\"ottcher, and Steffen Neumann "Highly sensitive feature detection for high resolution LC/MS" *BMC Bioinformatics* 2008, 9:504

See Also

`centWave` for the standard user interface method.

Other core peak detection functions: `do_findChromPeaks_centWaveWithPredIsoROIs`(), `do_findChromPeaks_massifqu`
`do_findChromPeaks_matchedFilter`(), `do_findPeaks_MSW`()

Examples

```
## Load the test file
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Subset to one file and restrict to a certain retention time range
data <- filterRt(filterFile(faahko_sub, 1), c(2500, 3000))

## Get m/z and intensity values
mzs <- mz(data)
ints <- intensity(data)

## Define the values per spectrum:
valsPerSpect <- lengths(mzs)

## Calling the function. We're using a large value for noise and prefilter
## to speed up the call in the example - in a real use case we would either
## set the value to a reasonable value or use the default value.
res <- do_findChromPeaks_centWave(mz = unlist(mzs), int = unlist(ints),
    scantime = rtime(data), valsPerSpect = valsPerSpect, noise = 10000,
    prefilter = c(3, 10000))
head(res)
```

`do_findChromPeaks_centWaveWithPredIsoROIs` *Core API function for two-step centWave peak detection with isotopes*

Description

The `do_findChromPeaks_centWaveWithPredIsoROIs` performs a two-step centWave based peak detection: chromatographic peaks are identified using centWave followed by a prediction of the location of the identified peaks' isotopes in the mz-retention time space. These locations are fed as *regions of interest* (ROIs) to a subsequent centWave run. All non overlapping peaks from these two peak detection runs are reported as the final list of identified peaks.

The `do_findChromPeaks_centWaveAddPredIsoROIs` performs centWave based peak detection based in regions of interest (ROIs) representing predicted isotopes for the peaks submitted with argument `peaks.`. The function returns a matrix with the identified peaks consisting of all input peaks and peaks representing predicted isotopes of these (if found by the centWave algorithm).

Usage

```
do_findChromPeaks_centWaveWithPredIsoROIs(
  mz,
  int,
  scantime,
  valsPerSpect,
  ppm = 25,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
  integrate = 1,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  roiList = list(),
  firstBaselineCheck = TRUE,
  roiScales = NULL,
  snthreshIsoROIs = 6.25,
  maxCharge = 3,
  maxIso = 5,
  mzIntervalExtension = TRUE,
  polarity = "unknown",
  extendLengthMSW = FALSE
)

do_findChromPeaks_addPredIsoROIs(
  mz,
```

```
  int,
  scantime,
  valsPerSpect,
  ppm = 25,
  peakwidth = c(20, 50),
  snthresh = 6.25,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
  integrate = 1,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  peaks. = NULL,
  maxCharge = 3,
  maxIso = 5,
  mzIntervalExtension = TRUE,
  polarity = "unknown"
)
```

Arguments

`mz`	Numeric vector with the individual m/z values from all scans/ spectra of one file/sample.
`int`	Numeric vector with the individual intensity values from all scans/spectra of one file/sample.
`scantime`	Numeric vector of length equal to the number of spectra/scans of the data representing the retention time of each scan.
`valsPerSpect`	Numeric vector with the number of values for each spectrum.
`ppm`	`numeric(1)` defining the maximal tolerated m/z deviation in consecutive scans in parts per million (ppm) for the initial ROI definition.
`peakwidth`	`numeric(2)` with the expected approximate peak width in chromatographic space. Given as a range (min, max) in seconds.
`snthresh`	For `do_findChromPeaks_addPredIsoROIs`: numeric(1) defining the signal to noise threshold for the centWave algorithm. For `do_findChromPeaks_centWaveWithPredIsoROIs`: numeric(1) defining the signal to noise threshold for the initial (first) centWave run.
`prefilter`	`numeric(2)`: `c(k, I)` specifying the prefilter step for the first analysis step (ROI detection). Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Name of the function to calculate the m/z center of the chromatographic peak. Allowed are: `"wMean"`: intensity weighted mean of the peak's m/z values, `"mean"`: mean of the peak's m/z values, `"apex"`: use the m/z value at the peak apex, `"wMeanApex3"`: intensity weighted mean of the m/z value at the peak apex and the m/z values left and right of it and `"meanApex3"`: mean of the m/z value of the peak apex and the m/z values left and right of it.

`integrate`	Integration method. For `integrate = 1` peak limits are found through descent on the mexican hat filtered data, for `integrate = 2` the descent is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak. This affects mostly the retention time position of the peak.
`noise`	`numeric(1)` allowing to set a minimum intensity required for centroids to be considered in the first analysis step (centroids with intensity `< noise` are omitted from ROI detection).
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`roiList`	An optional list of regions-of-interest (ROI) representing detected mass traces. If ROIs are submitted the first analysis step is omitted and chromatographic peak detection is performed on the submitted ROIs. Each ROI is expected to have the following elements specified: `scmin` (start scan index), `scmax` (end scan index), `mzmin` (minimum m/z), `mzmax` (maximum m/z), `length` (number of scans), `intensity` (summed intensity). Each ROI should be represented by a `list` of elements or a single row `data.frame`.
`firstBaselineCheck`	`logical(1)`. If `TRUE` continuous data within regions of interest is checked to be above the first baseline.
`roiScales`	Optional numeric vector with length equal to `roiList` defining the scale for each region of interest in `roiList` that should be used for the centWave-wavelets.
`snthreshIsoROIs`	`numeric(1)` defining the signal to noise ratio cutoff to be used in the second centWave run to identify peaks for predicted isotope ROIs.
`maxCharge`	`integer(1)` defining the maximal isotope charge. Isotopes will be defined for charges `1:maxCharge`.
`maxIso`	`integer(1)` defining the number of isotope peaks that should be predicted for each peak identified in the first centWave run.
`mzIntervalExtension`	`logical(1)` whether the mz range for the predicted isotope ROIs should be extended to increase detection of low intensity peaks.
`polarity`	`character(1)` specifying the polarity of the data. Currently not used, but has to be `"positive"`, `"negative"` or `"unknown"` if provided.
`extendLengthMSW`	Option to force centWave to use all scales when running centWave rather than truncating with the EIC length. Uses the "open" method to extend the EIC to a integer base-2 length prior to being passed to `convolve` rather than the default "reflect" method. See https://github.com/sneumann/xcms/issues/445 for more information.

`peaks.`	A matrix or `xcmsPeaks` object such as one returned by a call to `link{do_findChromPeaks_centWave}` or `link{findPeaks.centWave}` (both with `verboseColumns = TRUE`) with the peaks for which isotopes should be predicted and used for an additional peak detectoin using the centWave method. Required columns are: `"mz"`, `"mzmin"`, `"mzmax"`, `"scmin"`, `"scmax"`, `"scale"` and `"into"`.

Details

For more details on the centWave algorithm see `centWave`.

Value

A matrix, each row representing an identified chromatographic peak. All non-overlapping peaks identified in both centWave runs are reported. The matrix columns are:

mz Intensity weighted mean of m/z values of the peaks across scans.

mzmin Minimum m/z of the peaks.

mzmax Maximum m/z of the peaks.

rt Retention time of the peak's midpoint.

rtmin Minimum retention time of the peak.

rtmax Maximum retention time of the peak.

into Integrated (original) intensity of the peak.

intb Per-peak baseline corrected integrated peak intensity.

maxo Maximum intensity of the peak.

sn Signal to noise ratio, defined as `(maxo - baseline)/sd`, `sd` being the standard deviation of local chromatographic noise.

egauss RMSE of Gaussian fit.

Additional columns for `verboseColumns = TRUE`:

mu Gaussian parameter mu.

sigma Gaussian parameter sigma.

h Gaussian parameter h.

f Region number of the m/z ROI where the peak was localized.

dppm m/z deviation of mass trace across scans in ppm.

scale Scale on which the peak was localized.

scpos Peak position found by wavelet analysis (scan number).

scmin Left peak limit found by wavelet analysis (scan number).

scmax Right peak limit found by wavelet analysis (scan numer).

Author(s)

Hendrik Treutler, Johannes Rainer

See Also

Other core peak detection functions: `do_findChromPeaks_centWave()`, `do_findChromPeaks_massifquant()`, `do_findChromPeaks_matchedFilter()`, `do_findPeaks_MSW()`

`do_findChromPeaks_massifquant`

Core API function for massifquant peak detection

Description

Massifquant is a Kalman filter (KF)-based chromatographic peak detection for XC-MS data in centroid mode. The identified peaks can be further refined with the *centWave* method (see `do_findChromPeaks_centWave` for details on centWave) by specifying `withWave = TRUE`.

Usage

```
do_findChromPeaks_massifquant(
  mz,
  int,
  scantime,
  valsPerSpect,
  ppm = 10,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
  integrate = 1,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  criticalValue = 1.125,
  consecMissedLimit = 2,
  unions = 1,
  checkBack = 0,
  withWave = FALSE
)
```

Arguments

`mz`	Numeric vector with the individual m/z values from all scans/ spectra of one file/sample.
`int`	Numeric vector with the individual intensity values from all scans/spectra of one file/sample.
`scantime`	Numeric vector of length equal to the number of spectra/scans of the data representing the retention time of each scan.

`valsPerSpect`	Numeric vector with the number of values for each spectrum.
`ppm`	`numeric(1)` defining the maximal tolerated m/z deviation in consecutive scans in parts per million (ppm) for the initial ROI definition.
`peakwidth`	`numeric(2)` with the expected approximate peak width in chromatographic space. Given as a range (min, max) in seconds.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`prefilter`	`numeric(2)`: `c(k, I)` specifying the prefilter step for the first analysis step (ROI detection). Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Name of the function to calculate the m/z center of the chromatographic peak. Allowed are: `"wMean"`: intensity weighted mean of the peak's m/z values, `"mean"`: mean of the peak's m/z values, `"apex"`: use the m/z value at the peak apex, `"wMeanApex3"`: intensity weighted mean of the m/z value at the peak apex and the m/z values left and right of it and `"meanApex3"`: mean of the m/z value of the peak apex and the m/z values left and right of it.
`integrate`	Integration method. For `integrate = 1` peak limits are found through descent on the mexican hat filtered data, for `integrate = 2` the descent is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak. This affects mostly the retention time position of the peak.
`noise`	`numeric(1)` allowing to set a minimum intensity required for centroids to be considered in the first analysis step (centroids with intensity < `noise` are omitted from ROI detection).
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`criticalValue`	`numeric(1)`. Suggested values: (`0.1-3.0`). This setting helps determine the the Kalman Filter prediciton margin of error. A real centroid belonging to a bonafide peak must fall within the KF prediction margin of error. Much like in the construction of a confidence interval, `criticalVal` loosely translates to be a multiplier of the standard error of the prediction reported by the Kalman Filter. If the peak in the XC-MS sample have a small mass deviance in ppm error, a smaller critical value might be better and vice versa.
`consecMissedLimit`	`integer(1)` Suggested values: (`1,2,3`). While a peak is in the proces of being detected by a Kalman Filter, the Kalman Filter may not find a predicted centroid in every scan. After 1 or more consecutive failed predictions, this setting informs Massifquant when to stop a Kalman Filter from following a candidate peak.
`unions`	`integer(1)` set to `1` if apply t-test union on segmentation; set to `0` if no t-test to be applied on chromatographically continous peaks sharing same m/z range. Explanation: With very few data points, sometimes a Kalman Filter stops tracking a peak prematurely. Another Kalman Filter is instantiated and begins following

	the rest of the signal. Because tracking is done backwards to forwards, this algorithmic defect leaves a real peak divided into two segments or more. With this option turned on, the program identifies segmented peaks and combines them (merges them) into one with a two sample t-test. The potential danger of this option is that some truly distinct peaks may be merged.
`checkBack`	`integer(1)` set to `1` if turned on; set to `0` if turned off. The convergence of a Kalman Filter to a peak's precise m/z mapping is very fast, but sometimes it incorporates erroneous centroids as part of a peak (especially early on). The `scanBack` option is an attempt to remove the occasional outlier that lies beyond the converged bounds of the Kalman Filter. The option does not directly affect identification of a peak because it is a postprocessing measure; it has not shown to be a extremely useful thus far and the default is set to being turned off.
`withWave`	`logical(1)` if `TRUE`, the peaks identified first with Massifquant are subsequently filtered with the second step of the centWave algorithm, which includes wavelet estimation.

Details

This algorithm's performance has been tested rigorously on high resolution LC/OrbiTrap, TOF-MS data in centroid mode. Simultaneous kalman filters identify peaks and calculate their area under the curve. The default parameters are set to operate on a complex LC-MS Orbitrap sample. Users will find it useful to do some simple exploratory data analysis to find out where to set a minimum intensity, and identify how many scans an average peak spans. The `consecMissedLimit` parameter has yielded good performance on Orbitrap data when set to (`2`) and on TOF data it was found best to be at (`1`). This may change as the algorithm has yet to be tested on many samples. The `criticalValue` parameter is perhaps most dificult to dial in appropriately and visual inspection of peak identification is the best suggested tool for quick optimization. The `ppm` and `checkBack` parameters have shown less influence than the other parameters and exist to give users flexibility and better accuracy.

Value

A matrix, each row representing an identified chromatographic peak, with columns:

mz Intensity weighted mean of m/z values of the peaks across scans.

mzmin Minumum m/z of the peak.

mzmax Maximum m/z of the peak.

rtmin Minimum retention time of the peak.

rtmax Maximum retention time of the peak.

rt Retention time of the peak's midpoint.

into Integrated (original) intensity of the peak.

maxo Maximum intensity of the peak.

If `withWave` is set to `TRUE`, the result is the same as returned by the `do_findChromPeaks_centWave` method.

Author(s)

Christopher Conley

References

Conley CJ, Smith R, Torgrip RJ, Taylor RM, Tautenhahn R and Prince JT "Massifquant: open-source Kalman filter-based XC-MS isotope trace feature detection" *Bioinformatics* 2014, 30(18):2636-43.

See Also

`massifquant` for the standard user interface method.

Other core peak detection functions: `do_findChromPeaks_centWaveWithPredIsoROIs()`, `do_findChromPeaks_centWave`
`do_findChromPeaks_matchedFilter()`, `do_findPeaks_MSW()`

Examples

```
## Load the test file
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Subset to one file and restrict to a certain retention time range
data <- filterRt(filterFile(faahko_sub, 1), c(2500, 3000))

## Get m/z and intensity values
mzs <- mz(data)
ints <- intensity(data)

## Define the values per spectrum:
valsPerSpect <- lengths(mzs)

## Perform the peak detection using massifquant - setting prefilter to
## a high value to speed up the call for the example
res <- do_findChromPeaks_massifquant(mz = unlist(mzs), int = unlist(ints),
    scantime = rtime(data), valsPerSpect = valsPerSpect,
    prefilter = c(3, 10000))
head(res)
```

---

do_findChromPeaks_matchedFilter

Core API function for matchedFilter peak detection

Description

This function identifies peaks in the chromatographic time domain as described in [Smith 2006]. The intensity values are binned by cutting The LC/MS data into slices (bins) of a mass unit (`binSize` m/z) wide. Within each bin the maximal intensity is selected. The peak detection is then performed in each bin by extending it based on the `steps` parameter to generate slices comprising bins `current_bin - steps +1` to `current_bin + steps - 1`. Each of these slices is then filtered with matched filtration using a second-derative Gaussian as the model peak shape. After filtration peaks are detected using a signal-to-ration cut-off. For more details and illustrations see [Smith 2006].

Usage

```
do_findChromPeaks_matchedFilter(
  mz,
  int,
  scantime,
  valsPerSpect,
  binSize = 0.1,
  impute = "none",
  baseValue,
  distance,
  fwhm = 30,
  sigma = fwhm/2.3548,
  max = 5,
  snthresh = 10,
  steps = 2,
  mzdiff = 0.8 - binSize * steps,
  index = FALSE,
  sleep = 0
)
```

Arguments

`mz`	Numeric vector with the individual m/z values from all scans/ spectra of one file/sample.
`int`	Numeric vector with the individual intensity values from all scans/spectra of one file/sample.
`scantime`	Numeric vector of length equal to the number of spectra/scans of the data representing the retention time of each scan.
`valsPerSpect`	Numeric vector with the number of values for each spectrum.
`binSize`	`numeric(1)` specifying the width of the bins/slices in m/z dimension.
`impute`	Character string specifying the method to be used for missing value imputation. Allowed values are `"none"` (no linear interpolation), `"lin"` (linear interpolation), `"linbase"` (linear interpolation within a certain bin-neighborhood) and `"intlin"`. See `imputeLinInterpol` for more details.
`baseValue`	The base value to which empty elements should be set. This is only considered for `method = "linbase"` and corresponds to the `profBinLinBase`'s `baselevel` argument.
`distance`	For `method = "linbase"`: number of non-empty neighboring element of an empty element that should be considered for linear interpolation. See details section for more information.

`fwhm`	`numeric(1)` specifying the full width at half maximum of matched filtration gaussian model peak. Only used to calculate the actual sigma, see below.
`sigma`	`numeric(1)` specifying the standard deviation (width) of the matched filtration model peak.
`max`	`numeric(1)` representing the maximum number of peaks that are expected/will be identified per slice.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`steps`	`numeric(1)` defining the number of bins to be merged before filtration (i.e. the number of neighboring bins that will be joined to the slice in which filtration and peak detection will be performed).
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`index`	`logical(1)` specifying whether indicies should be returned instead of values for m/z and retention times.
`sleep`	`numeric(1)` defining the number of seconds to wait between iterations. Defaults to `sleep = 0`. If `> 0` a plot is generated visualizing the identified chromatographic peak. Note: this argument is for backward compatibility only and will be removed in future.

Details

The intensities are binned by the provided m/z values within each spectrum (scan). Binning is performed such that the bins are centered around the m/z values (i.e. the first bin includes all m/z values between `min(mz) - bin_size/2` and `min(mz) + bin_size/2`).

For more details on binning and missing value imputation see `binYonX` and `imputeLinInterpol` methods.

Value

A matrix, each row representing an identified chromatographic peak, with columns:

mz Intensity weighted mean of m/z values of the peak across scans.

mzmin Minimum m/z of the peak.

mzmax Maximum m/z of the peak.

rt Retention time of the peak's midpoint.

rtmin Minimum retention time of the peak.

rtmax Maximum retention time of the peak.

into Integrated (original) intensity of the peak.

intf Integrated intensity of the filtered peak.

maxo Maximum intensity of the peak.

maxf Maximum intensity of the filtered peak.

i Rank of peak in merged EIC (<= `max`).

sn Signal to noise ratio of the peak

Note

This function exposes core peak detection functionality of the *matchedFilter* method. While this function can be called directly, users will generally call the corresponding method for the data object instead (e.g. the `link{findPeaks.matchedFilter}` method).

Author(s)

Colin A Smith, Johannes Rainer

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" *Anal. Chem.* 2006, 78:779-787.

See Also

`binYonX` for a binning function, `imputeLinInterpol` for the interpolation of missing values. `matchedFilter` for the standard user interface method.

Other core peak detection functions: `do_findChromPeaks_centWaveWithPredIsoROIs()`, `do_findChromPeaks_centWave` `do_findChromPeaks_massifquant()`, `do_findPeaks_MSW()`

Examples

```
## Load the test file
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Subset to one file and restrict to a certain retention time range
data <- filterRt(filterFile(faahko_sub, 1), c(2500, 3000))

## Get m/z and intensity values
mzs <- mz(data)
ints <- intensity(data)

## Define the values per spectrum:
valsPerSpect <- lengths(mzs)

res <- do_findChromPeaks_matchedFilter(mz = unlist(mzs), int = unlist(ints),
    scantime = rtime(data), valsPerSpect = valsPerSpect)
head(res)
```

`do_findPeaks_MSW` *Core API function for single-spectrum non-chromatography MS data peak detection*

Description

This function performs peak detection in mass spectrometry direct injection spectrum using a wavelet based algorithm.

Usage

```
do_findPeaks_MSW(mz, int, snthresh = 3, verboseColumns = FALSE, ...)
```

Arguments

`mz`	Numeric vector with the individual m/z values from all scans/ spectra of one file/sample.
`int`	Numeric vector with the individual intensity values from all scans/spectra of one file/sample.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`...`	Additional parameters to be passed to the `peakDetectionCWT` function.

Details

This is a wrapper around the peak picker in Bioconductor's `MassSpecWavelet` package calling `peakDetectionCWT` and `tuneInPeakInfo` functions. See the *xcmsDirect* vignette for more information.

Value

A matrix, each row representing an identified peak, with columns:

mz m/z value of the peak at the centroid position.

mzmin Minimum m/z of the peak.

mzmax Maximum m/z of the peak.

rt Always `-1`.

rtmin Always `-1`.

rtmax Always `-1`.

into Integrated (original) intensity of the peak.

maxo Maximum intensity of the peak.

intf Always `NA`.

maxf Maximum MSW-filter response of the peak.

sn Signal to noise ratio.

Author(s)

Joachim Kutzera, Steffen Neumann, Johannes Rainer

See Also

`MSW` for the standard user interface method. `peakDetectionCWT` from the `MassSpecWavelet` package.

Other core peak detection functions: `do_findChromPeaks_centWaveWithPredIsoROIs()`, `do_findChromPeaks_centWave` `do_findChromPeaks_massifquant()`, `do_findChromPeaks_matchedFilter()`

---

do_groupChromPeaks_density

Core API function for peak density based chromatographic peak grouping

---

Description

The `do_groupChromPeaks_density` function performs chromatographic peak grouping based on the density (distribution) of peaks, found in different samples, along the retention time axis in slices of overlapping mz ranges.

Usage

```
do_groupChromPeaks_density(
  peaks,
  sampleGroups,
  bw = 30,
  minFraction = 0.5,
  minSamples = 1,
  binSize = 0.25,
  maxFeatures = 50,
  sleep = 0
)
```

Arguments

`peaks`	A `matrix` or `data.frame` with the mz values and retention times of the identified chromatographic peaks in all samples of an experiment. Required columns are `"mz"`, `"rt"` and `"sample"`. The latter should contain `numeric` values representing the index of the sample in which the peak was found.
`sampleGroups`	A vector of the same length than samples defining the sample group assignments (i.e. which samples belong to which sample group). This parameter is mandatory for the `PeakDensityParam` and has to be provided also if there is no sample grouping in the experiment (in which case all samples should be assigned to the same group).

`bw`	`numeric(1)` defining the bandwidth (standard deviation ot the smoothing kernel) to be used. This argument is passed to the [density() method.
`minFraction`	`numeric(1)` defining the minimum fraction of samples in at least one sample group in which the peaks have to be present to be considered as a peak group (feature).
`minSamples`	`numeric(1)` with the minimum number of samples in at least one sample group in which the peaks have to be detected to be considered a peak group (feature).
`binSize`	`numeric(1)` defining the size of the overlapping slices in mz dimension.
`maxFeatures`	`numeric(1)` with the maximum number of peak groups to be identified in a single mz slice.
`sleep`	`numeric(1)` defining the time to *sleep* between iterations and plot the result from the current iteration.

Details

For overlapping slices along the mz dimension, the function calculates the density distribution of identified peaks along the retention time axis and groups peaks from the same or different samples that are close to each other. See (Smith 2006) for more details.

Value

A `data.frame`, each row representing a (mz-rt) feature (i.e. a peak group) with columns:

- `"mzmed"`: median of the peaks' apex mz values.
- `"mzmin"`: smallest mz value of all peaks' apex within the feature.
- `"mzmax"`:largest mz value of all peaks' apex within the feature.
- `"rtmed"`: the median of the peaks' retention times.
- `"rtmin"`: the smallest retention time of the peaks in the group.
- `"rtmax"`: the largest retention time of the peaks in the group.
- `"npeaks"`: the total number of peaks assigned to the feature.
- `"peakidx"`: a `list` with the indices of all peaks in a feature in the `peaks` input matrix.

Note that this number can be larger than the total number of samples, since multiple peaks from the same sample could be assigned to a feature.

Note

The default settings might not be appropriate for all LC/GC-MS setups, especially the `bw` and `binSize` parameter should be adjusted accordingly.

Author(s)

Colin Smith, Johannes Rainer

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" Anal. Chem. 2006, 78:779-787.

See Also

Other core peak grouping algorithms: `do_groupChromPeaks_nearest()`, `do_groupPeaks_mzClust()`

Examples

```
## Load the test file
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Extract the matrix with the identified peaks from the xcmsSet:
pks <- chromPeaks(faahko_sub)

## Perform the peak grouping with default settings:
res <- do_groupChromPeaks_density(pks, sampleGroups = rep(1, 3))

## The feature definitions:
head(res)
```

`do_groupChromPeaks_nearest` *Core API function for chromatic peak grouping using a nearest neighbor approach*

Description

The `do_groupChromPeaks_nearest` function groups peaks across samples by creating a master peak list and assigning corresponding peaks from all samples to each peak group (i.e. feature). The method is inspired by the correspondence algorithm of mzMine (Katajamaa 2006).

Usage

```
do_groupChromPeaks_nearest(
  peaks,
  sampleGroups,
  mzVsRtBalance = 10,
  absMz = 0.2,
  absRt = 15,
  kNN = 10
)
```

Arguments

`peaks`	A `matrix` or `data.frame` with the mz values and retention times of the identified chromatographic peaks in all samples of an experiment. Required columns are `"mz"`, `"rt"` and `"sample"`. The latter should contain `numeric` values representing the index of the sample in which the peak was found.
`sampleGroups`	A vector of the same length than samples defining the sample group assignments (i.e. which samples belong to which sample group). This parameter is mandatory for the `PeakDensityParam` and has to be provided also if there is no sample grouping in the experiment (in which case all samples should be assigned to the same group).
`mzVsRtBalance`	`numeric(1)` representing the factor by which mz values are multiplied before calculating the (euclician) distance between two peaks.
`absMz`	`numeric(1)` maximum tolerated distance for mz values.
`absRt`	`numeric(1)` maximum tolerated distance for rt values.
`kNN`	`numeric(1)` representing the number of nearest neighbors to check.

Value

A `list` with elements `"featureDefinitions"` and `"peakIndex"`. `"featureDefinitions"` is a `matrix`, each row representing an (mz-rt) feature (i.e. peak group) with columns:

- `"mzmed"`: median of the peaks' apex mz values.
- `"mzmin"`: smallest mz value of all peaks' apex within the feature.
- `"mzmax"`:largest mz value of all peaks' apex within the feature.
- `"rtmed"`: the median of the peaks' retention times.
- `"rtmin"`: the smallest retention time of the peaks in the feature.
- `"rtmax"`: the largest retention time of the peaks in the feature.
- `"npeaks"`: the total number of peaks assigned to the feature.

`"peakIndex"` is a `list` with the indices of all peaks in a feature in the `peaks` input matrix.

References

Katajamaa M, Miettinen J, Oresic M: MZmine: Toolbox for processing and visualization of mass spectrometry based molecular profile data. Bioinformatics 2006, 22:634-636.

See Also

Other core peak grouping algorithms: `do_groupChromPeaks_density()`, `do_groupPeaks_mzClust()`

`do_groupPeaks_mzClust` *Core API function for peak grouping using mzClust*

Description

The `do_groupPeaks_mzClust` function performs high resolution correspondence on single spectra samples.

Usage

```
do_groupPeaks_mzClust(
  peaks,
  sampleGroups,
  ppm = 20,
  absMz = 0,
  minFraction = 0.5,
  minSamples = 1
)
```

Arguments

`peaks`	A `matrix` or `data.frame` with the mz values and retention times of the identified chromatographic peaks in all samples of an experiment. Required columns are `"mz"`, `"rt"` and `"sample"`. The latter should contain `numeric` values representing the index of the sample in which the peak was found.
`sampleGroups`	A vector of the same length than samples defining the sample group assignments (i.e. which samples belong to which sample group). This parameter is mandatory for the `PeakDensityParam` and has to be provided also if there is no sample grouping in the experiment (in which case all samples should be assigned to the same group).
`ppm`	`numeric(1)` representing the relative mz error for the clustering/grouping (in parts per million).
`absMz`	`numeric(1)` representing the absolute mz error for the clustering.
`minFraction`	`numeric(1)` defining the minimum fraction of samples in at least one sample group in which the peaks have to be present to be considered as a peak group (feature).
`minSamples`	`numeric(1)` with the minimum number of samples in at least one sample group in which the peaks have to be detected to be considered a peak group (feature).

Value

A `list` with elements `"featureDefinitions"` and `"peakIndex"`. `"featureDefinitions"` is a `matrix`, each row representing an (mz-rt) feature (i.e. peak group) with columns:

- `"mzmed"`: median of the peaks' apex mz values.
- `"mzmin"`: smallest mz value of all peaks' apex within the feature.

- `"mzmax"`: largest mz value of all peaks' apex within the feature.
- `"rtmed"`: always `-1`.
- `"rtmin"`: always `-1`.
- `"rtmax"`: always `-1`.
- `"npeaks"`: the total number of peaks assigned to the feature. Note that this number can be larger than the total number of samples, since multiple peaks from the same sample could be assigned to a group.

`"peakIndex"` is a `list` with the indices of all peaks in a peak group in the `peaks` input matrix.

References

Saira A. Kazmi, Samiran Ghosh, Dong-Guk Shin, Dennis W. Hill and David F. Grant
Alignment of high resolution mass spectra: development of a heuristic approach for metabolomics.
Metabolomics, Vol. 2, No. 2, 75-83 (2006)

See Also

Other core peak grouping algorithms: `do_groupChromPeaks_density()`, `do_groupChromPeaks_nearest()`

`estimatePrecursorIntensity` *Estimate precursor intensity for MS level 2 spectra*

Description

`estimatePrecursorIntensity` determines the precursor intensity for a MS 2 spectrum based on the intensity of the respective signal from the neighboring MS 1 spectra (i.e. based on the peak with the m/z matching the precursor m/z of the MS 2 spectrum). Based on parameter `method` either the intensity of the peak from the previous MS 1 scan is used (`method = "previous"`) or an interpolation between the intensity from the previous and subsequent MS1 scan is used (`method = "interpolation"`, which considers also the retention times of the two MS1 scans and the retention time of the MS2 spectrum).

Usage

```
estimatePrecursorIntensity(
  x,
  ppm = 10,
  method = c("previous", "interpolation"),
  BPPARAM = bpparam()
)
```

Arguments

`x`	`OnDiskMSnExp` or `XCMSnExp` object.
`ppm`	`numeric(1)` defining the maximal acceptable difference (in ppm) of the precursor m/z and the m/z of the corresponding peak in the MS 1 scan.
`method`	`character(1)` defining the method how the precursor intensity should be determined (see description above for details). Defaults to `method = "previous"`.
`BPPARAM`	parallel processing setup. See `bpparam()` for details.

Value

`numeric` with length equal to the number of spectra in `x`. `NA` is returned for MS 1 spectra or if no matching peak in a MS 1 scan can be found for an MS 2 spectrum

Author(s)

Johannes Rainer

`etg` *Empirically Transformed Gaussian function*

Description

A general function for asymmetric chromatographic peaks.

Usage

```
etg(x, H, t1, tt, k1, kt, lambda1, lambdat, alpha, beta)
```

Arguments

`x`	times to evaluate function at
`H`	peak height
`t1`	time of leading edge inflection point
`tt`	time of trailing edge inflection point
`k1`	leading edge parameter
`kt`	trailing edge parameter
`lambda1`	leading edge parameter
`lambdat`	trailing edge parameter
`alpha`	leading edge parameter
`beta`	trailing edge parameter

Value

The function evaluated at times `x`.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

References

Jianwei Li. Development and Evaluation of Flexible Empirical Peak Functions for Processing Chromatographic Peaks. Anal. Chem., 69 (21), 4452-4462, 1997. `http://dx.doi.org/10.1021/ac970481d`

`exportMetaboAnalyst` *Export data for use in MetaboAnalyst*

Description

Export the feature table for further analysis in the MetaboAnalyst software (or the `MetaboAnalystR` R package).

Usage

```
exportMetaboAnalyst(
  x,
  file = NULL,
  label,
  value = "into",
  digits = NULL,
  groupnames = FALSE,
  ...
)
```

Arguments

`x`	XCMSnExp object with identified chromatographic peaks grouped across samples.
`file`	`character(1)` defining the file name. If not specified, the `matrix` with the content is returned.
`label`	either `character(1)` specifying the phenodata column in `x` defining the sample grouping or a vector with the same length than samples in `x` defining the group assignment of the samples.
`value`	`character(1)` specifying the value to be returned for each feature. See `featureValues()` for more details.
`digits`	`integer(1)` defining the number of significant digits to be used for numeric. The default `NULL` uses `getOption("digits")`. See `format()` for more information.

`groupnames`	`logical(1)` whether row names of the resulting matrix should be the feature IDs (`groupnames = FALSE`; default) or IDs that are composed of the m/z and retention time of the features (in the format `M<m/z>T<rt>` (`groupnames = TRUE`). See help of the groupnames function for details.
`...`	additional parameters to be passed to the `featureValues()` function.

Value

If `file` is not specified, the function returns the `matrix` in the format supported by MetaboAnalyst.

Author(s)

Johannes Rainer

`extractMsData,OnDiskMSnExp-method`

DEPRECATED: Extract a `data.frame` *containing MS data*

Description

UPDATE: the `extractMsData` and `plotMsData` functions are deprecated and `as(x, "data.frame")` and `plot(x, type = "XIC")` (`x` being an `OnDiskMSnExp` or `XCMSnExp` object) should be used instead. See examples below. Be aware that filtering the raw object might however drop the adjusted retention times. In such cases it is advisable to use the `applyAdjustedRtime()` function prior to filtering.

Extract a `data.frame` of retention time, mz and intensity values from each file/sample in the provided rt-mz range (or for the full data range if `rt` and `mz` are not defined).

Usage

```
## S4 method for signature 'OnDiskMSnExp'
extractMsData(object, rt, mz, msLevel = 1L)

## S4 method for signature 'XCMSnExp'
extractMsData(
  object,
  rt,
  mz,
  msLevel = 1L,
  adjustedRtime = hasAdjustedRtime(object)
)
```

Arguments

`object`	A `XCMSnExp` or `OnDiskMSnExp` object.
`rt`	`numeric(2)` with the retention time range from which the data should be extracted.
`mz`	`numeric(2)` with the mz range.
`msLevel`	`integer` defining the MS level(s) to which the data should be sub-setted prior to extraction; defaults to `msLevel = 1L`.
`adjustedRtime`	(for `extractMsData,XCMSnExp`): `logical(1)` specifying if adjusted or raw retention times should be reported. Defaults to adjusted retention times, if these are present in `object`.

Value

A `list` of length equal to the number of samples/files in `object`. Each element being a `data.frame` with columns `"rt"`, `"mz"` and `"i"` with the retention time, mz and intensity tuples of a file. If no data is available for the mz-rt range in a file a `data.frame` with 0 rows is returned for that file.

Author(s)

Johannes Rainer

See Also

`XCMSnExp` for the data object.

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Extract the full MS data for a certain retention time range
## as a data.frame
tmp <- filterRt(faahko_sub, rt = c(2800, 2900))
ms_all <- as(tmp, "data.frame")
head(ms_all)
nrow(ms_all)
```

`feature-grouping` *Compounding of LC-MS features*

Description

Feature *compounding* aims at identifying and grouping LC-MS features representing different ions or adducts (including isotopes) of the same originating compound. The MsFeatures package provides a general framework and functionality to group features based on different properties. The `groupFeatures` methods for XCMSnExp objects implemented in `xcms` extend these to enable the *compounding* of LC-MS data. Note that these functions simply define feature groups but don't actually *aggregate* or combine the features.

See `MsFeatures::groupFeatures()` for an overview on the general feature grouping concept as well as details on the individual settings and parameters.

The available options for `groupFeatures` on `xcms` preprocessing results (i.e. on `XCMSnExp` objects after correspondence analysis with `groupChromPeaks()`) are:

- Grouping by similar retention times: `groupFeatures-similar-rtime()`.
- Grouping by similar feature values across samples: `AbundanceSimilarityParam()`.
- Grouping by similar peak shape of extracted ion chromatograms: `EicSimilarityParam()`.

An ideal workflow grouping features should sequentially perform the above methods (in the listed order).

Compounded feature groups can be accessed with the `featureGroups` function.

Usage

```
## S4 method for signature 'XCMSnExp'
featureGroups(object)

## S4 replacement method for signature 'XCMSnExp'
featureGroups(object) <- value
```

Arguments

`object`	an `XCMSnExp()` object.
`value`	for `featureGroups<-`: replacement for the feature groups in `object`. Has to be of length 1 or length equal to the number of features in `object`.

Author(s)

Johannes Rainer, Mar Garcia-Aloy, Vinicius Veri Hernandes

See Also

`plotFeatureGroups()` for visualization of grouped features.

`featureChromatograms` *Extract ion chromatograms for each feature*

Description

Extract ion chromatograms for features in an XCMSnExp object. The function returns for each feature its extracted ion chromatogram and all associated peaks with it. The chromatogram is extracted from the m/z - rt region including all chromatographic peaks of that features (i.e. based on the ranges of `"mzmin"`, `"mzmax"`, `"rtmin"`, `"rtmax"` of all chromatographic peaks of the feature).

By default only chromatographic peaks associated with a feature are included for an extracted ion chromatogram (parameter `include = "feature_only"`). By setting `include = "apex_within"` all chromatographic peaks (and eventually the feature which they are part of - if feature definitions are present) that have their apex position within the m/z - rt range from which the chromatogram is extracted are returned too. With `include = "any"` or `include = "all"` all chromatographic peaks (and eventually the feature in which they are present) overlapping the m/z and rt range will be returned.

Usage

```
featureChromatograms(
  x,
  expandRt = 0,
  aggregationFun = "max",
  features,
  include = c("feature_only", "apex_within", "any", "all"),
  filled = FALSE,
  n = length(fileNames(x)),
  value = c("maxo", "into"),
  expandMz = 0,
  ...
)
```

Arguments

`x`	`XCMSnExp` object with grouped chromatographic peaks.
`expandRt`	`numeric(1)` to expand the retention time range for each chromatographic peak by a constant value on each side.
`aggregationFun`	`character(1)` specifying the name that should be used to aggregate intensity values across the m/z value range for the same retention time. The default `"sum"` returns a base peak chromatogram.
`features`	`integer`, `character` or `logical` defining a subset of features for which chromatograms should be returned. Can be the index of the features in `featureDefinitions`, feature IDs (row names of `featureDefinitions`) or a logical vector.
`include`	`character(1)` defining which chromatographic peaks (and related feature definitions) should be included in the returned `XChromatograms()`. Defaults to `"feature_only"`; See description above for options and details.

`filled`	`logical(1)` whether filled-in peaks should be included in the result object. The default is `filled = FALSE`, i.e. only detected peaks are reported.
`n`	`integer(1)` to optionally specify the number of *top n* samples from which the EIC should be extracted.
`value`	`character(1)` specifying the column to be used to sort the samples. Can be either `"maxo"` (the default) or `"into"` to use the maximal peak intensity or the integrated peak area, respectively.
`expandMz`	`numeric(1)` to expand the m/z range for each chromatographic peak by a constant value on each side. Be aware that by extending the m/z range the extracted EIC might **no longer** represent the actual identified chromatographic peak because intensities of potential additional mass peaks within each spectra would be aggregated into the final reported intensity value per spectrum (retention time).
`...`	optional arguments to be passed along to the `chromatogram()` function.

Value

`XChromatograms()` object.

Note

When extracting EICs from only the top `n` samples it can happen that one or more of the features specified with `features` are dropped because they have no detected peak in the *top n* samples. The chance for this to happen is smaller if `x` contains also filled-in peaks (with `fillChromPeaks`).

Author(s)

Johannes Rainer

See Also

`filterColumnsKeepTop()` to filter the extracted EICs keeping only the *top n* columns (samples) with the highest intensity.

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Subset the object to a smaller retention time range
xdata <- filterRt(faahko_sub, c(2500, 3500))

xdata <- groupChromPeaks(xdata,
    param = PeakDensityParam(minFraction = 0.8, sampleGroups = rep(1, 3)))

## Get the feature definitions
```

```
featureDefinitions(xdata)

## Extract ion chromatograms for the first 3 features. Parameter
## `features` can be either the feature IDs or feature indices.
chrs <- featureChromatograms(xdata, features = 1:3)

## Plot the XIC for the first feature using different colors for each file
plot(chrs[1, ], col = c("red", "green", "blue"))
```

`featureSpectra` *Extract spectra associated with features*

Description

This function returns spectra associated with the identified features in the input object. By default, spectra are returned for all features (from all MS levels), but parameter `features` allows to specify selected features for which the result should be returned. Parameter `msLevel` allows to define whether MS level 1 or 2 spectra should be returned. For `msLevel = 1L` all MS1 spectra within the retention time range of each chromatographic peak (in that respective data file) associated with a feature are returned. Note that for samples in which no peak was identified (or even filled-in) no spectra are returned. For `msLevel = 2L` all MS2 spectra with a retention time within the retention time range and their precursor m/z within the m/z range of any chromatographic peak of a feature are returned. See also `chromPeakSpectra()` (used internally to extract spectra for each chromatographic peak of a feature) for additional information.

In contrast to the `chromPeakSpectra()` function, selecting a `method` different than `"all"` will not return a single spectrum per feature, but one spectrum per **chromatographic peak** assigned to the feature.

Note also that `msLevel = 1L` is only supported for `return.type = "List"` or `return.type = "Spectra"`.

Usage

```
featureSpectra(
  x,
  msLevel = 2L,
  expandRt = 0,
  expandMz = 0,
  ppm = 0,
  skipFilled = FALSE,
  return.type = c("MSpectra", "Spectra", "list", "List"),
  features = character(),
  ...
)
```

Arguments

`x`	XCMSnExp object with feature defitions available.

`msLevel`	`integer(1)` defining whether MS1 or MS2 spectra should be returned. `msLevel = 1` is currently only supported for `return.type` being `"Spectra"` or `"List"`.
`expandRt`	`numeric(1)` to expand the retention time range of each peak by a constant value on each side.
`expandMz`	`numeric(1)` to expand the m/z range of each peak by a constant value on each side.
`ppm`	`numeric(1)` to expand the m/z range of each peak (on each side) by a value dependent on the peak's m/z.
`skipFilled`	`logical(1)` whether spectra for filled-in peaks should be reported or not.
`return.type`	`character(1)` defining the result type. Defaults to `return.type = "MSpectra"` but `return.type = "Spectra"` or `return.type = "List"` are preferred. See below for more information.
`features`	`character`, `logical` or `integer` allowing to specify a subset of features in `featureDefinitions` for which spectra should be returned (providing either their ID, a logical vector same length than `nrow(featureDefinitions(x))` or their index in `featureDefinitions(x)`). This parameter overrides `skipFilled` and is only supported for `return.type` being either `"Spectra"` or `"List"`.
`...`	additional arguments to be passed along to `chromPeakSpectra()`, such as `method`.

Value

parameter `return.type` allow to specify the type of the returned object:

- `return.type = "MSpectra"`: a MSpectra object with elements being Spectrum objects. The result objects contains all spectra for all features. Metadata column `"feature_id"` provides the ID of the respective feature (i.e. its rowname in `featureDefinitions()`).
- `return.type = "Spectra"`: a `Spectra` object (defined in the `Spectra` package). The result contains all spectra for all features. Metadata column `"feature_id"` provides the ID of the respective feature (i.e. its rowname in `featureDefinitions()`.
- `return.type = "list"`: `list` of `list`s that are either of length 0 or contain Spectrum2 object(s) within the m/z-rt range. The length of the list matches the number of features.
- `return.type = "List"`: `List` of length equal to the number of features with MS level `msLevel` is returned with elements being either `NULL` (no spectrum found) or a `Spectra` object.

Author(s)

Johannes Rainer

`featureSummary` *Simple feature summaries*

Description

Simple function to calculate feature summaries. These include counts and percentages of samples in which a chromatographic peak is present for each feature and counts and percentages of samples in which more than one chromatographic peak was annotated to the feature. Also relative standard deviations (RSD) are calculated for the integrated peak areas per feature across samples. For ‘perSampleCounts = TRUE‘ also the individual chromatographic peak counts per sample are returned.

Usage

```
featureSummary(
  x,
  group,
  perSampleCounts = FALSE,
  method = "maxint",
  skipFilled = TRUE
)
```

Arguments

`x`	‘XCMSnExp‘ object with correspondence results.
`group`	‘numeric‘, ‘logical‘, ‘character‘ or ‘factor‘ with the same length than ‘x‘ has samples to aggregate counts by the groups defined in ‘group‘.
`perSampleCounts`	‘logical(1)‘ whether feature wise individual peak counts per sample should be returned too.
`method`	‘character‘ passed to the [featureValues()] function. See respective help page for more information.
`skipFilled`	‘logical(1)‘ whether filled-in peaks should be excluded (default) or included in the summary calculation.

Value

‘matrix‘ with one row per feature and columns:

- ‘"count"‘: the total number of samples in which a peak was found. - ‘"perc"‘: the percentage of samples in which a peak was found. - ‘"multi_count"‘: the total number of samples in which more than one peak was assigned to the feature. - ‘"multi_perc"‘: the percentage of those samples in which a peak was found, that have also multiple peaks annotated to the feature. Example: for a feature, at least one peak was detected in 50 samples. In 5 of them 2 peaks were assigned to the feature. ‘"multi_perc"‘ is in this case 10 - ‘"rsd"‘: relative standard deviation (coefficient of variation) of the integrated peak area of the feature’s peaks. - The same 4 columns are repeated for each unique element (level) in ‘group‘ if ‘group‘ was provided.

If ‘perSampleCounts = TRUE‘ also one column for each sample is returned with the peak counts per sample.

Author(s)

Johannes Rainer

`FillChromPeaksParam-class`

Integrate areas of missing peaks

Description

`expandMz`,`expandMz<-`: getter and setter for the `expandMz` slot of the object.

`expandRt`,`expandRt<-`: getter and setter for the `expandRt` slot of the object.

`ppm`,`ppm<-`: getter and setter for the `ppm` slot of the object.

Integrate signal in the mz-rt area of a feature (chromatographic peak group) for samples in which no chromatographic peak for this feature was identified and add it to the `chromPeaks()` matrix. Such *filled-in* peaks are indicated with a `TRUE` in column `"is_filled"` in the result object's `chromPeakData()` data frame.

Two different gap-filling approaches are implemented:

- `param = FillChromPeaksParam()`: the default of the original `xcms` code. Signal is integrated from the m/z and retention time range as defined in the `featureDefinitions()` data frame, i.e. from the `"rtmin"`, `"rtmax"`, `"mzmin"` and `"mzmax"`. See details below for more information and settings for this method.
- `param = ChromPeakAreaParam()`: the area from which the signal for a feature is integrated is defined based on the feature's chromatographic peak areas. The m/z range is by default defined as the the lower quartile of chromatographic peaks' `"mzmin"` value to the upper quartile of the chromatographic peaks' `"mzmax"` values. The retention time range for the area is defined analogously. Alternatively, by setting `mzmin = median`, `mzmax = median`, `rtmin = median` and `rtmax = median` in `ChromPeakAreaParam`, the median `"mzmin"`, `"mzmax"`, `"rtmin"` and `"rtmax"` values from all detected chromatographic peaks of a feature would be used instead. In contrast to the `FillChromPeaksParam` approach this method uses the actual identified chromatographic peaks of a feature to define the area from which the signal should be integrated.

Usage

```
FillChromPeaksParam(
  expandMz = 0,
  expandRt = 0,
  ppm = 0,
  fixedMz = 0,
  fixedRt = 0
)
```

```
fixedRt(object)

fixedMz(object)

ChromPeakAreaParam(
  mzmin = function(z) quantile(z, probs = 0.25),
  mzmax = function(z) quantile(z, probs = 0.75),
  rtmin = function(z) quantile(z, probs = 0.25),
  rtmax = function(z) quantile(z, probs = 0.75)
)

## S4 method for signature 'FillChromPeaksParam'
expandMz(object)

## S4 replacement method for signature 'FillChromPeaksParam'
expandMz(object) <- value

## S4 method for signature 'FillChromPeaksParam'
expandRt(object)

## S4 replacement method for signature 'FillChromPeaksParam'
expandRt(object) <- value

## S4 method for signature 'FillChromPeaksParam'
ppm(object)

## S4 replacement method for signature 'FillChromPeaksParam'
ppm(object) <- value

## S4 method for signature 'XCMSnExp,FillChromPeaksParam'
fillChromPeaks(object, param, msLevel = 1L, BPPARAM = bpparam())

## S4 method for signature 'XCMSnExp,ChromPeakAreaParam'
fillChromPeaks(object, param, msLevel = 1L, BPPARAM = bpparam())

## S4 method for signature 'XCMSnExp,missing'
fillChromPeaks(object, param, BPPARAM = bpparam(), msLevel = 1L)
```

Arguments

`expandMz`	for `FillChromPeaksParam`: `numeric(1)` defining the value by which the mz width of peaks should be expanded. Each peak is expanded in mz direction by `expandMz *` their original m/z width. A value of `0` means no expansion, a value of 1 grows each peak by `1 *` the m/z width of the peak resulting in peaks with twice their original size in m/z direction (expansion by half m/z width to both sides).
`expandRt`	for `FillChromPeaksParam`: `numeric(1)`, same as `expandMz` but for the retention time width.

`ppm`	for `FillChromPeaksParam`: `numeric(1)` optionally specifying a *ppm* by which the m/z width of the peak region should be expanded. For peaks with an m/z width smaller than `mean(c(mzmin, mzmax)) * ppm / 1e6`, the `mzmin` will be replaced by `mean(c(mzmin, mzmax)) - (mean(c(mzmin, mzmax)) * ppm / 2 / 1e6)` `mzmax` by `mean(c(mzmin, mzmax)) + (mean(c(mzmin, mzmax)) * ppm / 2 / 1e6)`. This is applied before eventually expanding the m/z width using the `expandMz` parameter.
`fixedMz`	for `FillChromPeaksParam`: `numeric(1)` defining a constant factor by which the m/z width of each feature is to be expanded. The m/z width is expanded on both sides by `fixedMz` (i.e. `fixedMz` is subtracted from the lower m/z and added to the upper m/z). This expansion is applied *after* `expandMz` and `ppm`.
`fixedRt`	for `FillChromPeaksParam`: `numeric(1)` defining a constant factor by which the retention time width of each factor is to be expanded. The rt width is expanded on both sides by `fixedRt` (i.e. `fixedRt` is subtracted from the lower rt and added to the upper rt). This expansion is applied *after* `expandRt`.
`object`	`XCMSnExp` object with identified and grouped chromatographic peaks.
`mzmin`	`function` to be applied to values in the `"mzmin"` column of all chromatographic peaks of a feature to define the lower m/z value of the area from which signal for the feature should be integrated. Defaults to `mzmin = function(z) quantile(z, probs = 0.25)` hence using the 25% quantile of all values.
`mzmax`	`function` to be applied to values in the `"mzmax"` column of all chromatographic peaks of a feature to define the upper m/z value of the area from which signal for the feature should be integrated. Defaults to `mzmax = function(z) quantile(z, probs = 0.75)` hence using the 75% quantile of all values.
`rtmin`	`function` to be applied to values in the `"rtmin"` column of all chromatographic peaks of a feature to define the lower rt value of the area from which signal for the feature should be integrated. Defaults to `rtmin = function(z) quantile(z, probs = 0.25)` hence using the 25% quantile of all values.
`rtmax`	`function` to be applied to values in the `"rtmax"` column of all chromatographic peaks of a feature to define the upper rt value of the area from which signal for the feature should be integrated. Defaults to `rtmax = function(z) quantile(z, probs = 0.75)` hence using the 75% quantile of all values.
`value`	The value for the slot.
`param`	`FillChromPeaksParam` or `ChromPeakAreaParam` object defining which approach should be used (see details section).
`msLevel`	`integer(1)` defining the MS level on which peak filling should be performed (defaults to `msLevel = 1L`). Only peak filling on one MS level at a time is supported, to fill in peaks for MS level 1 and 2 run first using `msLevel = 1` and then (on the returned result object) again with `msLevel = 2`.
`BPPARAM`	Parallel processing settings.

Details

After correspondence (i.e. grouping of chromatographic peaks across samples) there will always be features (peak groups) that do not include peaks from every sample. The `fillChromPeaks` method defines intensity values for such features in the missing samples by integrating the signal in the mz-rt region of the feature. Two different approaches to define this region are available: with `ChromPeakAreaParam` the region is defined based on the detected **chromatographic peaks** of a feature, while with `FillChromPeaksParam` the region is defined based on the m/z and retention times of the **feature** (which represent the m/z and retentention times of the apex position of the associated chromatographic peaks). For the latter approach various parameters are available to increase the area from which signal is to be integrated, either by a constant value (`fixedMz` and `fixedRt`) or by a feature-relative amount (`expandMz` and `expandRt`).

Adjusted retention times will be used if available.

Based on the peak finding algorithm that was used to identify the (chromatographic) peaks, different internal functions are used to guarantee that the integrated peak signal matches as much as possible the peak signal integration used during the peak detection. For peaks identified with the `matchedFilter()` method, signal integration is performed on the *profile matrix* generated with the same settings used also during peak finding (using the same `bin` size for example). For direct injection data and peaks identified with the `MSW` algorithm signal is integrated only along the mz dimension. For all other methods the complete (raw) signal within the area is used.

Value

The `FillChromPeaksParam` function returns a `FillChromPeaksParam` object.

A `XCMSnExp` object with previously missing chromatographic peaks for features filled into its `chromPeaks()` matrix.

Slots

`expandMz,expandRt,ppm,fixedMz,fixedRt` See corresponding parameter above.

`rtmin,rtmax,mzmin,mzmax` See corresponding parameter above.

Note

The reported `"mzmin"`, `"mzmax"`, `"rtmin"` and `"rtmax"` for the filled peaks represents the actual MS area from which the signal was integrated. Note that no peak is filled in if no signal was present in a file/sample in the respective mz-rt area. These samples will still show a `NA` in the matrix returned by the `featureValues()` method.

Author(s)

Johannes Rainer

See Also

`groupChromPeaks()` for methods to perform the correspondence.

featureArea for the function to define the m/z-retention time region for each feature.

Examples

```
## Load a test data set with identified chromatographic peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")
res <- faahko_sub

## Disable parallel processing for this example
register(SerialParam())

## Perform the correspondence. We assign all samples to the same group.
res <- groupChromPeaks(res,
    param = PeakDensityParam(sampleGroups = rep(1, length(fileNames(res)))))

## For how many features do we lack an integrated peak signal?
sum(is.na(featureValues(res)))

## Filling missing peak data using the peak area from identified
## chromatographic peaks.
res <- fillChromPeaks(res, param = ChromPeakAreaParam())

## How many missing values do we have after peak filling?
sum(is.na(featureValues(res)))

## Get the peaks that have been filled in:
fp <- chromPeaks(res)[chromPeakData(res)$is_filled, ]
head(fp)

## Get the process history step along with the parameters used to perform
## The peak filling:
ph <- processHistory(res, type = "Missing peak filling")[[1]]
ph

## The parameter class:
ph@param

## It is also possible to remove filled-in peaks:
res <- dropFilledChromPeaks(res)

sum(is.na(featureValues(res)))
```

`fillPeaks-methods` *Integrate areas of missing peaks*

Description

For each sample, identify peak groups where that sample is not represented. For each of those peak groups, integrate the signal in the region of that peak group and create a new peak.

Arguments

`object`	the `xcmsSet` object
`method`	the filling method

Details

After peak grouping, there will always be peak groups that do not include peaks from every sample. This method produces intensity values for those missing samples by integrating raw data in peak group region. According to the type of raw-data there are 2 different methods available. for filling gcms/lcms data the method "chrom" integrates raw-data in the chromatographic domain, whereas "MSW" is used for peaklists without retention-time information like those from direct-infusion spectra.

Value

A `xcmsSet` objects with filled in peak groups.

Methods

object = "xcmsSet" `fillPeaks(object, method="")`

See Also

`xcmsSet-class`, `getPeaks`

`fillPeaks.chrom-methods` *Integrate areas of missing peaks*

Description

For each sample, identify peak groups where that sample is not represented. For each of those peak groups, integrate the signal in the region of that peak group and create a new peak.

Arguments

`object`	the `xcmsSet` object
`nSlaves`	(DEPRECATED): number of slaves/cores to be used for parallel peak filling. MPI is used if installed, otherwise the snow package is employed for multicore support. If none of the two packages is available it uses the parallel package for parallel processing on multiple CPUs of the current machine. Users are advised to use the `BPPARAM` parameter instead.
`expand.mz`	Expansion factor for the m/z range used for integration.
`expand.rt`	Expansion factor for the rentention time range used for integration.
`BPPARAM`	allows to define a specific parallel processing setup for the current task (see `bpparam` from the `BiocParallel` package help more information). The default uses the globally defined parallel setup.

Details

After peak grouping, there will always be peak groups that do not include peaks from every sample. This method produces intensity values for those missing samples by integrating raw data in peak group region. In a given group, the start and ending retention time points for integration are defined by the median start and end points of the other detected peaks. The start and end m/z values are similarly determined. Intensities can be still be zero, which is a rather unusual intensity for a peak. This is the case if e.g. the raw data was threshholded, and the integration area contains no actual raw intensities, or if one sample is miscalibrated, such thet the raw data points are (just) outside the integration area.

Importantly, if retention time correction data is available, the alignment information is used to more precisely integrate the propper region of the raw data. If the corrected retention time is beyond the end of the raw data, the value will be not-a-number (NaN).

Value

A `xcmsSet` objects with filled in peak groups (into and maxo).

Methods

object = "xcmsSet" `fillPeaks.chrom(object, nSlaves=0,expand.mz=1,expand.rt=1, BPPARAM = bpparam())`

See Also

`xcmsSet-class`, `getPeaks` `fillPeaks`

`fillPeaks.MSW-methods` *Integrate areas of missing peaks in FTICR-MS data*

Description

For each sample, identify peak groups where that sample is not represented. For each of those peak groups, integrate the signal in the region of that peak group and create a new peak.

Arguments

`object`	the `xcmsSet` object

Details

After peak grouping, there will always be peak groups that do not include peaks from every sample. This method produces intensity values for those missing samples by integrating raw data in peak group region. In a given group, the start and ending m/z values for integration are defined by the median start and end points of the other detected peaks.

Value

A `xcmsSet` objects with filled in peak groups.

Methods

object = "xcmsSet" `fillPeaks.MSW(object)`

Note

In contrast to the `fillPeaks.chrom` method the maximum intensity reported in column `"maxo"` is not the maximum intensity measured in the expected peak area (defined by columns `"mzmin"` and `"mzmax"`), but the largest intensity of mz value(s) closest to the `"mzmed"` of the feature.

See Also

`xcmsSet-class`, `getPeaks` `fillPeaks`

`filterColumnsIntensityAbove,MChromatograms-method`
Filtering sets of chromatographic data

Description

These functions allow to filter (subset) `MChromatograms()` or `XChromatograms()` objects, i.e. sets of chromatographic data, without changing the data (intensity and retention times) within the individual chromatograms (`Chromatogram()` objects).

- `filterColumnsIntensityAbove`: subsets a `MChromatograms` objects keeping only columns (samples) for which `value` is larger than the provided `threshold` in `which` rows (i.e. if `which = "any"` a column is kept if **any** of the chromatograms in that column have a `value` larger than `threshold` or with `which = "all"` **all** chromatograms in that column fulfill this criteria). Parameter `value` allows to define on which value the comparison should be performed, with `value = "bpi"` the maximum intensity of each chromatogram is compared to `threshold`, with `value = "tic" the total sum of intensities of each chromatogram is compared to threshold. For` XChromatogramsobject`,`value = "maxo"andvalue = "into"`are supported which compares the largest` or the integrated peak area, respectively.
- `filterColumnsKeepTop`: subsets a `MChromatograms` object keeping the top `n` columns sorted by the value specified with `sortBy`. In detail, for each column the value defined by `sortBy` is extracted from each chromatogram and aggregated using the `aggregationFun`. Thus, by default, for each chromatogram the maximum intensity is determined (`sortBy = "bpi"`) and these values are summed up for chromatograms in the same column (`aggregationFun = sum`). The columns are then sorted by these values and the top `n` columns are retained in the returned `MChromatograms`. Similar to the `filterColumnsIntensityAbove` function, this function allows to use for `XChromatograms` objects to sort the columns by column `sortBy = "maxo"` or `sortBy = "into"` of the `chromPeaks` matrix.

Usage

```
## S4 method for signature 'MChromatograms'
filterColumnsIntensityAbove(
  object,
  threshold = 0,
  value = c("bpi", "tic"),
  which = c("any", "all")
)

## S4 method for signature 'MChromatograms'
filterColumnsKeepTop(
  object,
  n = 1L,
  sortBy = c("bpi", "tic"),
  aggregationFun = sum
)

## S4 method for signature 'XChromatograms'
filterColumnsIntensityAbove(
  object,
  threshold = 0,
  value = c("bpi", "tic", "maxo", "into"),
  which = c("any", "all")
)

## S4 method for signature 'XChromatograms'
filterColumnsKeepTop(
  object,
  n = 1L,
  sortBy = c("bpi", "tic", "maxo", "into"),
  aggregationFun = sum
)
```

Arguments

`object`	`MChromatograms()` or `XChromatograms()` object.
`threshold`	for `filterColumnsIntensityAbove`: `numeric(1)` with the threshold value to compare against.
`value`	`character(1)` defining which value should be used in the comparison or sorting. Can be `value = "bpi"` (default) to use the maximum intensity per chromatogram or `value = "tic"` to use the sum of intensities per chromatogram. For `XChromatograms()` objects also `value = "maxo"` and `value = "into"` is supported to use the maximum intensity or the integrated area of identified chromatographic peaks in each chromatogram.
`which`	for `filterColumnsIntensityAbove`: `character(1)` defining whether **any** (`which = "any"`, default) or **all** (`which = "all"`) chromatograms in a column have to fulfill the criteria for the column to be kept.

`n`	for `filterColumnsKeepTop`: `integer(1)` specifying the number of columns that should be returned. `n` will be rounded to the closest (larger) integer value.
`sortBy`	for `filterColumnsKeepTop`: the value by which columns should be ordered to determine the top `n` columns. Can be either `sortBy = "bpi"` (the default), in which case the maximum intensity of each column's chromatograms is used, or `sortBy = "tic"` to use the total intensity sum of all chromatograms. For `XChromatograms()` objects also `value = "maxo"` and `value = "into"` is supported to use the maximum intensity or the integrated area of identified chromatographic peaks in each chromatogram.
`aggregationFun`	for `filterColumnsKeepTop`: function to be used to aggregate (combine) the values from all chromatograms in each column. Defaults to `aggregationFun = sum` in which case the sum of the values is used to rank the columns. Alternatively the `mean`, `median` or similar function can be used.

Value

a filtered `MChromatograms` (or `XChromatograms`) object with the same number of rows (EICs) but eventually a lower number of columns (samples).

Author(s)

Johannes Rainer

Examples

```
chr1 <- Chromatogram(rtime = 1:10 + rnorm(n = 10, sd = 0.3),
    intensity = c(5, 29, 50, NA, 100, 12, 3, 4, 1, 3))
chr2 <- Chromatogram(rtime = 1:10 + rnorm(n = 10, sd = 0.3),
    intensity = c(80, 50, 20, 10, 9, 4, 3, 4, 1, 3))
chr3 <- Chromatogram(rtime = 3:9 + rnorm(7, sd = 0.3),
    intensity = c(53, 80, 130, 15, 5, 3, 2))

chrs <- MChromatograms(list(chr1, chr2, chr1, chr3, chr2, chr3),
    ncol = 3, byrow = FALSE)
chrs

#### filterColumnsIntensityAbove
##
## Keep all columns with for which the maximum intensity of any of its
## chromatograms is larger 90
filterColumnsIntensityAbove(chrs, threshold = 90)

## Require that ALL chromatograms in a column have a value larger 90
filterColumnsIntensityAbove(chrs, threshold = 90, which = "all")

## If none of the columns fulfills the criteria no columns are returned
filterColumnsIntensityAbove(chrs, threshold = 900)

## Filtering XChromatograms allow in addition to filter on the columns
## "maxo" or "into" of the identified chromatographic peaks within each
## chromatogram.
```

```
#### filterColumnsKeepTop
##
## Keep the 2 columns with the highest sum of maximal intensities in their
## chromatograms
filterColumnsKeepTop(chrs, n = 1)

## Keep the 50 percent of columns with the highest total sum of signal. Note
## that n will be rounded to the next larger integer value
filterColumnsKeepTop(chrs, n = 0.5 * ncol(chrs), sortBy = "tic")
```

`filterFeatureDefinitions` *XCMSnExp filtering and subsetting*

Description

The methods listed on this page allow to filter and subset XCMSnExp objects. Most of them are inherited from the OnDiskMSnExp object defined in the `MSnbase` package and have been adapted for `XCMSnExp` to enable correct subsetting of preprocessing results.

- `[`: subset a `XCMSnExp` object by spectra. Be aware that this removes **all** preprocessing results, except adjusted retention times if `keepAdjustedRtime = TRUE` is passed to the method.
- `[[`: extracts a single `Spectrum` object (defined in `MSnbase`). The reported retention time is the adjusted retention time if alignment has been performed.
- `filterChromPeaks`: subset the `chromPeaks` `matrix` in `object`. Parameter `method` allows to specify how the chromatographic peaks should be filtered. Currently, only `method = "keep"` is supported which allows to specify chromatographic peaks to keep with parameter `keep` (i.e. provide a `logical`, `integer` or `character` defining which chromatographic peaks to keep). Feature definitions (if present) are updated correspondingly.
- `filterFeatureDefinitions`: allows to subset the feature definitions of an `XCMSnExp` object. Parameter `features` allow to define which features to keep. It can be a `logical`, `integer` (index of features to keep) or `character` (feature IDs) vector.
- `filterFile`: allows to reduce the `XCMSnExp` to data from only selected files. Identified chromatographic peaks for these files are retained while correspondence results (feature definitions) are removed by default. To force keeping feature definitions use `keepFeatures = TRUE`. Adjusted retention times (if present) are retained by default if present. Use `keepAdjustedRtime = FALSE` to drop them.
- `filterMsLevel`: reduces the `XCMSnExp` object to spectra of the specified MS level(s). Chromatographic peaks and identified features are also subsetted to the respective MS level. See also the `filterMsLevel` documentation in `MSnbase` for details and examples.
- `filterMz`: filters the data set based on the provided m/z value range. All chromatographic peaks and features (grouped peaks) falling **completely** within the provided mz value range are retained (i.e. if their minimal m/z value is `>= mz[1]` and the maximal m/z value `<= mz[2]`. Adjusted retention times, if present, are kept.

- `filterRt`: filters the data set based on the provided retention time range. All chromatographic peaks and features (grouped peaks) **completely** within the specified retention time window are retained (i.e. if the retention time corresponding to the peak's apex is within the specified rt range). If retention time correction has been performed, the method will by default filter the object by adjusted retention times. The argument `adjusted` allows to specify manually whether filtering should be performed on raw or adjusted retention times. Filtering by retention time does not drop any preprocessing results nor does it remove or change alignment results (i.e. adjusted retention times). The method returns an empty object if no spectrum or feature is within the specified retention time range.
- `split`: splits an `XCMSnExp` object into a `list` of `XCMSnExp` objects based on the provided parameter `f`. Note that by default all pre-processing results are removed by the splitting, except adjusted retention times, if the optional argument `keepAdjustedRtime = TRUE` is provided.

Usage

```
filterFeatureDefinitions(x, features)

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

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

## S4 method for signature 'XCMSnExp'
filterMsLevel(object, msLevel., keepAdjustedRtime = hasAdjustedRtime(object))

## S4 method for signature 'XCMSnExp'
filterFile(
  object,
  file,
  keepAdjustedRtime = hasAdjustedRtime(object),
  keepFeatures = FALSE
)

## S4 method for signature 'XCMSnExp'
filterMz(object, mz, msLevel., ...)

## S4 method for signature 'XCMSnExp'
filterRt(object, rt, msLevel., adjusted = hasAdjustedRtime(object))

## S4 method for signature 'XCMSnExp,ANY'
split(x, f, drop = FALSE, ...)

## S4 method for signature 'XCMSnExp'
filterChromPeaks(
  object,
  keep = rep(TRUE, nrow(chromPeaks(object))),
  method = "keep",
```

```
  ...
)
```

Arguments

`x`	For `[` and `[[`: an `XCMSnExp` object.
`features`	For `filterFeatureDefinitions`: either a `integer` specifying the indices of the features (rows) to keep, a `logical` with a length matching the number of rows of `featureDefinitions` or a `character` with the feature (row) names.
`i`	For `[`: `numeric` or `logical` vector specifying to which spectra the data set should be reduced. For `[[`: a single integer or character.
`j`	For `[` and `[[`: not supported.
`...`	Optional additional arguments.
`drop`	For `[` and `[[`: not supported.
`object`	A XCMSnExp object.
`msLevel.`	For `filterMz`, `filterRt`: `numeric` defining the MS level(s) to which operations should be applied or to which the object should be subsetted.
`keepAdjustedRtime`	For `filterFile`, `filterMsLevel`, `[`, `split`: `logical(1)` defining whether the adjusted retention times should be kept, even if e.g. features are being removed (and the retention time correction was performed on these features).
`file`	For `filterFile`: `integer` defining the file index within the object to subset the object by file or `character` specifying the file names to sub set. The indices are expected to be increasingly ordered, if not they are ordered internally.
`keepFeatures`	For `filterFile`: `logical(1)` whether correspondence results (feature definitions) should be kept or dropped. Defaults to `keepFeatures = FALSE` hence feature definitions are removed from the returned object by default.
`mz`	For `filterMz`: `numeric(2)` defining the lower and upper mz value for the filtering.
`rt`	For `filterRt`: `numeric(2)` defining the retention time window (lower and upper bound) for the filtering.
`adjusted`	For `filterRt`: `logical` indicating whether the object should be filtered by original (`adjusted = FALSE`) or adjusted retention times (`adjusted = TRUE`). For `spectra`: whether the retention times in the individual `Spectrum` objects should be the adjusted or raw retention times.
`f`	For `split` a vector of length equal to the length of `x` defining how `x` should be splitted. It is converted internally to a `factor`.
`keep`	For `filterChromPeaks`: `logical`, `integer` or `character` defining which chromatographic peaks should be retained.
`method`	For `filterChromPeaks`: `character(1)` allowing to specify the method by which chromatographic peaks should be filtered. Currently only `method = "keep"` is supported (i.e. specify with parameter `keep` which chromatographic peaks should be retained).

Details

All subsetting methods try to ensure that the returned data is consistent. Correspondence results for example are removed by default if the data set is sub-setted by file, since the correspondence results are dependent on the files on which correspondence was performed. This can be changed by setting `keepFeatures = TRUE`. For adjusted retention times, most subsetting methods support the argument `keepAdjustedRtime` (even the `[` method) that forces the adjusted retention times to be retained even if the default would be to drop them.

Value

All methods return an XCMSnExp object.

Note

The `filterFile` method removes also process history steps not related to the files to which the object should be sub-setted and updates the `fileIndex` attribute accordingly. Also, the method does not allow arbitrary ordering of the files or re-ordering of the files within the object.

Note also that most of the filtering methods, and also the subsetting operations `[` drop all or selected preprocessing results. To consolidate the alignment results, i.e. ensure that adjusted retention times are always preserved, use the `applyAdjustedRtime()` function on the object that contains the alignment results. This replaces the raw retention times with the adjusted ones.

Author(s)

Johannes Rainer

See Also

XCMSnExp for base class documentation.

`XChromatograms()` for similar filter functions on `XChromatograms` objects.

Examples

```
## Loading a test data set with identified chromatographic peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Subset the dataset to the first and third file.
xod_sub <- filterFile(faahko_sub, file = c(1, 3))

## The number of chromatographic peaks per file for the full object
table(chromPeaks(faahko_sub)[, "sample"])

## The number of chromatographic peaks per file for the subset
table(chromPeaks(xod_sub)[, "sample"])
```

```
basename(fileNames(faahko_sub))
basename(fileNames(xod_sub))

## Filter on mz values; chromatographic peaks and features within the
## mz range are retained (as well as adjusted retention times).
xod_sub <- filterMz(faahko_sub, mz = c(300, 400))
head(chromPeaks(xod_sub))
nrow(chromPeaks(xod_sub))
nrow(chromPeaks(faahko_sub))

## Filter on rt values. All chromatographic peaks and features within the
## retention time range are retained. Filtering is performed by default on
## adjusted retention times, if present.
xod_sub <- filterRt(faahko_sub, rt = c(2700, 2900))

range(rtime(xod_sub))
head(chromPeaks(xod_sub))
range(chromPeaks(xod_sub)[, "rt"])

nrow(chromPeaks(faahko_sub))
nrow(chromPeaks(xod_sub))

## Extract a single Spectrum
faahko_sub[[4]]

## Subsetting using [ removes all preprocessing results - using
## keepAdjustedRtime = TRUE would keep adjusted retention times, if present.
xod_sub <- faahko_sub[fromFile(faahko_sub) == 1]
xod_sub

## Using split does also remove preprocessing results, but it supports the
## optional parameter keepAdjustedRtime.
## Split the object into a list of XCMSnExp objects, one per file
xod_list <- split(faahko_sub, f = fromFile(faahko_sub))
xod_list
```

`FilterIntensityParam` *Remove chromatographic peaks based on intensity*

Description

Remove chromatographic peaks with intensities below the specified threshold. By default, with `nValues = 1`, all peaks with an intensity `>= threshold` are retained. Parameter `value` allows to specify the column of the `chromPeaks()` matrix that should be used for the filtering (defaults to `value = "maxo"` and thus evaluating the maximal intensity for each peak). With `nValues > 1` it is possible to keep only peaks that have `nValues` intensities `>= threshold`. Note that this requires data import from the original MS files and run time of the call can thus be significantly larger. Also, for `nValues > 1` parameter `value` is ignored.

Usage

```
FilterIntensityParam(threshold = 0, nValues = 1L, value = "maxo")

## S4 method for signature 'XCMSnExp,FilterIntensityParam'
refineChromPeaks(
  object,
  param = FilterIntensityParam(),
  msLevel = 1L,
  BPPARAM = bpparam()
)
```

Arguments

`threshold`	`numeric(1)` defining the minimal required intensity for a peak to be retained. Defaults to `threshold = 0`.
`nValues`	`integer(1)` defining the number of data points (per chromatographic peak) that have to be `>= threshold`. Defaults to `nValues = 1`.
`value`	`character(1)` specifying the column in `chromPeaks()` that should be used for the comparison. This is ignored for `nValues > 1`.
`object`	XCMSnExp object with identified chromatographic peaks.
`param`	`FilterIntensityParam` object defining the settings for the method.
`msLevel`	`integer(1)` defining the MS level in which peaks should be filtered.
`BPPARAM`	parameter object to set up parallel processing. Uses the default parallel processing setup returned by `bpparam()`. See `bpparam()` for details and examples.

Value

`XCMSnExp` object with filtererd chromatographic peaks.

Author(s)

Johannes Rainer, Mar Garcia-Aloy

See Also

Other chromatographic peak refinement methods: `CleanPeaksParam`, `MergeNeighboringPeaksParam`

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Remove all peaks with a maximal intensity below 50000
```

```
res <- refineChromPeaks(faahko_sub, param = FilterIntensityParam(threshold = 50000))

nrow(chromPeaks(faahko_sub))
nrow(chromPeaks(res))

all(chromPeaks(res)[, "maxo"] > 50000)

## Keep only chromatographic peaks that have 3 signals above 20000; we
## perform this on the data of a single file.
xdata <- filterFile(faahko_sub)

res <- refineChromPeaks(xdata, FilterIntensityParam(threshold = 20000, nValues = 3))
nrow(chromPeaks(xdata))
nrow(chromPeaks(res))
```

`findChromPeaks,Chromatogram,CentWaveParam-method`

centWave-based peak detection in purely chromatographic data

Description

`findChromPeaks` on a Chromatogram or MChromatograms object with a CentWaveParam parameter object performs centWave-based peak detection on purely chromatographic data. See centWave for details on the method and CentWaveParam for details on the parameter class. Note that not all settings from the `CentWaveParam` will be used. See `peaksWithCentWave()` for the arguments used for peak detection on purely chromatographic data.

After chromatographic peak detection, identified peaks can also be *refined* with the `refineChromPeaks()` method, which can help to reduce peak detection artifacts.

Usage

```
## S4 method for signature 'Chromatogram,CentWaveParam'
findChromPeaks(object, param, ...)

## S4 method for signature 'MChromatograms,CentWaveParam'
findChromPeaks(object, param, BPPARAM = bpparam(), ...)

## S4 method for signature 'MChromatograms,MatchedFilterParam'
findChromPeaks(object, param, BPPARAM = BPPARAM, ...)
```

Arguments

`object`	a Chromatogram or MChromatograms object.
`param`	a CentWaveParam object specifying the settings for the peak detection. See `peaksWithCentWave()` for the description of arguments used for peak detection.
`...`	currently ignored.

`BPPARAM`	a parameter class specifying if and how parallel processing should be performed (only for `XChromatograms` objects). It defaults to `bpparam()`. See `bpparam()` for more information.

Value

If called on a `Chromatogram` object, the method returns an XChromatogram object with the identified peaks. See `peaksWithCentWave()` for details on the peak matrix content.

Author(s)

Johannes Rainer

See Also

`peaksWithCentWave()` for the downstream function and centWave for details on the method.

Examples

```
## Loading a test data set with identified chromatographic peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")
faahko_sub <- filterRt(faahko_sub, c(2500, 3700))

##
od <- as(filterFile(faahko_sub, 1L), "OnDiskMSnExp")

## Extract chromatographic data for a small m/z range
chr <- chromatogram(od, mz = c(272.1, 272.3))[1, 1]

## Identify peaks with default settings
xchr <- findChromPeaks(chr, CentWaveParam())
xchr

## Plot data and identified peaks.
plot(xchr)

## Perform peak detection on an MChromatograms object
od3 <- readMSData(c(system.file("cdf/KO/ko15.CDF", package = "faahKO"),
    system.file("cdf/KO/ko16.CDF", package = "faahKO"),
    system.file("cdf/KO/ko18.CDF", package = "faahKO")),
    mode = "onDisk")

## Disable parallel processing for this example
register(SerialParam())

## Extract chromatograms for a m/z - retention time slice
chrs <- chromatogram(od3, mz = 344, rt = c(2500, 3500))

## Perform peak detection using CentWave
xchrs <- findChromPeaks(chrs, param = CentWaveParam())
```

```
xchrs

## Extract the identified chromatographic peaks
chromPeaks(xchrs)

## plot the result
plot(xchrs)
```

`findChromPeaks,Chromatogram,MatchedFilterParam-method`
matchedFilter-based peak detection in purely chromatographic data

---

Description

`findChromPeaks` on a Chromatogram or MChromatograms object with a MatchedFilterParam parameter object performs matchedFilter-based peak detection on purely chromatographic data. See matchedFilter for details on the method and MatchedFilterParam for details on the parameter class. Note that not all settings from the `MatchedFilterParam` will be used. See `peaksWithMatchedFilter()` for the arguments used for peak detection on purely chromatographic data.

Usage

```
## S4 method for signature 'Chromatogram,MatchedFilterParam'
findChromPeaks(object, param, ...)
```

Arguments

`object`	a Chromatogram or MChromatograms object.
`param`	a MatchedFilterParam object specifying the settings for the peak detection. See `peaksWithMatchedFilter()` for the description of arguments used for peak detection.
`...`	currently ignored.

Value

If called on a `Chromatogram` object, the method returns a `matrix` with the identified peaks. See `peaksWithMatchedFilter()` for details on the matrix content.

Author(s)

Johannes Rainer

See Also

`peaksWithMatchedFilter()` for the downstream function and matchedFilter for details on the method.

Examples

```
## Loading a test data set with identified chromatographic peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")
faahko_sub <- filterRt(faahko_sub, c(2500, 3700))

##
od <- as(filterFile(faahko_sub, 1L), "OnDiskMSnExp")

## Extract chromatographic data for a small m/z range
chr <- chromatogram(od, mz = c(272.1, 272.3))[1, 1]

## Identify peaks with default settings
xchr <- findChromPeaks(chr, MatchedFilterParam())

## Plot the identified peaks
plot(xchr)
```

`findChromPeaks-centWave`

Chromatographic peak detection using the centWave method

Description

The centWave algorithm perform peak density and wavelet based chromatographic peak detection for high resolution LC/MS data in centroid mode [Tautenhahn 2008].

The `CentWaveParam` class allows to specify all settings for a chromatographic peak detection using the centWave method. Instances should be created with the `CentWaveParam` constructor.

The `detectChromPeaks,OnDiskMSnExp,CentWaveParam` method performs chromatographic peak detection using the *centWave* algorithm on all samples from an `OnDiskMSnExp` object. `OnDiskMSnExp` objects encapsule all experiment specific data and load the spectra data (mz and intensity values) on the fly from the original files applying also all eventual data manipulations.

`ppm,ppm<-`: getter and setter for the `ppm` slot of the object.

`peakwidth,peakwidth<-`: getter and setter for the `peakwidth` slot of the object.

`snthresh,snthresh<-`: getter and setter for the `snthresh` slot of the object.

`prefilter,prefilter<-`: getter and setter for the `prefilter` slot of the object.

`mzCenterFun,mzCenterFun<-`: getter and setter for the `mzCenterFun` slot of the object.

`integrate,integrate<-`: getter and setter for the `integrate` slot of the object.

`mzdiff,mzdiff<-`: getter and setter for the `mzdiff` slot of the object.

`fitgauss,fitgauss<-`: getter and setter for the `fitgauss` slot of the object.

`noise,noise<-`: getter and setter for the `noise` slot of the object.

`verboseColumns,verboseColumns<-`: getter and setter for the `verboseColumns` slot of the object.

`roiList`,`roiList<-`: getter and setter for the `roiList` slot of the object.

`fistBaselineCheck`,`firstBaselineCheck<-`: getter and setter for the `firstBaselineCheck` slot of the object.

`roiScales`,`roiScales<-`: getter and setter for the `roiScales` slot of the object.

Usage

```
CentWaveParam(
  ppm = 25,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
  integrate = 1L,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  roiList = list(),
  firstBaselineCheck = TRUE,
  roiScales = numeric(),
  extendLengthMSW = FALSE
)

## S4 method for signature 'OnDiskMSnExp,CentWaveParam'
findChromPeaks(
  object,
  param,
  BPPARAM = bpparam(),
  return.type = "XCMSnExp",
  msLevel = 1L,
  ...
)

## S4 method for signature 'CentWaveParam'
ppm(object)

## S4 replacement method for signature 'CentWaveParam'
ppm(object) <- value

## S4 method for signature 'CentWaveParam'
peakwidth(object)

## S4 replacement method for signature 'CentWaveParam'
peakwidth(object) <- value

## S4 method for signature 'CentWaveParam'
snthresh(object)
```

```
## S4 replacement method for signature 'CentWaveParam'
snthresh(object) <- value

## S4 method for signature 'CentWaveParam'
prefilter(object)

## S4 replacement method for signature 'CentWaveParam'
prefilter(object) <- value

## S4 method for signature 'CentWaveParam'
mzCenterFun(object)

## S4 replacement method for signature 'CentWaveParam'
mzCenterFun(object) <- value

## S4 method for signature 'CentWaveParam'
integrate(f)

## S4 replacement method for signature 'CentWaveParam'
integrate(object) <- value

## S4 method for signature 'CentWaveParam'
mzdiff(object)

## S4 replacement method for signature 'CentWaveParam'
mzdiff(object) <- value

## S4 method for signature 'CentWaveParam'
fitgauss(object)

## S4 replacement method for signature 'CentWaveParam'
fitgauss(object) <- value

## S4 method for signature 'CentWaveParam'
noise(object)

## S4 replacement method for signature 'CentWaveParam'
noise(object) <- value

## S4 method for signature 'CentWaveParam'
verboseColumns(object)

## S4 replacement method for signature 'CentWaveParam'
verboseColumns(object) <- value

## S4 method for signature 'CentWaveParam'
roiList(object)
```

```
## S4 replacement method for signature 'CentWaveParam'
roiList(object) <- value

## S4 method for signature 'CentWaveParam'
firstBaselineCheck(object)

## S4 replacement method for signature 'CentWaveParam'
firstBaselineCheck(object) <- value

## S4 method for signature 'CentWaveParam'
roiScales(object)

## S4 replacement method for signature 'CentWaveParam'
roiScales(object) <- value
```

Arguments

`ppm`	`numeric(1)` defining the maximal tolerated m/z deviation in consecutive scans in parts per million (ppm) for the initial ROI definition.
`peakwidth`	`numeric(2)` with the expected approximate peak width in chromatographic space. Given as a range (min, max) in seconds.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`prefilter`	`numeric(2)`: `c(k, I)` specifying the prefilter step for the first analysis step (ROI detection). Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Name of the function to calculate the m/z center of the chromatographic peak. Allowed are: `"wMean"`: intensity weighted mean of the peak's m/z values, `"mean"`: mean of the peak's m/z values, `"apex"`: use the m/z value at the peak apex, `"wMeanApex3"`: intensity weighted mean of the m/z value at the peak apex and the m/z values left and right of it and `"meanApex3"`: mean of the m/z value of the peak apex and the m/z values left and right of it.
`integrate`	Integration method. For `integrate = 1` peak limits are found through descent on the mexican hat filtered data, for `integrate = 2` the descent is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak. This affects mostly the retention time position of the peak.
`noise`	`numeric(1)` allowing to set a minimum intensity required for centroids to be considered in the first analysis step (centroids with intensity < `noise` are omitted from ROI detection).
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.

`roiList`	An optional list of regions-of-interest (ROI) representing detected mass traces. If ROIs are submitted the first analysis step is omitted and chromatographic peak detection is performed on the submitted ROIs. Each ROI is expected to have the following elements specified: `scmin` (start scan index), `scmax` (end scan index), `mzmin` (minimum m/z), `mzmax` (maximum m/z), `length` (number of scans), `intensity` (summed intensity). Each ROI should be represented by a `list` of elements or a single row `data.frame`.
`firstBaselineCheck`	`logical(1)`. If `TRUE` continuous data within regions of interest is checked to be above the first baseline.
`roiScales`	Optional numeric vector with length equal to `roiList` defining the scale for each region of interest in `roiList` that should be used for the centWave-wavelets.
`extendLengthMSW`	Option to force centWave to use all scales when running centWave rather than truncating with the EIC length. Uses the "open" method to extend the EIC to a integer base-2 length prior to being passed to `convolve` rather than the default "reflect" method. See https://github.com/sneumann/xcms/issues/445 for more information.
`object`	For `findChromPeaks`: an `OnDiskMSnExp` object containing the MS- and all other experiment-relevant data. For all other methods: a parameter object.
`param`	An `CentWaveParam` object containing all settings for the centWave algorithm.
`BPPARAM`	A parameter class specifying if and how parallel processing should be performed. It defaults to `bpparam`. See documentation of the `BiocParallel` for more details. If parallel processing is enabled, peak detection is performed in parallel on several of the input samples.
`return.type`	Character specifying what type of object the method should return. Can be either `"XCMSnExp"` (default), `"list"` or `"xcmsSet"`.
`msLevel`	`integer(1)` defining the MS level on which the peak detection should be performed. Defaults to `msLevel = 1`.
`...`	ignored.
`value`	The value for the slot.
`f`	For `integrate`: a `CentWaveParam` object.

Details

The centWave algorithm is most suitable for high resolution LC/{TOF,OrbiTrap,FTICR}-MS data in centroid mode. In the first phase the method identifies *regions of interest* (ROIs) representing mass traces that are characterized as regions with less than `ppm` m/z deviation in consecutive scans in the LC/MS map. In detail, starting with a single m/z, a ROI is extended if a m/z can be found in the next scan (spectrum) for which the difference to the mean m/z of the ROI is smaller than the user defined `ppm` of the m/z. The mean m/z of the ROI is then updated considering also the newly included m/z value.

These ROIs are then, after some cleanup, analyzed using continuous wavelet transform (CWT) to locate chromatographic peaks on different scales. The first analysis step is skipped, if regions of interest are passed *via* the `param` parameter.

Parallel processing (one process per sample) is supported and can be configured either by the `BPPARAM` parameter or by globally defining the parallel processing mode using the `register` method from the `BiocParallel` package.

Value

The `CentWaveParam` function returns a `CentWaveParam` class instance with all of the settings specified for chromatographic peak detection by the centWave method.

For `findChromPeaks`: if `return.type = "XCMSnExp"` an `XCMSnExp` object with the results of the peak detection. If `return.type = "list"` a list of length equal to the number of samples with matrices specifying the identified peaks. If `return.type = "xcmsSet"` an `xcmsSet` object with the results of the peak detection.

Slots

`ppm,peakwidth,snthresh,prefilter,mzCenterFun,integrate,mzdiff,fitgauss,noise,verboseColumns,roiList`
See corresponding parameter above. Slots values should exclusively be accessed *via* the corresponding getter and setter methods listed above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `findPeaks` methods. It supports peak detection on `OnDiskMSnExp` objects (defined in the `MSnbase` package). All of the settings to the centWave algorithm can be passed with a `CentWaveParam` object.

Author(s)

Ralf Tautenhahn, Johannes Rainer

References

Ralf Tautenhahn, Christoph B\"ottcher, and Steffen Neumann "Highly sensitive feature detection for high resolution LC/MS" *BMC Bioinformatics* 2008, 9:504

See Also

The `do_findChromPeaks_centWave` core API function and `findPeaks.centWave` for the old user interface.

`peaksWithCentWave` for functions to perform centWave peak detection in purely chromatographic data.

`XCMSnExp` for the object containing the results of the peak detection.

Other peak detection methods: `chromatographic-peak-detection`, `findChromPeaks-centWaveWithPredIsoROIs`, `findChromPeaks-massifquant`, `findChromPeaks-matchedFilter`, `findPeaks-MSW`

Examples

```
## Create a CentWaveParam object. Note that the noise is set to 10000 to
## speed up the execution of the example - in a real use case the default
## value should be used, or it should be set to a reasonable value.
cwp <- CentWaveParam(ppm = 20, noise = 10000, prefilter = c(3, 10000))
## Change snthresh parameter
snthresh(cwp) <- 25
cwp

## Perform the peak detection using centWave on some of the files from the
## faahKO package. Files are read using the readMSData from the MSnbase
## package
library(faahKO)
library(xcms)
fls <- dir(system.file("cdf/KO", package = "faahKO"), recursive = TRUE,
           full.names = TRUE)
raw_data <- readMSData(fls[1], mode = "onDisk")

## Perform the peak detection using the settings defined above.
res <- findChromPeaks(raw_data, param = cwp)
head(chromPeaks(res))
```

`findChromPeaks-centWaveWithPredIsoROIs`

Two-step centWave peak detection considering also isotopes

Description

This method performs a two-step centWave-based chromatographic peak detection: in a first centWave run peaks are identified for which then the location of their potential isotopes in the mz-retention time is predicted. A second centWave run is then performed on these *regions of interest* (ROIs). The final list of chromatographic peaks comprises all non-overlapping peaks from both centWave runs.

The `CentWavePredIsoParam` class allows to specify all settings for the two-step centWave-based peak detection considering also predicted isotopes of peaks identified in the first centWave run. Instances should be created with the `CentWavePredIsoParam` constructor. See also the documentation of the `CentWaveParam` for all methods and arguments this class inherits.

The `findChromPeaks,OnDiskMSnExp,CentWavePredIsoParam` method performs a two-step centWave-based chromatographic peak detection on all samples from an `OnDiskMSnExp` object. `OnDiskMSnExp` objects encapsule all experiment specific data and load the spectra data (mz and intensity values) on the fly from the original files applying also all eventual data manipulations.

`snthreshIsoROIs,snthreshIsoROIs<-`: getter and setter for the `snthreshIsoROIs` slot of the object.

`maxCharge,maxCharge<-`: getter and setter for the `maxCharge` slot of the object.

`maxIso,maxIso<-`: getter and setter for the `maxIso` slot of the object.

`mzIntervalExtension`,`mzIntervalExtension<-`: getter and setter for the `mzIntervalExtension` slot of the object.

`polarity`,`polarity<-`: getter and setter for the `polarity` slot of the object.

Usage

```
CentWavePredIsoParam(
  ppm = 25,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
  integrate = 1L,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  roiList = list(),
  firstBaselineCheck = TRUE,
  roiScales = numeric(),
  snthreshIsoROIs = 6.25,
  maxCharge = 3,
  maxIso = 5,
  mzIntervalExtension = TRUE,
  polarity = "unknown"
)

## S4 method for signature 'OnDiskMSnExp,CentWavePredIsoParam'
findChromPeaks(
  object,
  param,
  BPPARAM = bpparam(),
  return.type = "XCMSnExp",
  msLevel = 1L,
  ...
)

## S4 method for signature 'CentWavePredIsoParam'
snthreshIsoROIs(object)

## S4 replacement method for signature 'CentWavePredIsoParam'
snthreshIsoROIs(object) <- value

## S4 method for signature 'CentWavePredIsoParam'
maxCharge(object)

## S4 replacement method for signature 'CentWavePredIsoParam'
maxCharge(object) <- value
```

```
## S4 method for signature 'CentWavePredIsoParam'
maxIso(object)

## S4 replacement method for signature 'CentWavePredIsoParam'
maxIso(object) <- value

## S4 method for signature 'CentWavePredIsoParam'
mzIntervalExtension(object)

## S4 replacement method for signature 'CentWavePredIsoParam'
mzIntervalExtension(object) <- value

## S4 method for signature 'CentWavePredIsoParam'
polarity(object)

## S4 replacement method for signature 'CentWavePredIsoParam'
polarity(object) <- value
```

Arguments

`ppm`	`numeric(1)` defining the maximal tolerated m/z deviation in consecutive scans in parts per million (ppm) for the initial ROI definition.
`peakwidth`	`numeric(2)` with the expected approximate peak width in chromatographic space. Given as a range (min, max) in seconds.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`prefilter`	`numeric(2)`: `c(k, I)` specifying the prefilter step for the first analysis step (ROI detection). Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Name of the function to calculate the m/z center of the chromatographic peak. Allowed are: `"wMean"`: intensity weighted mean of the peak's m/z values, `"mean"`: mean of the peak's m/z values, `"apex"`: use the m/z value at the peak apex, `"wMeanApex3"`: intensity weighted mean of the m/z value at the peak apex and the m/z values left and right of it and `"meanApex3"`: mean of the m/z value of the peak apex and the m/z values left and right of it.
`integrate`	Integration method. For `integrate = 1` peak limits are found through descent on the mexican hat filtered data, for `integrate = 2` the descent is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak. This affects mostly the retention time position of the peak.

`noise`	`numeric(1)` allowing to set a minimum intensity required for centroids to be considered in the first analysis step (centroids with intensity < `noise` are omitted from ROI detection).
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`roiList`	An optional list of regions-of-interest (ROI) representing detected mass traces. If ROIs are submitted the first analysis step is omitted and chromatographic peak detection is performed on the submitted ROIs. Each ROI is expected to have the following elements specified: `scmin` (start scan index), `scmax` (end scan index), `mzmin` (minimum m/z), `mzmax` (maximum m/z), `length` (number of scans), `intensity` (summed intensity). Each ROI should be represented by a `list` of elements or a single row `data.frame`.
`firstBaselineCheck`	`logical(1)`. If `TRUE` continuous data within regions of interest is checked to be above the first baseline.
`roiScales`	Optional numeric vector with length equal to `roiList` defining the scale for each region of interest in `roiList` that should be used for the centWave-wavelets.
`snthreshIsoROIs`	`numeric(1)` defining the signal to noise ratio cutoff to be used in the second centWave run to identify peaks for predicted isotope ROIs.
`maxCharge`	`integer(1)` defining the maximal isotope charge. Isotopes will be defined for charges `1:maxCharge`.
`maxIso`	`integer(1)` defining the number of isotope peaks that should be predicted for each peak identified in the first centWave run.
`mzIntervalExtension`	`logical(1)` whether the mz range for the predicted isotope ROIs should be extended to increase detection of low intensity peaks.
`polarity`	`character(1)` specifying the polarity of the data. Currently not used, but has to be `"positive"`, `"negative"` or `"unknown"` if provided.
`object`	For `findChromPeaks`: an `OnDiskMSnExp` object containing the MS- and all other experiment-relevant data. For all other methods: a parameter object.
`param`	An `CentWavePredIsoParam` object with the settings for the chromatographic peak detection algorithm.
`BPPARAM`	A parameter class specifying if and how parallel processing should be performed. It defaults to `bpparam`. See documentation of the `BiocParallel` for more details. If parallel processing is enabled, peak detection is performed in parallel on several of the input samples.
`return.type`	Character specifying what type of object the method should return. Can be either `"XCMSnExp"` (default), `"list"` or `"xcmsSet"`.
`msLevel`	`integer(1)` defining the MS level on which the peak detection should be performed. Defaults to `msLevel = 1`.
`...`	ignored.
`value`	The value for the slot.

Details

See `centWave` for details on the centWave method.

Parallel processing (one process per sample) is supported and can be configured either by the `BPPARAM` parameter or by globally defining the parallel processing mode using the `register` method from the `BiocParallel` package.

Value

The `CentWavePredIsoParam` function returns a `CentWavePredIsoParam` class instance with all of the settings specified for the two-step centWave-based peak detection considering also isotopes.

For `findChromPeaks`: if `return.type = "XCMSnExp"` an `XCMSnExp` object with the results of the peak detection. If `return.type = "list"` a list of length equal to the number of samples with matrices specifying the identified peaks. If `return.type = "xcmsSet"` an `xcmsSet` object with the results of the peak detection.

Slots

`ppm,peakwidth,snthresh,prefilter,mzCenterFun,integrate,mzdiff,fitgauss,noise,verboseColumns,roiList`
See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `findPeaks` methods. It supports chromatographic peak detection on `OnDiskMSnExp` objects (defined in the `MSnbase` package). All of the settings to the algorithm can be passed with a `CentWavePredIsoParam` object.

Author(s)

Hendrik Treutler, Johannes Rainer

See Also

The `do_findChromPeaks_centWaveWithPredIsoROIs` core API function and `findPeaks.centWave` for the old user interface. `CentWaveParam` for the class the `CentWavePredIsoParam` extends.

`XCMSnExp` for the object containing the results of the peak detection.

Other peak detection methods: `chromatographic-peak-detection`, `findChromPeaks-centWave`, `findChromPeaks-massifquant`, `findChromPeaks-matchedFilter`, `findPeaks-MSW`

Examples

```
## Create a param object
p <- CentWavePredIsoParam(maxCharge = 4)
## Change snthresh parameter
snthresh(p) <- 25
p
```

`findChromPeaks-massifquant`

Chromatographic peak detection using the massifquant method

Description

Massifquant is a Kalman filter (KF)-based chromatographic peak detection for XC-MS data in centroid mode. The identified peaks can be further refined with the *centWave* method (see `findChromPeaks-centWave` for details on centWave) by specifying `withWave = TRUE`.

The `MassifquantParam` class allows to specify all settings for a chromatographic peak detection using the massifquant method eventually in combination with the centWave algorithm. Instances should be created with the `MassifquantParam` constructor.

The `findChromPeaks,OnDiskMSnExp,MassifquantParam` method performs chromatographic peak detection using the *massifquant* algorithm on all samples from an `OnDiskMSnExp` object. `OnDiskMSnExp` objects encapsule all experiment specific data and load the spectra data (mz and intensity values) on the fly from the original files applying also all eventual data manipulations.

`ppm,ppm<-`: getter and setter for the `ppm` slot of the object.

`peakwidth,peakwidth<-`: getter and setter for the `peakwidth` slot of the object.

`snthresh,snthresh<-`: getter and setter for the `snthresh` slot of the object.

`prefilter,prefilter<-`: getter and setter for the `prefilter` slot of the object.

`mzCenterFun,mzCenterFun<-`: getter and setter for the `mzCenterFun` slot of the object.

`integrate,integrate<-`: getter and setter for the `integrate` slot of the object.

`mzdiff,mzdiff<-`: getter and setter for the `mzdiff` slot of the object.

`fitgauss,fitgauss<-`: getter and setter for the `fitgauss` slot of the object.

`noise,noise<-`: getter and setter for the `noise` slot of the object.

`verboseColumns,verboseColumns<-`: getter and setter for the `verboseColumns` slot of the object.

`criticalValue,criticalValue<-`: getter and setter for the `criticalValue` slot of the object.

`consecMissedLimit,consecMissedLimit<-`: getter and setter for the `consecMissedLimit` slot of the object.

`unions,unions<-`: getter and setter for the `unions` slot of the object.

`checkBack,checkBack<-`: getter and setter for the `checkBack` slot of the object.

`withWave,withWave<-`: getter and setter for the `withWave` slot of the object.

Usage

```
MassifquantParam(
  ppm = 25,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  mzCenterFun = "wMean",
```

```
  integrate = 1L,
  mzdiff = -0.001,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  criticalValue = 1.125,
  consecMissedLimit = 2,
  unions = 1,
  checkBack = 0,
  withWave = FALSE
)

## S4 method for signature 'OnDiskMSnExp,MassifquantParam'
findChromPeaks(
  object,
  param,
  BPPARAM = bpparam(),
  return.type = "XCMSnExp",
  msLevel = 1L,
  ...
)

## S4 method for signature 'MassifquantParam'
ppm(object)

## S4 replacement method for signature 'MassifquantParam'
ppm(object) <- value

## S4 method for signature 'MassifquantParam'
peakwidth(object)

## S4 replacement method for signature 'MassifquantParam'
peakwidth(object) <- value

## S4 method for signature 'MassifquantParam'
snthresh(object)

## S4 replacement method for signature 'MassifquantParam'
snthresh(object) <- value

## S4 method for signature 'MassifquantParam'
prefilter(object)

## S4 replacement method for signature 'MassifquantParam'
prefilter(object) <- value

## S4 method for signature 'MassifquantParam'
mzCenterFun(object)
```

```
## S4 replacement method for signature 'MassifquantParam'
mzCenterFun(object) <- value

## S4 method for signature 'MassifquantParam'
integrate(f)

## S4 replacement method for signature 'MassifquantParam'
integrate(object) <- value

## S4 method for signature 'MassifquantParam'
mzdiff(object)

## S4 replacement method for signature 'MassifquantParam'
mzdiff(object) <- value

## S4 method for signature 'MassifquantParam'
fitgauss(object)

## S4 replacement method for signature 'MassifquantParam'
fitgauss(object) <- value

## S4 method for signature 'MassifquantParam'
noise(object)

## S4 replacement method for signature 'MassifquantParam'
noise(object) <- value

## S4 method for signature 'MassifquantParam'
verboseColumns(object)

## S4 replacement method for signature 'MassifquantParam'
verboseColumns(object) <- value

## S4 method for signature 'MassifquantParam'
criticalValue(object)

## S4 replacement method for signature 'MassifquantParam'
criticalValue(object) <- value

## S4 method for signature 'MassifquantParam'
consecMissedLimit(object)

## S4 replacement method for signature 'MassifquantParam'
consecMissedLimit(object) <- value

## S4 method for signature 'MassifquantParam'
unions(object)
```

```
## S4 replacement method for signature 'MassifquantParam'
unions(object) <- value

## S4 method for signature 'MassifquantParam'
checkBack(object)

## S4 replacement method for signature 'MassifquantParam'
checkBack(object) <- value

## S4 method for signature 'MassifquantParam'
withWave(object)

## S4 replacement method for signature 'MassifquantParam'
withWave(object) <- value
```

Arguments

`ppm`	`numeric(1)` defining the maximal tolerated m/z deviation in consecutive scans in parts per million (ppm) for the initial ROI definition.
`peakwidth`	`numeric(2)`. Only the first element is used by massifquant, which specifices the minimum peak length in time scans. For `withWave = TRUE` the second argument represents the maximum peak length subject to being greater than the mininum peak length (see also documentation of `do_findChromPeaks_centWave`).
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`prefilter`	`numeric(2)`. The first argument is only used if (`withWave = TRUE`); see `findChromPeaks-centWave` for details. The second argument specifies the minimum threshold for the maximum intensity of a chromatographic peak that must be met.
`mzCenterFun`	Name of the function to calculate the m/z center of the chromatographic peak. Allowed are: `"wMean"`: intensity weighted mean of the peak's m/z values, `"mean"`: mean of the peak's m/z values, `"apex"`: use the m/z value at the peak apex, `"wMeanApex3"`: intensity weighted mean of the m/z value at the peak apex and the m/z values left and right of it and `"meanApex3"`: mean of the m/z value of the peak apex and the m/z values left and right of it.
`integrate`	Integration method. For `integrate = 1` peak limits are found through descent on the mexican hat filtered data, for `integrate = 2` the descent is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`mzdiff`	`numeric(1)` representing the minimum difference in m/z dimension required for peaks with overlapping retention times; can be negative to allow overlap. During peak post-processing, peaks defined to be overlapping are reduced to the one peak with the largest signal.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak. This affects mostly the retention time position of the peak.
`noise`	`numeric(1)` allowing to set a minimum intensity required for centroids to be considered in the first analysis step (centroids with intensity < `noise` are omitted from ROI detection).

`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`criticalValue`	`numeric(1)`. Suggested values: (`0.1-3.0`). This setting helps determine the the Kalman Filter prediciton margin of error. A real centroid belonging to a bonafide peak must fall within the KF prediction margin of error. Much like in the construction of a confidence interval, `criticalVal` loosely translates to be a multiplier of the standard error of the prediction reported by the Kalman Filter. If the peak in the XC-MS sample have a small mass deviance in ppm error, a smaller critical value might be better and vice versa.
`consecMissedLimit`	`integer(1)` Suggested values: (`1,2,3`). While a peak is in the proces of being detected by a Kalman Filter, the Kalman Filter may not find a predicted centroid in every scan. After 1 or more consecutive failed predictions, this setting informs Massifquant when to stop a Kalman Filter from following a candidate peak.
`unions`	`integer(1)` set to `1` if apply t-test union on segmentation; set to `0` if no t-test to be applied on chromatographically continous peaks sharing same m/z range. Explanation: With very few data points, sometimes a Kalman Filter stops tracking a peak prematurely. Another Kalman Filter is instantiated and begins following the rest of the signal. Because tracking is done backwards to forwards, this algorithmic defect leaves a real peak divided into two segments or more. With this option turned on, the program identifies segmented peaks and combines them (merges them) into one with a two sample t-test. The potential danger of this option is that some truly distinct peaks may be merged.
`checkBack`	`integer(1)` set to `1` if turned on; set to `0` if turned off. The convergence of a Kalman Filter to a peak's precise m/z mapping is very fast, but sometimes it incorporates erroneous centroids as part of a peak (especially early on). The `scanBack` option is an attempt to remove the occasional outlier that lies beyond the converged bounds of the Kalman Filter. The option does not directly affect identification of a peak because it is a postprocessing measure; it has not shown to be a extremely useful thus far and the default is set to being turned off.
`withWave`	`logical(1)` if `TRUE`, the peaks identified first with Massifquant are subsequently filtered with the second step of the centWave algorithm, which includes wavelet estimation.
`object`	For `findChromPeaks`: an `OnDiskMSnExp` object containing the MS- and all other experiment-relevant data. For all other methods: a parameter object.
`param`	An `MassifquantParam` object containing all settings for the massifquant algorithm.
`BPPARAM`	A parameter class specifying if and how parallel processing should be performed. It defaults to `bpparam`. See documentation of the `BiocParallel` for more details. If parallel processing is enabled, peak detection is performed in parallel on several of the input samples.
`return.type`	Character specifying what type of object the method should return. Can be either `"XCMSnExp"` (default), `"list"` or `"xcmsSet"`.
`msLevel`	`integer(1)` defining the MS level on which the peak detection should be performed. Defaults to `msLevel = 1`.

`...`	ignored.
`value`	The value for the slot.
`f`	For `integrate`: a `MassifquantParam` object.

Details

This algorithm's performance has been tested rigorously on high resolution LC/OrbiTrap, TOF-MS data in centroid mode. Simultaneous kalman filters identify chromatographic peaks and calculate their area under the curve. The default parameters are set to operate on a complex LC-MS Orbitrap sample. Users will find it useful to do some simple exploratory data analysis to find out where to set a minimum intensity, and identify how many scans an average peak spans. The `consecMissedLimit` parameter has yielded good performance on Orbitrap data when set to (`2`) and on TOF data it was found best to be at (`1`). This may change as the algorithm has yet to be tested on many samples. The `criticalValue` parameter is perhaps most dificult to dial in appropriately and visual inspection of peak identification is the best suggested tool for quick optimization. The `ppm` and `checkBack` parameters have shown less influence than the other parameters and exist to give users flexibility and better accuracy.

Parallel processing (one process per sample) is supported and can be configured either by the `BPPARAM` parameter or by globally defining the parallel processing mode using the `register` method from the `BiocParallel` package.

Value

The `MassifquantParam` function returns a `MassifquantParam` class instance with all of the settings specified for chromatographic peak detection by the *massifquant* method.

For `findChromPeaks`: if `return.type = "XCMSnExp"` an `XCMSnExp` object with the results of the peak detection. If `return.type = "list"` a list of length equal to the number of samples with matrices specifying the identified peaks. If `return.type = "xcmsSet"` an `xcmsSet` object with the results of the peak detection.

Slots

`ppm,peakwidth,snthresh,prefilter,mzCenterFun,integrate,mzdiff,fitgauss,noise,verboseColumns,critica`
: See corresponding parameter above. Slots values should exclusively be accessed *via* the corresponding getter and setter methods listed above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `findPeaks` methods. It supports chromatographic peak detection on `OnDiskMSnExp` objects (defined in the `MSnbase` package). All of the settings to the massifquant and centWave algorithm can be passed with a `MassifquantParam` object.

Author(s)

Christopher Conley, Johannes Rainer

References

Conley CJ, Smith R, Torgrip RJ, Taylor RM, Tautenhahn R and Prince JT "Massifquant: open-source Kalman filter-based XC-MS isotope trace feature detection" *Bioinformatics* 2014, 30(18):2636-43.

See Also

The `do_findChromPeaks_massifquant` core API function and `findPeaks.massifquant` for the old user interface.

`XCMSnExp` for the object containing the results of the peak detection.

Other peak detection methods: `chromatographic-peak-detection`, `findChromPeaks-centWaveWithPredIsoROIs`, `findChromPeaks-centWave`, `findChromPeaks-matchedFilter`, `findPeaks-MSW`

Examples

```
## Create a MassifquantParam object.
mqp <- MassifquantParam()
## Change snthresh prefilter parameters
snthresh(mqp) <- 30
prefilter(mqp) <- c(6, 10000)
mqp

## Perform the peak detection using massifquant on the files from the
## faahKO package. Files are read using the readMSData from the MSnbase
## package
library(faahKO)
library(MSnbase)
fls <- dir(system.file("cdf/KO", package = "faahKO"), recursive = TRUE,
           full.names = TRUE)
raw_data <- readMSData(fls[1], mode = "onDisk")
## Perform the peak detection using the settings defined above.
res <- findChromPeaks(raw_data, param = mqp)
head(chromPeaks(res))
```

`findChromPeaks-matchedFilter`

Peak detection in the chromatographic time domain

Description

The *matchedFilter* algorithm identifies peaks in the chromatographic time domain as described in [Smith 2006]. The intensity values are binned by cutting The LC/MS data into slices (bins) of a mass unit (`binSize` m/z) wide. Within each bin the maximal intensity is selected. The chromatographic peak detection is then performed in each bin by extending it based on the `steps` parameter to generate slices comprising bins `current_bin - steps +1` to `current_bin + steps - 1`. Each of these slices is then filtered with matched filtration using a second-derative Gaussian as the model peak shape. After filtration peaks are detected using a signal-to-ratio cut-off. For more details and illustrations see [Smith 2006].

The `MatchedFilterParam` class allows to specify all settings for a chromatographic peak detection using the matchedFilter method. Instances should be created with the `MatchedFilterParam` constructor.

The `findChromPeaks,OnDiskMSnExp,MatchedFilterParam` method performs peak detection using the *matchedFilter* algorithm on all samples from an `OnDiskMSnExp` object. `OnDiskMSnExp` objects encapsule all experiment specific data and load the spectra data (mz and intensity values) on the fly from the original files applying also all eventual data manipulations.

`binSize`,`binSize<-`: getter and setter for the `binSize` slot of the object.

`impute`,`impute<-`: getter and setter for the `impute` slot of the object.

`baseValue`,`baseValue<-`: getter and setter for the `baseValue` slot of the object.

`distance`,`distance<-`: getter and setter for the `distance` slot of the object.

`fwhm`,`fwhm<-`: getter and setter for the `fwhm` slot of the object.

`sigma`,`sigma<-`: getter and setter for the `sigma` slot of the object.

`max`,`max<-`: getter and setter for the `max` slot of the object.

`snthresh`,`snthresh<-`: getter and setter for the `snthresh` slot of the object.

`steps`,`steps<-`: getter and setter for the `steps` slot of the object.

`mzdiff`,`mzdiff<-`: getter and setter for the `mzdiff` slot of the object.

`index`,`index<-`: getter and setter for the `index` slot of the object.

Usage

```
MatchedFilterParam(
  binSize = 0.1,
  impute = "none",
  baseValue = numeric(),
  distance = numeric(),
  fwhm = 30,
  sigma = fwhm/2.3548,
  max = 5,
  snthresh = 10,
  steps = 2,
  mzdiff = 0.8 - binSize * steps,
  index = FALSE
)

## S4 method for signature 'OnDiskMSnExp,MatchedFilterParam'
findChromPeaks(
  object,
  param,
  BPPARAM = bpparam(),
  return.type = "XCMSnExp",
  msLevel = 1L,
  ...
)
```

```
## S4 method for signature 'MatchedFilterParam'
binSize(object)

## S4 replacement method for signature 'MatchedFilterParam'
binSize(object) <- value

## S4 method for signature 'MatchedFilterParam'
impute(object)

## S4 replacement method for signature 'MatchedFilterParam'
impute(object) <- value

## S4 method for signature 'MatchedFilterParam'
baseValue(object)

## S4 replacement method for signature 'MatchedFilterParam'
baseValue(object) <- value

## S4 method for signature 'MatchedFilterParam'
distance(object)

## S4 replacement method for signature 'MatchedFilterParam'
distance(object) <- value

## S4 method for signature 'MatchedFilterParam'
fwhm(object)

## S4 replacement method for signature 'MatchedFilterParam'
fwhm(object) <- value

## S4 method for signature 'MatchedFilterParam'
sigma(object)

## S4 replacement method for signature 'MatchedFilterParam'
sigma(object) <- value

## S4 method for signature 'MatchedFilterParam'
max(x)

## S4 replacement method for signature 'MatchedFilterParam'
max(object) <- value

## S4 method for signature 'MatchedFilterParam'
snthresh(object)

## S4 replacement method for signature 'MatchedFilterParam'
snthresh(object) <- value
```

```
## S4 method for signature 'MatchedFilterParam'
steps(object)

## S4 replacement method for signature 'MatchedFilterParam'
steps(object) <- value

## S4 method for signature 'MatchedFilterParam'
mzdiff(object)

## S4 replacement method for signature 'MatchedFilterParam'
mzdiff(object) <- value

## S4 method for signature 'MatchedFilterParam'
index(object)

## S4 replacement method for signature 'MatchedFilterParam'
index(object) <- value
```

Arguments

`binSize`	`numeric(1)` specifying the width of the bins/slices in m/z dimension.
`impute`	Character string specifying the method to be used for missing value imputation. Allowed values are `"none"` (no linear interpolation), `"lin"` (linear interpolation), `"linbase"` (linear interpolation within a certain bin-neighborhood) and `"intlin"`. See `imputeLinInterpol` for more details.
`baseValue`	The base value to which empty elements should be set. This is only considered for `method = "linbase"` and corresponds to the `profBinLinBase`'s `baselevel` argument.
`distance`	For `method = "linbase"`: number of non-empty neighboring element of an empty element that should be considered for linear interpolation. See details section for more information.
`fwhm`	`numeric(1)` specifying the full width at half maximum of matched filtration gaussian model peak. Only used to calculate the actual sigma, see below.
`sigma`	`numeric(1)` specifying the standard deviation (width) of the matched filtration model peak.
`max`	`numeric(1)` representing the maximum number of peaks that are expected/will be identified per slice.
`snthresh`	`numeric(1)` defining the signal to noise cutoff to be used in the chromatographic peak detection step.
`steps`	`numeric(1)` defining the number of bins to be merged before filtration (i.e. the number of neighboring bins that will be joined to the slice in which filtration and peak detection will be performed).
`mzdiff`	`numeric(1)` defining the minimum difference in m/z for peaks with overlapping retention times
`index`	`logical(1)` specifying whether indicies should be returned instead of values for m/z and retention times.

`object`	For `findChromPeaks`: an `OnDiskMSnExp` object containing the MS- and all other experiment-relevant data. For all other methods: a parameter object.
`param`	An `MatchedFilterParam` object containing all settings for the matchedFilter algorithm.
`BPPARAM`	A parameter class specifying if and how parallel processing should be performed. It defaults to `bpparam`. See documentation of the `BiocParallel` for more details. If parallel processing is enabled, peak detection is performed in parallel on several of the input samples.
`return.type`	Character specifying what type of object the method should return. Can be either `"XCMSnExp"` (default), `"list"` or `"xcmsSet"`.
`msLevel`	`integer(1)` defining the MS level on which the peak detection should be performed. Defaults to `msLevel = 1`.
`...`	ignored.
`value`	The value for the slot.
`x`	For `max`: a `MatchedFilterParam` object.

Details

The intensities are binned by the provided m/z values within each spectrum (scan). Binning is performed such that the bins are centered around the m/z values (i.e. the first bin includes all m/z values between `min(mz) - bin_size/2` and `min(mz) + bin_size/2`).

For more details on binning and missing value imputation see `binYonX` and `imputeLinInterpol` methods.

Parallel processing (one process per sample) is supported and can be configured either by the `BPPARAM` parameter or by globally defining the parallel processing mode using the `register` method from the `BiocParallel` package.

Value

The `MatchedFilterParam` function returns a `MatchedFilterParam` class instance with all of the settings specified for chromatographic detection by the *matchedFilter* method.

For `findChromPeaks`: if `return.type = "XCMSnExp"` an `XCMSnExp` object with the results of the peak detection. If `return.type = "list"` a list of length equal to the number of samples with matrices specifying the identified peaks. If `return.type = "xcmsSet"` an `xcmsSet` object with the results of the peak detection.

Slots

`binSize,impute,baseValue,distance,fwhm,sigma,max,snthresh,steps,mzdiff,index` See corresponding parameter above. Slots values should exclusively be accessed *via* the corresponding getter and setter methods listed above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `findPeaks` methods. It supports chromatographic peak detection on `OnDiskMSnExp` objects (defined in the `MSnbase` package). All of the settings to the matchedFilter algorithm can be passed with a `MatchedFilterParam` object.

Author(s)

Colin A Smith, Johannes Rainer

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" *Anal. Chem.* 2006, 78:779-787.

See Also

The `do_findChromPeaks_matchedFilter` core API function and `findPeaks.matchedFilter` for the old user interface.

`peaksWithMatchedFilter` for functions to perform matchedFilter peak detection in purely chromatographic data.

`XCMSnExp` for the object containing the results of the chromatographic peak detection.

Other peak detection methods: `chromatographic-peak-detection`, `findChromPeaks-centWaveWithPredIsoROIs`, `findChromPeaks-centWave`, `findChromPeaks-massifquant`, `findPeaks-MSW`

Examples

```
## Create a MatchedFilterParam object. Note that we use a unnecessarily large
## binSize parameter to reduce the run-time of the example.
mfp <- MatchedFilterParam(binSize = 5)
## Change snthresh parameter
snthresh(mfp) <- 15
mfp

## Perform the peak detection using matchecFilter on the files from the
## faahKO package. Files are read using the readMSData from the MSnbase
## package
library(faahKO)
library(MSnbase)
fls <- dir(system.file("cdf/KO", package = "faahKO"), recursive = TRUE,
           full.names = TRUE)
raw_data <- readMSData(fls[1], mode = "onDisk")
## Perform the chromatographic peak detection using the settings defined
## above. Note that we are also disabling parallel processing in this
## example by registering a "SerialParam"
res <- findChromPeaks(raw_data, param = mfp)
head(chromPeaks(res))
```

`findChromPeaksIsolationWindow`

Data independent acquisition (DIA): peak detection in isolation windows

---

Description

The `findChromPeaksIsolationWindow` function allows to perform a chromatographic peak detection in MS level > 1 spectra of certain isolation windows (e.g. SWATH pockets). The function performs a peak detection, separately for all spectra belonging to the same isolation window and adds them to the `chromPeaks()` matrix of the result object, information about the isolation window they were detected in is added to `chromPeakData()`. Note that peak detection with this method does not remove previously identified chromatographic peaks (e.g. on MS1 level using the `findChromPeaks()` function but adds newly identified peaks to the existing `chromPeaks()` matrix.

Isolation windows can be defined with the `isolationWindow` parameter, that by default uses the definition of `isolationWindowTargetMz()`, i.e. chromatographic peak detection is performed for all spectra with the same isolation window target m/z (seprarately for each file). The parameter `param` allows to define and configure the peak detection algorithm (see `findChromPeaks()` for more information).

Usage

```
findChromPeaksIsolationWindow(
  object,
  param,
  msLevel = 2L,
  isolationWindow = isolationWindowTargetMz(object),
  ...
)
```

Arguments

`object`	`OnDiskMSnExp` or `XCMSnExp` object with the DIA data.
`param`	Peak detection parameter object, such as a CentWaveParam object defining and configuring the chromographic peak detection algorithm. See also `findChromPeaks()` for more details.
`msLevel`	`integer(1)` specifying the MS level in which the peak detection should be performed. By default `msLevel = 2L`.
`isolationWindow`	`factor` or similar defining the isolation windows in which the peak detection should be performed with length equal to the number of spectra in `object`.
`...`	currently not used.

Value

An `XCMSnExp` object with the chromatographic peaks identified in spectra of each isolation window from each file added to the `chromPeaks` matrix. Isolation window definition for each identified peak are stored as additional columns in `chromPeakData()`.

Author(s)

Johannes Rainer, Michael Witting

See Also

`reconstructChromPeakSpectra()` for the function to reconstruct MS2 spectra for each MS1 chromatographic peak.

`findMZ` *Find fragment ions in xcmsFragment objects*

Description

This is a method to find a fragment mass with a ppm window in a xcmsFragment object

Usage

```
findMZ(object, find, ppmE=25, print=TRUE)
```

Arguments

`object`	xcmsFragment object type
`find`	The fragment ion to be found
`ppmE`	the ppm error window for searching
`print`	If we should print a nice little report

Details

The method simply searches for a given fragment ion in an xcmsFragment object type given a certain ppm error window

Value

A data frame with the following columns:

`PrecursorMz`	The precursor m/z of the fragment
`MSnParentPeakID`	An index ID of the location of the precursor peak in the xcmsFragment object
`msLevel`	The level of the found fragment ion
`rt`	the Retention time of the found ion

`mz`	the actual m/z of the found fragment ion
`intensity`	The intensity of the fragment ion
`sample`	Which sample the fragment ion came from
`GroupPeakMSn`	an ID if the peaks were grouped by an xcmsSet grouping
`CollisionEnergy`	The collision energy of the precursor scan

Author(s)

H. Paul Benton, `<hpaul.beonton08@imperial.ac.uk>`

References

H. Paul Benton, D.M. Wong, S.A.Strauger, G. Siuzdak "XCMS^2" Analytical Chemistry 2008

See Also

`findneutral`,

Examples

```
## Not run:
    library(msdata)
    mzMLpath <- system.file("iontrap", package = "msdata")
    mzMLfiles<-list.files(mzMLpath, pattern = "extracted.mzML",
                            recursive = TRUE, full.names = TRUE)
    xs <- xcmsSet(mzMLfiles, method = "MS1")
    ##takes only one file from the file set
    xfrag <- xcmsFragments(xs)
    found<-findMZ(xfrag, 657.3433, 50)

## End(Not run)
```

`findneutral` *Find neutral losses in xcmsFragment objects*

Description

This is a method to find a neutral loss with a ppm window in a xcmsFragment object

Usage

```
findneutral(object, find, ppmE=25, print=TRUE)
```

Arguments

`object`	xcmsFragment object type
`find`	The neutral loss to be found
`ppmE`	the ppm error window for searching
`print`	If we should print a nice little report

Details

The method searches for a given neutral loss in an xcmsFragment object type given a certain ppm error window. The neutral losses are generated between neighbouring ions. The resulting data frame shows the whole scan in which the neutral loss was found.

Value

A data frame with the following columns:

`PrecursorMz`	The precursor m/z of the neutral losses
`MSnParentPeakID`	An index ID of the location of the precursor peak in the xcmsFragment object
`msLevel`	The level of the found fragment ion
`rt`	the Retention time of the found ion
`mz`	the actual m/z of the found fragment ion
`intensity`	The intensity of the fragment ion
`sample`	Which sample the fragment ion came from
`GroupPeakMSn`	an ID if the peaks were grouped by an xcmsSet grouping
`CollisionEnergy`	The collision energy of the precursor scan

Author(s)

H. Paul Benton, <`hpbenton@scripps.edu`>

References

H. Paul Benton, D.M. Wong, S.A.Strauger, G. Siuzdak "XCMS^2" Analytical Chemistry 2008

See Also

`findMZ`,

Examples

```
## Not run:
    library(msdata)
    mzMLpath <- system.file("iontrap", package = "msdata")
    mzMLfiles<-list.files(mzMLpath, pattern = "extracted.mzML",
                            recursive = TRUE, full.names = TRUE)
    xs <- xcmsSet(mzMLfiles, method = "MS1")
    ##takes only one file from the file set
    xfrag <- xcmsFragments(xs)
    found<-findneutral(xfrag, 58.1455, 50)

## End(Not run)
```

`findPeaks-methods` *Feature detection for GC/MS and LC/MS Data - methods*

Description

A number of peak pickers exist in XCMS. `findPeaks` is the generic method.

Arguments

`object`	`xcmsRaw-class` object
`method`	Method to use for peak detection. See details.
`...`	Optional arguments to be passed along

Details

Different algorithms can be used by specifying them with the `method` argument. For example to use the matched filter approach described by Smith et al (2006) one would use: `findPeaks(object, method="matchedFilter")`. This is also the default.

Further arguments given by `...` are passed through to the function implementing the `method`.

A character vector of *nicknames* for the algorithms available is returned by `getOption("BioC")$xcms$findPeaks.methods`. If the nickname of a method is called "centWave", the help page for that specific method can be accessed with `?findPeaks.centWave`.

Value

A matrix with columns:

`mz`	weighted (by intensity) mean of peak m/z across scans
`mzmin`	m/z of minimum step
`mzmax`	m/z of maximum step
`rt`	retention time of peak midpoint
`rtmin`	leading edge of peak retention time

`rtmax`	trailing edge of peak retention time
`into`	integrated area of original (raw) peak
`maxo`	maximum intensity of original (raw) peak

and additional columns depending on the choosen method.

Methods

object = "xcmsRaw" `findPeaks(object, ...)`

See Also

`findPeaks.matchedFilter` `findPeaks.centWave` `findPeaks.addPredictedIsotopeFeatures` `findPeaks.centWaveWithPredictedIsotopeROIs` `xcmsRaw-class`

`findPeaks-MSW` *Single-spectrum non-chromatography MS data peak detection*

Description

Perform peak detection in mass spectrometry direct injection spectrum using a wavelet based algorithm.

The `MSWParam` class allows to specify all settings for a peak detection using the MSW method. Instances should be created with the `MSWParam` constructor.

The `findChromPeaks,OnDiskMSnExp,MSWParam` method performs peak detection in single-spectrum non-chromatography MS data using functionality from the `MassSpecWavelet` package on all samples from an `OnDiskMSnExp` object. `OnDiskMSnExp` objects encapsule all experiment specific data and load the spectra data (mz and intensity values) on the fly from the original files applying also all eventual data manipulations.

`snthresh,snthresh<-`: getter and setter for the `snthresh` slot of the object.

`verboseColumns,verboseColumns<-`: getter and setter for the `verboseColumns` slot of the object.

`scales,scales<-`: getter and setter for the `scales` slot of the object.

`nearbyPeak,nearbyPeak<-`: getter and setter for the `nearbyPeak` slot of the object.

`peakScaleRange,peakScaleRange<-`: getter and setter for the `peakScaleRange` slot of the object.

`ampTh,ampTh<-`: getter and setter for the `ampTh` slot of the object.

`minNoiseLevel,minNoiseLevel<-`: getter and setter for the `minNoiseLevel` slot of the object.

`ridgeLength,ridgeLength<-`: getter and setter for the `ridgeLength` slot of the object.

`peakThr,peakThr<-`: getter and setter for the `peakThr` slot of the object.

`tuneIn,tuneIn<-`: getter and setter for the `tuneIn` slot of the object.

`addParams,addParams<-`: getter and setter for the `addParams` slot of the object. This slot stores optional additional parameters to be passed to the `identifyMajorPeaks` and `sav.gol` functions from the `MassSpecWavelet` package.

Usage

```
MSWParam(
  snthresh = 3,
  verboseColumns = FALSE,
  scales = c(1, seq(2, 30, 2), seq(32, 64, 4)),
  nearbyPeak = TRUE,
  peakScaleRange = 5,
  ampTh = 0.01,
  minNoiseLevel = ampTh/snthresh,
  ridgeLength = 24,
  peakThr = NULL,
  tuneIn = FALSE,
  ...
)

## S4 method for signature 'OnDiskMSnExp,MSWParam'
findChromPeaks(
  object,
  param,
  BPPARAM = bpparam(),
  return.type = "XCMSnExp",
  msLevel = 1L,
  ...
)

## S4 method for signature 'MSWParam'
snthresh(object)

## S4 replacement method for signature 'MSWParam'
snthresh(object) <- value

## S4 method for signature 'MSWParam'
verboseColumns(object)

## S4 replacement method for signature 'MSWParam'
verboseColumns(object) <- value

## S4 method for signature 'MSWParam'
scales(object)

## S4 replacement method for signature 'MSWParam'
scales(object) <- value

## S4 method for signature 'MSWParam'
nearbyPeak(object)

## S4 replacement method for signature 'MSWParam'
nearbyPeak(object) <- value
```

```
## S4 method for signature 'MSWParam'
peakScaleRange(object)

## S4 replacement method for signature 'MSWParam'
peakScaleRange(object) <- value

## S4 method for signature 'MSWParam'
ampTh(object)

## S4 replacement method for signature 'MSWParam'
ampTh(object) <- value

## S4 method for signature 'MSWParam'
minNoiseLevel(object)

## S4 replacement method for signature 'MSWParam'
minNoiseLevel(object) <- value

## S4 method for signature 'MSWParam'
ridgeLength(object)

## S4 replacement method for signature 'MSWParam'
ridgeLength(object) <- value

## S4 method for signature 'MSWParam'
peakThr(object)

## S4 replacement method for signature 'MSWParam'
peakThr(object) <- value

## S4 method for signature 'MSWParam'
tuneIn(object)

## S4 replacement method for signature 'MSWParam'
tuneIn(object) <- value

## S4 method for signature 'MSWParam'
addParams(object)

## S4 replacement method for signature 'MSWParam'
addParams(object) <- value
```

Arguments

`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`verboseColumns`	`logical(1)` whether additional peak meta data columns should be returned.
`scales`	Numeric defining the scales of the continuous wavelet transform (CWT).

`nearbyPeak`	logical(1) whether to include nearby peaks of major peaks.
`peakScaleRange`	numeric(1) defining the scale range of the peak (larger than 5 by default).
`ampTh`	numeric(1) defining the minimum required relative amplitude of the peak (ratio of the maximum of CWT coefficients).
`minNoiseLevel`	numeric(1) defining the minimum noise level used in computing the SNR.
`ridgeLength`	numeric(1) defining the minimum highest scale of the peak in 2-D CWT coefficient matrix.
`peakThr`	numeric(1) with the minimum absolute intensity (above baseline) of peaks to be picked. If provided, the smoothing function `sav.gol` function (in the `MassSpecWavelet`) package is called to estimate the local intensity.
`tuneIn`	logical(1) whther to tune in the parameter estimation of the detected peaks.
`...`	Additional parameters to be passed to the `identifyMajorPeaks` and `sav.gol` functions from the `MassSpecWavelet` package.
`object`	For `findChromPeaks`: an `OnDiskMSnExp` object containing the MS- and all other experiment-relevant data. For all other methods: a parameter object.
`param`	An `MSWParam` object containing all settings for the algorithm.
`BPPARAM`	A parameter class specifying if and how parallel processing should be performed. It defaults to `bpparam`. See documentation of the `BiocParallel` for more details. If parallel processing is enabled, peak detection is performed in parallel on several of the input samples.
`return.type`	Character specifying what type of object the method should return. Can be either `"XCMSnExp"` (default), `"list"` or `"xcmsSet"`.
`msLevel`	`integer(1)` defining the MS level on which the peak detection should be performed. Defaults to `msLevel = 1`.
`value`	The value for the slot.

Details

This is a wrapper for the peak picker in Bioconductor's `MassSpecWavelet` package calling `peakDetectionCWT` and `tuneInPeakInfo` functions. See the *xcmsDirect* vignette for more information.

Parallel processing (one process per sample) is supported and can be configured either by the `BPPARAM` parameter or by globally defining the parallel processing mode using the `register` method from the `BiocParallel` package.

Value

The `MSWParam` function returns a `MSWParam` class instance with all of the settings specified for peak detection by the *MSW* method.

For `findChromPeaks`: if `return.type = "XCMSnExp"` an `XCMSnExp` object with the results of the peak detection. If `return.type = "list"` a list of length equal to the number of samples with matrices specifying the identified peaks. If `return.type = "xcmsSet"` an `xcmsSet` object with the results of the detection.

Slots

`snthresh,verboseColumns,scales,nearbyPeak,peakScaleRange,ampTh,minNoiseLevel,ridgeLength,peakThr,tu`
See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `findPeaks` methods. It supports peak detection on `OnDiskMSnExp` objects (defined in the `MSnbase` package). All of the settings to the algorithm can be passed with a `MSWParam` object.

Author(s)

Joachim Kutzera, Steffen Neumann, Johannes Rainer

See Also

The `do_findPeaks_MSW` core API function and `findPeaks.MSW` for the old user interface.

`XCMSnExp` for the object containing the results of the peak detection.

Other peak detection methods: `chromatographic-peak-detection`, `findChromPeaks-centWaveWithPredIsoROIs`, `findChromPeaks-centWave`, `findChromPeaks-massifquant`, `findChromPeaks-matchedFilter`

Examples

```
## Create a MSWParam object
mp <- MSWParam()
## Change snthresh parameter
snthresh(mp) <- 15
mp

## Loading a small subset of direct injection, single spectrum files
library(msdata)
fticrf <- list.files(system.file("fticr-mzML", package = "msdata"),
                    recursive = TRUE, full.names = TRUE)
fticr <- readMSData(fticrf[1], msLevel. = 1, mode = "onDisk")

## Perform the MSW peak detection on these:
p <- MSWParam(scales = c(1, 7), peakThr = 80000, ampTh = 0.005,
             SNR.method = "data.mean", winSize.noise = 500)
fticr <- findChromPeaks(fticr, param = p)

head(chromPeaks(fticr))
```

`findPeaks.addPredictedIsotopeFeatures-methods` *Feature detection based on predicted isotope features for high resolution LC/MS data*

Description

Peak density and wavelet based feature detection aiming at isotope peaks for high resolution LC/MS data in centroid mode

Arguments

`object`	`xcmsSet` object
`ppm`	maxmial tolerated m/z deviation in consecutive scans, in ppm (parts per million)
`peakwidth`	Chromatographic peak width, given as range (min,max) in seconds
`prefilter`	`prefilter=c(k,I)`. Prefilter step for the first phase. Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Function to calculate the m/z center of the feature: `wMean` intensity weighted mean of the feature m/z values, `mean` mean of the feature m/z values, `apex` use m/z value at peak apex, `wMeanApex3` intensity weighted mean of the m/z value at peak apex and the m/z value left and right of it, `meanApex3` mean of the m/z value at peak apex and the m/z value left and right of it.
`integrate`	Integration method. If `=1` peak limits are found through descent on the mexican hat filtered data, if `=2` the descent is done on the real data. Method 2 is very accurate but prone to noise, while method 1 is more robust to noise but less exact.
`mzdiff`	minimum difference in m/z for peaks with overlapping retention times, can be negative to allow overlap
`fitgauss`	logical, if TRUE a Gaussian is fitted to each peak
`scanrange`	scan range to process
`noise`	optional argument which is useful for data that was centroided without any intensity threshold, centroids with intensity < `noise` are omitted from ROI detection
`sleep`	number of seconds to pause between plotting peak finding cycles
`verbose.columns`	logical, if TRUE additional peak meta data columns are returned
`xcmsPeaks`	peak list picked using the `centWave` algorithm with parameter `verbose.columns` set to TRUE (columns `scmin` and `scmax` needed)
`snthresh`	signal to noise ratio cutoff, definition see below.
`maxcharge`	max. number of the isotope charge.
`maxiso`	max. number of the isotope peaks to predict for each detected feature.
`mzIntervalExtension`	logical, if TRUE predicted isotope ROIs (regions of interest) are extended in the m/z dimension to increase the detection of low intensity and hence noisy peaks.

Details

This algorithm is most suitable for high resolution LC/{TOF,OrbiTrap,FTICR}-MS data in centroid mode. In the first phase of the method isotope ROIs (regions of interest) in the LC/MS map are predicted. In the second phase these mass traces are further analysed. Continuous wavelet transform (CWT) is used to locate chromatographic peaks on different scales. The resulting peak list and the given peak list (`xcmsPeaks`) are merged and redundant peaks are removed.

Value

A matrix with columns:

`mz`	weighted (by intensity) mean of peak m/z across scans
`mzmin`	m/z peak minimum
`mzmax`	m/z peak maximum
`rt`	retention time of peak midpoint
`rtmin`	leading edge of peak retention time
`rtmax`	trailing edge of peak retention time
`into`	integrated peak intensity
`intb`	baseline corrected integrated peak intensity
`maxo`	maximum peak intensity
`sn`	Signal/Noise ratio, defined as `(maxo - baseline)/sd`, where `maxo` is the maximum peak intensity, `baseline` the estimated baseline value and `sd` the standard deviation of local chromatographic noise.
`egauss`	RMSE of Gaussian fit
	if `verbose.columns` is `TRUE` additionally :
`mu`	Gaussian parameter mu
`sigma`	Gaussian parameter sigma
`h`	Gaussian parameter h
`f`	Region number of m/z ROI where the peak was localised
`dppm`	m/z deviation of mass trace across scans in ppm
`scale`	Scale on which the peak was localised
`scpos`	Peak position found by wavelet analysis
`scmin`	Left peak limit found by wavelet analysis (scan number)
`scmax`	Right peak limit found by wavelet analysis (scan number)

Methods

object = "xcmsRaw" `findPeaks.centWave(object, ppm=25, peakwidth=c(20,50), prefilter=c(3,100), mzCenterFun="wMean", integrate=1, mzdiff=-0.001, fitgauss=FALSE, scanrange= numeric(), noise=0, sleep=0, verbose.columns=FALSE, xcmsPeaks, snthresh=6.25, maxcharge=3, maxiso=5, mzIntervalExtension=TRUE)`

Author(s)

Ralf Tautenhahn

References

Ralf Tautenhahn, Christoph B\"ottcher, and Steffen Neumann "Highly sensitive feature detection for high resolution LC/MS" BMC Bioinformatics 2008, 9:504\ Hendrik Treutler and Steffen Neumann. "Prediction, detection, and validation of isotope clusters in mass spectrometry data" Submitted to Metabolites 2016, Special Issue "Bioinformatics and Data Analysis"

See Also

`findPeaks.centWave` `findPeaks-methods` `xcmsRaw-class`

`findPeaks.centWave-methods`

Feature detection for high resolution LC/MS data

Description

Peak density and wavelet based feature detection for high resolution LC/MS data in centroid mode

Arguments

`object`	`xcmsSet` object
`ppm`	maxmial tolerated m/z deviation in consecutive scans, in ppm (parts per million)
`peakwidth`	Chromatographic peak width, given as range (min,max) in seconds
`snthresh`	signal to noise ratio cutoff, definition see below.
`prefilter`	`prefilter=c(k,I)`. Prefilter step for the first phase. Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Function to calculate the m/z center of the feature: `wMean` intensity weighted mean of the feature m/z values, `mean` mean of the feature m/z values, `apex` use m/z value at peak apex, `wMeanApex3` intensity weighted mean of the m/z value at peak apex and the m/z value left and right of it, `meanApex3` mean of the m/z value at peak apex and the m/z value left and right of it.
`integrate`	Integration method. If `=1` peak limits are found through descent on the mexican hat filtered data, if `=2` the descent is done on the real data. Method 2 is very accurate but prone to noise, while method 1 is more robust to noise but less exact.
`mzdiff`	minimum difference in m/z for peaks with overlapping retention times, can be negative to allow overlap
`fitgauss`	logical, if TRUE a Gaussian is fitted to each peak
`scanrange`	scan range to process

`noise`	optional argument which is useful for data that was centroided without any intensity threshold, centroids with intensity < `noise` are omitted from ROI detection
`sleep`	number of seconds to pause between plotting peak finding cycles
`verbose.columns`	logical, if TRUE additional peak meta data columns are returned
`ROI.list`	A optional list of ROIs that represents detected mass traces (ROIs). If this list is empty (default) then centWave detects the mass trace ROIs, otherwise this step is skipped and the supplied ROIs are used in the peak detection phase. Each ROI object in the list has the following slots: `scmin` start scan index, `scmax` end scan index, `mzmin` minimum m/z, `mzmax` maximum m/z, `length` number of scans, `intensity` summed intensity.
`firstBaselineCheck`	logical, if TRUE continuous data within ROI is checked to be above 1st baseline
`roiScales`	numeric, optional vector of scales for each ROI in `ROI.list` to be used for the centWave-wavelets

Details

This algorithm is most suitable for high resolution LC/{TOF,OrbiTrap,FTICR}-MS data in centroid mode. In the first phase of the method mass traces (characterised as regions with less than `ppm` m/z deviation in consecutive scans) in the LC/MS map are located. In the second phase these mass traces are further analysed. Continuous wavelet transform (CWT) is used to locate chromatographic peaks on different scales.

Value

A matrix with columns:

`mz`	weighted (by intensity) mean of peak m/z across scans
`mzmin`	m/z peak minimum
`mzmax`	m/z peak maximum
`rt`	retention time of peak midpoint
`rtmin`	leading edge of peak retention time
`rtmax`	trailing edge of peak retention time
`into`	integrated peak intensity
`intb`	baseline corrected integrated peak intensity
`maxo`	maximum peak intensity
`sn`	Signal/Noise ratio, defined as `(maxo - baseline)/sd`, where `maxo` is the maximum peak intensity, `baseline` the estimated baseline value and `sd` the standard deviation of local chromatographic noise.
`egauss`	RMSE of Gaussian fit
	if `verbose.columns` is `TRUE` additionally :
`mu`	Gaussian parameter mu

`sigma`	Gaussian parameter sigma
`h`	Gaussian parameter h
`f`	Region number of m/z ROI where the peak was localised
`dppm`	m/z deviation of mass trace across scans in ppm
`scale`	Scale on which the peak was localised
`scpos`	Peak position found by wavelet analysis
`scmin`	Left peak limit found by wavelet analysis (scan number)
`scmax`	Right peak limit found by wavelet analysis (scan number)

Methods

object = "xcmsRaw" `findPeaks.centWave(object, ppm=25, peakwidth=c(20,50), snthresh=10, prefilter=c(3,100), mzCenterFun="wMean", integrate=1, mzdiff=-0.001, fitgauss=FALSE, scanrange= numeric(), noise=0, sleep=0, verbose.columns=FALSE, ROI.list=list()), firstBaselineCheck=TRUE, roiScales=NULL`

Author(s)

Ralf Tautenhahn

References

Ralf Tautenhahn, Christoph B\"ottcher, and Steffen Neumann "Highly sensitive feature detection for high resolution LC/MS" BMC Bioinformatics 2008, 9:504

See Also

`centWave` for the new user interface. `findPeaks-methods` `xcmsRaw-class`

`findPeaks.centWaveWithPredictedIsotopeROIs-methods`

Feature detection with centWave and additional isotope features

Description

Peak density and wavelet based feature detection for high resolution LC/MS data in centroid mode with additional peak picking of isotope features on basis of isotope peak predictions

Arguments

`object`	`xcmsSet` object
`ppm`	maxmial tolerated m/z deviation in consecutive scans, in ppm (parts per million)
`peakwidth`	Chromatographic peak width, given as range (min,max) in seconds
`snthresh`	signal to noise ratio cutoff, definition see below.

`prefilter`	`prefilter=c(k,I)`. Prefilter step for the first phase. Mass traces are only retained if they contain at least `k` peaks with intensity >= `I`.
`mzCenterFun`	Function to calculate the m/z center of the feature: `wMean` intensity weighted mean of the feature m/z values, `mean` mean of the feature m/z values, `apex` use m/z value at peak apex, `wMeanApex3` intensity weighted mean of the m/z value at peak apex and the m/z value left and right of it, `meanApex3` mean of the m/z value at peak apex and the m/z value left and right of it.
`integrate`	Integration method. If `=1` peak limits are found through descent on the mexican hat filtered data, if `=2` the descent is done on the real data. Method 2 is very accurate but prone to noise, while method 1 is more robust to noise but less exact.
`mzdiff`	minimum difference in m/z for peaks with overlapping retention times, can be negative to allow overlap
`fitgauss`	logical, if TRUE a Gaussian is fitted to each peak
`scanrange`	scan range to process
`noise`	optional argument which is useful for data that was centroided without any intensity threshold, centroids with intensity < `noise` are omitted from ROI detection
`sleep`	number of seconds to pause between plotting peak finding cycles
`verbose.columns`	logical, if TRUE additional peak meta data columns are returned
`ROI.list`	A optional list of ROIs that represents detected mass traces (ROIs). If this list is empty (default) then centWave detects the mass trace ROIs, otherwise this step is skipped and the supplied ROIs are used in the peak detection phase. Each ROI object in the list has the following slots: `scmin` start scan index, `scmax` end scan index, `mzmin` minimum m/z, `mzmax` maximum m/z, `length` number of scans, `intensity` summed intensity.
`firstBaselineCheck`	logical, if TRUE continuous data within ROI is checked to be above 1st baseline
`roiScales`	numeric, optional vector of scales for each ROI in `ROI.list` to be used for the centWave-wavelets
`snthreshIsoROIs`	signal to noise ratio cutoff for predicted isotope ROIs, definition see below.
`maxcharge`	max. number of the isotope charge.
`maxiso`	max. number of the isotope peaks to predict for each detected feature.
`mzIntervalExtension`	logical, if TRUE predicted isotope ROIs (regions of interest) are extended in the m/z dimension to increase the detection of low intensity and hence noisy peaks.

Details

This algorithm is most suitable for high resolution LC/{TOF,OrbiTrap,FTICR}-MS data in centroid mode. The `centWave` algorithm is applied in two peak picking steps as follows. In the first peak picking step ROIs (regions of interest, characterised as regions with less than `ppm` m/z deviation in consecutive scans) in the LC/MS map are located and further analysed using continuous wavelet transform (CWT) for the localization of chromatographic peaks on different scales. In the second peak picking step isotope ROIs in the LC/MS map are predicted further analysed using continuous wavelet transform (CWT) for the localization of chromatographic peaks on different scales. The peak lists resulting from both peak picking steps are merged and redundant peaks are removed.

Value

A matrix with columns:

`mz`	weighted (by intensity) mean of peak m/z across scans
`mzmin`	m/z peak minimum
`mzmax`	m/z peak maximum
`rt`	retention time of peak midpoint
`rtmin`	leading edge of peak retention time
`rtmax`	trailing edge of peak retention time
`into`	integrated peak intensity
`intb`	baseline corrected integrated peak intensity
`maxo`	maximum peak intensity
`sn`	Signal/Noise ratio, defined as `(maxo - baseline)/sd`, where `maxo` is the maximum peak intensity, `baseline` the estimated baseline value and `sd` the standard deviation of local chromatographic noise.
`egauss`	RMSE of Gaussian fit
	if `verbose.columns` is `TRUE` additionally :
`mu`	Gaussian parameter mu
`sigma`	Gaussian parameter sigma
`h`	Gaussian parameter h
`f`	Region number of m/z ROI where the peak was localised
`dppm`	m/z deviation of mass trace across scans in ppm
`scale`	Scale on which the peak was localised
`scpos`	Peak position found by wavelet analysis
`scmin`	Left peak limit found by wavelet analysis (scan number)
`scmax`	Right peak limit found by wavelet analysis (scan number)

Methods

object = "xcmsRaw" `findPeaks.centWaveWithPredictedIsotopeROIs(object, ppm=25, peakwidth=c(20,50), snthresh=10, prefilter=c(3,100), mzCenterFun="wMean", integrate=1, mzdiff=-0.001, fitgauss=FALSE, scanrange= numeric(), noise=0, sleep=0, verbose.columns=FALSE, ROI.list=list(), firstBaselineCheck=TRUE, roiScales=NULL, snthreshIsoROIs=6.25, maxcharge=3, maxiso=5, mzIntervalExtension=TRUE)`

Author(s)

Ralf Tautenhahn

References

Ralf Tautenhahn, Christoph B\"ottcher, and Steffen Neumann "Highly sensitive feature detection for high resolution LC/MS" BMC Bioinformatics 2008, 9:504\ Hendrik Treutler and Steffen Neumann. "Prediction, detection, and validation of isotope clusters in mass spectrometry data" Submitted to Metabolites 2016, Special Issue "Bioinformatics and Data Analysis"

See Also

`do_findChromPeaks_centWaveWithPredIsoROIs` for the corresponding core API function. `findPeaks.addPredictedIso` `findPeaks.centWave` `findPeaks-methods` `xcmsRaw-class`

`findPeaks.massifquant-methods`

Feature detection for XC-MS data.

Description

Massifquant is a Kalman filter (KF) based feature detection for XC-MS data in centroid mode (currently in experimental stage). Optionally allows for calling the method "centWave" on features discovered by Massifquant to further refine the feature detection; to do so, supply any additional parameters specific to centWave (even more experimental). The method may be conveniently called through the xcmsSet(...) method.

Arguments

	The following arguments are specific to Massifquant. Any additional arguments supplied must correspond as specified by the method findPeaks.centWave.
`object`	An xcmsRaw object.
`criticalValue`	Numeric: Suggested values: (0.1-3.0). This setting helps determine the the Kalman Filter prediciton margin of error. A real centroid belonging to a bonafide feature must fall within the KF prediction margin of error. Much like in the construction of a confidence interval, criticalVal loosely translates to be a multiplier of the standard error of the prediction reported by the Kalman Filter. If the features in the XC-MS sample have a small mass deviance in ppm error, a smaller critical value might be better and vice versa.
`consecMissedLimit`	Integer: Suggested values:(1,2,3). While a feature is in the proces of being detected by a Kalman Filter, the Kalman Filter may not find a predicted centroid in every scan. After 1 or more consecutive failed predictions, this setting informs Massifquant when to stop a Kalman Filter from following a candidate feature.
`prefilter`	Numeric Vector: (Positive Integer, Positive Numeric): The first argument is only used if (withWave = 1); see centWave for details. The second argument specifies the minimum threshold for the maximum intensity of a feature that must be met.

`peakwidth`	Integer Vector: (Positive Integer, Positive Integer): Only the first argument is used for Massifquant, which specifices the minimum feature length in time scans. If centWave is used, then the second argument is the maximum feature length subject to being greater than the mininum feature length.
`ppm`	The minimum estimated parts per million mass resolution a feature must possess.
`unions`	Integer: set to 1 if apply t-test union on segmentation; set to 0 if no t-test to be applied on chromatographically continous features sharing same m/z range. Explanation: With very few data points, sometimes a Kalman Filter stops tracking a feature prematurely. Another Kalman Filter is instantiated and begins following the rest of the signal. Because tracking is done backwards to forwards, this algorithmic defect leaves a real feature divided into two segments or more. With this option turned on, the program identifies segmented features and combines them (merges them) into one with a two sample t-test. The potential danger of this option is that some truly distinct features may be merged.
`withWave`	Integer: set to 1 if turned on; set to 0 if turned off. Allows the user to find features first with Massifquant and then filter those features with the second phase of centWave, which includes wavelet estimation.
`checkBack`	Integer: set to 1 if turned on; set to 0 if turned off. The convergence of a Kalman Filter to a feature's precise m/z mapping is very fast, but sometimes it incorporates erroneous centroids as part of a feature (especially early on). The "scanBack" option is an attempt to remove the occasional outlier that lies beyond the converged bounds of the Kalman Filter. The option does not directly affect identification of a feature because it is a postprocessing measure; it has not shown to be a extremely useful thus far and the default is set to being turned off.

Details

This algorithm's performance has been tested rigorously on high resolution LC/{OrbiTrap, TOF}-MS data in centroid mode. Simultaneous kalman filters identify features and calculate their area under the curve. The default parameters are set to operate on a complex LC-MS Orbitrap sample. Users will find it useful to do some simple exploratory data analysis to find out where to set a minimum intensity, and identify how many scans an average feature spans. The "consecMissedLimit" parameter has yielded good performance on Orbitrap data when set to (2) and on TOF data it was found best to be at (1). This may change as the algorithm has yet to be tested on many samples. The "criticalValue" parameter is perhaps most dificult to dial in appropriately and visual inspection of peak identification is the best suggested tool for quick optimization. The "ppm" and "checkBack" parameters have shown less influence than the other parameters and exist to give users flexibility and better accuracy.

Value

If the method findPeaks.massifquant(...) is used, then a matrix is returned with rows corresponding to features, and properties of the features listed with the following column names. Otherwise, if centWave feature is used also (withWave = 1), or Massifquant is called through the xcmsSet(...) method, then their corresponding return values are used.

`mz`	weighted m/z mean (weighted by intensity) of the feature

`mzmin`	m/z lower boundary of the feature
`mzmax`	m/z upper boundary of the feature
`rtmin`	starting scan time of the feature
`rtmax`	starting scan time of the feature
`into`	the raw quantitation (area under the curve) of the feature.
`area`	feature area that is not normalized by the scan rate.

Methods

object = "xcmsRaw" `findPeaks.massifquant(object, ppm=10, peakwidth=c(20,50), snthresh=10, prefilter=c(3,100), mzCenterFun="wMean", integrate=1, mzdiff=-0.001, fitgauss=FALSE, scanrange= numeric(), noise=0, sleep=0, verbose.columns=FALSE, criticalValue = 1.125, consecMissedLimit = 2, unions = 1, checkBack = 0, withWave = 0)`

Author(s)

Christopher Conley

References

Submitted for review. Christopher Conley, Ralf J .O Torgrip. Ryan Taylor, and John T. Prince. "Massifquant: open-source Kalman filter based XC-MS feature detection". August 2013.

See Also

`centWave` for the new user interface. `findPeaks-methods` `xcmsSet` `xcmsRaw` `xcmsRaw-class`

Examples

```
library(faahKO)
library(xcms)
#load all the wild type and Knock out samples
cdfpath <- system.file("cdf", package = "faahKO")
## Subset to only the first 2 files.
cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)[1:2]

## Run the massifquant analysis. Setting the noise level to 10000 to speed up
## execution of the examples - in a real use case it should be set to a reasoable
## value.
xset <- xcmsSet(cdffiles, method = "massifquant",
                consecMissedLimit = 1,
                snthresh = 10,
                criticalValue = 1.73,
                ppm = 10,
                peakwidth= c(30, 60),
                prefilter= c(1,3000),
                noise = 10000,
                withWave = 0)
```

`findPeaks.matchedFilter,xcmsRaw-method`
Peak detection in the chromatographic time domain

Description

Find peaks in the chromatographic time domain of the profile matrix. For more details see `do_findChromPeaks_matchedFil`

Usage

```
## S4 method for signature 'xcmsRaw'
findPeaks.matchedFilter(
  object,
  fwhm = 30,
  sigma = fwhm/2.3548,
  max = 5,
  snthresh = 10,
  step = 0.1,
  steps = 2,
  mzdiff = 0.8 - step * steps,
  index = FALSE,
  sleep = 0,
  scanrange = numeric()
)
```

Arguments

`object`	The `xcmsRaw` object on which peak detection should be performed.
`fwhm`	`numeric(1)` specifying the full width at half maximum of matched filtration gaussian model peak. Only used to calculate the actual sigma, see below.
`sigma`	`numeric(1)` specifying the standard deviation (width) of the matched filtration model peak.
`max`	`numeric(1)` representing the maximum number of peaks that are expected/will be identified per slice.
`snthresh`	`numeric(1)` defining the signal to noise cutoff to be used in the chromatographic peak detection step.
`step`	numeric(1) specifying the width of the bins/slices in m/z dimension.
`steps`	`numeric(1)` defining the number of bins to be merged before filtration (i.e. the number of neighboring bins that will be joined to the slice in which filtration and peak detection will be performed).
`mzdiff`	`numeric(1)` defining the minimum difference in m/z for peaks with overlapping retention times
`index`	`logical(1)` specifying whether indicies should be returned instead of values for m/z and retention times.

`sleep`	(DEPRECATED). The use of this parameter is highly discouraged, as it could cause problems in parallel processing mode.
`scanrange`	Numeric vector defining the range of scans to which the original `object` should be sub-setted before peak detection.

Value

A matrix, each row representing an intentified chromatographic peak, with columns:

mz Intensity weighted mean of m/z values of the peak across scans.

mzmin Minimum m/z of the peak.

mzmax Maximum m/z of the peak.

rt Retention time of the peak's midpoint.

rtmin Minimum retention time of the peak.

rtmax Maximum retention time of the peak.

into Integrated (original) intensity of the peak.

intf Integrated intensity of the filtered peak.

maxo Maximum intensity of the peak.

maxf Maximum intensity of the filtered peak.

i Rank of peak in merged EIC (<= `max`).

sn Signal to noise ratio of the peak.

Author(s)

Colin A. Smith

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" *Anal. Chem.* 2006, 78:779-787. @family Old peak detection methods

See Also

`matchedFilter` for the new user interface. `xcmsRaw`, `do_findChromPeaks_matchedFilter` for the core function performing the peak detection.

`findPeaks.MS1-methods` *Collecting MS1 precursor peaks*

Description

Collecting Tandem MS or MS^n Mass Spectrometry precursor peaks as annotated in XML raw file

Arguments

`object`	`xcmsRaw` object

Details

Some mass spectrometers can acquire MS1 and MS2 (or MS^n scans) quasi simultaneously, e.g. in data dependent tandem MS or DDIT mode.

Since xcmsFragments attaches *all* MS^n peaks to MS1 peaks in xcmsSet, it is important that findPeaks and xcmsSet do not miss any MS1 precursor peak.

To be sure that all MS1 precursor peaks are in an xcmsSet, findPeaks.MS1 does not do an actual peak picking, but simply uses the annotation stored in mzXML, mzData or mzML raw files.

This relies on the following XML tags:

mzData: `<spectrum id="463"> <spectrumInstrument msLevel="2"> <cvParam cvLabel="psi" accession="PSI:1000039" name="TimeInSeconds" value="92.7743"/> </spectrumInstrument> <precursor msLevel="1" spectrumRef="461"> <cvParam cvLabel="psi" accession="PSI:1000040" name="MassToChargeRatio" value="462.091"/> <cvParam cvLabel="psi" accession="PSI:1000042" name="Intensity" value="366.674"/> </precursor> </spectrum>`

mzXML: `<scan num="17" msLevel="2" retentionTime="PT1.5224S"> <precursorMz precursorIntensity="125245` `</scan>`

Several mzXML and mzData converters are known to create incomplete files, either without intensities (they will be set to 0) or without the precursor retention time (then a reasonably close rt will be chosen. NYI).

Value

A matrix with columns:

`mz, mzmin, mzmax`	annotated MS1 precursor selection mass
`rt, rtmin, rtmax`	annotated MS1 precursor retention time
`into, maxo, sn`	annotated MS1 precursor intensity

Methods

object = "xcmsRaw" `findPeaks.MS1(object)`

Author(s)

Steffen Neumann, <`sneumann@ipb-halle.de`>

See Also

`findPeaks-methods` `xcmsRaw-class`

`findPeaks.MSW,xcmsRaw-method`

Peak detection for single-spectrum non-chromatography MS data

Description

This method performs peak detection in mass spectrometry direct injection spectrum using a wavelet based algorithm.

Usage

```
## S4 method for signature 'xcmsRaw'
findPeaks.MSW(object, snthresh = 3, verbose.columns = FALSE, ...)
```

Arguments

`object`	The `xcmsRaw` object on which peak detection should be performed.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff.
`verbose.columns`	Logical whether additional peak meta data columns should be returned.
`...`	Additional parameters to be passed to the `identifyMajorPeaks` and `sav.gol` functions from the `MassSpecWavelet` package.

Details

This is a wrapper around the peak picker in Bioconductor's `MassSpecWavelet` package calling `peakDetectionCWT` and `tuneInPeakInfo` functions.

Value

A matrix, each row representing an intentified peak, with columns:

mz m/z value of the peak at the centroid position.

mzmin Minimum m/z of the peak.

mzmax Maximum m/z of the peak.

rt Always `-1`.

rtmin Always `-1`.

rtmax Always `-1`.

into Integrated (original) intensity of the peak.

maxo Maximum intensity of the peak.

intf Always `NA`.

maxf Maximum MSW-filter response of the peak.

sn Signal to noise ratio.

Author(s)

Joachim Kutzera, Steffen Neumann, Johannes Rainer

See Also

`MSW` for the new user interface, `do_findPeaks_MSW` for the downstream analysis function or `peakDetectionCWT` from the `MassSpecWavelet` for details on the algorithm and additionally supported parameters.

`GenericParam-class` *Generic parameter class*

Description

The `GenericParam` class allows to store generic parameter information such as the name of the function that was/has to be called (slot `fun`) and its arguments (slot `args`). This object is used to track the process history of the data processings of an `XCMSnExp` object. This is in contrast to e.g. the `CentWaveParam` object that is passed to the actual processing method.

Usage

```
GenericParam(fun = character(), args = list())
```

Arguments

`fun`	`character` representing the name of the function.
`args`	`list` (ideally named) with the arguments to the function.

Value

The `GenericParam` function returns a `GenericParam` object.

Slots

`fun` `character` specifying the function name.

`args` `list` (ideally named) with the arguments to the function.

Author(s)

Johannes Rainer

See Also

`processHistory` for how to access the process history of an `XCMSnExp` object.

Examples

```
prm <- GenericParam(fun = "mean")

prm <- GenericParam(fun = "mean", args = list(na.rm = TRUE))
```

`getEIC-methods` *Get extracted ion chromatograms for specified m/z ranges*

Description

Generate multiple extracted ion chromatograms for m/z values of interest. For `xcmsSet` objects, reread original raw data and apply precomputed retention time correction, if applicable.

Note that this method will *always* return profile, not raw data (with profile data being the binned data along M/Z). See details for further information.

Arguments

`object`	the `xcmsRaw` or `xcmsSet` object
`mzrange`	Either a two column matrix with minimum or maximum m/z or a matrix of any dimensions containing columns `mzmin` and `mzmax`. If not specified, the method for `xcmsRaw` returns the base peak chromatogram (BPC, i.e. the most intense signal for each RT across all m/z). For `xcmsSet` objects the group data will be used if `mzrange` is not provided.
`rtrange`	A two column matrix the same size as `mzrange` with minimum and maximum retention times between which to return EIC data points. If not specified, the method returns the chromatogram for the full RT range. For `xcmsSet` objects, it may also be a single number specifying the time window around the peak to return EIC data points
`step`	step (bin) size to use for profile generation. Note that a value of `step = 0` is not supported.
`groupidx`	either character vector with names or integer vector with indicies of peak groups for which to get EICs
`sampleidx`	either character vector with names or integer vector with indicies of samples for which to get EICs
`rt`	`"corrected"` for using corrected retention times, or `"raw"` for using raw retention times

Details

In contrast to the `rawEIC` method, that extracts the actual raw values, this method extracts them from the object's profile matrix (or if the provided `step` argument does not match the `profStep` of the `object` the profile matrix is calculated on the fly and the values returned).

Value

For `xcmsSet` and `xcmsRaw` objects, an `xcmsEIC` object.

Methods

object = "xcmsRaw" `getEIC(object, mzrange, rtrange = NULL, step = 0.1)`

object = "xcmsSet" `getEIC(object, mzrange, rtrange = 200, groupidx, sampleidx = sampnames(object), rt = c("corrected", "raw"))`

See Also

`xcmsRaw-class`, `xcmsSet-class`, `xcmsEIC-class`, `rawEIC`

`getPeaks-methods` *Get peak intensities for specified regions*

Description

Integrate extracted ion chromatograms in pre-defined defined regions. Return output similar to `findPeaks`.

Arguments

`object`	the `xcmsSet` object
`peakrange`	matrix or data frame with 4 columns: `mzmin`, `mzmax`, `rtmin`, `rtmax` (they must be in that order or named)
`step`	step size to use for profile generation

Value

A matrix with columns:

`i`	rank of peak identified in merged EIC (<= `max`), always `NA`
`mz`	weighted (by intensity) mean of peak m/z across scans
`mzmin`	m/z of minimum step
`mzmax`	m/z of maximum step
`ret`	retention time of peak midpoint
`retmin`	leading edge of peak retention time
`retmax`	trailing edge of peak retention time
`into`	integrated area of original (raw) peak
`intf`	integrated area of filtered peak, always `NA`
`maxo`	maximum intensity of original (raw) peak
`maxf`	maximum intensity of filtered peak, always `NA`

Methods

object = "xcmsRaw" `getPeaks(object, peakrange, step = 0.1)`

See Also

`xcmsRaw-class`

`getScan-methods` *Get m/z and intensity values for a single mass scan*

Description

Return the data from a single mass scan using the numeric index of the scan as a reference.

Arguments

`object`	the `xcmsRaw` object
`scan`	integer index of scan. if negative, the index numbered from the end
`mzrange`	limit data points returned to those between in the range, `range(mzrange)`

Value

A matrix with two columns:

`mz`	m/z values
`intensity`	intensity values

Methods

object = "xcmsRaw" `getScan(object, scan, mzrange = numeric())` `getMsnScan(object, scan, mzrange = numeric())`

See Also

`xcmsRaw-class`, `getSpec`

`getSpec-methods` *Get average m/z and intensity values for multiple mass scans*

Description

Return full-resolution averaged data from multiple mass scans.

Arguments

`object`	the `xcmsRaw` object
`...`	arguments passed to `profRange` used to sepecify the spectral segments of interest for averaging

Details

Based on the mass points from the spectra selected, a master unique list of masses is generated. Every spectra is interpolated at those masses and then averaged.

Value

A matrix with two columns:

`mz`	m/z values
`intensity`	intensity values

Methods

object = "xcmsRaw" `getSpec(object, ...)`

See Also

`xcmsRaw-class`, `profRange`, `getScan`

`getXcmsRaw-methods` *Load the raw data for one or more files in the xcmsSet*

Description

Reads the raw data applies evential retention time corrections and waters Lock mass correction and returns it as an `xcmsRaw` object (or list of `xcmsRaw` objects) for one or more files of the `xcmsSet` object.

Arguments

`object`	the `xcmsSet` object
`sampleidx`	The index of the sample for which the raw data should be returned. Can be a single number or a numeric vector with the indices. Alternatively, the file name can be specified.
`profmethod`	The profile method.
`profstep`	The profile step.
`rt`	Whether corrected or raw retention times should be returned.
`...`	Additional arguments submitted to the `xcmsRaw` function.

Value

A single `xcmsRaw` object or a list of `xcmsRaw` objects.

Methods

object = "xcmsSet" `getXcmsRaw(object, sampleidx=1, profmethod=profinfo(object)$method, profstep=profinfo(object)$step, rt=c("corrected", "raw"), ... )`

Author(s)

Johannes Rainer, `<johannes.rainer@eurac.edu>`

See Also

`xcmsRaw-class`,

`group-methods` *Group peaks from different samples together*

Description

A number of grouping (or alignment) methods exist in XCMS. `group` is the generic method.

Arguments

`object`	`xcmsSet-class` object
`method`	Method to use for grouping. See details.
`...`	Optional arguments to be passed along

Details

Different algorithms can be used by specifying them with the `method` argument. For example to use the density-based approach described by Smith et al (2006) one would use: `group(object, method="density")`. This is also the default.

Further arguments given by `...` are passed through to the function implementing the `method`.

A character vector of *nicknames* for the algorithms available is returned by `getOption("BioC")$xcms$group.methods`. If the nickname of a method is called "mzClust", the help page for that specific method can be accessed with `?group.mzClust`.

Value

An `xcmsSet` object with peak group assignments and statistics.

Methods

object = "xcmsSet" `group(object, ...)`

See Also

`group.density` `group.mzClust` `group.nearest` `xcmsSet-class`,

`group.density` *Group peaks from different samples together*

Description

Group peaks together across samples using overlapping m/z bins and calculation of smoothed peak distributions in chromatographic time.

Arguments

`object`	the `xcmsSet` object
`minfrac`	minimum fraction of samples necessary in at least one of the sample groups for it to be a valid group
`minsamp`	minimum number of samples necessary in at least one of the sample groups for it to be a valid group
`bw`	bandwidth (standard deviation or half width at half maximum) of gaussian smoothing kernel to apply to the peak density chromatogram
`mzwid`	width of overlapping m/z slices to use for creating peak density chromatograms and grouping peaks across samples
`max`	maximum number of groups to identify in a single m/z slice
`sleep`	seconds to pause between plotting successive steps of the peak grouping algorithm. peaks are plotted as points showing relative intensity. identified groups are flanked by dotted vertical lines.

Value

An `xcmsSet` object with peak group assignments and statistics.

Methods

object = "xcmsSet" `group(object, bw = 30, minfrac = 0.5, minsamp = 1, mzwid = 0.25, max = 50, sleep = 0)`

See Also

`do_groupChromPeaks_density` for the core API function performing the analysis. `xcmsSet-class`, `density`

`group.mzClust` *Group Peaks via High Resolution Alignment*

Description

Runs high resolution alignment on single spectra samples stored in a given xcmsSet.

Arguments

`object`	a xcmsSet with peaks
`mzppm`	the relative error used for clustering/grouping in ppm (parts per million)
`mzabs`	the absolute error used for clustering/grouping
`minsamp`	set the minimum number of samples in one bin
`minfrac`	set the minimum fraction of each class in one bin

Value

Returns a xcmsSet with slots groups and groupindex set.

Methods

object = "xcmsSet" `group(object, method="mzClust", mzppm = 20, mzabs = 0, minsamp = 1, minfrac=0)`

References

Saira A. Kazmi, Samiran Ghosh, Dong-Guk Shin, Dennis W. Hill and David F. Grant
Alignment of high resolution mass spectra: development of a heuristic approach for metabolomics.
Metabolomics, Vol. 2, No. 2, 75-83 (2006)

See Also

`xcmsSet-class`,

Examples

```
## Not run:
library(msdata)
mzMLpath <- system.file("fticr-mzML", package = "msdata")
mzMLfiles <- list.files(mzMLpath, recursive = TRUE, full.names = TRUE)

xs <- xcmsSet(method="MSW", files=mzMLfiles, scales=c(1,7),
              SNR.method='data.mean' , winSize.noise=500,
              peakThr=80000,  amp.Th=0.005)

xsg <- group(xs, method="mzClust")

## End(Not run)
```

`group.nearest` *Group peaks from different samples together*

Description

Group peaks together across samples by creating a master peak list and assigning corresponding peaks from all samples. It is inspired by the alignment algorithm of mzMine. For further details check `http://mzmine.sourceforge.net/` and

Katajamaa M, Miettinen J, Oresic M: MZmine: Toolbox for processing and visualization of mass spectrometry based molecular profile data. Bioinformatics (Oxford, England) 2006, 22:634?636.

Currently, there is no equivalent to `minfrac` or `minsamp`.

Arguments

`object`	the `xcmsSet` object
`mzVsRTbalance`	Multiplicator for mz value before calculating the (euclidean) distance between two peaks.
`mzCheck`	Maximum tolerated distance for mz.
`rtCheck`	Maximum tolerated distance for RT.
`kNN`	Number of nearest Neighbours to check

Value

An `xcmsSet` object with peak group assignments and statistics.

Methods

object = "xcmsSet" `group(object, mzVsRTbalance=10, mzCheck=0.2, rtCheck=15, kNN=10)`

See Also

`xcmsSet-class`, `group.density` and `group.mzClust`

Examples

```
## Not run: library(xcms)
    library(faahKO)
    ## These files do not have this problem to correct for
    ## but just for an example
    cdfpath <- system.file("cdf", package = "faahKO")
    cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)

    xset<-xcmsSet(cdffiles)

    gxset<-group(xset, method="nearest")
    nrow(gxset@groups) == 1096 ## the number of features before minFrac

    post.minFrac<-function(object, minFrac=0.5){
        ix.minFrac<-sapply(1:length(unique(sampclass(object))),
                           function(x, object, mf){
                               meta<-groups(object)
                               minFrac.idx<-numeric(length=nrow(meta))
                               idx<-which(
                                   meta[,levels(sampclass(object))[x]] >=
                                   mf*length(which(levels(sampclass(object))[x]
                                                   == sampclass(object)) ))
                               minFrac.idx[idx]<-1
                               return(minFrac.idx)
                           }, object, minFrac)
        ix.minFrac<-as.logical(apply(ix.minFrac, 1, sum))
        ix<-which(ix.minFrac == TRUE)
        return(ix)
    }

    ## using the above function we can get a post processing minFrac
    idx<-post.minFrac(gxset)

    gxset.post<-gxset ## copy the xcmsSet object
    gxset.post@groupidx<-gxset@groupidx[idx]
    gxset.post@groups<-gxset@groups[idx,]

    nrow(gxset.post@groups) == 465 ## number of features after minFrac

## End(Not run)
```

`groupChromPeaks` *Correspondence: Chromatographic peak grouping methods.*

Description

The `groupChromPeaks` method(s) perform the correspondence, i.e. the grouping of chromatographic peaks within and between samples. These methods are part of the modernized `xcms` user interface. The resulting peak groups are referred to as (mz-rt) features and can be accessed *via* the `featureDefinitions` method on the result object.

The implemented peak grouping methods are:

density peak grouping based on time dimension peak densities. See `groupChromPeaks-density` for more details.

mzClust high resolution peak grouping for single spectra (direct infusion) MS data. See `groupChromPeaks-mzClust` for more details.

nearest chromatographic peak grouping based on their proximity in the mz-rt space. See `groupChromPeaks-nearest` for more details.

Author(s)

Johannes Rainer

See Also

`featureDefinitions` and `featureValues,XCMSnExp-method` for methods to access peak grouping results.

`featureChromatograms` to extract ion chromatograms for each feature.

`group` for the *old* peak grouping methods.

Other peak grouping methods: `groupChromPeaks-density`, `groupChromPeaks-mzClust`, `groupChromPeaks-nearest`

---

groupChromPeaks-density

Peak grouping based on time dimension peak densities

Description

This method performs performs correspondence (chromatographic peak grouping) based on the density (distribution) of identified peaks along the retention time axis within slices of overlapping mz ranges. All peaks (from the same or from different samples) being close on the retention time axis are grouped into a feature (*peak group*).

The `PeakDensityParam` class allows to specify all settings for the peak grouping based on peak densities along the time dimension. Instances should be created with the `PeakDensityParam()` constructor.

`sampleGroups`,`sampleGroups<-`: getter and setter for the `sampleGroups` slot of the object. Its length should match the number of samples in the experiment and it should not contain `NA`s.

`bw`,`bw<-`: getter and setter for the `bw` slot of the object.

`minFraction`,`minFraction<-`: getter and setter for the `minFraction` slot of the object.

`minSamples`,`minSamples<-`: getter and setter for the `minSamples` slot of the object.

`binSize`,`binSize<-`: getter and setter for the `binSize` slot of the object.

`maxFeatures`,`maxFeatures<-`: getter and setter for the `maxFeatures` slot of the object.

`groupChromPeaks,XCMSnExp,PeakDensityParam`: performs correspondence (peak grouping within and across samples) within in mz dimension overlapping slices of MS data based on the density distribution of the identified chromatographic peaks in the slice along the time axis.

The correspondence analysis can be performed on chromatographic peaks of any MS level (if present and if chromatographic peak detection has been performed for that MS level) defining features combining these peaks. The MS level can be selected with the parameter `msLevel`. By default, calling `groupChromPeaks` will remove any previous correspondence results. This can be disabled with `add = TRUE`, which will add newly defined features to already present feature definitions.

Usage

```
PeakDensityParam(
  sampleGroups = numeric(),
  bw = 30,
  minFraction = 0.5,
  minSamples = 1,
  binSize = 0.25,
  maxFeatures = 50
)

## S4 method for signature 'PeakDensityParam'
sampleGroups(object)

## S4 replacement method for signature 'PeakDensityParam'
sampleGroups(object) <- value

## S4 method for signature 'PeakDensityParam'
bw(object)

## S4 replacement method for signature 'PeakDensityParam'
bw(object) <- value

## S4 method for signature 'PeakDensityParam'
minFraction(object)

## S4 replacement method for signature 'PeakDensityParam'
minFraction(object) <- value

## S4 method for signature 'PeakDensityParam'
minSamples(object)

## S4 replacement method for signature 'PeakDensityParam'
minSamples(object) <- value

## S4 method for signature 'PeakDensityParam'
binSize(object)

## S4 replacement method for signature 'PeakDensityParam'
binSize(object) <- value

## S4 method for signature 'PeakDensityParam'
maxFeatures(object)
```

```
## S4 replacement method for signature 'PeakDensityParam'
maxFeatures(object) <- value

## S4 method for signature 'XCMSnExp,PeakDensityParam'
groupChromPeaks(object, param, msLevel = 1L, add = FALSE)
```

Arguments

`sampleGroups`	A vector of the same length than samples defining the sample group assignments (i.e. which samples belong to which sample group). This parameter is mandatory for the `PeakDensityParam` and has to be provided also if there is no sample grouping in the experiment (in which case all samples should be assigned to the same group).
`bw`	`numeric(1)` defining the bandwidth (standard deviation ot the smoothing kernel) to be used. This argument is passed to the [density() method.
`minFraction`	`numeric(1)` defining the minimum fraction of samples in at least one sample group in which the peaks have to be present to be considered as a peak group (feature).
`minSamples`	`numeric(1)` with the minimum number of samples in at least one sample group in which the peaks have to be detected to be considered a peak group (feature).
`binSize`	`numeric(1)` defining the size of the overlapping slices in mz dimension.
`maxFeatures`	`numeric(1)` with the maximum number of peak groups to be identified in a single mz slice.
`object`	For `groupChromPeaks`: an XCMSnExp object containing the results from a previous peak detection analysis (see `findChromPeaks()`). `For all other methods: a `PeakDensityParam` object.`
`value`	The value for the slot.
`param`	A `PeakDensityParam` object containing all settings for the peak grouping algorithm.
`msLevel`	`integer(1)` (default `msLevel = 1L`) defining the MS level on which the correspondence should be performed. It is required that chromatographic peaks of the respective MS level are present.
`add`	`logical(1)` (default `add = FALSE`) allowing to perform an additional round of correspondence (e.g. on a different MS level) and add features to the already present feature definitions.

Value

The `PeakDensityParam` function returns a `PeakDensityParam` class instance with all of the settings specified for chromatographic peak alignment based on peak densities. Note that argument `sampleGroups` is mandatory and should represent either the sample grouping in the experiment. It's length has to match the number of sample in the experiments.

For `groupChromPeaks`: a XCMSnExp object with the results of the correspondence analysis. The definition of the resulting mz-rt features can be accessed with the `featureDefinitions()` method

Slots

`sampleGroups`,`bw`,`minFraction`,`minSamples`,`binSize`,`maxFeatures` See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface. All of the settings to the algorithm can be passed with a `PeakDensityParam` object.

Author(s)

Colin Smith, Johannes Rainer

References

Colin A. Smith, Elizabeth J. Want, Grace O'Maille, Ruben Abagyan and Gary Siuzdak. "XCMS: Processing Mass Spectrometry Data for Metabolite Profiling Using Nonlinear Peak Alignment, Matching, and Identification" Anal. Chem. 2006, 78:779-787.

See Also

The `do_groupChromPeaks_density()` core API function and `group.density()` for the old user interface.

`plotChromPeakDensity()` to plot peak densities and evaluate different algorithm settings.

`featureDefinitions()` and `featureValues()` for methods to access the features (i.e. the peak grouping results).

XCMSnExp for the object containing the results of the correspondence.

`plotChromPeakDensity()` for plotting chromatographic peak density with the possibility to test different parameter settings.

Other peak grouping methods: `groupChromPeaks-mzClust`, `groupChromPeaks-nearest`, `groupChromPeaks()`

Examples

```
## Create a PeakDensityParam object
p <- PeakDensityParam(binSize = 0.05, sampleGroups = c(1, 1, 2, 2))
## Change hte minSamples slot
minSamples(p) <- 3
p

##############################
## Chromatographic peak detection and grouping.
##
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())
```

```
res <- faahko_sub

head(chromPeaks(res))
## The number of peaks identified per sample:
table(chromPeaks(res)[, "sample"])

## Performing the chromatographic peak grouping. Assigning all samples to
## the same sample group.
fdp <- PeakDensityParam(sampleGroups = rep(1, length(fileNames(res))))
res <- groupChromPeaks(res, fdp)

## The definition of the features (peak groups):
featureDefinitions(res)

## Using the featureValues method to extract a matrix with the
## intensities of the features per sample.
head(featureValues(res, value = "into"))

## The process history:
processHistory(res)
```

---

groupChromPeaks-mzClust

High resolution peak grouping for single spectra samples

Description

This method performs high resolution correspondence for single spectra samples.

The `MzClustParam` class allows to specify all settings for the peak grouping based on the *mzClust* algorithm. Instances should be created with the `MzClustParam` constructor.

`sampleGroups`,`sampleGroups<-`: getter and setter for the `sampleGroups` slot of the object.

`ppm`,`ppm<-`: getter and setter for the `ppm` slot of the object.

`absMz`,`absMz<-`: getter and setter for the `absMz` slot of the object.

`minFraction`,`minFraction<-`: getter and setter for the `minFraction` slot of the object.

`minSamples`,`minSamples<-`: getter and setter for the `minSamples` slot of the object.
`groupChromPeaks,XCMSnExp,MzClustParam`: performs high resolution peak grouping for single spectrum metabolomics data.

Usage

```
MzClustParam(
  sampleGroups = numeric(),
  ppm = 20,
  absMz = 0,
  minFraction = 0.5,
  minSamples = 1
```

```
)

## S4 method for signature 'MzClustParam'
sampleGroups(object)

## S4 replacement method for signature 'MzClustParam'
sampleGroups(object) <- value

## S4 method for signature 'MzClustParam'
ppm(object)

## S4 replacement method for signature 'MzClustParam'
ppm(object) <- value

## S4 method for signature 'MzClustParam'
absMz(object)

## S4 replacement method for signature 'MzClustParam'
absMz(object) <- value

## S4 method for signature 'MzClustParam'
minFraction(object)

## S4 replacement method for signature 'MzClustParam'
minFraction(object) <- value

## S4 method for signature 'MzClustParam'
minSamples(object)

## S4 replacement method for signature 'MzClustParam'
minSamples(object) <- value

## S4 method for signature 'XCMSnExp,MzClustParam'
groupChromPeaks(object, param, msLevel = 1L)
```

Arguments

`sampleGroups`	A vector of the same length than samples defining the sample group assignments (i.e. which samples belong to which sample group). This parameter is mandatory for the `PeakDensityParam` and has to be provided also if there is no sample grouping in the experiment (in which case all samples should be assigned to the same group).
`ppm`	`numeric(1)` representing the relative mz error for the clustering/grouping (in parts per million).
`absMz`	`numeric(1)` representing the absolute mz error for the clustering.
`minFraction`	`numeric(1)` defining the minimum fraction of samples in at least one sample group in which the peaks have to be present to be considered as a peak group (feature).

`minSamples`	`numeric(1)` with the minimum number of samples in at least one sample group in which the peaks have to be detected to be considered a peak group (feature).
`object`	For `groupChromPeaks`: an XCMSnExp object containing the results from a previous chromatographic peak detection analysis (see `findChromPeaks()`). For all other methods: a \`MzClustParam\` object.
`value`	The value for the slot.
`param`	A `MzClustParam` object containing all settings for the peak grouping algorithm.
`msLevel`	`integer(1)` defining the MS level. Currently only MS level 1 is supported.

Value

The `MzClustParam` function returns a `MzClustParam` class instance with all of the settings specified for high resolution single spectra peak alignment.

For `groupChromPeaks`: a XCMSnExp object with the results of the peak grouping step (i.e. the features). These can be accessed with the `featureDefinitions()` method.

Slots

`sampleGroups,ppm,absMz,minFraction,minSamples` See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface which will eventually replace the `group()` methods. All of the settings to the algorithm can be passed with a MzClustParam object.

Calling `groupChromPeaks` on an `XCMSnExp` object will cause all eventually present previous correspondence results to be dropped.

References

Saira A. Kazmi, Samiran Ghosh, Dong-Guk Shin, Dennis W. Hill and David F. Grant
Alignment of high resolution mass spectra: development of a heuristic approach for metabolomics.
Metabolomics, Vol. 2, No. 2, 75-83 (2006)

See Also

The `do_groupPeaks_mzClust()` core API function and `group.mzClust()` for the old user interface.

`featureDefinitions()` and `featureValues()` for methods to access peak grouping results (i.e. the features).

XCMSnExp for the object containing the results of the peak grouping.

Other peak grouping methods: `groupChromPeaks-density`, `groupChromPeaks-nearest`, `groupChromPeaks()`

Examples

```
## Loading a small subset of direct injection, single spectrum files
library(msdata)
fticrf <- list.files(system.file("fticr-mzML", package = "msdata"),
                    recursive = TRUE, full.names = TRUE)
fticr <- readMSData(fticrf[1:2], msLevel. = 1, mode = "onDisk")

## Disable parallel processing for this example
register(SerialParam())

## Perform the MSW peak detection on these:
p <- MSWParam(scales = c(1, 7), peakThr = 80000, ampTh = 0.005,
             SNR.method = "data.mean", winSize.noise = 500)
fticr <- findChromPeaks(fticr, param = p)

head(chromPeaks(fticr))

## Now create the MzClustParam parameter object: we're assuming here that
## both samples are from the same sample group.
p <- MzClustParam(sampleGroups = c(1, 1))

fticr <- groupChromPeaks(fticr, param = p)

## Get the definition of the features.
featureDefinitions(fticr)
```

`groupChromPeaks-nearest`

Peak grouping based on proximity in the mz-rt space

Description

This method is inspired by the grouping algorithm of mzMine (Katajamaa 2006) and performs correspondence based on proximity of peaks in the space spanned by retention time and mz values. The method creates first a *master peak list* consisting of all chromatographic peaks from the sample in which most peaks were identified, and starting from that, calculates distances to peaks from the sample with the next most number of peaks. If peaks are closer than the defined threshold they are grouped together.

The `NearestPeaksParam` class allows to specify all settings for the peak grouping based on the *nearest* algorithm. Instances should be created with the `NearestPeaksParam` constructor.

`sampleGroups`,`sampleGroups<-`: getter and setter for the `sampleGroups` slot of the object.

`mzVsRtBalance`,`mzVsRtBalance<-`: getter and setter for the `mzVsRtBalance` slot of the object.

`absMz`,`absMz<-`: getter and setter for the `absMz` slot of the object.

`absRt`,`absRt<-`: getter and setter for the `absRt` slot of the object.

`kNN`,`kNN<-`: getter and setter for the `kNN` slot of the object.

`groupChromPeaks,XCMSnExp,NearestPeaksParam`: performs peak grouping based on the proximity between chromatographic peaks from different samples in the mz-rt range.

The correspondence analysis can be performed on chromatographic peaks of any MS level (if present and if chromatographic peak detection has been performed for that MS level) defining features combining these peaks. The MS level can be selected with the parameter `msLevel`. By default, calling `groupChromPeaks` will remove any previous correspondence results. This can be disabled with `add = TRUE`, which will add newly defined features to already present feature definitions.

Usage

```
NearestPeaksParam(
  sampleGroups = numeric(),
  mzVsRtBalance = 10,
  absMz = 0.2,
  absRt = 15,
  kNN = 10
)

## S4 method for signature 'NearestPeaksParam'
sampleGroups(object)

## S4 replacement method for signature 'NearestPeaksParam'
sampleGroups(object) <- value

## S4 method for signature 'NearestPeaksParam'
mzVsRtBalance(object)

## S4 replacement method for signature 'NearestPeaksParam'
mzVsRtBalance(object) <- value

## S4 method for signature 'NearestPeaksParam'
absMz(object)

## S4 replacement method for signature 'NearestPeaksParam'
absMz(object) <- value

## S4 method for signature 'NearestPeaksParam'
absRt(object)

## S4 replacement method for signature 'NearestPeaksParam'
absRt(object) <- value

## S4 method for signature 'NearestPeaksParam'
kNN(object)

## S4 replacement method for signature 'NearestPeaksParam'
kNN(object) <- value
```

```
## S4 method for signature 'XCMSnExp,NearestPeaksParam'
groupChromPeaks(object, param, msLevel = 1L, add = FALSE)
```

Arguments

`sampleGroups`	A vector of the same length than samples defining the sample group assignments (i.e. which samples belong to which sample group). This parameter is mandatory for the `PeakDensityParam` and has to be provided also if there is no sample grouping in the experiment (in which case all samples should be assigned to the same group).
`mzVsRtBalance`	`numeric(1)` representing the factor by which mz values are multiplied before calculating the (euclician) distance between two peaks.
`absMz`	`numeric(1)` maximum tolerated distance for mz values.
`absRt`	`numeric(1)` maximum tolerated distance for rt values.
`kNN`	`numeric(1)` representing the number of nearest neighbors to check.
`object`	For `groupChromPeaks`: an XCMSnExp object containing the results from a previous chromatographic peak detection analysis (see `findChromPeaks()`). ``For all other methods: a `NearestPeaksParam` object.``
`value`	The value for the slot.
`param`	A `PeakDensityParam` object containing all settings for the peak grouping algorithm.
`msLevel`	`integer(1)` defining the MS level. Currently only MS level 1 is supported.
`add`	`logical(1)` (default `add = FALSE`) allowing to perform an additional round of correspondence (e.g. on a different MS level) and add features to the already present feature definitions.

Value

The `NearestPeaksParam` function returns a `NearestPeaksParam` class instance with all of the settings specified for peak alignment based on peak proximity.

For `groupChromPeaks`: a XCMSnExp object with the results of the peak grouping/correspondence step (i.e. the mz-rt features). These can be accessed with the `featureDefinitions()` method.

Slots

`sampleGroups,mzVsRtBalance,absMz,absRt,kNN` See corresponding parameter above.

Note

These methods and classes are part of the updated and modernized `xcms` user interface. All of the settings to the algorithm can be passed with a `NearestPeaksParam` object.

References

Katajamaa M, Miettinen J, Oresic M: MZmine: Toolbox for processing and visualization of mass spectrometry based molecular profile data. Bioinformatics 2006, 22:634-636.

See Also

The `do_groupChromPeaks_nearest()` core API function.

`featureDefinitions()` and `featureValues()` for methods to access peak grouping results (i.e. the features).

XCMSnExp for the object containing the results of the peak grouping.

Other peak grouping methods: `groupChromPeaks-density`, `groupChromPeaks-mzClust`, `groupChromPeaks()`

Examples

```
## Create a NearestPeaksParam object
p <- NearestPeaksParam(kNN = 3)
p

## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")
res <- faahko_sub

## Disable parallel processing for this example
register(SerialParam())

head(chromPeaks(res))
## The number of peaks identified per sample:
table(chromPeaks(res)[, "sample"])

## Performing the peak grouping
p <- NearestPeaksParam()
res <- groupChromPeaks(res, param = p)

## The results from the peak grouping:
featureDefinitions(res)

## Using the featureValues method to extract a matrix with the intensities of
## the features per sample.
head(featureValues(res, value = "into"))

## The process history:
processHistory(res)
```

`groupFeatures-abundance-correlation`

Compounding/feature grouping based on similarity of abundances across samples

Description

Features from the same originating compound are expected to have similar intensities across samples. This method this groups features based on similarity of abundances (i.e. *feature values*) across samples. See also `AbundanceSimilarityParam()` for additional information and details.

This help page lists parameters specific for `xcms` result objects (i.e. the `XCMSnExp()` object). Documentation of the parameters for the similarity calculation is available in the `AbundanceSimilarityParam()` help page in the `MsFeatures` package.

Usage

```
## S4 method for signature 'XCMSnExp,AbundanceSimilarityParam'
groupFeatures(
  object,
  param,
  msLevel = 1L,
  method = c("medret", "maxint", "sum"),
  value = "into",
  intensity = "into",
  filled = TRUE,
  ...
)
```

Arguments

`object`	`XCMSnExp()` object containing also correspondence results.
`param`	`AbudanceSimilarityParam` object with the settings for the method. See `AbundanceSimilarityParam()` for details on the grouping method and its parameters.
`msLevel`	`integer(1)` defining the MS level on which the features should be grouped.
`method`	`character(1)` passed to the `featureValues` call. See `featureValues()` for details. Defaults to `method = "medret"`.
`value`	`character(1)` passed to the `featureValues` call. See `featureValues()` for details. Defaults to `value = "into"`.
`intensity`	`character(1)` passed to the `featureValues` call. See `featureValues()` for details. Defaults to `intensity = "into"`.
`filled`	`logical(1)` whether filled-in values should be included in the correlation analysis. Defaults to `filled = TRUE`.
`...`	additional parameters passed to the `groupFeatures` method for `matrix`.

Value

input `XCMSnExp` with feature group definitions added to a column `"feature_group"` in its `featureDefinitions` data frame.

Author(s)

Johannes Rainer

See Also

feature-grouping for a general overview.

Other feature grouping methods: `groupFeatures-eic-similarity`, `groupFeatures-similar-rtime`

Examples

```
library(MsFeatures)
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Group chromatographic peaks across samples
xodg <- groupChromPeaks(faahko_sub, param = PeakDensityParam(sampleGroups = rep(1, 3)))

## Group features based on correlation of feature values (integrated
## peak area) across samples. Note that there are many missing values
## in the feature value which influence grouping of features in the present
## data set.
xodg_grp <- groupFeatures(xodg,
    param = AbundanceSimilarityParam(threshold = 0.8))
table(featureDefinitions(xodg_grp)$feature_group)

## Group based on the maximal peak intensity per feature
xodg_grp <- groupFeatures(xodg,
    param = AbundanceSimilarityParam(threshold = 0.8, value = "maxo"))
table(featureDefinitions(xodg_grp)$feature_group)
```

---

groupFeatures-eic-similarity

Compounding/feature grouping based on similarity of extracted ion chromatograms

---

Description

Features from the same originating compound are expected to share their elution pattern (i.e. chromatographic peak shape) with it. Thus, this methods allows to group features based on similarity of their extracted ion chromatograms (EICs). The similarity calculation is performed separately for each sample with the similarity score being aggregated across samples for the final generation of the similarity matrix on which the grouping (considering parameter `threshold`) will be performed.

The `compareChromatograms()` function is used for similarity calculation which by default calculates the Pearson's correlation coefficient. The settings for `compareChromatograms` can be specified with parameters `ALIGNFUN`, `ALIGNFUNARGS`, `FUN` and `FUNARGS`. `ALIGNFUN` defaults to `alignRt()` and is the function used to *align* the chromatograms before comparison. `ALIGNFUNARGS` allows to specify additional arguments for the `ALIGNFUN` function. It defaults to `ALIGNFUNARGS = list(tolerance`

`= 0, method = "closest")` which ensures that data points from the same spectrum (scan, i.e. with the same retention time) are compared between the EICs from the same sample. Parameter `FUN` defines the function to calculate the similarity score and defaults to `FUN = cor` and `FUNARGS` allows to pass additional arguments to this function (defaults to `FUNARGS = list(use = "pairwise.complete.obs")`. See also `compareChromatograms()` for more information.

The grouping of features based on the EIC similarity matrix is performed with the function specified with parameter `groupFun` which defaults to `groupFun = groupSimilarityMatrix` which groups all rows (features) in the similarity matrix with a similarity score larger than `threshold` into the same cluster. This creates clusters of features in which **all** features have a similarity score `>= threshold` with **any** other feature in that cluster. See `groupSimilarityMatrix()` for details. Additional parameters to that function can be passed with the `...` argument.

This feature grouping should be called **after** an initial feature grouping by retention time (see `SimilarRtimeParam()`). The feature groups defined in columns `"feature_group"` of `featureDefinitions(object)` (for features matching `msLevel`) will be used and refined by this method. Features with a value of `NA` in `featureDefinitions(object)$feature_group` will be skipped/not considered for feature grouping.

Usage

```
EicSimilarityParam(
  threshold = 0.9,
  n = 1,
  onlyPeak = TRUE,
  value = c("maxo", "into"),
  groupFun = groupSimilarityMatrix,
  ALIGNFUN = alignRt,
  ALIGNFUNARGS = list(tolerance = 0, method = "closest"),
  FUN = cor,
  FUNARGS = list(use = "pairwise.complete.obs"),
  ...
)

## S4 method for signature 'XCMSnExp,EicSimilarityParam'
groupFeatures(object, param, msLevel = 1L)
```

Arguments

`threshold`	`numeric(1)` with the minimal required similarity score to group featues. This is passed to the `groupFun` function.
`n`	`numeric(1)` defining the total number of samples per feature group on which this similarity calculation should be performed. This value is rounded up to the next larger integer value.
`onlyPeak`	`logical(1)` whether the correlation should be performed only on the signals within the identified chromatographic peaks (`onlyPeak = TRUE`, default) or all the signal from the extracted ion chromatogram.
`value`	`character(1)` defining whether samples should be grouped based on the sum of the maximal peak intensity (`value = "maxo"`, the default) or the integrated peak area (`value = "into"`) for a feature.

`groupFun`	`function` defining the function to be used to group rows based on a pairwise similarity matrix. Defaults to `groupSimilarityMatrix()`.
`ALIGNFUN`	`function` defining the function to be used to *align* chromatograms prior similarity calculation. Defaults to `ALIGNFUN = alignRt`. See `alignRt()` and `compareChromatograms()` for more information.
`ALIGNFUNARGS`	**named** `list` with arguments for `ALIGNFUN`. Defaults to `ALIGNFUNARGS = list(tolerance = 0, method = "closest")`.
`FUN`	`function` defining the function to be used to calculate a similarity between (aligned) chromatograms. Defaults to `FUN = cor`. See `cor()` and `compareChromatograms()` for more information.
`FUNARGS`	**named** `list` with arguments for `FUN`. Defaults to `FUN = list(use = "pairwise.complete.obs")`.
`...`	for `EicSimilarityParam`: additional arguments to be passed to `groupFun` and `featureChromatograms` (such as `expandRt` to expand the retention time range of each feature).
`object`	`XCMSnExp()` object containing also correspondence results.
`param`	`EicSimilarityParam` object with the settings for the method.
`msLevel`	`integer(1)` defining the MS level on which the features should be grouped.

Value

input `XCMSnExp` with feature groups added (i.e. in column `"feature_group"` of its `featureDefinitions` data frame.

Note

While being possible to be performed on the full data set without prior feature grouping, this is not suggested for the following reasons: I) the selection of the top `n` samples with the highest signal for the *feature group* will be biased by very abundant compounds as this is performed on the full data set (i.e. the samples with the highest overall intensities are used for correlation of all features) and II) it is computationally much more expensive because a pairwise correlation between all features has to be performed.

It is also suggested to perform the correlation on a subset of samples per feature with the highest intensities of the peaks (for that feature) although it would also be possible to run the correlation on all samples by setting `n` equal to the total number of samples in the data set. EIC correlation should however be performed ideally on samples in which the original compound is highly abundant to avoid correlation of missing values or noisy peak shapes as much as possible.

By default also the signal which is outside identified chromatographic peaks is excluded from the correlation.

Author(s)

Johannes Rainer

See Also

feature-grouping for a general overview.

Other feature grouping methods: `groupFeatures-abundance-correlation`, `groupFeatures-similar-rtime`

Examples

```
library(MsFeatures)
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Group chromatographic peaks across samples
xodg <- groupChromPeaks(faahko_sub, param = PeakDensityParam(sampleGroups = rep(1, 3)))

## Performing a feature grouping based on EIC similarities on a single
## sample
xodg_grp <- groupFeatures(xodg, param = EicSimilarityParam(n = 1))

table(featureDefinitions(xodg_grp)$feature_group)

## Usually it is better to perform this correlation on pre-grouped features
## e.g. based on similar retention time.
xodg_grp <- groupFeatures(xodg, param = SimilarRtimeParam(diffRt = 4))
xodg_grp <- groupFeatures(xodg_grp, param = EicSimilarityParam(n = 1))

table(featureDefinitions(xodg_grp)$feature_group)
```

`groupFeatures-similar-rtime` *Compounding/feature grouping based on similar retention times*

Description

Group features based on similar retention time. This method is supposed to be used as an initial *crude* grouping of features based on the median retention time of all their chromatographic peaks. All features with a difference in their retention time which is <= parameter `diffRt` of the parameter object are grouped together. If a column `"feature_group"` is found in `featureDefinitions()` this is further sub-grouped by this method.

See `MsFeatures::SimilarRtimeParam()` in `MsFeatures` for more details.

Usage

```
## S4 method for signature 'XCMSnExp,SimilarRtimeParam'
groupFeatures(object, param, msLevel = 1L, ...)
```

Arguments

`object`	`XCMSnExp()` object containing also correspondence results.

`param`	`SimilarRtimeParam` object with the settings for the method. See `MsFeatures::SimilarRtimeParam()` for details and options.
`msLevel`	`integer(1)` defining the MS level on which the features should be grouped.
`...`	passed to the `groupFeatures` function on numeric values.

Value

input `XCMSnExp` with feature groups added (i.e. in column `"feature_group"` of its `featureDefinitions` data frame.

Author(s)

Johannes Rainer

See Also

Other feature grouping methods: `groupFeatures-abundance-correlation`, `groupFeatures-eic-similarity`

Examples

```
library(MsFeatures)
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Group chromatographic peaks across samples
xodg <- groupChromPeaks(faahko_sub, param = PeakDensityParam(sampleGroups = rep(1, 3)))

## Group features based on similar retention time (i.e. difference <= 2 seconds)
xodg_grp <- groupFeatures(xodg, param = SimilarRtimeParam(diffRt = 2))

## Feature grouping get added to the featureDefinitions in column "feature_group"
head(featureDefinitions(xodg_grp)$feature_group)

table(featureDefinitions(xodg_grp)$feature_group)
length(unique(featureDefinitions(xodg_grp)$feature_group))

## Using an alternative groupiing method that creates larger groups
xodg_grp <- groupFeatures(xodg,
    param = SimilarRtimeParam(diffRt = 2, groupFun = MsCoreUtils::group))

length(unique(featureDefinitions(xodg_grp)$feature_group))
```

`groupnames,XCMSnExp-method` *Generate unique group (feature) names based on mass and retention time*

Description

`groupnames` generates names for the identified features from the correspondence analysis based in their mass and retention time. This generates feature names that are equivalent to the group names of the *old* user interface (aka xcms1).

Usage

```
## S4 method for signature 'XCMSnExp'
groupnames(object, mzdec = 0, rtdec = 0, template = NULL)
```

Arguments

`object`	`XCMSnExp` object containing correspondence results.
`mzdec`	`integer(1)` with the number of decimal places to use for m/z (defaults to `0`).
`rtdec`	`integer(1)` with the number of decimal places to use for the retention time (defaults to `0`).
`template`	`character` with existing group names whose format should be emulated.

Value

`character` with unique names for each feature in `object`. The format is `M(m/z)T(time in seconds)`.

See Also

XCMSnExp.

`groupnames-methods` *Generate unque names for peak groups*

Description

Allow linking of peak group data between classes using unique group names that remain the same as long as no re-grouping occurs.

Arguments

`object`	the `xcmsSet` or `xcmsEIC` object
`mzdec`	number of decimal places to use for m/z
`rtdec`	number of decimal places to use for retention time
`template`	a character vector with existing group names whose format should be emulated

Value

A character vector with unique names for each peak group in the object. The format is `M[m/z]T[time in seconds]`.

Methods

object = "xcmsSet" `(object, mzdec = 0, rtdec = 0, template = NULL)`

object = "xcmsEIC" `(object)`

See Also

`xcmsSet-class`, `xcmsEIC-class`

`groupOverlaps` *Group overlapping ranges*

Description

`groupOverlaps` identifies overlapping ranges in the input data and groups them by returning their indices in `xmin` `xmax`.

Usage

```
groupOverlaps(xmin, xmax)
```

Arguments

`xmin`	`numeric` (same length than `xmax`) with the lower boundary of the range.
`xmax`	`numeric` (same length than `xmin`) with the upper boundary of the range.

Value

`list` with the indices of grouped elements.

Author(s)

Johannes Rainer

Examples

```
x <- c(2, 12, 34.2, 12.4)
y <- c(3, 16, 35, 36)

groupOverlaps(x, y)
```

`groupval-methods` *Extract a matrix of peak values for each group*

Description

Generate a matrix of peak values with rows for every group and columns for every sample. The value included in the matrix can be any of the columns from the `xcmsSet` peaks slot matrix. Collisions where more than one peak from a single sample are in the same group get resolved with one of several user-selectable methods.

Arguments

`object`	the `xcmsSet` object
`method`	conflict resolution method, `"medret"` to use the peak closest to the median retention time or `"maxint"` to use the peak with the highest intensity
`value`	name of peak column to enter into returned matrix, or `"index"` for index to the corresponding row in the peaks slot matrix
`intensity`	if `method == "maxint"`, name of peak column to use for intensity

Value

A matrix with with rows for every group and columns for every sample. Missing peaks have `NA` values.

Methods

object = "xcmsSet" `groupval(object, method = c("medret", "maxint"), value = "index", intensity = "into")`

See Also

`xcmsSet-class`

`highlightChromPeaks` *Add definition of chromatographic peaks to an extracted chromatogram plot*

Description

The `highlightChromPeaks` function adds chromatographic peak definitions to an existing plot, such as one created by the `plot` method on a `Chromatogram` or `MChromatograms` object.

Usage

```
highlightChromPeaks(
  x,
  rt,
  mz,
  peakIds = character(),
  border = rep("00000040", length(fileNames(x))),
  lwd = 1,
  col = NA,
  type = c("rect", "point", "polygon"),
  whichPeaks = c("any", "within", "apex_within"),
  ...
)
```

Arguments

`x`	For `highlightChromPeaks`: `XCMSnExp` object with the detected peaks.
`rt`	For `highlightChromPeaks`: `numeric(2)` with the retention time range from which peaks should be extracted and plotted.
`mz`	`numeric(2)` with the mz range from which the peaks should be extracted and plotted.
`peakIds`	`character` defining the IDs (i.e. rownames of the peak in the `chromPeaks` table) of the chromatographic peaks to be highlighted in a plot.
`border`	colors to be used to color the border of the rectangles/peaks. Has to be equal to the number of samples in `x`.
`lwd`	`numeric(1)` defining the width of the line/border.
`col`	For `highlightChromPeaks`: color to be used to fill the rectangle (if `type = "rect"`) or the peak (for `type = "polygon"`).
`type`	the plotting type. See `plot` in base grapics for more details. For `highlightChromPeaks`: `character(1)` defining how the peak should be highlighted: `type = "rect"` draws a rectangle representing the peak definition, `type = "point"` indicates a chromatographic peak with a single point at the position of the peak's `"rt"` and `"maxo"` and `type = "polygon"` will highlight the peak shape. For `type = "polygon"` the color of the border and area can be defined with parameters `"border"` and `"col"`, respectively.
`whichPeaks`	`character(1)` specifying how peaks are called to be located within the region defined by `mz` and `rt`. Can be one of `"any"`, `"within"`, and `"apex_within"` for all peaks that are even partially overlapping the region, peaks that are completely within the region, and peaks for which the apex is within the region. This parameter is passed to the `type` argument of the `chromPeaks` function. See related documentation for more information and examples.
`...`	additional parameters to the `matplot` or `plot` function.

Author(s)

Johannes Rainer

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Extract the ion chromatogram for one chromatographic peak in the data.
chrs <- chromatogram(faahko_sub, rt = c(2700, 2900), mz = 335)

plot(chrs)

## Extract chromatographic peaks for the mz/rt range (if any).
chromPeaks(faahko_sub, rt = c(2700, 2900), mz = 335)

## Highlight the chromatographic peaks in the area
## Show the peak definition with a rectangle
highlightChromPeaks(faahko_sub, rt = c(2700, 2900), mz = 335)

## Color the actual peak
highlightChromPeaks(faahko_sub, rt = c(2700, 2900), mz = 335,
    col = c("#ff000020", "#00ff0020"), type = "polygon")
```

`image-methods` *Plot log intensity image of a xcmsRaw object*

Description

Create log intensity false-color image of a xcmsRaw object plotted with m/z and retention time axes

Arguments

`x`	xcmsRaw object
`col`	vector of colors to use for for the image
`...`	arguments for `profRange`

Methods

x = "xcmsRaw" `image(x, col = rainbow(256), ...)`

Author(s)

Colin A. Smith, <`csmith@scripps.edu`>

See Also

`xcmsRaw-class`

`imputeLinInterpol` *Impute values for empty elements in a vector using linear interpolation*

Description

This function provides missing value imputation based on linear interpolation and resembles some of the functionality of the `profBinLin` and `profBinLinBase` functions deprecated from version 1.51 on.

Usage

```
imputeLinInterpol(
  x,
  baseValue,
  method = "lin",
  distance = 1L,
  noInterpolAtEnds = FALSE
)
```

Arguments

`x`	A numeric vector with eventual missing (`NA`) values.
`baseValue`	The base value to which empty elements should be set. This is only considered for `method = "linbase"` and corresponds to the `profBinLinBase`'s `baselevel` argument.
`method`	One of `"none"`, `"lin"` or `"linbase"`.
`distance`	For `method = "linbase"`: number of non-empty neighboring element of an empty element that should be considered for linear interpolation. See details section for more information.
`noInterpolAtEnds`	For `method = "lin"`: Logical indicating whether linear interpolation should also be performed at the ends of the data vector (i.e. if missing values are present at the beginning or the end of the vector).

Details

Values for NAs in input vector `x` can be imputed using methods `"lin"` and `"linbase"`:

`impute = "lin"` uses simple linear imputation to derive a value for an empty element in input vector `x` from its neighboring non-empty elements. This method is equivalent to the linear interpolation in the `profBinLin` method. Whether interpolation is performed if missing values are present at the beginning and end of `x` can be set with argument `noInterpolAtEnds`. By default interpolation is also performed at the ends interpolating from `0` at the beginning and towards `0` at the end. For `noInterpolAtEnds = TRUE` no interpolation is performed at both ends replacing the missing values at the beginning and/or the end of `x` with `0`.

`impute = "linbase"` uses linear interpolation to impute values for empty elements within a user-definable proximity to non-empty elements and setting the element's value to the `baseValue` otherwise. The default for the `baseValue` is half of the smallest value in `x` (`NA`s being removed). Whether linear interpolation based imputation is performed for a missing value depends on the `distance` argument. Interpolation is only performed if one of the next `distance` closest neighbors to the current empty element has a value other than `NA`. No interpolation takes place for `distance = 0`, while `distance = 1` means that the value for an empty element is interpolated from directly adjacent non-empty elements while, if the next neighbors of the current empty element are also `NA`, it's vale is set to `baseValue`. This corresponds to the linear interpolation performed by the `profBinLinBase` method. For more details see examples below.

Value

A numeric vector with empty values imputed based on the selected `method`.

Author(s)

Johannes Rainer

Examples

```
#######
## Impute missing values by linearly interpolating from neighboring
## non-empty elements
x <- c(3, NA, 1, 2, NA, NA, 4, NA, NA, NA, 3, NA, NA, NA, NA, 2)
imputeLinInterpol(x, method = "lin")
## visualize the interpolation:
plot(x = 1:length(x), y = x)
points(x = 1:length(x), y = imputeLinInterpol(x, method = "lin"), type = "l", col = "grey")

## If the first or last elements are NA, interpolation is performed from 0
## to the first non-empty element.
x <- c(NA, 2, 1, 4, NA)
imputeLinInterpol(x, method = "lin")
## visualize the interpolation:
plot(x = 1:length(x), y = x)
points(x = 1:length(x), y = imputeLinInterpol(x, method = "lin"), type = "l", col = "grey")

## If noInterpolAtEnds is TRUE no interpolation is performed at both ends
imputeLinInterpol(x, method = "lin", noInterpolAtEnds = TRUE)

######
## method = "linbase"
## "linbase" performs imputation by interpolation for empty elements based on
## 'distance' adjacent non-empty elements, setting all remaining empty elements
## to the baseValue
x <- c(3, NA, 1, 2, NA, NA, 4, NA, NA, NA, 3, NA, NA, NA, NA, 2)
## Setting distance = 0 skips imputation by linear interpolation
imputeLinInterpol(x, method = "linbase", distance = 0)

## With distance = 1 for all empty elements next to a non-empty element the value
## is imputed by linear interpolation.
```

```
xInt <- imputeLinInterpol(x, method = "linbase", distance = 1L)
xInt

plot(x = 1:length(x), y = x, ylim = c(0, max(x, na.rm = TRUE)))
points(x = 1:length(x), y = xInt, type = "l", col = "grey")

## Setting distance = 2L would cause that for all empty elements for which the
## distance to the next non-empty element is <= 2 the value is imputed by
## linear interpolation:
xInt <- imputeLinInterpol(x, method = "linbase", distance = 2L)
xInt

plot(x = 1:length(x), y = x, ylim = c(0, max(x, na.rm = TRUE)))
points(x = 1:length(x), y = xInt, type = "l", col = "grey")
```

`imputeRowMin` *Replace missing values with a proportion of the row minimum*

Description

`imputeRowMin` imputes missing values in `x` by replacing `NA`s in each row with a proportion of the minimal value for that row (i.e. `min_fraction * min(x[i, ])`).

Usage

```
imputeRowMin(x, min_fraction = 1/2)
```

Arguments

`x`	`matrix` with abundances, rows being features/metabolites and columns samples.
`min_fraction`	`numeric(1)` with the fraction of the row minimum that should be used to replace `NA` values in that row.

Author(s)

Johannes Rainer

See Also

`imputeLCMD` package for more left censored imputation functions.

Other imputation functions: `imputeRowMinRand()`

Examples

```
library(faahKO)
data("faahko")

xset <- group(faahko)
mat <- groupval(xset, value = "into")

mat_imp <- imputeRowMin(mat)

head(mat)
head(mat_imp)

## Replace with 1/8 of the row mimimum
head(imputeRowMin(mat, min_fraction = 1/8))
```

`imputeRowMinRand` *Impute missing values with random numbers based on the row minimum*

Description

Replace missing values with random numbers. When using the `method = "mean_sd"`, random numbers will be generated from a normal distribution based on (a fraction of) the row min and a standard deviation estimated from the linear relationship between row standard deviation and mean of the full data set. Parameter `sd_fraction` allows to further reduce the estimated standard deviation. When using the method `method = "from_to"`, random numbers between 2 specific values will be generated.

Usage

```
imputeRowMinRand(
  x,
  method = c("mean_sd", "from_to"),
  min_fraction = 1/2,
  min_fraction_from = 1/1000,
  sd_fraction = 1,
  abs = TRUE
)
```

Arguments

`x`	`matrix` with abundances, rows being features/metabolites and columns samples.
`method`	method `character(1)` defining the imputation method. See description for details. Defaults to `method = "mean_sd"`.
`min_fraction`	`numeric(1)` with the fraction of the row minimum that should be used to replace `NA` values in that row in case that `mean_sd` method is specified. When using `from_to` method, this value will be the one used to calculate the maximum value for replace `NA` values in that row.

`min_fraction_from`	`numeric(1)` with the fraction of the row minimum that should be used to calculate the minimum value for replace `NA` values in that row. This parameter is used only in case that `from_to` method is specified.
`sd_fraction`	`numeric(1)` factor to reduce the estimated standard deviation. This parameter is used only in case that `mean_sd` method is specified.
`abs`	`logical(1)` to force imputed values to be strictly positive.

Details

For method **mean_sd**, imputed values are taken from a normal distribution with mean being a user defined fraction of the row minimum and the standard deviation estimated for that mean based on the linear relationship between row standard deviations and row means in the full matrix `x`.

To largely avoid imputed values being negative or larger than the *real* values, the standard deviation for the random number generation is estimated ignoring the intercept of the linear model estimating the relationship between standard deviation and mean. If `abs = TRUE` `NA` values are replaced with the absolute value of the random values.

For method **from_to**, imputed values are taken between 2 user defined fractions of the row minimum.

Author(s)

Johannes Rainer, Mar Garcia-Aloy

See Also

`imputeLCMD` package for more left censored imputation functions.

Other imputation functions: `imputeRowMin()`

Examples

```
library(faahKO)
data("faahko")

xset <- group(faahko)
mat <- groupval(xset, value = "into")

## Estimate the relationship between row sd and mean. The standard deviation
## of the random distribution is estimated on this relationship.
mns <- rowMeans(mat, na.rm = TRUE)
sds <- apply(mat, MARGIN = 1, sd, na.rm = TRUE)
plot(mns, sds)
abline(lm(sds ~ mns))

mat_imp_meansd <- imputeRowMinRand(mat, method = "mean_sd")
mat_imp_fromto <- imputeRowMinRand(mat, method = "from_to")

head(mat)
head(mat_imp_meansd)
head(mat_imp_fromto)
```

`isolationWindowTargetMz,OnDiskMSnExp-method` *Extract isolation window target m/z definition*

Description

`isolationWindowTargetMz` extracts the isolation window target m/z definition for each spectrum in `object`.

Usage

```
## S4 method for signature 'OnDiskMSnExp'
isolationWindowTargetMz(object)
```

Arguments

`object`	OnDiskMSnExp object.

Value

a `numeric` of length equal to the number of spectra in `object` with the isolation window target m/z or `NA` if not specified/available.

Author(s)

Johannes Rainer

`levelplot-methods` *Plot log intensity image of a xcmsRaw object*

Description

Create an image of the raw (profile) data m/z against retention time, with the intensity color coded.

Arguments

`x`	xcmsRaw object.
`log`	Whether the intensity should be log transformed.
`col.regions`	The color ramp that should be used for encoding of the intensity.
`rt`	wheter the original (`rt="raw"`) or the corrected (`rt="corrected"`) retention times should be used.
`...`	Arguments for `profRange`.

Methods

x = "xcmsRaw" `levelplot(x, log=TRUE, col.regions=colorRampPalette(brewer.pal(9, "YlOrRd"))(256), ...)`

x = "xcmsSet" `levelplot(x, log=TRUE, col.regions=colorRampPalette(brewer.pal(9, "YlOrRd"))(256), rt="raw", ...)`

Author(s)

Johannes Rainer, <johannes.rainer@eurac.edu>

See Also

`xcmsRaw-class`, `xcmsSet-class`

`loadRaw-methods` *Read binary data from a source*

Description

This function extracts the raw data which will be used an `xcmsRaw` object. Further processing of data is done in the `xcmsRaw` constructor.

Arguments

`object`	Specification of a data source (such as a file name or database query)

Details

The implementing methods decide how to gather the data.

Value

A list containing elements describing the data source. The `rt`, `scanindex`, `tic`, and `acquisitionNum` components each have one entry per scan. They are *parallel* in the sense that `rt[1]`, `scanindex[1]`, and `acquisitionNum[1]` all refer to the same scan. The list containst the following components:

`rt`	Numeric vector with acquisition time (in seconds) for each scan
`tic`	Numeric vector with Total Ion Count for each scan
`scanindex`	Integer vector with starting positions of each scan in the `mz` and `intensity` components. It is an exclusive offset, so `scanindex[i]` is the offset in `mz` and `intensity` *before* the beginning of scan `i`. This means that the `mz` (respectively `intensity`) values for scan `i` would be from `scanindex[i] + 1` to `scanindex[i + 1]`
`mz`	Concatenated vector of m/z values for all scans
`intensity`	Concatenated vector of intensity values for all scans

Methods

`signature(object = "xcmsSource")` Uses `loadRaw,xcmsSource-method` to extract raw data. Subclasses of `xcmsSource` can provide different ways of fetching data.

Author(s)

Daniel Hackney, `<dan@haxney.org>`

See Also

`xcmsRaw-class`, `xcmsSource`

`manualChromPeaks` *Manual peak integration and feature definition*

Description

The `manualChromPeaks` function allows to manually define chromatographic peaks which are added to the object's `chromPeaks` matrix. In contrast to `findChromPeaks()`, no *peak detection* is performed (e.g. using an algorithm such as *centWave*) but the peak is added as defined by the user. Note that a peak will not be added if no signal (intensity) was found in a sample within the provided boundaries.

Because chromatographic peaks are added to eventually previously identified peaks, it is suggested to run `refineChromPeaks()` with the `MergeNeighboringPeaksParam()` approach to merge potentially overlapping peaks.

The `manualFeatures` function allows to manually group identified chromatographic peaks into features by providing their index in the object's `chromPeaks` matrix.

Usage

```
manualChromPeaks(
  object,
  chromPeaks = matrix(),
  samples = seq_along(fileNames(object)),
  BPPARAM = bpparam(),
  msLevel = 1L
)

manualFeatures(object, peakIdx = list(), msLevel = 1L)
```

Arguments

`object`	`XCMSnExp` or `OnDiskMSnExp` object.
`chromPeaks`	`matrix` defining the boundaries of the chromatographic peaks, one row per chromatographic peak, columns `"mzmin"`, `"mzmax"`, `"rtmin"` and `"rtmax"` defining the m/z and retention time region of each peak.

`samples`	optional `integer` to select samples in which the peak integration should be performed. By default performed in all samples.
`BPPARAM`	parallel processing settings (see `bpparam()` for details).
`msLevel`	`integer(1)` defining the MS level in which peak integration should be performed.
`peakIdx`	for `nabbyakFeatyres`: `list` of `integer` vectors with the indices of chromatographic peaks in the object's `chromPeaks` matrix that should be grouped into features.

Value

`XCMSnExp` with the manually added chromatographic peaks or features.

Author(s)

Johannes Rainer

`medianFilter` *Apply a median filter to a matrix*

Description

For each element in a matix, replace it with the median of the values around it.

Usage

```
medianFilter(x, mrad, nrad)
```

Arguments

`x`	numeric matrix to median filter
`mrad`	number of rows on either side of the value to use for median calculation
`nrad`	number of rows on either side of the value to use for median calculation

Value

A matrix whose values have been median filtered

Author(s)

Colin A. Smith, <`csmith@scripps.edu`>

Examples

```
mat <- matrix(1:25, nrow=5)
mat
medianFilter(mat, 1, 1)
```

`MergeNeighboringPeaksParam` *Merge neighboring and overlapping chromatographic peaks*

Description

Peak detection sometimes fails to identify a chromatographic peak correctly, especially for broad peaks and if the peak shape is irregular (mostly for HILIC data). In such cases several smaller peaks are reported. Also, peak detection can result in partially or completely overlapping peaks. To reduce such peak detection artifacts, this function merges chromatographic peaks which are overlapping or close in rt and m/z dimension considering also the measured signal intensities in the region between them.

Chromatographic peaks are first expanded in m/z and retention time dimension (based on parameters `expandMz`, `ppm` and `expandRt`) and subsequently grouped into sets of merge candidates if they are (after expansion) overlapping in both m/z and rt (within the same sample). Candidate peaks are merged if the average intensity of the 3 data points in the middle position between them (i.e. at half the distance between `"rtmax"` of the first and `"rtmin"` of the second peak) is larger than a certain proportion (`minProp`) of the smaller maximal intensity (`"maxo"`) of both peaks. In cases in which this calculated mid point is **not** located between the apexes of the two peaks (e.g. if the peaks are largely overlapping) the average signal intensity at half way between the apexes is used instead. Candidate peaks are not joined if all 3 data points between them have `NA` intensities. The joined peaks get the `"mz"`, `"rt"`, `"sn"` and `"maxo"` values from the peak with the largest signal (`"maxo"`) as well as its row in the metadata data frame of the peak (`chromPeakData`). The `"rtmin"`, `"rtmax"` of the merged peaks are updated and `"into"` is recalculated based on all the signal between `"rtmin"` and `"rtmax"` of the new merged peak. See details for information on the `"mzmin"` and `"mzmax"` values of the merged peak.

Usage

```
MergeNeighboringPeaksParam(
  expandRt = 2,
  expandMz = 0,
  ppm = 10,
  minProp = 0.75
)

## S4 method for signature 'XCMSnExp,MergeNeighboringPeaksParam'
refineChromPeaks(
  object,
  param = MergeNeighboringPeaksParam(),
  msLevel = 1L,
  BPPARAM = bpparam()
)
```

Arguments

`expandRt`	`numeric(1)` defining by how many seconds the retention time window is expanded on both sides to check for overlapping peaks.
`expandMz`	`numeric(1)` constant value by which the m/z range of each chromatographic peak is expanded (on both sides!) to check for overlapping peaks.
`ppm`	`numeric(1)` defining a m/z relative value (in parts per million) by which the m/z range of each chromatographic peak is expanded to check for overlapping peaks.
`minProp`	`numeric(1)` between `0` and `1` representing the proporion of intensity to be required for peaks to be joined. See description for more details. The default (`minProp = 0.75`) means that peaks are only joined if the signal half way between then is larger 75% of the smallest of the two peak's `"maxo"` (maximal intensity at peak apex).
`object`	XCMSnExp object with identified chromatographic peaks.
`param`	`MergeNeighboringPeaksParam` object defining the settings for the method.
`msLevel`	`integer` defining for which MS level(s) the chromatographic peaks should be merged.
`BPPARAM`	parameter object to set up parallel processing. Uses the default parallel processing setup returned by `bpparam()`. See `bpparam()` for details and examples.

Details

For each set of candidate peaks an ion chromatogram is extracted using the range of retention times and m/z values of these peaks. The m/z range for the extracted ion chromatogram is expanded by `expandMz` and `ppm` (on both sides) to reduce the possibility of missing signal intensities between candidate peaks (variance of measured m/z values for lower intensities is larger than for higher intensities and thus data points not being part of identified chromatographic peaks tend to have m/z values outside of the m/z range of the candidate peaks - especially for ToF instruments). This also ensures that all data points from the same ion are considered for the peak integration of merged peaks. The smallest and largest m/z value of all data points used in the peak integration of the merged peak are used as the merged peak's m/z range (i.e. columns `"mzmin"` and `"mzmax"`).

Value

`XCMSnExp` object with chromatographic peaks matching the defined conditions being merged.

Note

Note that **each** peak gets expanded by `expandMz` and `expandRt`, thus peaks differing by `2 * expandMz` (or `expandRt`) will be identified as *overlapping*. As an example: m/z max of one peak is 12.2, m/z min of another one is 12.4, if `expandMz = 0.1` the m/z max of the first peak will be 12.3 and the m/z min of the second one 12.3, thus both are considered overlapping.

`refineChromPeaks` methods will always remove feature definitions, because a call to this method can change or remove identified chromatographic peaks, which may be part of features.

Merging of chromatographic peaks is performed along the retention time axis, i.e. candidate peaks are first ordered by their `"rtmin"` value. The signals at half way between the first and the second candidate peak are then compared to the smallest `"maxo"` of both and the two peaks are then merged if the average signal between the peaks is larger `minProp`. For merging any additional peak in a candidate peak list the `"maxo"` of that peak and the newly merged peak are considered.

Author(s)

Johannes Rainer, Mar Garcia-Aloy

See Also

Other chromatographic peak refinement methods: `CleanPeaksParam`, `FilterIntensityParam`

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Subset to a single file
xd <- filterFile(faahko_sub, file = 1)

## Example of a split peak that will be merged
mzr <- 305.1 + c(-0.01, 0.01)
chr <- chromatogram(xd, mz = mzr, rt = c(2700, 3700))
plot(chr)

## Combine the peaks
res <- refineChromPeaks(xd, param = MergeNeighboringPeaksParam(expandRt = 4))
chr_res <- chromatogram(res, mz = mzr, rt = c(2700, 3700))
plot(chr_res)

## Example of a peak that was not merged, because the signal between them
## is lower than the cut-off minProp
mzr <- 496.2 + c(-0.01, 0.01)
chr <- chromatogram(xd, mz = mzr, rt = c(3200, 3500))
plot(chr)
chr_res <- chromatogram(res, mz = mzr, rt = c(3200, 3500))
plot(chr_res)
```

`msn2xcmsRaw` *Copy MSn data in an xcmsRaw to the MS slots*

Description

The MS2 and MSn data is stored in separate slots, and can not directly be used by e.g. findPeaks(). `msn2xcmsRaw()` will copy the MSn spectra into the "normal" `xcmsRaw` slots.

Usage

```
msn2xcmsRaw(xmsn)
```

Arguments

`xmsn`	an object of class `xcmsRaw` that contains spectra read with includeMSn=TRUE

Details

The default gap value is determined from the 90th percentile of the pair-wise differences between adjacent mass values.

Value

An xcmsRaw object

Author(s)

Steffen Neumann <`sneumann@ipb-halle.de`>

See Also

`xcmsRaw`,

Examples

```
msnfile <- system.file("microtofq/MSMSpos20_6.mzML", package = "msdata")
xrmsn <- xcmsRaw(msnfile, includeMSn=TRUE)
xr <- msn2xcmsRaw(xrmsn)
p <- findPeaks(xr, method="centWave")
```

`overlappingFeatures` *Identify overlapping features*

Description

`overlappingFeatures` identifies features that are overlapping or close in the m/z - rt space.

Usage

```
overlappingFeatures(x, expandMz = 0, expandRt = 0, ppm = 0)
```

Arguments

`x`	`XCMSnExp` with the features.
`expandMz`	`numeric(1)` with the value to expand each feature (on each side) in m/z dimension before identifying overlapping features. The resulting `"mzmin"` for the feature is thus `mzmin - expandMz` and the `"mzmax"` `mzmax + expandMz`.
`expandRt`	`numeric(1)` with the value to expand each feature (on each side) in retention time dimension before identifying overlapping features. The resulting `"rtmin"` for the feature is thus `rtmin - expandRt` and the `"rtmax"` `rtmax + expandRt`.
`ppm`	`numeric(1)` to grow the m/z width of the feature by a relative value: `mzmin - mzmin * ppm / 2e6`, `mzmax + mzmax * ppm / 2e6`. Each feature is thus expanded in m/z dimension by ppm/2 on each side before identifying overlapping features.

Value

`list` with indices of features (in `featureDefinitions()`) that are overlapping.

Author(s)

Johannes Rainer

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Correspondence analysis
xdata <- groupChromPeaks(faahko_sub, param = PeakDensityParam(sampleGroups = c(1, 1, 1)))

## Identify overlapping features
overlappingFeatures(xdata)

## Identify features that are separated on retention time by less than
## 2 minutes
overlappingFeatures(xdata, expandRt = 60)
```

`peakPlots-methods` *Plot a grid of a large number of peaks*

Description

Plot extracted ion chromatograms for many peaks simultaneously, indicating peak integration start and end points with vertical grey lines.

Arguments

`object`	the `xcmsRaw` object
`peaks`	matrix with peak information as produced by `findPeaks`
`figs`	two-element vector describing the number of rows and the number of columns of peaks to plot, if missing then an approximately square grid that will fit the number of peaks supplied
`width`	width of chromatogram retention time to plot for each peak

Details

This function is intended to help graphically analyze the results of peak picking. It can help estimate the number of false positives and improper integration start and end points. Its output is very compact and tries to waste as little space as possible. Each plot is labeled with rounded m/z and retention time separated by a space.

Methods

`signature(object = "xcmsSet") plotPeaks(object, peaks, figs, width = 200)`

See Also

`xcmsRaw-class`, `findPeaks`, `split.screen`

`peaksWithCentWave` *Identify peaks in chromatographic data using centWave*

Description

`peaksWithCentWave` identifies (chromatographic) peaks in purely chromatographic data, i.e. based on intensity and retention time values without m/z values.

Usage

```
peaksWithCentWave(
  int,
  rt,
  peakwidth = c(20, 50),
  snthresh = 10,
  prefilter = c(3, 100),
  integrate = 1,
  fitgauss = FALSE,
  noise = 0,
  verboseColumns = FALSE,
  firstBaselineCheck = TRUE,
  extendLengthMSW = FALSE,
  ...
)
```

Arguments

`int`	`numeric` with intensity values.
`rt`	`numeric` with the retention time for the intensities. Length has to be equal to `length(int)`.
`peakwidth`	`numeric(2)` with the lower and upper bound of the expected peak width.
`snthresh`	`numeric(1)` defining the signal to noise ratio cutoff. Peaks with a signal to noise ratio < `snthresh` are omitted.
`prefilter`	`numeric(2)` (`c(k, I)`): only regions of interest with at least `k` centroids with signal `>= I` are returned in the first step.
`integrate`	`numeric(1)`, integration method. For `integrate = 1` peak limits are found through descending on the mexican hat filtered data, for `integrate = 2` the descend is done on the real data. The latter method is more accurate but prone to noise, while the former is more robust, but less exact.
`fitgauss`	`logical(1)` whether or not a Gaussian should be fitted to each peak.
`noise`	`numeric(1)` defining the minimum required intensity for centroids to be considered in the first analysis step (definition of the *regions of interest*).
`verboseColumns`	`logical(1)`: whether additional peak meta data columns should be returned.
`firstBaselineCheck`	`logical(1)`. If `TRUE` continuous data within regions of interest is checked to be above the first baseline.
`extendLengthMSW`	`logical(1)`. If `TRUE` the "open" method of EIC extension is used, rather than the default "reflect" method.
`...`	currently ignored.

Details

The method uses the same algorithm for the peak detection than centWave, employs however a different approach to identify the initial regions in which the peak detection is performed (i.e. the *regions of interest* ROI). The method first identifies all local maxima in the chromatographic data and defines the corresponding positions +/- `peakwidth[2]` as the ROIs. Noise estimation bases also on these ROIs and can thus be different from centWave resulting in different signal to noise ratios.

Value

A matrix, each row representing an identified chromatographic peak, with columns:

- `"rt"`: retention time of the peak's midpoint (time of the maximum signal).
- `"rtmin"`: minimum retention time of the peak.
- `"rtmax"`: maximum retention time of the peak.
- `"into"`: integrated (original) intensity of the peak.
- `"intb"`: per-peak baseline corrected integrated peak intensity.
- `"maxo"`: maximum (original) intensity of the peak.

- `"sn"`: signal to noise ratio of the peak defined as `(maxo - baseline)/sd` with `sd` being the standard deviation of the local chromatographic noise.

Additional columns for `verboseColumns = TRUE`:

- `"mu"`: gaussian parameter mu.
- `"sigma"`: gaussian parameter sigma.
- `"h"`: gaussian parameter h.
- `"f"`: region number of the m/z ROI where the peak was localized.
- `"dppm"`: m/z deviation of mass trace across scans in ppm (always `NA`).
- `"scale"`: scale on which the peak was localized.
- `"scpos"`: peak position found by wavelet analysis (index in `int`).
- `"scmin"`: left peak limit found by wavelet analysis (index in `int`).
- `"scmax"`: right peak limit found by wavelet analysis (index in `int`).

Author(s)

Johannes Rainer

See Also

centWave for a detailed description of the peak detection method.

Other peak detection functions for chromatographic data: `peaksWithMatchedFilter()`

Examples

```
## Reading a file
library(xcms)
od <- readMSData(system.file("cdf/KO/ko15.CDF", package = "faahKO"),
    mode = "onDisk")

## Extract chromatographic data for a small m/z range
mzr <- c(272.1, 272.2)
chr <- chromatogram(od, mz = mzr, rt = c(3000, 3300))[1, 1]

int <- intensity(chr)
rt <- rtime(chr)

## Plot the region
plot(chr, type = "h")

## Identify peaks in the chromatographic data
pks <- peaksWithCentWave(intensity(chr), rtime(chr))
pks

## Highlight the peaks
rect(xleft = pks[, "rtmin"], xright = pks[, "rtmax"],
    ybottom = rep(0, nrow(pks)), ytop = pks[, "maxo"], col = "#ff000040",
    border = "#00000040")
```

`peaksWithMatchedFilter`

Identify peaks in chromatographic data using matchedFilter

Description

The function performs peak detection using the matchedFilter algorithm on chromatographic data (i.e. with only intensities and retention time).

Usage

```
peaksWithMatchedFilter(
  int,
  rt,
  fwhm = 30,
  sigma = fwhm/2.3548,
  max = 20,
  snthresh = 10,
  ...
)
```

Arguments

`int`	`numeric` with intensity values.
`rt`	`numeric` with the retention time for the intensities. Length has to be equal to `length(int)`.
`fwhm`	`numeric(1)` specifying the full width at half maximum of matched filtration gaussian model peak. Only used to calculate the actual sigma, see below.
`sigma`	`numeric(1)` specifying the standard deviation (width) of the matched filtration model peak.
`max`	`numeric(1)` with the maximal number of peaks that are expected/ will bbe detected in the data
`snthresh`	`numeric(1)` defining the signal to noise cut-off to be used in the peak detection step.
`...`	currently ignored.

Value

A matrix, each row representing an identified chromatographic peak, with columns:

- `"rt"`: retention time of the peak's midpoint (time of the maximum signal).
- `"rtmin"`: minimum retention time of the peak.
- `"rtmax"`: maximum retention time of the peak.
- `"into"`: integrated (original) intensity of the peak.

- `"intf"`: integrated intensity of the filtered peak.
- `"maxo"`: maximum (original) intensity of the peak.
- `"maxf""` maximum intensity of the filtered peak.
- `"sn"`: signal to noise ratio of the peak.

Author(s)

Johannes Rainer

See Also

matchedFilter for a detailed description of the peak detection method.

Other peak detection functions for chromatographic data: `peaksWithCentWave()`

Examples

```
## Load the test file
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Subset to one file and drop identified chromatographic peaks
data <- dropChromPeaks(filterFile(faahko_sub, 1))

## Extract chromatographic data for a small m/z range
chr <- chromatogram(data, mz = c(272.1, 272.3), rt = c(3000, 3200))[1, 1]

pks <- peaksWithMatchedFilter(intensity(chr), rtime(chr))
pks

## Plotting the data
plot(rtime(chr), intensity(chr), type = "h")
rect(xleft = pks[, "rtmin"], xright = pks[, "rtmax"], ybottom = c(0, 0),
    ytop = pks[, "maxo"], border = "red")
```

`peakTable-methods` *Create report of aligned peak intensities*

Description

Create a report showing all aligned peaks.

Arguments

`object`	the `xcmsSet` object
`filebase`	base file name to save report, `.tsv` file and `_eic` will be appended to this name for the tabular report and EIC directory, respectively. if blank nothing will be saved
`...`	arguments passed down to `groupval`, which provides the actual intensities.

Details

This method handles creation of summary reports similar to `diffreport`. It returns a summary report that can optionally be written out to a tab-separated file.

If a base file name is provided, the report (see Value section) will be saved to a tab separated file.

Value

A data frame with the following columns:

`mz`	median m/z of peaks in the group
`mzmin`	minimum m/z of peaks in the group
`mzmax`	maximum m/z of peaks in the group
`rt`	median retention time of peaks in the group
`rtmin`	minimum retention time of peaks in the group
`rtmax`	maximum retention time of peaks in the group
`npeaks`	number of peaks assigned to the group
`Sample Classes`	number samples from each sample class represented in the group
`...`	one column for every sample class
`Sample Names`	integrated intensity value for every sample
`...`	one column for every sample

Methods

object = "xcmsSet" `peakTable(object, filebase = character(), ...)`

See Also

`xcmsSet-class`,

Examples

```
## Not run:
library(faahKO)
cdfpath <- system.file("cdf", package = "faahKO")
cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)
xs<-xcmsSet(cdf files)
xs<-group(xs)
peakTable(xs, filebase="peakList")

## End(Not run)
```

`phenoDataFromPaths` *Derive experimental design from file paths*

Description

The `phenoDataFromPaths` function builds a `data.frame` representing the experimental design from the folder structure in which the files of the experiment are located.

Usage

```
phenoDataFromPaths(paths)
```

Arguments

`paths`	`character` representing the file names (including the full path) of the experiment's files.

Note

This function is used by the *old* `xcmsSet` function to guess the experimental design (i.e. group assignment of the files) from the folders in which the files of the experiment can be found.

Examples

```
## List the files available in the faahKO package
base_dir <- system.file("cdf", package = "faahKO")
cdf_files <- list.files(base_dir, recursive = TRUE, full.names = TRUE)
```

`plot.xcmsEIC` *Plot extracted ion chromatograms from multiple files*

Description

Batch plot a list of extracted ion chromatograms to the current graphics device.

Arguments

`x`	the `xcmsEIC` object
`y`	optional `xcmsSet` object with peak integration data
`groupidx`	either character vector with names or integer vector with indicies of peak groups for which to plot EICs
`sampleidx`	either character vector with names or integer vector with indicies of samples for which to plot EICs

`rtrange`	a two column matrix with minimum and maximum retention times between which to return EIC data points if it has the same number of rows as the number groups in the `xcmsEIC` object, then `sampleidx` is used to subset it. otherwise, it is repeated over the length of `sampleidx` it may also be a single number specifying the time window around the peak for which to plot EIC data
`col`	color to use for plotting extracted ion chromatograms. if missing and `y` is specified, colors are taken from `unclass(sampclass(y))` and the default palette if it is the same length as the number groups in the `xcmsEIC` object, then `sampleidx` is used to subset it. otherwise, it is repeated over the length of `sampleidx`
`legtext`	text to use for legend. if `NULL` and `y` is specified, legend text is taken from the sample class information found in the `xcmsSet`
`peakint`	logical, plot integrated peak area with darkened lines (requires that `y` also be specified)
`sleep`	seconds to pause between plotting EICs
`...`	other graphical parameters

Value

A `xcmsSet` object.

Methods

x = "xcmsEIC" `plot.xcmsEIC(x, y, groupidx = groupnames(x), sampleidx = sampnames(x), rtrange = x@rtrange, col = rep(1, length(sampleidx)), legtext = NULL, peakint = TRUE, sleep = 0, ...)`

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`xcmsEIC-class`, `png`, `pdf`, `postscript`,

`plotAdjustedRtime` *Visualization of alignment results*

Description

Plot the difference between the adjusted and the raw retention time (y-axis) for each file along the (adjusted or raw) retention time (x-axis). If alignment was performed using the `adjustRtime-peakGroups` method, also the features (peak groups) used for the alignment are shown.

Usage

```
plotAdjustedRtime(
  object,
  col = "#00000080",
  lty = 1,
  lwd = 1,
  type = "l",
  adjustedRtime = TRUE,
 xlab = ifelse(adjustedRtime, yes = expression(rt[adj]), no = expression(rt[raw])),
  ylab = expression(rt[adj] - rt[raw]),
  peakGroupsCol = "#00000060",
  peakGroupsPch = 16,
  peakGroupsLty = 3,
  ylim,
  ...
)
```

Arguments

`object`	A `XCMSnExp` object with the alignment results.
`col`	colors to be used for the lines corresponding to the individual samples.
`lty`	line type to be used for the lines of the individual samples.
`lwd`	line width to be used for the lines of the individual samples.
`type`	plot type to be used. See help on the `par` function for supported values.
`adjustedRtime`	logical(1) whether adjusted or raw retention times should be shown on the x-axis.
`xlab`	the label for the x-axis.
`ylab`	the label for the y-axis.
`peakGroupsCol`	color to be used for the peak groups (only used if alignment was performed using the `adjustRtime-peakGroups` method.
`peakGroupsPch`	point character (`pch`) to be used for the peak groups (only used if alignment was performed using the `adjustRtime-peakGroups` method.
`peakGroupsLty`	line type (`lty`) to be used to connect points for each peak groups (only used if alignment was performed using the `adjustRtime-peakGroups` method.
`ylim`	optional `numeric(2)` with the upper and lower limits on the y-axis.
`...`	Additional arguments to be passed down to the `plot` function.

Author(s)

Johannes Rainer

See Also

`adjustRtime` for all retention time correction/ alignment methods.

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## Performing the peak grouping using the "peak density" method.
p <- PeakDensityParam(sampleGroups = c(1, 1, 1))
res <- groupChromPeaks(faahko_sub, param = p)

## Perform the retention time adjustment using peak groups found in both
## files.
fgp <- PeakGroupsParam(minFraction = 1)
res <- adjustRtime(res, param = fgp)

## Visualize the impact of the alignment.
plotAdjustedRtime(res, adjusted = FALSE)
grid()
```

`plotChrom-methods` *Plot extracted ion chromatograms from the profile matrix*

Description

Uses the pre-generated profile mode matrix to plot averaged or base peak extracted ion chromatograms over a specified mass range.

Arguments

`object`	the `xcmsRaw` object
`base`	logical, plot a base-peak chromatogram
`ident`	logical, use mouse to identify and label peaks
`fitgauss`	logical, fit a gaussian to the largest peak
`vline`	numeric vector with locations of vertical lines
`...`	arguments passed to `profRange`

Value

If `ident == TRUE`, an integer vector with the indecies of the points that were identified. If `fitgauss == TRUE`, a `nls` model with the fitted gaussian. Otherwise a two-column matrix with the plotted points.

Methods

object = "xcmsRaw" `plotChrom(object, base = FALSE, ident = FALSE, fitgauss = FALSE, vline = numeric(0), ...)`

See Also

`xcmsRaw-class`

`plotChromatogramsOverlay`

Plot multiple chromatograms into the same plot

Description

`plotOverlay` draws chromatographic peak data from multiple (different) extracted ion chromatograms (EICs) into the same plot. This allows to directly compare the peak shape of these EICs in the same sample. In contrast to the `plot` function for `MChromatograms()` object, which draws the data from the same EIC across multiple samples in the same plot, this function draws the different EICs from the same sample into the same plot.

If `plotChromatogramsOverlay` is called on a `XChromatograms` object any present chromatographic peaks will also be highlighted/drawn depending on the parameters `peakType`, `peakCol`, `peakBg` and `peakPch` (see also help on the `plot` function for `XChromatogram()` object for details).

Usage

```
## S4 method for signature 'MChromatograms'
plotChromatogramsOverlay(
  object,
  col = "#00000060",
  type = "l",
  main = NULL,
  xlab = "rtime",
  ylab = "intensity",
  xlim = numeric(),
  ylim = numeric(),
  stacked = 0,
  transform = identity,
  ...
)

## S4 method for signature 'XChromatograms'
plotChromatogramsOverlay(
  object,
  col = "#00000060",
  type = "l",
  main = NULL,
```

```
  xlab = "rtime",
  ylab = "intensity",
  xlim = numeric(),
  ylim = numeric(),
  peakType = c("polygon", "point", "rectangle", "none"),
  peakBg = NULL,
  peakCol = NULL,
  peakPch = 1,
  stacked = 0,
  transform = identity,
  ...
)
```

Arguments

`object`	`MChromatograms()` or `XChromatograms()` object.
`col`	definition of the color in which the chromatograms should be drawn. Can be of length 1 or equal to `nrow(object)` to plot each overlayed chromatogram in a different color.
`type`	`character(1)` defing the type of the plot. By default (`type = "l"`) each chromatogram is drawn as a line.
`main`	optional title of the plot. If not defined, the range of m/z values is used.
`xlab`	`character(1)` defining the x-axis label.
`ylab`	`character(1)` defining the y-axis label.
`xlim`	optional `numeric(2)` defining the x-axis limits.
`ylim`	optional `numeric(2)` defining the y-axis limits.
`stacked`	`numeric(1)` defining the part (proportion) of the y-axis to use to *stack* EICs depending on their m/z values. If `stacked = 0` (the default) no stacking is performed. With `stacked = 1` half of the y-axis is used for stacking and half for the intensity y-axis (i.e. the ratio between stacking and intensity y-axis is 1:1). Note that if `stacking` is different from 0 no y-axis and label are drawn.
`transform`	`function` to transform the intensity values before plotting. Defaults to `transform = identity` which plots the data as it is. With `transform = log10` intensity values would be log10 transformed before plotting.
`...`	optional arguments to be passed to the plotting functions (see help on the base R `plot` function.
`peakType`	if `object` is a `XChromatograms` object: how chromatographic peaks should be drawn: `peakType = "polygon"` (the default): label the full chromatographic peak area, `peakType = "rectangle"`: indicate the chromatographic peak by a rectangle and `peakType = "point"`: label the chromatographic peaks' apex position with a point.
`peakBg`	if `object` is a `XChromatograms` object: definition of background color(s) for each chromatographic peak. Has to be either of length 1 or equal to the number of peaks in `object`. If not specified, the peak will be drawn in the color defined by `col`.

`peakCol`	if `object` is a `XChromatograms` object: definition of color(s) for each chromatographic peak. Has to be either of length 1 or equal to the number of peaks in `object`. If not specified, the peak will be drawn in the color defined by `col`.
`peakPch`	if `object` is a `XChromatograms` object: *point character* to be used to label the apex position of the chromatographic peak if `peakType = "point"`.

Value

silently returns a `list` (length equal to `ncol(object)` of `numeric` (length equal to `nrow(object)`) with the y position of each EIC.

Author(s)

Johannes Rainer

Examples

```
## Load preprocessed data and extract EICs for some features.
library(xcms)
data(xdata)
## Update the path to the files for the local system
dirname(xdata) <- c(rep(system.file("cdf", "KO", package = "faahKO"), 4),
                    rep(system.file("cdf", "WT", package = "faahKO"), 4))
## Subset to the first 3 files.
xdata <- filterFile(xdata, 1:3, keepFeatures = TRUE)

## Define features for which to extract EICs
fts <- c("FT097", "FT163", "FT165")
chrs <- featureChromatograms(xdata, features = fts)

plotChromatogramsOverlay(chrs)

## plot the overlay of EICs in the first sample
plotChromatogramsOverlay(chrs[, 1])

## Define a different color for each feature (row in chrs). By default, also
## all chromatographic peaks of a feature is labeled in the same color.
plotChromatogramsOverlay(chrs[, 1],
    col = c("#ff000040", "#00ff0040", "#0000ff40"))

## Alternatively, we can define a color for each individual chromatographic
## peak and provide this with the `peakBg` and `peakCol` parameters.
chromPeaks(chrs[, 1])

## Use a color for each of the two identified peaks in that sample
plotChromatogramsOverlay(chrs[, 1],
    col = c("#ff000040", "#00ff0040", "#0000ff40"),
    peakBg = c("#ffff0020", "#00ffff20"))

## Plotting the data in all samples.
plotChromatogramsOverlay(chrs,
    col = c("#ff000040", "#00ff0040", "#0000ff40"))
```

```
## Creating a "stacked" EIC plot: the EICs are placed along the y-axis
## relative to their m/z value. With `stacked = 1` the y-axis is split in
## half, the lower half being used for the stacking of the EICs, the upper
## half being used for the *original* intensity axis.
res <- plotChromatogramsOverlay(chrs[, 1], stacked = 1,
    col = c("#ff000040", "#00ff0040", "#0000ff40"))
## add horizontal lines for the m/z values of each EIC
abline(h = res[[1]], col = "grey", lty = 2)

## Note that this type of visualization is different than the conventional
## plot function for chromatographic data, which will draw the EICs for
## multiple samples into the same plot
plot(chrs)

## Converting the object to a MChromatograms without detected peaks
chrs <- as(chrs, "MChromatograms")

plotChromatogramsOverlay(chrs,
    col = c("#ff000040", "#00ff0040", "#0000ff40"))
```

`plotChromPeakDensity,XCMSnExp-method`
Plot chromatographic peak density along the retention time axis

---

Description

Plot the density of chromatographic peaks along the retention time axis and indicate which peaks would be (or were) grouped into the same feature based using the *peak density* correspondence method. Settings for the *peak density* method can be passed with an PeakDensityParam object to parameter `param`. If the `object` contains correspondence results and the correspondence was performed with the *peak groups* method, the results from that correspondence can be visualized setting `simulate = FALSE`.

Usage

```
## S4 method for signature 'XCMSnExp'
plotChromPeakDensity(
  object,
  mz,
  rt,
  param,
  simulate = TRUE,
  col = "#00000080",
  xlab = "retention time",
  ylab = "sample",
  xlim = range(rt),
  main = NULL,
```

```
  type = c("any", "within", "apex_within"),
  ...
)
```

Arguments

`object`	A XCMSnExp object with identified chromatographic peaks.
`mz`	`numeric(2)` defining an mz range for which the peak density should be plotted.
`rt`	`numeric(2)` defining an optional rt range for which the peak density should be plotted. Defaults to the absolute retention time range of `object`.
`param`	PeakDensityParam from which parameters for the *peak density* correspondence algorithm can be extracted. If not provided and if `object` contains feature definitions with the correspondence/ peak grouping being performed by the *peak density* method, the corresponding parameter class stored in `object` is used.
`simulate`	`logical(1)` defining whether correspondence should be simulated within the specified m/z / rt region or (with `simulate = FALSE`) whether the results from an already performed correspondence should be shown.
`col`	Color to be used for the individual samples. Length has to be 1 or equal to the number of samples in `object`.
`xlab`	`character(1)` with the label for the x-axis.
`ylab`	`character(1)` with the label for the y-axis.
`xlim`	`numeric(2)` representing the limits for the x-axis. Defaults to the range of the `rt` parameter.
`main`	`character(1)` defining the title of the plot. By default (for `main = NULL`) the mz-range is used.
`type`	`character(1)` specifying how peaks are called to be located within the region defined by `mz` and `rt`. Can be one of `"any"`, `"within"`, and `"apex_within"` for all peaks that are even partially overlapping the region, peaks that are completely within the region, and peaks for which the apex is within the region. This parameter is passed to the chromPeaks function. See related documentation for more information and examples.
`...`	Additional parameters to be passed to the `plot` function. Data point specific parameters such as `bg` or `pch` have to be of length 1 or equal to the number of samples.

Details

The `plotChromPeakDensity` function allows to evaluate different settings for the *peak density* on an mz slice of interest (e.g. containing chromatographic peaks corresponding to a known metabolite). The plot shows the individual peaks that were detected within the specified `mz` slice at their retention time (x-axis) and sample in which they were detected (y-axis). The density function is plotted as a black line. Parameters for the `density` function are taken from the `param` object. Grey rectangles indicate which chromatographic peaks would be grouped into a feature by the `peak density` correspondence method. Parameters for the algorithm are also taken from `param`. See `groupChromPeaks-density()` for more information about the algorithm and its supported settings.

Value

The function is called for its side effect, i.e. to create a plot.

Author(s)

Johannes Rainer

See Also

`groupChromPeaks-density()` for details on the *peak density* correspondence method and supported settings.

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Plot the chromatographic peak density for a specific mz range to evaluate
## different peak density correspondence settings.
mzr <- c(305.05, 305.15)

plotChromPeakDensity(faahko_sub, mz = mzr, pch = 16,
    param = PeakDensityParam(sampleGroups = rep(1, length(fileNames(faahko_sub)))))
```

`plotChromPeaks` *General visualizations of peak detection results*

Description

`plotChromPeaks` plots the identified chromatographic peaks from one file into the plane spanned by the retention time and mz dimension (x-axis representing the retention time and y-axis mz). Each chromatographic peak is plotted as a rectangle representing its width in rt and mz dimension.

This plot is supposed to provide some initial overview of the chromatographic peak detection results.

`plotChromPeakImage` plots the number of detected peaks for each sample along the retention time axis as an *image* plot, i.e. with the number of peaks detected in each bin along the retention time represented with the color of the respective cell.

Usage

```
plotChromPeaks(
  x,
  file = 1,
  xlim = NULL,
```

```
    ylim = NULL,
    add = FALSE,
    border = "#00000060",
    col = NA,
    xlab = "retention time",
    ylab = "mz",
    main = NULL,
    ...
  )

  plotChromPeakImage(
    x,
    binSize = 30,
    xlim = NULL,
    log = FALSE,
    xlab = "retention time",
    yaxt = par("yaxt"),
    main = "Chromatographic peak counts",
    ...
  )
```

Arguments

`x`	`XCMSnExp` object.
`file`	For `plotChromPeaks`: `numeric(1)` specifying the index of the file within `x` for which the plot should be created. Defaults to `1`.
`xlim`	`numeric(2)` specifying the x-axis limits (retention time dimension). Defaults to `NULL` in which case the full retention time range of the file is used.
`ylim`	For `plotChromPeaks`: `numeric(2)` specifying the y-axis limits (mz dimension). Defaults to `NULL` in which case the full mz range of the file is used.
`add`	For `plotChromPeaks`: `logical(1)` whether the plot should be added or created as a new plot.
`border`	For `plotChromPeaks`: the color for the rectangles' border.
`col`	For `plotChromPeaks`: the color to be used to fill the rectangles.
`xlab`	`character(1)` defining the x-axis label.
`ylab`	For `plotChromPeaks`: `character(1)` defining the y-axis label.
`main`	`character(1)` defining the plot title. By default (i.e. `main = NULL` the name of the file will be used as title.
`...`	Additional arguments passed to the `plot` (for `plotChromPeaks`) and `image` (for `plotChromPeakImage`) functions. Ignored if `add = TRUE`.
`binSize`	For `plotChromPeakImage`: `numeric(1)` defining the size of the bins along the x-axis (retention time). Defaults to `binSize = 30`, peaks within each 30 seconds will thus counted and plotted.
`log`	For `plotChromPeakImage`: `logical(1)` whether the peak counts should be log2 transformed before plotting.

`yaxt`	For `plotChromPeakImage`: `character(1)` defining whether y-axis labels should be added. To disable the y-axis use `yaxt = "n"`. For any other value of `yaxt` the axis will be drawn. See `par` help page for more details.

Details

The width and line type of the rectangles indicating the detected chromatographic peaks for the `plotChromPeaks` function can be specified using the `par` function, i.e. with `par(lwd = 3)` and `par(lty = 2)`, respectively.

Author(s)

Johannes Rainer

See Also

`highlightChromPeaks` for the function to highlight detected chromatographic peaks in extracted ion chromatogram plots.

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## plotChromPeakImage: plot an image for the identified peaks per file
plotChromPeakImage(faahko_sub)

## Show all detected chromatographic peaks from the first file
plotChromPeaks(faahko_sub)

## Plot all detected peaks from the second file and restrict the plot to a
## mz-rt slice
plotChromPeaks(faahko_sub, file = 2, xlim = c(3500, 3600), ylim = c(400, 600))
```

`plotEIC-methods` *Plot extracted ion chromatograms for specified m/z range*

Description

Plot extracted ion chromatogram for m/z values of interest. The raw data is used in contrast to `plotChrom` which uses data from the profile matrix.

Arguments

`object`	`xcmsRaw` object
`mzrange`	m/z range for EIC. Uses the full m/z range by default.
`rtrange`	retention time range for EIC. Uses the full retention time range by default.
`scanrange`	scan range for EIC
`mzdec`	Number of decimal places of title m/z values in the eic plot.
`type`	Speficies how the data should be plotted (by default as a line).
`add`	If the EIC should be added to an existing plot.
`...`	Additional parameters passed to the plotting function (e.g. `col` etc).

Value

A two-column matrix with the plotted points.

Methods

object = "xcmsRaw" `plotEIC(object, mzrange = numeric(), rtrange = numeric(), scanrange = numeric(), mzdec=2, type="l", add=FALSE, ...)`

Author(s)

Ralf Tautenhahn

See Also

`rawEIC`, `xcmsRaw-class`

`plotFeatureGroups` *Plot feature groups in the m/z-retention time space*

Description

`plotFeatureGroups` visualizes defined feature groups in the m/z by retention time space. Features are indicated by points with features from the same feature group being connected by a line. See `featureGroups()` for details on and options for feature grouping.

Usage

```
plotFeatureGroups(
  x,
  xlim = numeric(),
  ylim = numeric(),
  xlab = "retention time",
  ylab = "m/z",
  pch = 4,
```

```
  col = "#00000060",
  type = "o",
  main = "Feature groups",
  featureGroups = character()
)
```

Arguments

`x`	`XCMSnExp()` object with grouped features (i.e. after calling `groupFeatures()`.
`xlim`	`numeric(2)` with the lower and upper limit for the x-axis.
`ylim`	`numeric(2)` with the lower and upper limit for the y-axis.
`xlab`	`character(1)` with the label for the x-axis.
`ylab`	`character(1)` with the label for the y-axis.
`pch`	the plotting character. Defaults to `pch = 4` i.e. plotting features as crosses. See `par()` for more information.
`col`	color to be used to draw the features. At present only a single color is supported.
`type`	plotting type (see `par()`). Defaults to `type = "o"` which draws each feature as a point and connecting the features of the same feature group with a line.
`main`	`character(1)` with the title of the plot.
`featureGroups`	optional `character` of feature group IDs to draw only specified feature group(s). If not provided, all feature groups are drawn.

Author(s)

Johannes Rainer

`plotMsData` *DEPRECATED: Create a plot that combines a XIC and a mz/rt 2D plot for one sample*

Description

UPDATE: please use `plot(x, type = "XIC")` from the `MSnbase` package instead. See examples below.

The `plotMsData` creates a plot that combines an (base peak) extracted ion chromatogram on top (rt against intensity) and a plot of rt against m/z values at the bottom.

Usage

```
plotMsData(
  x,
  main = "",
  cex = 1,
  mfrow = c(2, 1),
  grid.color = "lightgrey",
  colramp = colorRampPalette(rev(brewer.pal(9, "YlGnBu")))
)
```

Arguments

`x`	`data.frame` such as returned by the `extractMsData()` function. Only a single `data.frame` is supported.
`main`	`character(1)` specifying the title.
`cex`	`numeric(1)` defining the size of points. Passed directly to the `plot` function.
`mfrow`	`numeric(2)` defining the plot layout. This will be passed directly to `par(mfrow = mfrow)`. See `par` for more information. Setting `mfrow = NULL` avoids calling `par(mfrow = mfrow)` hence allowing to pre-define the plot layout.
`grid.color`	a color definition for the grid line (or `NA` to skip creating them).
`colramp`	a *color ramp palette* to be used to color the data points based on their intensity. See argument `col.regions` in lattice::level.colors documentation.

Author(s)

Johannes Rainer

Examples

```
## Read two files from the faahKO package
library(faahKO)
library(magrittr)
cdfs <- dir(system.file("cdf", package = "faahKO"), full.names = TRUE,
    recursive = TRUE)[1:2]
raw_data <- readMSData(cdfs, mode = "onDisk")

## Subset the object to a rt and mz range and plot the data.
raw_data %>%
    filterRt(rt = c(2700, 2900)) %>%
    filterMz(mz = c(334.9, 335.1)) %>%
    plot(type = "XIC")
```

`plotPeaks-methods` *Plot a grid of a large number of peaks*

Description

Plot extracted ion chromatograms for many peaks simultaneously, indicating peak integration start and end points with vertical grey lines.

Arguments

`object`	the xcmsRaw object
`peaks`	matrix with peak information as produced by `findPeaks`
`figs`	two-element vector describing the number of rows and the number of columns of peaks to plot, if missing then an approximately square grid that will fit the number of peaks supplied
`width`	width of chromatogram retention time to plot for each peak

Details

This function is intended to help graphically analyze the results of peak picking. It can help estimate the number of false positives and improper integration start and end points. Its output is very compact and tries to waste as little space as possible. Each plot is labeled with rounded m/z and retention time separated by a space.

Methods

object = "xcmsRaw" `plotPeaks(object, peaks, figs, width = 200)`

See Also

`xcmsRaw-class`, `findPeaks`, `split.screen`

`plotQC` *Plot m/z and RT deviations for QC purposes without external reference data*

Description

Use "democracy" to determine the average m/z and RT deviations for a grouped xcmsSet, and dependency on sample or absolute m/z

Usage

```
plotQC(object, sampNames, sampColors, sampOrder, what)
```

Arguments

`object`	A grouped `xcmsSet`
`sampNames`	Override sample names (e.g. with simplified names)
`sampColors`	Provide a set of colors (default: monochrome ?)
`sampOrder`	Override the order of samples, e.g. to bring them in order of measurement to detect time drift
`what`	A vector of which QC plots to generate. "mzdevhist": histogram of mz deviations. Should be gaussian shaped. If it is multimodal, then some peaks seem to have a systematically higher m/z deviation "rtdevhist": histogram of RT deviations. Should be gaussian shaped. If it is multimodal, then some peaks seem to have a systematically higher RT deviation "mzdevmass": Shows whether m/z deviations are absolute m/z dependent, could indicate miscalibration "mzdevtime": Shows whether m/z deviations are RT dependent, could indicate instrument drift "mzdevsample": median mz deviation for each sample, indicates outliers "rtdevsample": median RT deviation for each sample, indicates outliers

Details

plotQC() is a warpper to create a set of diagnostic plots. For the m/z deviations, the median of all m/z withon one group are assumed.

Value

List with four matrices, each of dimension features * samples: "mz": median mz deviation for each sample "mzdev": median mz deviation for each sample "rt": median RT deviation for each sample "rtdev": median RT deviation for each sample

Author(s)

Michael Wenk, Michael Wenk <michael.wenk@student.uni-halle.de>

Examples

```
library(faahKO)
xsg <- group(faahko)

plotQC(xsg, what="mzdevhist")
plotQC(xsg, what="rtdevhist")
plotQC(xsg, what="mzdevmass")
plotQC(xsg, what="mzdevtime")
plotQC(xsg, what="mzdevsample")
plotQC(xsg, what="rtdevsample")
```

`plotRaw-methods` *Scatterplot of raw data points*

Description

Produce a scatterplot showing raw data point location in retention time and m/z. This plot is more useful for centroided data than continuum data.

Arguments

`object`	the `xcmsRaw` object
`mzrange`	numeric vector of length >= 2 whose range will be used to select the masses to plot
`rtrange`	numeric vector of length >= 2 whose range will be used to select the retention times to plot
`scanrange`	numeric vector of length >= 2 whose range will be used to select scans to plot
`log`	logical, log transform intensity
`title`	main title of the plot

Value

A matrix with the points plotted.

Methods

object = "xcmsRaw" `plotRaw(object, mzrange = numeric(), rtrange = numeric(), scanrange = numeric(), log=FALSE, title='Raw Data')`

See Also

`xcmsRaw-class`

`plotrt-methods` *Plot retention time deviation profiles*

Description

Use corrected retention times for each sample to calculate retention time deviation profiles and plot each on the same graph.

Arguments

`object`	the `xcmsSet` object
`col`	vector of colors for plotting each sample
`ty`	vector of line and point types for plotting each sample
`leg`	logical plot legend with sample labels
`densplit`	logical, also plot peak overall peak density

Methods

object = "xcmsSet" `plotrt(object, col = NULL, ty = NULL, leg = TRUE, densplit = FALSE)`

See Also

`xcmsSet-class`, `retcor`

`plotScan-methods` *Plot a single mass scan*

Description

Plot a single mass scan using the impulse representation. Most useful for centroided data.

Arguments

`object`	the `xcmsRaw` object
`scan`	integer with number of scan to plot
`mzrange`	numeric vector of length >= 2 whose range will be used to select masses to plot
`ident`	logical, use mouse to interactively identify and label individual masses

Methods

object = "xcmsRaw" `plotScan(object, scan, mzrange = numeric(), ident = FALSE)`

See Also

`xcmsRaw-class`

`plotSpec-methods` *Plot mass spectra from the profile matrix*

Description

Uses the pre-generated profile mode matrix to plot mass spectra over a specified retention time range.

Arguments

`object`	the `xcmsRaw` object
`ident`	logical, use mouse to identify and label peaks
`vline`	numeric vector with locations of vertical lines
`...`	arguments passed to `profRange`

Value

If `ident == TRUE`, an integer vector with the indecies of the points that were identified. Otherwise a two-column matrix with the plotted points.

Methods

object = "xcmsRaw" `plotSpec(object, ident = FALSE, vline = numeric(0), ...)`

See Also

`xcmsRaw-class`

`plotSurf-methods` *Plot profile matrix 3D surface using OpenGL*

Description

This method uses the rgl package to create interactive three dimensonal representations of the profile matrix. It uses the terrain color scheme.

Arguments

`object`	the `xcmsRaw` object
`log`	logical, log transform intensity
`aspect`	numeric vector with aspect ratio of the m/z, retention time and intensity components of the plot
`...`	arguments passed to `profRange`

Details

The rgl package is still in development and imposes some limitations on the output format. A bug in the axis label code means that the axis labels only go from 0 to the aspect ratio constant of that axis. Additionally the axes are not labeled with what they are.

It is important to only plot a small portion of the profile matrix. Large portions can quickly overwhelm your CPU and memory.

Methods

object = "xcmsRaw" `plotSurf(object, log = FALSE, aspect = c(1, 1, .5), ...)`

See Also

`xcmsRaw-class`

`plotTIC-methods` *Plot total ion count*

Description

Plot chromatogram of total ion count. Optionally allow identification of target peaks and viewing/identification of individual spectra.

Arguments

`object`	the `xcmsRaw` object
`ident`	logical, use mouse to identify and label chromatographic peaks
`msident`	logical, use mouse to identify and label spectral peaks

Value

If `ident == TRUE`, an integer vector with the indecies of the points that were identified. Otherwise a two-column matrix with the plotted points.

Methods

object = "xcmsRaw" `plotTIC(object, ident = FALSE, msident = FALSE)`

See Also

`xcmsRaw-class`

`ProcessHistory-class` *Tracking data processing*

Description

Objects of the type `ProcessHistory` allow to keep track of any data processing step in an metabolomics experiment. They are created by the data processing methods, such as `findChromPeaks` and added to the corresponding results objects. Thus, usually, users don't need to create them.

The `XProcessHistory` extends the `ProcessHistory` by adding a slot `param` that allows to store the actual parameter class of the processing step.

`processParam`, `processParam<-`: get or set the parameter class from an `XProcessHistory` object.

`msLevel`: returns the MS level on which a certain analysis has been performed, or `NA` if not defined.

The `processType` method returns a character specifying the processing step *type*.

The `processDate` extracts the start date of the processing step.

The `processInfo` extracts optional additional information on the processing step.

The `fileIndex` extracts the indices of the files on which the processing step was applied.

Usage

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

## S4 method for signature 'XProcessHistory'
show(object)

## S4 method for signature 'XProcessHistory'
processParam(object)

## S4 method for signature 'XProcessHistory'
msLevel(object)

## S4 method for signature 'ProcessHistory'
processType(object)

## S4 method for signature 'ProcessHistory'
processDate(object)

## S4 method for signature 'ProcessHistory'
processInfo(object)

## S4 method for signature 'ProcessHistory'
fileIndex(object)
```

Arguments

`object`	A `ProcessHistory` or `XProcessHistory` object.

Value

For `processParam`: a parameter object extending the `Param` class.

The `processType` method returns a character string with the processing step type.

The `processDate` method returns a character string with the time stamp of the processing step start.

The `processInfo` method returns a character string with optional additional informations.

The `fileIndex` method returns a integer vector with the index of the files/samples on which the processing step was applied.

Slots

`type` character(1): string defining the type of the processing step. This string has to match predefined values. Use `processHistoryTypes` to list them.

`date` character(1): date time stamp when the processing step was started.

`info` character(1): optional additional information.

`fileIndex` integer of length 1 or > 1 to specify on which samples of the object the processing was performed.

`error` (ANY): used to store eventual calculation errors.

`param` (Param): an object of type `Param` (e.g. `CentWaveParam`) specifying the settings of the processing step.

`msLevel: integer` definining the MS level(s) on which the analysis was performed.

Author(s)

Johannes Rainer

`profMat-xcmsSet` *The profile matrix*

Description

The *profile* matrix is an n x m matrix, n (rows) representing equally spaced m/z values (bins) and m (columns) the retention time of the corresponding scans. Each cell contains the maximum intensity measured for the specific scan and m/z values falling within the m/z bin.

The `profMat` method creates a new profile matrix or returns the profile matrix within the object's `@env` slot, if available. Settings for the profile matrix generation, such as `step` (the bin size), `method` or additional settings are extracted from the respective slots of the `xcmsRaw` object. Alternatively it is possible to specify all of the settings as additional parameters.

Usage

```
## S4 method for signature 'xcmsRaw'
profMat(object, method, step, baselevel, basespace, mzrange.)
```

Arguments

`object`	The `xcmsRaw` object.
`method`	The profile matrix generation method. Allowed are `"bin"`, `"binlin"`, `"binlinbase"` and `"intlin"`. See details section for more information.
`step`	numeric(1) representing the m/z bin size.
`baselevel`	numeric(1) representing the base value to which empty elements (i.e. m/z bins without a measured intensity) should be set. Only considered if `method = "binlinbase"`. See `baseValue` parameter of `imputeLinInterpol` for more details.
`basespace`	numeric(1) representing the m/z length after which the signal will drop to the base level. Linear interpolation will be used between consecutive data points falling within `2 * basespace` to each other. Only considered if `method = "binlinbase"`. If not specified, it defaults to `0.075`. Internally this parameter is translated into the `distance` parameter of the `imputeLinInterpol` function by `distance = floor(basespace / step)`. See `distance` parameter of `imputeLinInterpol` for more details.
`mzrange.`	Optional numeric(2) manually specifying the mz value range to be used for binnind. If not provided, the whole mz value range is used.

Details

Profile matrix generation methods:

- **bin** The default profile matrix generation method that does a simple binning, i.e. aggregating of intensity values falling within an m/z bin.
- **binlin** Binning followed by linear interpolation to impute missing values. The value for m/z bins without a measured intensity are inferred by a linear interpolation between neighboring bins with a measured intensity.
- **binlinbase** Binning followed by a linear interpolation to impute values for empty elements (m/z bins) within a user-definable proximity to non-empty elements while stetting the element's value to the `baselevel` otherwise. See `impute = "linbase"` parameter of `imputeLinInterpol` for more details.
- **intlin** Set the elements' values to the integral of the linearly interpolated data from plus to minus half the step size.

Value

`profMat` returns the profile matrix (rows representing scans, columns equally spaced m/z values).

Note

From `xcms` version 1.51.1 on only the `profMat` method should be used to extract the profile matrix instead of the previously default way to access it directly *via* `object@env$profile`.

Author(s)

Johannes Rainer

See Also

`xcmsRaw`, `binYonX` and `imputeLinInterpol` for the employed binning and missing value imputation methods, respectively. `profMat,XCMSnExp-method` for the method on `XCMSnExp` objects.

Examples

```
file <- system.file('cdf/KO/ko15.CDF', package = "faahKO")
## Load the data without generating the profile matrix (profstep = 0)
xraw <- xcmsRaw(file, profstep = 0)
## Extract the profile matrix
profmat <- profMat(xraw, step = 0.3)
dim(profmat)
## If not otherwise specified, the settings from the xraw object are used:
profinfo(xraw)
## To extract a profile matrix with linear interpolation use
profmat <- profMat(xraw, step = 0.3, method = "binlin")
## Alternatively, the profMethod of the xraw objects could be changed
profMethod(xraw) <- "binlin"
profmat_2 <- profMat(xraw, step = 0.3)
all.equal(profmat, profmat_2)
```

`profMedFilt-methods` *Median filtering of the profile matrix*

Description

Apply a median filter of given size to a profile matrix.

Arguments

`object`	the `xcmsRaw` object
`massrad`	number of m/z grid points on either side to use for median calculation
`scanrad`	number of scan grid points on either side to use for median calculation

Methods

object = "xcmsRaw" `profMedFilt(object, massrad = 0, scanrad = 0)`

See Also

`xcmsRaw-class`, `medianFilter`

`profMethod-methods` *Get and set method for generating profile data*

Description

These methods get and set the method for generating profile (matrix) data from raw mass spectral data. It can currently be `bin`, `binlin`, `binlinbase`, or `intlin`.

Methods

object = "xcmsRaw" `profMethod(object)`

See Also

`xcmsRaw-class`, `profMethod`, `profBin`, `plotSpec`, `plotChrom`, `findPeaks`

`profRange-methods` *Specify a subset of profile mode data*

Description

Specify a subset of the profile mode matrix given a mass, time, or scan range. Allow flexible user entry for other functions.

Arguments

`object`	the `xcmsRaw` object
`mzrange`	single numeric mass or vector of masses
`rtrange`	single numeric time (in seconds) or vector of times
`scanrange`	single integer scan index or vector of indecies
`...`	arguments to other functions

Details

This function handles selection of mass/time subsets of the profile matrix for other functions. It allows the user to specify such subsets in a variety of flexible ways with minimal typing.

Because R does partial argument matching, `mzrange`, `scanrange`, and `rtrange` can be specified in short form using `m=`, `s=`, and `t=`, respectively. If both a `scanrange` and `rtrange` are specified, then the `rtrange` specification takes precedence.

When specifying ranges, you may either enter a single number or a numeric vector. If a single number is entered, then the closest single scan or mass value is selected. If a vector is entered, then the range is set to the `range()` of the values entered. That allows specification of ranges using shortened, slightly non-standard syntax. For example, one could specify 400 to 500 seconds using any of the following: `t=c(400,500)`, `t=c(500,400)`, or `t=400:500`. Use of the sequence operator (`:`) can save several keystrokes when specifying ranges. However, while the sequence operator works well for specifying integer ranges, fractional ranges do not always work as well.

Value

A list with the folloing items:

`mzrange`	numeric vector with start and end mass
`masslab`	textual label of mass range
`massidx`	integer vector of mass indecies
`scanrange`	integer vector with stat ane end scans
`scanlab`	textual label of scan range
`scanidx`	integer vector of scan range
`rtrange`	numeric vector of start and end times
`timelab`	textual label of time range

Methods

object = "xcmsRaw" `profRange(object, mzrange = numeric(), rtrange = numeric(), scanrange = numeric(), ...)`

See Also

`xcmsRaw-class`

`profStep-methods` *Get and set m/z step for generating profile data*

Description

These methods get and set the m/z step for generating profile (matrix) data from raw mass spectral data. Smaller steps yield more precision at the cost of greater memory usage.

Methods

object = "xcmsRaw" `profStep(object)`

See Also

`xcmsRaw-class`, `profMethod`

Examples

```
## Not run:
library(faahKO)
cdfpath <- system.file("cdf", package = "faahKO")
cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)
xset <- xcmsRaw(cdffiles[1])

xset
plotSurf(xset, mass=c(200,500))

profStep(xset)<-0.1 ## decrease the bin size to get better resolution
plotSurf(xset, mass=c(200, 500))
##works nicer on high resolution data.

## End(Not run)
```

`quantify,XCMSnExp-method` *Accessing mz-rt feature data values*

Description

`featureValues,XCMSnExp` : extract a `matrix` for feature values with rows representing features and columns samples. Parameter `value` allows to define which column from the `chromPeaks` matrix should be returned. Multiple chromatographic peaks from the same sample can be assigned to a feature. Parameter `method` allows to specify the method to be used in such cases to chose from which of the peaks the value should be returned. Parameter ‘msLevel‘ allows to choose a specific MS level for which feature values should be returned (given that features have been defined for that MS level).

`quantify,XCMSnExp`: return the preprocessing results as an `SummarizedExperiment` object containing the feature abundances as assay matrix, the feature definitions (returned by `featureDefinitions`) as `rowData` and the phenotype information as `colData`. This is an ideal container for further processing of the data. Internally, the `featureValues` method is used to extract the feature abundances, parameters for that method can be passed to `quantify` with `...`.

Usage

```
## S4 method for signature 'XCMSnExp'
quantify(object, ...)

## S4 method for signature 'XCMSnExp'
featureValues(
  object,
  method = c("medret", "maxint", "sum"),
  value = "into",
  intensity = "into",
  filled = TRUE,
  missing = NA,
  msLevel = integer()
)
```

Arguments

`object`	A `XCMSnExp` object providing the feature definitions.
`...`	For `quantify`: additional parameters to be passed on to the `featureValues` method.
`method`	`character` specifying the method to resolve multi-peak mappings within the same sample, i.e. to define the *representative* peak for a feature in samples where more than one peak was assigned to the feature. If `"medret"`: select the peak closest to the median retention time of the feature. If `"maxint"`: select the peak yielding the largest signal. If `"sum"`: sum the values (only if `value` is `"into"` or `"maxo"`.

`value`	`character` specifying the name of the column in `chromPeaks(object)` that should be returned. Defaults to `"into"` in which case the integrated peak area is returned. To get the index of the peak in the `chromPeaks(object)` matrix use `"index"`.
`intensity`	`character` specifying the name of the column in the `chromPeaks(objects)` matrix containing the intensity value of the peak that should be used for the conflict resolution if `method = "maxint"`.
`filled`	`logical(1)` specifying whether values for filled-in peaks should be returned or not. If `filled = FALSE`, an `NA` is returned in the matrix for the respective peak. See `fillChromPeaks` for details on peak filling.
`missing`	how missing values should be reported. Allowed values are `NA` (the default), a `numeric` or `missing = "rowmin_half"`. The latter replaces any `NA` with half of the row's minimal (non-missing) value.
`msLevel`	for ‘featureValues‘: ‘integer‘ defining the MS level(s) for which feature values should be returned. By default, values for features defined for all MS levels are returned.

Value

For `featureValues`: a `matrix` with feature values, columns representing samples, rows features. The order of the features matches the order found in the `featureDefinitions(object)` `DataFrame`. The rownames of the `matrix` are the same than those of the `featureDefinitions` `DataFrame`. `NA` is reported for features without corresponding chromatographic peak in the respective sample(s).

For `quantify`: a `SummarizedExperiment` representing the preprocessing results.

Note

This method is equivalent to the `groupval` for `xcmsSet` objects. Note that `missing = 0` should be used to get the same behaviour as `groupval`, i.e. report missing values as 0 after a call to `fillPeaks`.

Author(s)

Johannes Rainer

See Also

`XCMSnExp` for information on the data object.

`featureDefinitions` to extract the `DataFrame` with the feature definitions.

`featureChromatograms` to extract ion chromatograms for each feature.

`hasFeatures` to evaluate whether the `XCMSnExp` provides feature definitions.

`groupval` for the equivalent method on `xcmsSet` objects.

`rawEIC-methods` *Get extracted ion chromatograms for specified m/z range*

Description

Generate extracted ion chromatogram for m/z values of interest. The raw data is used in contrast to `getEIC` which uses data from the profile matrix (i.e. values binned along the M/Z dimension).

Arguments

`object`	`xcmsRaw` object
`mzrange`	m/z range for EIC
`rtrange`	retention time range for EIC
`scanrange`	scan range for EIC

Value

A list of :

`scan`	scan number
`intensity`	added intensity values

Methods

object = "xcmsRaw" `rawEIC(object, mzrange = numeric(), rtrange = numeric(), scanrange = numeric())`

Author(s)

Ralf Tautenhahn

See Also

`xcmsRaw-class`

`rawMat-methods` *Get a raw data matrix*

Description

Returns a matrix with columns for time, m/z, and intensity that represents the raw data from a chromatography mass spectrometry experiment.

Arguments

`object`	The container of the raw data
`mzrange`	Subset by m/z range
`rtrange`	Subset by retention time range
`scanrange`	Subset by scan index range
`log`	Whether to log transform the intensities

Value

A numeric matrix with three columns: time, mz and intensity.

Methods

object = "xcmsRaw" `rawMat(object, mzrange = numeric(), rtrange = numeric(), scanrange = numeric(), log=FALSE)`

Author(s)

Michael Lawrence

See Also

`plotRaw` for plotting the raw intensities

`reconstructChromPeakSpectra` *Data independent acquisition (DIA): reconstruct MS2 spectra*

Description

Reconstructs MS2 spectra for each MS1 chromatographic peak (if possible) for data independent acquisition (DIA) data (such as SWATH). See the *LC-MS/MS analysis* vignette for more details and examples.

Usage

```
reconstructChromPeakSpectra(
  object,
  expandRt = 0,
  diffRt = 2,
  minCor = 0.8,
  intensity = "maxo",
  peakId = rownames(chromPeaks(object, msLevel = 1L)),
  BPPARAM = bpparam(),
  return.type = c("MSpectra", "Spectra")
)
```

Arguments

`object`	`XCMSnExp` with identified chromatographic peaks.
`expandRt`	`numeric(1)` allowing to expand the retention time range for extracted ion chromatograms by a constant value (for the peak shape correlation). Defaults to `expandRt = 0` hence correlates only the signal included in the identified chromatographic peaks.
`diffRt`	`numeric(1)` defining the maximal allowed difference between the retention time of the chromatographic peak (apex) and the retention times of MS2 chromatographic peaks (apex) to consider them as representing candidate fragments of the original ion.
`minCor`	`numeric(1)` defining the minimal required correlation coefficient for MS2 chromatographic peaks to be considered for MS2 spectrum reconstruction.
`intensity`	`character(1)` defining the column in the `chromPeaks` matrix that should be used for the intensities of the reconstructed spectra's peaks. The same value from the MS1 chromatographic peaks will be used as `precursorIntensity` of the resulting spectra.
`peakId`	optional `character` vector with peak IDs (i.e. rownames of `chromPeaks`) of MS1 peaks for which MS2 spectra should be reconstructed. By default they are reconstructed for all MS1 chromatographic peaks.
`BPPARAM`	parallel processing setup. See `bpparam()` for more information.
`return.type`	`character(1)` defining the type of the returned object. Can be either `return.type = "MSpectra"` (the default) to return a `MSnbase::MSpectra` object or `return.type = "Spectra"` for the newer `Spectra::Spectra` object.

Details

In detail, the function performs for each MS1 chromatographic peak:

- Identify all MS2 chromatographic peaks from the isolation window containing the m/z of the ion (i.e. the MS1 chromatographic peak) with approximately the same retention time than the MS1 peak (accepted rt shift can be specified with the `diffRt` parameter).
- Correlate the peak shapes of the candidate MS2 chromatographic peaks with the peak shape of the MS1 peak retaining only MS2 chromatographic peaks for which the correlation is `> minCor`.

- Reconstruct the MS2 spectrum using the m/z of all above selected MS2 chromatographic peaks and their intensity (either `"maxo"` or `"into"`). Each MS2 chromatographic peak selected for an MS1 peak will thus represent one **mass peak** in the reconstructed spectrum.

The resulting `MSpectra` object provides also the peak IDs of the MS2 chromatographic peaks for each spectrum as well as their correlation value.

Value

Depending on `return.type`:

- `MSpectra()` with the reconstructed MS2 spectra for all MS1 peaks in `object`. Contains empty Spectrum2 objects for MS1 peaks for which reconstruction was not possible (either no MS2 signal was recorded or the correlation of the MS2 chromatographic peaks with the MS1 chromatographic peak was below threshold `minCor`. `MSpectra` metadata columns `"ms2_peak_id"` and `"ms2_peak_cor"` (of type `CharacterList()` and `NumericList()` with length equal to the number of peaks per reconstructed MS2 spectrum) providing the IDs and the correlation of the MS2 chromatographic peaks from which the MS2 spectrum was reconstructed. As retention time the median retention times of all MS2 chromatographic peaks used for the spectrum reconstruction is reported. The MS1 chromatographic peak intensity is reported as the reconstructed spectrum's `precursorIntensity` value (see parameter `intensity` above).
- `Spectra` object (defined in the `Spectra` package). The same content and information than above.

Author(s)

Johannes Rainer, Michael Witting

See Also

`findChromPeaksIsolationWindow()` for the function to perform MS2 peak detection in DIA isolation windows and for examples.

`removeIntensity,Chromatogram-method`

Remove intensities from chromatographic data

Description

`removeIntensities` allows to remove intensities from chromatographic data matching certain conditions (depending on parameter `which`). The intensities are actually not *removed* but replaced with `NA_real_`. To actually **remove** the intensities (and the associated retention times) use `clean()` afterwards.

Parameter `which` allows to specify which intensities should be replaced by `NA_real_`. By default (`which = "below_threshod"` intensities below `threshold` are removed. If `x` is a `XChromatogram` or `XChromatograms` object (and hence provides also chromatographic peak definitions within the object) `which = "outside_chromPeak"` can be selected which removes any intensity which is outside the boundaries of identified chromatographic peak(s) in the chromatographic data.

Note that `filterIntensity()` might be a better approach to subset/filter chromatographic data.

Usage

```
## S4 method for signature 'Chromatogram'
removeIntensity(object, which = "below_threshold", threshold = 0)

## S4 method for signature 'MChromatograms'
removeIntensity(object, which = "below_threshold", threshold = 0)

## S4 method for signature 'XChromatogram'
removeIntensity(
  object,
  which = c("below_threshold", "outside_chromPeak"),
  threshold = 0
)
```

Arguments

`object`	an object representing chromatographic data. Can be a `Chromatogram()`, `MChromatograms()`, `XChromatogram()` or `XChromatograms()` object.
`which`	`character(1)` defining the condition to remove intensities. See description for details and options.
`threshold`	`numeric(1)` defining the threshold below which intensities are removed (if `which = "below_threshold"`).

Value

the input object with matching intensities being replaced by `NA`.

Author(s)

Johannes Rainer

Examples

```
chr <- Chromatogram(rtime = 1:10 + rnorm(n = 10, sd = 0.3),
    intensity = c(5, 29, 50, NA, 100, 12, 3, 4, 1, 3))

## Remove all intensities below 20
res <- removeIntensity(chr, threshold = 20)
intensity(res)
```

`retcor-methods` *Correct retention time from different samples*

Description

To correct differences between retention times between different samples, a number of of methods exist in XCMS. `retcor` is the generic method.

Arguments

`object`	`xcmsSet-class` object
`method`	Method to use for retention time correction. See details.
`...`	Optional arguments to be passed along

Details

Different algorithms can be used by specifying them with the `method` argument. For example to use the approach described by Smith et al (2006) one would use: `retcor(object, method="loess")`. This is also the default.

Further arguments given by `...` are passed through to the function implementing the `method`.

A character vector of *nicknames* for the algorithms available is returned by `getOption("BioC")$xcms$retcor.methods`. If the nickname of a method is called "loess", the help page for that specific method can be accessed with `?retcor.loess`.

Value

An `xcmsSet` object with corrected retntion times.

Methods

object = "xcmsSet" `retcor(object, ...)`

See Also

`retcor.loess` `retcor.obiwarp` `xcmsSet-class`,

`retcor.obiwarp` *Align retention times across samples with Obiwarp*

Description

Calculate retention time deviations for each sample. It is based on the code at `http://obi-warp.sourceforge.net/`. However, this function is able to align multiple samples, by a center-star strategy.

For the original publication see

Chromatographic Alignment of ESI-LC-MS Proteomics Data Sets by Ordered Bijective Interpolated Warping John T. Prince and, Edward M. Marcotte Analytical Chemistry 2006 78 (17), 6140-6152

Arguments

`object`	the `xcmsSet` object
`plottype`	if `deviation` plot retention time deviation
`profStep`	step size (in m/z) to use for profile generation from the raw data files
`center`	the index of the sample all others will be aligned to. If center==NULL, the sample with the most peaks is chosen as default.
`col`	vector of colors for plotting each sample
`ty`	vector of line and point types for plotting each sample
`response`	Responsiveness of warping. 0 will give a linear warp based on start and end points. 100 will use all bijective anchors
`distFunc`	DistFunc function: cor (Pearson's R) or cor_opt (default, calculate only 10% diagonal band of distance matrix, better runtime), cov (covariance), prd (product), euc (Euclidean distance)
`gapInit`	Penalty for Gap opening, see below
`gapExtend`	Penalty for Gap enlargement, see below
`factorDiag`	Local weighting applied to diagonal moves in alignment.
`factorGap`	Local weighting applied to gap moves in alignment.
`localAlignment`	Local rather than global alignment
`initPenalty`	Penalty for initiating alignment (for local alignment only) Default: 0 Default gap penalties: (gapInit, gapExtend) [by distFunc type]: 'cor' = '0.3,2.4' 'cov' = '0,11.7' 'prd' = '0,7.8' 'euc' = '0.9,1.8'

Value

An `xcmsSet` object

Methods

object = "xcmsSet" retcor(object, method="obiwarp", plottype = c("none", "deviation"), profStep=1, center=NULL, col = NULL, ty = NULL, response=1, distFunc="cor_opt", gapInit=NULL, gapExtend=NULL, factorDiag=2, factorGap=1, localAlignment=0, initPenalty=0)

See Also

`xcmsSet-class`,

`retcor.peakgroups-methods` *Align retention times across samples*

Description

These two methods use “well behaved” peak groups to calculate retention time deviations for every time point of each sample. Use smoothed deviations to align retention times.

Arguments

`object`	the `xcmsSet` object
`missing`	number of missing samples to allow in retention time correction groups
`extra`	number of extra peaks to allow in retention time correction correction groups
`smooth`	either `"loess"` for non-linear alignment or `"linear"` for linear alignment
`span`	degree of smoothing for local polynomial regression fitting
`family`	if `gaussian` fitting is by least-squares with no outlier removal, and if `symmetric` a re-descending M estimator is used with Tukey’s biweight function, allowing outlier removal
`plottype`	if `deviation` plot retention time deviation points and regression fit, and if `mdevden` also plot peak overall peak density and retention time correction peak density
`col`	vector of colors for plotting each sample
`ty`	vector of line and point types for plotting each sample

Value

An `xcmsSet` object

Methods

object = "xcmsSet" `retcor(object, missing = 1, extra = 1, smooth = c("loess", "linear"), span = .2, family = c("gaussian", "symmetric"), plottype = c("none", "deviation", "mdevden"), col = NULL, ty = NULL)`

See Also

`xcmsSet-class`, `loess` `retcor.obiwarp`

`retexp` *Set retention time window to a specified width*

Description

Expands (or contracts) the retention time window in each row of a matrix as defined by the `retmin` and `retmax` columns.

Usage

```
retexp(peakrange, width = 200)
```

Arguments

`peakrange`	maxtrix with columns `retmin` and `retmax`
`width`	new width for the window

Value

The altered matrix.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`getEIC`

`rla` *Calculate relative log abundances `rla` calculates the relative log abundances (RLA, see reference) on a `numeric` vector.*

Description

Calculate relative log abundances

`rla` calculates the relative log abundances (RLA, see reference) on a `numeric` vector.

`rowRla` calculates row-wise RLAs.

Usage

```
rla(x, group, log.transform = TRUE)

rowRla(x, group, log.transform = TRUE)
```

Arguments

`x`	`numeric` (for `rla`) or `matrix` (for `rowRla`) with the abundances (in natural scale) on which the RLA should be calculated.
`group`	`factor`, `numeric` or `character` with the same length than `x` that groups values in `x`. If omitted all values are considered to be from the same group.
`log.transform`	`logical(1)` whether `x` should be log2 transformed. Set to `log.transform = FALSE` if `x` is already in log scale.

Details

The RLA is defines as the (log) abundance of an analyte relative to the median across all abundances of the same group.

Value

`numeric` of the same length than `x` (for `rla`) or `matrix` with the same dimensions than `x` (for `rowRla`).

Author(s)

Johannes Rainer

References

De Livera AM, Dias DA, De Souza D, Rupasinghe T, Pyke J, Tull D, Roessner U, McConville M, Speed TP. Normalizing and integrating metabolomics data. *Anal Chem* 2012 Dec 18;84(24):10768-76.

Examples

```
x <- c(3, 4, 5, 1, 2, 3, 7, 8, 9)

grp <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)

rla(x, grp)
```

`sampnames-methods` *Get sample names*

Description

Return sample names for an object

Value

A character vector with sample names.

Methods

object = "xcmsEIC" sampnames(object)

object = "xcmsSet" sampnames(object)

See Also

`xcmsSet-class`, `xcmsEIC-class`

`showError,xcmsSet-method`

Extract processing errors

Description

If peak detection is performed with `findPeaks` setting argument `stopOnError = FALSE` eventual errors during the process do not cause to stop the processing but are recorded inside of the resulting `xcmsSet` object. These errors can be accessed with the `showError` method.

Usage

```
## S4 method for signature 'xcmsSet'
showError(object, message. = TRUE, ...)
```

Arguments

`object`	An `xcmsSet` object.
`message.`	Logical indicating whether only the error message, or the error itself should be returned.
`...`	Additional arguments.

Value

A list of error messages (if `message. = TRUE`) or errors or an empty list if no errors are present.

Author(s)

Johannes Rainer

`specDist-methods` *Distance methods for xcmsSet, xcmsRaw and xsAnnotate*

Description

There are several methods for calculating a distance between two sets of peaks in xcms. `specDist` is the generic method.

Arguments

`object`	a xcmsSet or xcmsRaw.
`method`	Method to use for distance calculation. See details.
`...`	mzabs, mzppm and parameters for the distance function.

Details

Different algorithms can be used by specifying them with the `method` argument. For example to use the "meanMZmatch" approach with xcmsSet one would use: `specDist(object, peakIDs1, peakIDs2, method="meanMZmatch")`. This is also the default.

Further arguments given by `...` are passed through to the function implementing the `method`.

A character vector of *nicknames* for the algorithms available is returned by `getOption("BioC")$xcms$specDist.methods`. If the nickname of a method is called "meanMZmatch", the help page for that specific method can be accessed with `?specDist.meanMZmatch`.

Value

`mzabs`	maximum absolute deviation for two matching peaks
`mzppm`	relative deviations in ppm for two matching peaks
`symmetric`	use symmetric pairwise m/z-matches only, or each match

Methods

object = "xcmsSet" `specDist(object, peakIDs1, peakIDs2,...)`

object = "xsAnnotate" `specDist(object, PSpec1, PSpec2,...)`

Author(s)

Joachim Kutzera, `<jkutzer@ipb-halle.de>`

`specDist.cosine` *a Distance function based on matching peaks*

Description

This method calculates the distance of two sets of peaks using the cosine-distance.

Usage

```
specDist.cosine(peakTable1, peakTable2, mzabs=0.001, mzppm=10, mzExp=0.6,
                intExp=3, nPdiff=2, nPmin=8, symmetric=FALSE)
```

Arguments

`peakTable1`	a Matrix containing at least m/z-values, row must be called "mz"
`peakTable2`	the matrix for the other mz-values
`mzabs`	maximum absolute deviation for two matching peaks
`mzppm`	relative deviations in ppm for two matching peaks
`symmetric`	use symmetric pairwise m/z-matches only, or each match
`mzExp`	the exponent used for mz
`intExp`	the exponent used for intensity
`nPdiff`	the maximum nrow-difference of the two peaktables
`nPmin`	the minimum absolute sum of peaks from both praktables

Details

The result is the cosine-distance of the product from weighted factors of mz and intensity from matching peaks in the two peaktables. The factors are calculated as wFact = mz^mzExp * int^intExp. if no distance is calculated (for example because no matching peaks were found) the return-value is NA.

Methods

peakTable1 = "matrix", peakTable2 = "matrix" `specDist.cosine(peakTable1, peakTable2, mzabs = 0.001, mzppm = 10, mzExp = 0.6, intExp = 3, nPdiff = 2, nPmin = 8, symmetric = FALSE)`

Author(s)

Joachim Kutzera, `<jkutzer@ipb-halle.de>`

`specDist.meanMZmatch` *a Distance function based on matching peaks*

Description

This method calculates the distance of two sets of peaks.

Usage

```
specDist.meanMZmatch(peakTable1, peakTable2, matchdist=1, matchrate=1,
                     mzabs=0.001, mzppm=10, symmetric=TRUE)
```

Arguments

`peakTable1`	a Matrix containing at least m/z-values, row must be called "mz"
`peakTable2`	the matrix for the other mz-values
`mzabs`	maximum absolute deviation for two matching peaks
`mzppm`	relative deviations in ppm for two matching peaks
`symmetric`	use symmetric pairwise m/z-matches only, or each match
`matchdist`	the weight for value one (see details)
`matchrate`	the weight for value two

Details

The result of the calculation is a weighted sum of two values. Value one is the mean absolute difference of the matching peaks, value two is the relation of matching peaks and non matching peaks. if no distance is calculated (for example because no matching peaks were found) the return-value is NA.

Methods

peakTable1 = "matrix", peakTable2 = "matrix" `specDist.meanMZmatch(peakTable1, peakTable2, matchdist=1, matchrate=1, mzabs=0.001, mzppm=10, symmetric=TRUE)`

Author(s)

Joachim Kutzera, `<jkutzer@ipb-halle.de>`

`specDist.peakCount-methods` *a Distance function based on matching peaks*

Description

This method calculates the distance of two sets of peaks by just returning the number of matching peaks (m/z-values).

Usage

```
specDist.peakCount(peakTable1, peakTable2, mzabs=0.001, mzppm=10, symmetric=FALSE)
```

Arguments

`peakTable1`	a Matrix containing at least m/z-values, row must be called "mz"
`peakTable2`	the matrix for the other mz-values
`mzabs`	maximum absolute deviation for two matching peaks
`mzppm`	relative deviations in ppm for two matching peaks
`symmetric`	use symmetric pairwise m/z-matches only, or each match

Methods

peakTable1 = "matrix", peakTable2 = "matrix" `specDist.peakCount(peakTable1, peakTable2, mzppm=10,symmetric=FALSE )`

Author(s)

Joachim Kutzera, `<jkutzer@ipb-halle.de>`

`specNoise` *Calculate noise for a sparse continuum mass spectrum*

Description

Given a sparse continuum mass spectrum, determine regions where no signal is present, substituting half of the minimum intensity for those regions. Calculate the noise level as the weighted mean of the regions with signal and the regions without signal. If there is only one raw peak, return zero.

Usage

```
specNoise(spec, gap = quantile(diff(spec[, "mz"]), 0.9))
```

Arguments

`spec`	matrix with named columns `mz` and `intensity`
`gap`	threshold above which to data points are considerd to be separated by a blank region and not bridged by an interpolating line

Details

The default gap value is determined from the 90th percentile of the pair-wise differences between adjacent mass values.

Value

A numeric noise level

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`getSpec`, `specPeaks`

`specPeaks` *Identify peaks in a sparse continuum mode spectrum*

Description

Given a spectrum, identify and list significant peaks as determined by several criteria.

Usage

```
specPeaks(spec, sn = 20, mzgap = 0.2)
```

Arguments

`spec`	matrix with named columns `mz` and `intensity`
`sn`	minimum signal to noise ratio
`mzgap`	minimal distance between adjacent peaks, with smaller peaks being excluded

Details

Peaks must meet two criteria to be considered peaks: 1) Their s/n ratio must exceed a certain threshold. 2) They must not be within a given distance of any greater intensity peaks.

Value

A matrix with columns:

`mz`	m/z at maximum peak intensity
`intensity`	maximum intensity of the peak
`fwhm`	full width at half max of the peak

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`getSpec`, `specNoise`

`split.xcmsRaw` *Divide an xcmsRaw object*

Description

Divides the scans from a `xcmsRaw` object into a list of multiple objects. MS^n data is discarded.

Arguments

`x`	`xcmsRaw` object
`f`	factor such that `factor(f)` defines the scans which go into the new `xcmsRaw` objects
`drop`	logical indicating if levels that do not occur should be dropped (if 'f' is a 'factor' or a list).
`...`	further potential arguments passed to methods.

Value

A list of `xcmsRaw` objects.

Methods

xr = "xcmsRaw" `split(x, f, drop = TRUE, ...)`

Author(s)

Steffen Neumann, `<sneumann(at)ipb-halle.de>`

See Also

`xcmsRaw-class`

`split.xcmsSet` *Divide an xcmsSet object*

Description

Divides the samples and peaks from a `xcmsSet` object into a list of multiple objects. Group data is discarded.

Arguments

`xs`	`xcmsSet` object
`f`	factor such that `factor(f)` defines the grouping
`drop`	logical indicating if levels that do not occur should be dropped (if 'f' is a 'factor' or a list).
`...`	further potential arguments passed to methods.

Value

A list of `xcmsSet` objects.

Methods

xs = "xcmsSet" `split(x, f, drop = TRUE, ...)`

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`xcmsSet-class`

`SSgauss` *Gaussian Model*

Description

This `selfStart` model evalueates the Gaussian model and its gradient. It has an `initial` attribute that will evalueate the inital estimates of the parameters `mu`, `sigma`, and `h`.

Usage

```
SSgauss(x, mu, sigma, h)
```

Arguments

`x`	a numeric vector of values at which to evaluate the model
`mu`	mean of the distribution function
`sigma`	standard deviation of the distribution fuction
`h`	height of the distribution function

Details

Initial values for `mu` and `h` are chosen from the maximal value of `x`. The initial value for `sigma` is determined from the area under `x` divided by `h*sqrt(2*pi)`.

Value

A numeric vector of the same length as `x`. It is the value of the expression `h*exp(-(x-mu)^2/(2*sigma^2)`, which is a modified gaussian function where the maximum height is treated as a separate parameter not dependent on `sigma`. If arguments `mu`, `sigma`, and `h` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`nls`, `selfStart`

`stitch-methods` *Correct gaps in data*

Description

Fixes gaps in data due to calibration scans or lock mass. Automatically detects file type and calls the relevant method. The mzXML file keeps the data the same length in time but overwrites the lock mass scans. The netCDF version adds the scans back into the data thereby increasing the length of the data and correcting for the unseen gap.

Arguments

`object`	An `xcmsRaw-class` object
`lockMass`	A dataframe of locations of the gaps
`freq`	The intervals of the lock mass scans
`start`	The starting lock mass scan location, default is 1

Details

`makeacqNum` takes locates the gap using the starting lock mass scan and it's intervals. This data frame is then used in `stitch` to correct for the gap caused by the lock mass. Correction works by using scans from either side of the gap to fill it in.

Value

`stitch` A corrected `xcmsRaw-class` object `makeacqNum` A numeric vector of scan locations corresponding to lock Mass scans

Methods

object = "xcmsRaw" `stitch(object, lockMass=numeric())`

object = "xcmsRaw" `makeacqNum(object, freq=numeric(), start=1)`

Author(s)

Paul Benton, `<hpaul.benton08@imperial.ac.uk>`

Examples

```
## Not run: library(xcms)
    library(faahKO)
    ## These files do not have this problem to correct for but just
    ## for an example
    cdfpath <- system.file("cdf", package = "faahKO")
    cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)
    xr<-xcmsRaw(cdffiles[1])
    xr
    ##Lets assume that the lockmass starts at 1 and is every 100 scans
    lockMass<-xcms:::makeacqNum(xr, freq=100, start=1)
    ## these are equcal
    lockmass<-AutoLockMass(xr)
    ob<-stitch(xr, lockMass)
    ob

    ## plot the old data before correction
    foo<-rawEIC(xr, m=c(200,210), scan=c(80,140))
    plot(foo$scan, foo$intensity, type="h")

    ## plot the new corrected data to see what changed
    foo<-rawEIC(ob, m=c(200,210), scan=c(80,140))
    plot(foo$scan, foo$intensity, type="h")

## End(Not run)
```

`updateObject,xcmsSet-method` *Update an `xcmsSet` object*

Description

This method updates an *old* `xcmsSet` object to the latest definition.

Usage

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

Arguments

`object`	The `xcmsSet` object to update.
`...`	Optional additional arguments. Currently ignored.
`verbose`	Currently ignored.

Value

An updated `xcmsSet` containing all data from the input object.

Author(s)

Johannes Rainer

`useOriginalCode` *Enable usage of old xcms code*

Description

This function allows to enable the usage of old, partially deprecated code from xcms by setting a corresponding global option. See details for functions affected.

Usage

```
useOriginalCode(x)
```

Arguments

`x`	`logical(1)` to specify whether or not original old code should be used in corresponding functions. If not provided the function simply returns the value of the global option.

Details

The functions/methods that are affected by this option are:

- do_findChromPeaks_matchedFilter: use the original code that iteratively creates a subset of the binned (profile) matrix. This is helpful for computers with limited memory or matchedFilter settings with a very small bin size.
- getPeaks

Value

`logical(1)` indicating whether old code is being used.

Note

For parallel processing using the SOCKS method (e.g. by `SnowParam()` on Windows computers) this option might not be passed to the individual R processes performing the calculations. In such cases it is suggested to specify the option manually and system-wide by adding the line `options(XCMSuseOriginalCode = TRUE)` in a file called *.Rprofile* in the folder in which new R processes are started (usually the user's home directory; to ensure that the option is correctly read add a new line to the file too). See also Startup from the base R documentation on how to specify system-wide options for R.

Usage of old code is strongly dicouraged. This function is thought to be used mainly in the transition phase from xcms to xcms version 3.

Author(s)

Johannes Rainer

`verify.mzQuantM` *Verify an mzQuantML file*

Description

Export in XML data formats: verify the written data

Usage

```
verify.mzQuantML(filename, xsdfilename)
```

Arguments

`filename`	filename (may include full path) for the output file. Pipes or URLs are not allowed.
`xsdfilename`	Filename of the XSD to verify against (may include full path)

Details

The verify.mzQuantML() function will verify an PSI standard format mzQuantML document against the XSD schemda, see http://www.psidev.info/mzquantml

Value

None.

See Also

`write.mzQuantML`

`write.cdf-methods` *Save an xcmsRaw object to file*

Description

Write the raw data to a (simple) CDF file.

Arguments

`object`	the `xcmsRaw` object
`filename`	filename (may include full path) for the CDF file. Pipes or URLs are not allowed.

Details

Currently the only application known to read the resulting file is XCMS. Others, especially those which build on the AndiMS library, will refuse to load the output.

Value

None.

Methods

object = "xcmsRaw" `write.cdf(object, filename)`

See Also

`xcmsRaw-class`, `xcmsRaw`,

`write.mzdata-methods` *Save an xcmsRaw object to a file*

Description

Write the raw data to a (simple) mzData file.

Arguments

`object`	the `xcmsRaw` object
`filename`	filename (may include full path) for the mzData file. Pipes or URLs are not allowed.

Details

This function will export a given xcmsRaw object to an mzData file. The mzData file will contain a <spectrumList> containing the <spectrum> with mass and intensity values in 32 bit precision. Other formats are currently not supported. Any header information (e.g. additional <software> information or <cvParams>) will be lost. Currently, also any MSn information will not be stored.

Value

None.

Methods

object = "xcmsRaw" `write.mzdata(object, filename)`

See Also

`xcmsRaw-class`, `xcmsRaw`,

`write.mzQuantML-methods`

Save an xcmsSet object to an PSI mzQuantML file

Description

Export in XML data formats: Write the processed data in an xcmsSet to mzQuantML.

Arguments

`object`	the `xcmsRaw` or `xcmsSet` object
`filename`	filename (may include full path) for the output file. Pipes or URLs are not allowed.

Details

The write.mzQuantML() function will write a (grouped) xcmsSet into the PSI standard format mzQuantML, see `http://www.psidev.info/mzquantml`

Value

None.

Methods

object = "xcmsSet" `write.mzQuantML(object, filename)`

See Also

`xcmsSet-class`, `xcmsSet`, `verify.mzQuantML`,

`writeMSData,XCMSnExp,character-method`

Export MS data to mzML/mzXML files

Description

`writeMSData` exports mass spectrometry data in mzML or mzXML format. If adjusted retention times are present, these are used as retention time of the exported spectra.

Usage

```
## S4 method for signature 'XCMSnExp,character'
writeMSData(
  object,
  file,
  outformat = c("mzml", "mzxml"),
  copy = FALSE,
  software_processing = NULL,
  ...
)
```

Arguments

`object`	XCMSnExp object with the mass spectrometry data.
`file`	`character` with the file name(s). The length of this parameter has to match the number of files/samples of `object`.
`outformat`	`character(1)` defining the format of the output files (either `"mzml"` or `"mzxml"`).
`copy`	`logical(1)` if metadata (data processing, software used, original file names etc) should be copied from the original files.

`software_processing`	optionally provide specific data processing steps. See documentation of the `software_processing` parameter of `mzR::writeMSData()`.
`...`	Additional parameters to pass down to the `writeMSData()` function in the `MSnbase` package, such as `outformat` to specify the output format (`"mzml"` or `"mzxml"`) or `copy` to specify whether general information from the original MS data files (such as data processing, software etc) should be copied to the new files.

Author(s)

Johannes Rainer

See Also

`writeMSData()` function in the `MSnbase` package.

`writeMzTab` *Save a grouped xcmsSet object in mzTab-1.1 format file*

Description

Write the grouped xcmsSet to an mzTab file.

Arguments

`object`	the `xcmsSet` object
`filename`	filename (may include full path) for the mzTab file. Pipes or URLs are not allowed.

Details

The mzTab file format for MS-based metabolomics (and proteomics) is a lightweight supplement to the existing standard XML-based file formats (mzML, mzIdentML, mzQuantML), providing a comprehensive summary, similar in concept to the supplemental material of a scientific publication. mzTab files from xcms contain small molecule sections together with experimental metadata and basic quantitative information. The format is intended to store a simple summary of the final results.

Value

None.

Usage

object = "xcmsSet" `writeMzTab(object, filename)`

See Also

`xcmsSet-class`, `xcmsSet`,

Examples

```
library(faahKO)
xs <- group(faahko)

mzt <- data.frame(character(0))
mzt <- xcms:::mzTabHeader(mzt,
                 version="1.1.0", mode="Complete", type="Quantification",
                 description="faahKO",
                 xset=xs)
mzt <- xcms:::mzTabAddSME(mzt, xs)

xcms:::writeMzTab(mzt, "faahKO.mzTab")
```

`XChromatograms` *Containers for chromatographic and peak detection data*

Description

The `XChromatogram` object allows to store chromatographic data (e.g. an extracted ion chromatogram) along with identified chromatographic peaks within that data. The object inherits all functions from the `Chromatogram()` object in the `MSnbase` package.

Multiple `XChromatogram` objects can be stored in a `XChromatograms` object. This class extends `MChromatograms()` from the `MSnbase` package and allows thus to arrange chromatograms in a matrix-like structure, columns representing samples and rows m/z-retention time ranges.

All functions are described (grouped into topic-related sections) after the **Arguments** section.

Usage

```
XChromatograms(data, phenoData, featureData, chromPeaks, chromPeakData, ...)

XChromatogram(
  rtime = numeric(),
  intensity = numeric(),
  mz = c(NA_real_, NA_real_),
  filterMz = c(NA_real_, NA_real_),
  precursorMz = c(NA_real_, NA_real_),
  productMz = c(NA_real_, NA_real_),
  fromFile = integer(),
  aggregationFun = character(),
  msLevel = 1L,
  chromPeaks,
  chromPeakData
)

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

```
## S4 method for signature 'XChromatogram'
chromPeaks(
  object,
  rt = numeric(),
  mz = numeric(),
  ppm = 0,
  type = c("any", "within", "apex_within"),
  msLevel
)

## S4 replacement method for signature 'XChromatogram'
chromPeaks(object) <- value

## S4 method for signature 'XChromatogram,ANY'
plot(
  x,
  col = "#00000060",
  lty = 1,
  type = "l",
  xlab = "retention time",
  ylab = "intensity",
  main = NULL,
  peakType = c("polygon", "point", "rectangle", "none"),
  peakCol = "#00000060",
  peakBg = "#00000020",
  peakPch = 1,
  ...
)

## S4 method for signature 'XChromatogram'
filterMz(object, mz, ...)

## S4 method for signature 'XChromatogram'
filterRt(object, rt, ...)

## S4 method for signature 'XChromatogram'
hasChromPeaks(object)

## S4 method for signature 'XChromatogram'
dropFilledChromPeaks(object)

## S4 method for signature 'XChromatogram'
chromPeakData(object)

## S4 replacement method for signature 'XChromatogram'
chromPeakData(object) <- value

## S4 method for signature 'XChromatogram,MergeNeighboringPeaksParam'
```

```
refineChromPeaks(object, param = MergeNeighboringPeaksParam())

## S4 method for signature 'XChromatogram'
filterChromPeaks(object, method = c("keepTop"), ...)

## S4 method for signature 'XChromatogram'
transformIntensity(object, FUN = identity)

## S4 method for signature 'XChromatograms'
show(object)

## S4 method for signature 'XChromatograms'
hasChromPeaks(object)

## S4 method for signature 'XChromatograms'
hasFilledChromPeaks(object)

## S4 method for signature 'XChromatograms'
chromPeaks(
  object,
  rt = numeric(),
  mz = numeric(),
  ppm = 0,
  type = c("any", "within", "apex_within"),
  msLevel
)

## S4 method for signature 'XChromatograms'
chromPeakData(object)

## S4 method for signature 'XChromatograms'
filterMz(object, mz, ...)

## S4 method for signature 'XChromatograms'
filterRt(object, rt, ...)

## S4 method for signature 'XChromatograms,ANY'
plot(
  x,
  col = "#00000060",
  lty = 1,
  type = "l",
  xlab = "retention time",
  ylab = "intensity",
  main = NULL,
  peakType = c("polygon", "point", "rectangle", "none"),
  peakCol = "#00000060",
  peakBg = "#00000020",
```

```
  peakPch = 1,
  ...
)

## S4 method for signature 'XChromatograms'
processHistory(object, fileIndex, type)

## S4 method for signature 'XChromatograms'
hasFeatures(object, ...)

## S4 method for signature 'XChromatograms'
dropFeatureDefinitions(object, ...)

## S4 method for signature 'XChromatograms,PeakDensityParam'
groupChromPeaks(object, param)

## S4 method for signature 'XChromatograms'
featureDefinitions(
  object,
  mz = numeric(),
  rt = numeric(),
  ppm = 0,
  type = c("any", "within", "apex_within")
)

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

## S4 method for signature 'XChromatograms'
featureValues(
  object,
  method = c("medret", "maxint", "sum"),
  value = "into",
  intensity = "into",
  missing = NA,
  ...
)

## S4 method for signature 'XChromatograms'
plotChromPeakDensity(
  object,
  param,
  col = "#00000060",
  xlab = "retention time",
  main = NULL,
  peakType = c("polygon", "point", "rectangle", "none"),
  peakCol = "#00000060",
  peakBg = "#00000020",
```

```
  peakPch = 1,
  simulate = TRUE,
  ...
)

## S4 method for signature 'XChromatograms'
dropFilledChromPeaks(object)

## S4 method for signature 'XChromatograms,MergeNeighboringPeaksParam'
refineChromPeaks(object, param = MergeNeighboringPeaksParam())

## S4 method for signature 'XChromatograms'
filterChromPeaks(object, method = c("keepTop"), ...)

## S4 method for signature 'XChromatograms'
transformIntensity(object, FUN = identity)
```

Arguments

`data`	For `XChromatograms`: `list` of `Chromatogram` or `XChromatogram` objects.
`phenoData`	For `XChromatograms`: either a `data.frame`, `AnnotatedDataFrame` or `NAnnotatedDataFrame` describing the phenotypical information of the samples.
`featureData`	For `XChromatograms`: either a `data.frame` or `AnnotatedDataFrame` with additional information for each row of chromatograms.
`chromPeaks`	For `XChromatogram`: `matrix` with required columns `"rt"`, `"rtmin"`, `"rtmax"`, `"into"`, `"maxo"` and `"sn"`. For `XChromatograms`: `list`, same length than `data`, with the chromatographic peaks for each chromatogram. Each element has to be a `matrix`, the ordering has to match the order of the chromatograms in `data`.
`chromPeakData`	For `XChromatogram`: `DataFrame` with optional additional annotations for each chromatographic peak. The number of rows has to match the number of chromatographic peaks.
`...`	For `filterChromPeaks`: additional parameters defining how to filter chromatographic peaks. See function description below for details.
`rtime`	For `XChromatogram`: `numeric` with the retention times (length has to be equal to the length of `intensity`).
`intensity`	For `XChromatogram`: `numeric` with the intensity values (length has to be equal to the length of `rtime`). `For `featureValues`: `character(1)` specifying the name of the column in `chromPeaks(object)` containing the intensity value of the peak that should be used for the `method = "maxint"` conflict resolution if.`
`mz`	For `XChromatogram`: `numeric(2)` representing the m/z value range (min, max) on which the chromatogram was created. This is supposed to contain the *real* range of m/z values in contrast to the `filterMz` below. For `chromPeaks` and `featureDefinitions`: `numeric(2)` defining the m/z range for which chromatographic peaks or features should be returned. For `filterMz`: `numeric(2)` defining the m/z range for which chromatographic peaks should be retained.#'

filterMz	For XChromatogram: numeric(2) representing the m/z value range (min, max) that was used to filter the original object on m/z dimension. If not applicable use filterMz = c(0, 0).
precursorMz	For XChromatogram: numeric(2) for SRM/MRM transitions. Represents the mz of the precursor ion. See details for more information.
productMz	For XChromatogram: numeric(2) for SRM/MRM transitions. Represents the mz of the product. See details for more information.
fromFile	For XChromatogram: integer(1) the index of the file within the OnDiskMSnExp or MSnExp object from which the chromatogram was extracted.
aggregationFun	For XChromatogram: character(1) specifying the function that was used to aggregate intensity values for the same retention time across the m/z range.
msLevel	For XChromatogram: integer with the MS level from which the chromatogram was extracted. For chromPeaks and chromPeakData: extract chromatographic peaks of a certain MS level.
object	An XChromatogram or XChromatograms object.
rt	For chromPeaks and featureDefinitions: numeric(2) defining the retention time range for which chromatographic peaks or features should be returned. For filterRt: numeric(2) defining the retention time range to reduce object to.
ppm	For chromPeaks and featureDefinitions: numeric(1) defining a ppm to expand the provided m/z range.
type	For chromPeaks and featureDefinitions: character(1) defining which peaks or features to return if rt or mz is provided: "any" (default) return all peaks that are even partially overlapping with rt, "within" return peaks that are completely within rt and "apex_within" return peaks which apex is within rt. For \`plot\`: what type of plot should be used for the chromatogram (such as \`"l"\` for lines, \`"p"\` for points etc), see help of [plot()] in the \`graphics\` package for more details. For \`processHistory\`: restrict returned processing steps to specific types. Use [processHistoryTypes()] to list all supported values.
value	For chromPeaks<-: a numeric matrix with required columns "rt", "rtmin", "rtmax", "into" and "maxo". For \`featureValues\`: \`character(1)\` specifying the name of the column in \`chromPeaks(object)\` that should be returned or \`"index"\` (default) to return the index of the peak associated with the feature in each sample. To return the integrated peak area instead of the index use \`value = "into"\`.
x	For plot: an XChromatogram or XChromatograms object.
col	For plot: the color to be used to draw the chromatogram.
lty	For plot and plotChromPeakDensity: the line type.
xlab	For plot and plotChromPeakDensity: the x axis label.
ylab	For plot: the y axis label.
main	For plot and plotChromPeakDensity: an optional title for the plot.

`peakType`	For `plot` and `plotChromPeakDensity`: `character(1)` defining how (and if) identified chromatographic peak within the chromatogram should be plotted. Options are `"polygon"` (default): draw the peak borders with the `peakCol` color and fill the peak area with the `peakBg` color, `"point"`: indicate the peak's apex with a point, `"rectangle"`: draw a rectangle around the identified peak and `"none"`: don't draw peaks.
`peakCol`	For `plot` and `plotChromPeakDensity`: the foreground color for the peaks. For `peakType = "polygon"` and `peakType = "rectangle"` this is the color for the border. Use `NA` to not use a foreground color. This should either be a single color or a vector of colors with the same length than `chromPeaks(x)` has rows.
`peakBg`	For `plot` and `plotChromPeakDensity`: the background color for the peaks. For `peakType = "polygon"` and `peakType = "rectangle"` the peak are or rectangle will be filled with this color. Use `NA` to skip. This should be either a single color or a vector of colors with the same length than `chromPeaks(x)` has rows.
`peakPch`	For `plot` and `plotChromPeakDensity`: the point character to be used for `peakType = "point"`. See `plot()` in the `graphics` package for more details.
`param`	For `groupChromPeaks` and `plotChromPeakDensity`: a `PeakDensityParam()` object with the settings for the *peak density* correspondence analysis algorithm.
`method`	For `featureValues`: `character(1)` specifying the method to resolve multi-peak mappings within the sample sample, i.e. to select the *representative* peak for a feature for which more than one peak was assigned in one sample. Options are `"medret"` (default): select the peak closest to the median retention time of the feature, `"maxint"`: select the peak with the largest signal and `"sum"`: sum the values of all peaks (only if `value` is `"into"` or `"maxo"`). For `filterChromPeaks`: `character(1)` defining the method that should be used to filter chromatographic peaks. See help on `filterChromPeaks` below for details.
`FUN`	For `transformIntensity`: a function to transform the intensity values of `object`.
`fileIndex`	For `processHistory`: optional `integer` specifying the index of the files/samples for which the ProcessHistory objects should be returned.
`i`	For `[`: `integer` with the row indices to subset the `XChromatograms` object.
`j`	For `[`: `integer` with the column indices to subset the `XChromatograms` object.
`drop`	For `[`: `logical(1)` whether the dimensionality should be dropped (if possible). Defaults to `drop = TRUE`, thus, if length of `i` and `j` is 1 a `XChromatogram` is returned. Note that `drop` is ignored if length of `i` or `j` is larger than 1, thus a `XChromatograms` is returned.
`missing`	For `featureValues`: how missing values should be reported. Allowed values are `NA` (default), a `numeric(1)` to replace `NA`s with that value or `missing = "rowmin_half"` to replace `NA`s with half of the row's minimal (non-missing) value.
`simulate`	For `plotChromPeakDensity`: `logical(1)` whether a correspondence analysis should be *simulated* based on the available data and the provided `PeakDensityParam()` `param` argument. See section *Correspondence analysis* for details.

Value

See help of the individual functions.

Creation of objects

Objects can be created with the contructor function `XChromatogram` and `XChromatograms`, respectively. Also, they can be coerced from Chromatogram or `MChromatograms()` objects using `as(object, "XChromatogram")` or `as(object, "XChromatograms")`.

Filtering and subsetting

Besides classical subsetting with `[` specific filter operations on `MChromatograms()` and `XChromatograms` objects are available. See `filterColumnsIntensityAbove()` for more details.

- `[` allows to subset a `XChromatograms` object by row (`i`) and column (`j`), with `i` and `j` being of type `integer`. The `featureDefinitions` will also be subsetted accordingly and the `peakidx` column updated.
- `filterMz` filters the chromatographic peaks within an `XChromatogram` or `XChromatograms`, if a column `"mz"` is present in the `chromPeaks` matrix. This would be the case if the `XChromatogram` was extracted from an `XCMSnExp()` object with the `chromatogram()` function. All chromatographic peaks with their m/z within the m/z range defined by `mz` will be retained. Also feature definitions (if present) will be subset accordingly. The function returns a filtered `XChromatogram` or `XChromatograms` object.
- `filterRt` filters chromatogram(s) by the provided retention time range. All eventually present chromatographic peaks with their apex within the retention time range specified with `rt` will be retained. Also feature definitions, if present, will be filtered accordingly. The function returns a filtered `XChromatogram` or `XChromatograms` object.

Accessing data

See also help of Chromatogram in the `MSnbase` package for general information and data access. The methods listed here are specific for `XChromatogram` and `XChromatograms` objects.

- `chromPeaks`, `chromPeaks<-`: extract or set the matrix with the chromatographic peak definitions. Parameter `rt` allows to specify a retention time range for which peaks should be returned along with parameter `type` that defines how *overlapping* is defined (parameter description for details). For `XChromatogram` objects the function returns a `matrix` with columns `"rt"` (retention time of the peak apex), `"rtmin"` (the lower peak boundary), `"rtmax"` (the upper peak boundary), `"into"` (the ingegrated peak signal/area of the peak), `"maxo"` (the maximum instensity of the peak and `"sn"` (the signal to noise ratio). Note that, depending on the peak detection algorithm, the matrix may contain additional columns. For `XChromatograms` objects the `matrix` contains also columns `"row"` and `"column"` specifying in which chromatogram of `object` the peak was identified. Chromatographic peaks are ordered by row.
- `chromPeakData`, `chromPeakData<-`: extract or set the `DataFrame()` with optional chromatographic peak annotations.
- `hasChromPeaks`: infer whether a `XChromatogram` (or `XChromatograms`) has chromatographic peaks. For `XChromatogram`: returns a `logical(1)`, for `XChromatograms`: returns a `matrix`, same dimensions than `object` with either `TRUE` or `FALSE` if chromatographic peaks are available in the chromatogram at the respective position.
- `hasFilledChromPeaks`: whether a `XChromatogram` (or a `XChromatogram` in a `XChromatograms`) has filled-in chromatographic peaks. For `XChromatogram`: returns a `logical(1)`, for `XChromatograms`:

returns a `matrix`, same dimensions than `object` with either `TRUE` or `FALSE` if chromatographic peaks are available in the chromatogram at the respective position.

- `dropFilledChromPeaks`: removes filled-in chromatographic peaks. See `dropFilledChromPeaks()` help for `XCMSnExp()` objects for more information.
- `hasFeatures`: for `XChromatograms` objects only: if correspondence analysis has been performed and m/z-rt feature definitions are present. Returns a `logical(1)`.
- `dropFeatureDefinitions`: for `XChrmomatograms` objects only: delete any correspondence analysis results (and related process history).
- `featureDefinitions`: for `XChromatograms` objects only. Extract the results from the correspondence analysis (performed with `groupChromPeaks`). Returns a `DataFrame` with the properties of the defined m/z-rt features: their m/z and retention time range. Columns `peakidx` and `row` contain the index of the chromatographic peaks in the `chromPeaks` matrix associated with the feature and the row in the `XChromatograms` object in which the feature was defined. Similar to the `chromPeaks` method it is possible to filter the returned feature matrix with the `mz`, `rt` and `ppm` parameters.
- `featureValues`: for `XChromatograms` objects only. Extract the abundance estimates for the individuals features. Note that by default (with parameter `value = "index"` a `matrix` of indices of the peaks in the `chromPeaks` matrix associated to the feature is returned. To extract the integrated peak area use `value = "into"`. The function returns a `matrix` with one row per feature (in `featureDefinitions`) and each column being a sample (i.e. column of `object`). For features without a peak associated in a certain sample `NA` is returned. This can be changed with the `missing` argument of the function.
- `filterChromPeaks`: *filters* chromatographic peaks in `object` depending on parameter `method` and method-specific parameters passed as additional arguments with `...`. Available methods are:
  - `method = "keepTop"`: keep top `n` (default `n = 1L`) peaks in each chromatogram ordered by column `order` (defaults to `order = "maxo"`). Parameter `decreasing` (default `decreasing = TRUE`) can be used to order peaks in descending (`decreasing = TRUE`) or ascending (`decreasing = FALSE`) order to keep the top `n` peaks with largest or smallest values, respectively.
- `processHistory`: returns a `list` of ProcessHistory objects representing the individual performed processing steps. Optional parameters `type` and `fileIndex` allow to further specify which processing steps to return.

Manipulating data

- `transformIntensity`: transforms the intensity values of the chromatograms with provided function `FUN`. See `transformIntensity()` in the `MSnbase` package for details. For `XChromatogram` and `XChromatograms` in addition to the intensity values also columns `"into"` and `"maxo"` in the object's `chromPeaks` matrix are transformed by the same function.

Plotting and visualizing

- `plot` draws the chromatogram and highlights in addition any chromatographic peaks present in the `XChromatogram` or `XChromatograms` (unless `peakType = "none"` was specified). To draw peaks in different colors a vector of color definitions with length equal to `nrow(chromPeaks(x))` has to be submitted with `peakCol` and/or `peakBg` defining one color for each peak (in the order as peaks are in `chromPeaks(x))`. For base peak chromatograms or total ion chromatograms it might be better to set `peakType = "none"` to avoid generating busy plots.

- `plotChromPeakDensity`: visualize *peak density*-based correspondence analysis results. See section *Correspondence analysis* for more details.

Chromatographic peak detection

See findChromPeaks-Chromatogram-CentWaveParam for information.

After chromatographic peak detection it is also possible to *refine* identified chromatographic peaks with the `refineChromPeaks` method (e.g. to reduce peak detection artifacts). Currently, only peak refinement using the *merge neighboring peaks* method is available (see `MergeNeighboringPeaksParam()` for a detailed description of the approach.

Correspondence analysis

Identified chromatographic peaks in an `XChromatograms` object can be grouped into *features* with the `groupChromPeaks` function. Currently, such a correspondence analysis can be performed with the *peak density* method (see groupChromPeaks for more details) specifying the algorithm settings with a `PeakDensityParam()` object. A correspondence analysis is performed separately for each row in the `XChromatograms` object grouping chromatographic peaks across samples (columns).

The analysis results are stored in the returned `XChromatograms` object and can be accessed with the `featureDefinitions` method which returns a `DataFrame` with one row for each feature. Column `"row"` specifies in which row of the `XChromatograms` object the feature was identified.

The `plotChromPeakDensity` method can be used to visualize *peak density* correspondence results, or to *simulate* a peak density correspondence analysis on chromatographic data. The resulting plot consists of two panels, the upper panel showing the chromatographic data as well as the identified chromatographic peaks, the lower panel the distribution of peaks (the peak density) along the retention time axis. This plot shows each peak as a point with it's peak's retention time on the x-axis, and the sample in which it was found on the y-axis. The distribution of peaks along the retention time axis is visualized with a density estimate. Grouped chromatographic peaks are indicated with grey shaded rectangles. Parameter `simulate` allows to define whether the correspondence analysis should be simulated (`simulate=TRUE`, based on the available data and the provided `PeakDensityParam()` parameter class) or not (`simulate=FALSE`). For the latter it is assumed that a correspondence analysis has been performed with the *peak density* method on the `object`. See examples below.

Abundance estimates for each feature can be extracted with the `featureValues` function using parameter `value = "into"` to extract the integrated peak area for each feature. The result is a `matrix`, columns being samples and rows features.

Note

Highlighting the peak area(s) in an `XChromatogram` or `XChromatograms` object (`plot` with `peakType = "polygon"`) draws a polygon representing the displayed chromatogram from the peak's minimal retention time to the maximal retention time. If the `XChromatograms` was extracted from an `XCMSnExp()` object with the `chromatogram()` function this might not represent the actual identified peak area if the m/z range that was used to extract the chromatogram was larger than the peak's m/z.

Author(s)

Johannes Rainer

See Also

findChromPeaks-centWave for peak detection on `MChromatograms()` objects.

Examples

```
## ---- Creation of XChromatograms ----
##
## Create a XChromatograms from Chromatogram objects
dta <- list(Chromatogram(rtime = 1:7, c(3, 4, 6, 12, 8, 3, 2)),
    Chromatogram(1:10, c(4, 6, 3, 4, 7, 13, 43, 34, 23, 9)))

## Create an XChromatograms without peak data
xchrs <- XChromatograms(dta)

## Create an XChromatograms with peaks data
pks <- list(matrix(c(4, 2, 5, 30, 12, NA), nrow = 1,
    dimnames = list(NULL, c("rt", "rtmin", "rtmax", "into", "maxo", "sn"))),
    NULL)
xchrs <- XChromatograms(dta, chromPeaks = pks)

## Create an XChromatograms from XChromatogram objects
dta <- lapply(dta, as, "XChromatogram")
chromPeaks(dta[[1]]) <- pks[[1]]

xchrs <- XChromatograms(dta, nrow = 1)

hasChromPeaks(xchrs)

## Loading a test data set with identified chromatographic peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Subset the dataset to the first and third file.
xod_sub <- filterFile(faahko_sub, file = c(1, 3))

od <- as(xod_sub, "OnDiskMSnExp")

## Extract chromatograms for a m/z - retention time slice
chrs <- chromatogram(od, mz = 344, rt = c(2500, 3500))
chrs

## --------------------------------------------------- ##
##       Chromatographic peak detection                ##
## --------------------------------------------------- ##
## Perform peak detection using CentWave
xchrs <- findChromPeaks(chrs, param = CentWaveParam())
xchrs
```

```
## Do we have chromatographic peaks?
hasChromPeaks(xchrs)

## Process history
processHistory(xchrs)

## The chromatographic peaks, columns "row" and "column" provide information
## in which sample the peak was identified.
chromPeaks(xchrs)

## Spectifically extract chromatographic peaks for one sample/chromatogram
chromPeaks(xchrs[1, 2])

## Plot the results
plot(xchrs)

## Plot the results using a different color for each sample
sample_colors <- c("#ff000040", "#00ff0040", "#0000ff40")
cols <- sample_colors[chromPeaks(xchrs)[, "column"]]
plot(xchrs, col = sample_colors, peakBg = cols)

## Indicate the peaks with a rectangle
plot(xchrs, col = sample_colors, peakCol = cols, peakType = "rectangle",
    peakBg = NA)

## ---------------------------------------------------- ##
##       Correspondence analysis                        ##
## ---------------------------------------------------- ##
## Group chromatographic peaks across samples
prm <- PeakDensityParam(sampleGroup = rep(1, 2))
res <- groupChromPeaks(xchrs, param = prm)

hasFeatures(res)
featureDefinitions(res)

## Plot the correspondence results. Use simulate = FALSE to show the
## actual results. Grouped chromatographic peaks are indicated with
## grey shaded rectangles.
plotChromPeakDensity(res, simulate = FALSE)

## Simulate a correspondence analysis based on different settings. Larger
## bw will increase the smoothing of the density estimate hence grouping
## chromatographic peaks that are more apart on the retention time axis.
prm <- PeakDensityParam(sampleGroup = rep(1, 3), bw = 60)
plotChromPeakDensity(res, param = prm)

## Delete the identified feature definitions
res <- dropFeatureDefinitions(res)
hasFeatures(res)

## Create a XChromatogram object
pks <- matrix(nrow = 1, ncol = 6)
```

```
colnames(pks) <- c("rt", "rtmin", "rtmax", "into", "maxo", "sn")
pks[, "rtmin"] <- 2
pks[, "rtmax"] <- 9
pks[, "rt"] <- 4
pks[, "maxo"] <- 19
pks[, "into"] <- 93

xchr <- XChromatogram(rtime = 1:10,
    intensity = c(4, 8, 14, 19, 18, 12, 9, 8, 5, 2),
    chromPeaks = pks)
xchr

## Add arbitrary peak annotations
df <- DataFrame(peak_id = c("a"))
xchr <- XChromatogram(rtime = 1:10,
    intensity = c(4, 8, 14, 19, 18, 12, 9, 8, 5, 2),
    chromPeaks = pks, chromPeakData = df)
xchr
chromPeakData(xchr)

## Extract the chromatographic peaks
chromPeaks(xchr)

## Plotting of a single XChromatogram object
## o Don't highlight chromatographic peaks
plot(xchr, peakType = "none")

## o Indicate peaks with a polygon
plot(xchr)

## Add a second peak to the data.
pks <- rbind(chromPeaks(xchr), c(7, 7, 10, NA, 15, NA))
chromPeaks(xchr) <- pks

## Plot the peaks in different colors
plot(xchr, peakCol = c("#ff000080", "#0000ff80"),
    peakBg = c("#ff000020", "#0000ff20"))

## Indicate the peaks as rectangles
plot(xchr, peakCol = c("#ff000060", "#0000ff60"), peakBg = NA,
    peakType = "rectangle")

## Filter the XChromatogram by retention time
xchr_sub <- filterRt(xchr, rt = c(4, 6))
xchr_sub
plot(xchr_sub)
```

`xcms-deprecated` *Deprecated functions in package 'xcms'*

Description

These functions are provided for compatibility with older versions of 'xcms' only, and will be defunct at the next release.

Details

The following functions/methods are deprecated.

- `profBin`, `profBinM`, `profBinLin`, `profBinLinM`, `profBinLinBase`, `profBinLinBaseM` have been deprecated and `binYonX` in combination with `imputeLinInterpol` should be used instead.
- `extractMsData`: replaced by `as(x, "data.frame")`.
- `plotMsData`: replaced by `plot(x, type = "XIC")`.

`xcmsEIC-class` *Class xcmsEIC, a class for multi-sample extracted ion chromatograms*

Description

This class is used to store and plot parallel extracted ion chromatograms from multiple sample files. It integrates with the `xcmsSet` class to display peak area integrated during peak identification or fill-in.

Objects from the Class

Objects can be created with the `getEIC` method of the `xcmsSet` class. Objects can also be created by calls of the form `new("xcmsEIC", ...)`.

Slots

`eic`**:** list containing named entries for every sample. for each entry, a list of two column EIC matricies with retention time and intensity

`mzrange`**:** two column matrix containing starting and ending m/z for each EIC

`rtrange`**:** two column matrix containing starting and ending time for each EIC

`rt`**:** either `"raw"` or `"corrected"` to specify retention times contained in the object

`groupnames`**:** group names from `xcmsSet` object used to generate EICs

Methods

groupnames `signature(object = "xcmsEIC")`: get `groupnames` slot

mzrange `signature(object = "xcmsEIC")`: get `mzrange` slot

plot `signature(x = "xcmsEIC")`: plot the extracted ion chromatograms

rtrange `signature(object = "xcmsEIC")`: get `rtrange` slot

sampnames `signature(object = "xcmsEIC")`: get sample names

Note

No notes yet.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`getEIC`

`xcmsFileSource-class` *Base class for loading raw data from a file*

Description

Data sources which read data from a file should inherit from this class. The `xcms` package provides classes to read from `netCDF`, `mzData`, `mzXML`, and `mzML` files using `xcmsFileSource`.

This class should be considered virtual and will not work if passed to `loadRaw-methods`. The reason it is not explicitly virtual is that there does not appear to be a way for a class to be both virtual and have a data part (which lets functions treat objects as if they were character strings).

This class validates that a file exists at the path given.

Objects from the Class

`xcmsFileSource` objects should not be instantiated directly. Instead, create subclasses and instantiate those.

Slots

`.Data`: Object of class `"character"`. File path of a file from which to read raw data as the object's data part

Extends

Class `"character"`, from data part. Class `"xcmsSource"`, directly.

Methods

`xcmsSource signature(object = "character")`: Create an `xcmsFileSource` object referencing the given file name.

Author(s)

Daniel Hackney `<dan@haxney.org>`

See Also

`xcmsSource`

`xcmsFragments` *Constructor for xcmsFragments objects which holds Tandem MS peaks*

Description

EXPERIMANTAL FEATURE

xcmsFragments is an object similar to xcmsSet, which holds peaks picked (or collected) from one or several xcmsRaw objects.

There are still discussions going on about the exact API for MS^n data, so this is likely to change in the future. The code is not yet pipeline-ified.

Usage

```
xcmsFragments(xs, ...)
```

Arguments

`xs`	A `xcmsSet-class` object which contains picked ms1-peaks from one or several experiments
`...`	further arguments to the `collect` method

Details

After running collect(xFragments,xSet) The peaktable of the xcmsFragments includes the ms1Peaks from all experinemts stored in a xcmsSet-object. Further it contains the relevant MSn-peaks from the xcmsRaw-objects, which were created temporarily with the paths in xcmsSet.

Value

An `xcmsFragments` object.

Author(s)

Joachim Kutzera, Steffen Neumann, `<sneumann@ipb-halle.de>`

See Also

`xcmsFragments-class`, `collect`

`xcmsFragments-class` *Class xcmsFragments, a class for handling Tandem MS and MS^n data*

Description

This class is similar to `xcmsSet` because it stores peaks from a number of individual files. However, xcmsFragments keeps Tandem MS and e.g. Ion Trap or Orbitrap MS^n peaks, including the parent ion relationships.

Objects from the Class

Objects can be created with the `xcmsFragments` constructor and filled with peaks using the collect method.

Slots

`peaks`**:** matrix with colmns peakID (MS1 parent in corresponding xcmsSet), MSnParentPeakID (parent peak within this xcmsFragments), msLevel (e.g. 2 for Tandem MS), rt (retention time in case of LC data), mz (fragment mass-to-charge), intensity (peak intensity extracted from the original `xcmsSet`), sample (the index of the rawData-file).

`MS2spec`**:** This is a list of matrixes. Each matrix in the list is a single collected spectra from `collect`. The column ID's are mz, intensity, and full width half maximum(fwhm). The fwhm column is only relevant if the spectra came from profile data.

`specinfo`**:** This is a matrix with reference data for the spectra in MS2spec. The column id's are preMZ, AccMZ, rtmin, rtmax, ref, CollisionEnergy. The preMZ is precursor mass from the MS1 scan. This mass is given by the XML file. With some instruments this mass is only given as nominal mass, therefore a AccMZ is given which is a weighted average mass from the MS1 scan of the collected spectra. The retention time is given by rtmin and rtmax. The ref column is a pointer to the MS2spec matrix spectra. The collisionEnergy column is the collision Energy for the spectra.

Methods

collect `signature(object = "xcmsFragments")`: gets a xcmsSet-object, collects ms1-peaks from it and the msn-peaks from the corresponding xcmsRaw-files.

plotTree `signature(object = "xcmsFragments")`: prints a (text based) pseudo-tree of the peak-table to display the dependencies of the peaks among each other.

show `signature(object = "xcmsFragments")`: print a human-readable description of this object to the console.

Author(s)

S. Neumann, J. Kutzera

See Also

`xcmsRaw`

`XCMSnExp-class` *Data container storing xcms preprocessing results*

Description

The `XCMSnExp` object is a container for the results of a G/LC-MS data preprocessing that comprises chromatographic peak detection, alignment and correspondence. These results can be accessed with the `chromPeaks`, `adjustedRtime` and `featureDefinitions` functions; see below (after the Usage, Arguments, Value and Slots sections) for more details). Along with the results, the object contains the processing history that allows to track each processing step along with the used settings. This can be extracted with the `processHistory` method. `XCMSnExp` objects, by directly extending the `OnDiskMSnExp` object from the `MSnbase` package, inherit all of its functionality and allows thus an easy access to the full raw data at any stage of an analysis. To support interaction with packages requiring the *old* objects, `XCMSnExp` objects can be coerced into `xcmsSet` objects using the `as` method (see examples below). All preprocessing results will be passed along to the resulting `xcmsSet` object.

General functions for `XCMSnExp` objects are (see further below for specific function to handle chromatographic peak data, alignment and correspondence results):

`processHistoryTypes` returns the available *types* of process histories. These can be passed with argument `type` to the `processHistory` method to extract specific process step(s).

`hasFilledChromPeaks`: whether filled-in peaks are present or not.

`featureArea` extracts the m/z - retention time region for each feature. This area is defined by the m/z - retention time regions of all chromatographic peaks associated with a feature. Parameters `mzmin`, `mzmax`, `rtmin` and `rtmax` allow to define functions how the corresponding value is calculated from the individual values (such as the `"rtmin"`) of all chromatographic peaks of that feature. By default the median `"rtmin"`, `"rtmax"`, `"mzmin"` and `"mzmax"` is reported. Parameter `features` allows to provide feature IDs for which the area should be extracted. By default it is extracted for all features.

`profMat`: creates a *profile matrix*, which is a n x m matrix, n (rows) representing equally spaced m/z values (bins) and m (columns) the retention time of the corresponding scans. Each cell contains the maximum intensity measured for the specific scan and m/z values. See `profMat` for more details and description of the various binning methods.

`hasAdjustedRtime`: whether the object provides adjusted retention times.

`hasFeatures`: whether the object contains correspondence results (i.e. features).

`hasChromPeaks`: whether the object contains peak detection results.

`hasFilledChromPeaks`: whether the object contains any filled-in chromatographic peaks.

`adjustedRtime`,`adjustedRtime<-`: extract/set adjusted retention times. `adjustedRtime<-` should not be called manually, it is called internally by the `adjustRtime` methods. For `XCMSnExp` objects, `adjustedRtime<-` does also apply retention time adjustments to eventually present chromatographic peaks. The `bySample` parameter allows to specify whether the adjusted retention time should be grouped by sample (file).

`featureDefinitions`, `featureDefinitions<-`: extract or set the correspondence results, i.e. the mz-rt features (peak groups). Similar to the `chromPeaks` it is possible to extract features for specified m/z and/or rt ranges. The function supports also the parameter `type` that allows to specify which features to be returned if any of `rt` or `mz` is specified. For details see help of `chromPeaks`. See also `featureSummary` for a function to calculate simple feature summaries.

`chromPeaks`, `chromPeaks<-`: extract or set the matrix containing the information on identified chromatographic peaks. Rownames of the matrix represent unique IDs of the respective peaks within the experiment. Parameter `bySample` allows to specify whether peaks should be returned ungrouped (default `bySample = FALSE`) or grouped by sample (`bySample = TRUE`). The `chromPeaks<-` method for `XCMSnExp` objects removes also all correspondence (peak grouping) and retention time correction (alignment) results. The optional arguments `rt`, `mz`, `ppm` and `type` allow to extract only chromatographic peaks overlapping the defined retention time and/or m/z ranges. Argument `type` allows to define how *overlapping* is determined: for `type == "any"` (the default), all peaks that are even partially overlapping the region are returned (i.e. for which either `"mzmin"` or `"mzmax"` of the `chromPeaks` or `featureDefinitions` matrix are within the provided m/z range), for `type == "within"` the full peak has to be within the region (i.e. both `"mzmin"` and `"mzmax"` have to be within the m/z range) and for `type == "apex_within"` the peak's apex position (highest signal of the peak) has to be within the region (i.e. the peak's or features m/z has to be within the m/z range). See description of the return value for details on the returned matrix. Users usually don't have to use the `chromPeaks<-` method directly as detected chromatographic peaks are added to the object by the `findChromPeaks` method. Also, `chromPeaks<-` will replace any existing `chromPeakData`.

`chromPeakData` and `chromPeakData<-` allow to get or set arbitrary chromatographic peak annotations. These are returned or ar returned as a `DataFrame`. Note that the number of rows and the rownames of the `DataFrame` have to match those of `chromPeaks`.

`rtime`: extracts the retention time for each scan. The `bySample` parameter allows to return the values grouped by sample/file and `adjusted` whether adjusted or raw retention times should be returned. By default the method returns adjusted retention times, if they are available (i.e. if retention times were adjusted using the `adjustRtime` method).

`mz`: extracts the mz values from each scan of all files within an `XCMSnExp` object. These values are extracted from the original data files and eventual processing steps are applied *on the fly*. Using the `bySample` parameter it is possible to switch from the default grouping of mz values by spectrum/scan to a grouping by sample/file.

`intensity`: extracts the intensity values from each scan of all files within an `XCMSnExp` object. These values are extracted from the original data files and eventual processing steps are applied *on the fly*. Using the `bySample` parameter it is possible to switch from the default grouping of intensity values by spectrum/scan to a grouping by sample/file.

`spectra`: extracts the `Spectrum` objects containing all data from `object`. The values are extracted from the original data files and eventual processing steps are applied *on the fly*. By setting `bySample = TRUE`, the spectra are returned grouped by sample/file. If the `XCMSnExp` object contains adjusted retention times, these are returned by default in the `Spectrum` objects (can be overwritten by setting `adjusted = FALSE`).

`processHistory`: returns a `list` of `ProcessHistory` objects (or objects inheriting from this base class) representing the individual processing steps that have been performed, eventually along with their settings (`Param` parameter class). Optional arguments `fileIndex`, `type` and `msLevel` allow to restrict to process steps of a certain type or performed on a certain file or MS level.

`dropChromPeaks`: drops any identified chromatographic peaks and returns the object without that information. Note that for `XCMSnExp` objects the method drops by default also results from a correspondence (peak grouping) analysis. Adjusted retention times are removed if the alignment has been performed *after* peak detection. This can be overruled with `keepAdjustedRtime = TRUE`.

`dropFeatureDefinitions`: drops the results from a correspondence (peak grouping) analysis, i.e. the definition of the mz-rt features and returns the object without that information. Note that for `XCMSnExp` objects the method will also by default drop retention time adjustment results, if these were performed after the last peak grouping (i.e. which base on the results from the peak grouping that are going to be removed). All related process history steps are removed too as well as eventually filled in peaks (by `fillChromPeaks`). The parameter `keepAdjustedRtime` can be used to avoid removal of adjusted retention times.

`dropAdjustedRtime`: drops any retention time adjustment information and returns the object without adjusted retention time. For `XCMSnExp` objects, this also reverts the retention times reported for the chromatographic peaks in the peak matrix to the original, raw, ones (after chromatographic peak detection). Note that for `XCMSnExp` objects the method drops also all peak grouping results if these were performed *after* the retention time adjustment. All related process history steps are removed too.

`findChromPeaks` performs chromatographic peak detection on the provided `XCMSnExp` objects. For more details see the method for `XCMSnExp`. Note that by default (with parameter `add = FALSE`) previous peak detection results are removed. Use `add = TRUE` to perform a second round of peak detection and add the newly identified peaks to the previous peak detection results. Correspondence results (features) are always removed prior to peak detection. Previous alignment (retention time adjustment) results are kept, i.e. chromatographic peak detection is performed using adjusted retention times if the data was first aligned using e.g. obiwarp (`adjustRtime-obiwarp`).

`dropFilledChromPeaks`: drops any filled-in chromatographic peaks (filled in by the `fillChromPeaks` method) and all related process history steps.

`spectrapply` applies the provided function to each `Spectrum` in the object and returns its results. If no function is specified the function simply returns the `list` of `Spectrum` objects.

`XCMSnExp` objects can be combined with the `c` function. This combines identified chromatographic peaks and the objects' pheno data but discards alignment results or feature definitions.

`plot` plots the spectrum data (see `plot` for `MSnExp` objects in the `MSnbase` package for more details. For `type = "XIC"`, identified chromatographic peaks will be indicated as rectangles with border color `peakCol`.

Usage

```
processHistoryTypes()

hasFilledChromPeaks(object)

featureArea(
  object,
  mzmin = median,
  mzmax = median,
  rtmin = median,
  rtmax = median,
  msLevel = unique(msLevel(object)),
  features = character()
```

```
)

## S4 method for signature 'OnDiskMSnExp'
profMat(
  object,
  method = "bin",
  step = 0.1,
  baselevel = NULL,
  basespace = NULL,
  mzrange. = NULL,
  fileIndex,
  ...
)

## S4 method for signature 'XCMSnExp'
show(object)

## S4 method for signature 'XCMSnExp'
hasAdjustedRtime(object)

## S4 method for signature 'XCMSnExp'
hasFeatures(object, msLevel = integer())

## S4 method for signature 'XCMSnExp'
hasChromPeaks(object, msLevel = integer())

## S4 method for signature 'XCMSnExp'
hasFilledChromPeaks(object)

## S4 method for signature 'XCMSnExp'
adjustedRtime(object, bySample = FALSE)

## S4 replacement method for signature 'XCMSnExp'
adjustedRtime(object) <- value

## S4 method for signature 'XCMSnExp'
featureDefinitions(
  object,
  mz = numeric(),
  rt = numeric(),
  ppm = 0,
  type = c("any", "within", "apex_within"),
  msLevel = integer()
)

## S4 replacement method for signature 'XCMSnExp'
featureDefinitions(object) <- value
```

```
## S4 method for signature 'XCMSnExp'
chromPeaks(
  object,
  bySample = FALSE,
  rt = numeric(),
  mz = numeric(),
  ppm = 0,
  msLevel = integer(),
  type = c("any", "within", "apex_within"),
  isFilledColumn = FALSE
)

## S4 replacement method for signature 'XCMSnExp'
chromPeaks(object) <- value

## S4 method for signature 'XCMSnExp'
rtime(object, bySample = FALSE, adjusted = hasAdjustedRtime(object))

## S4 method for signature 'XCMSnExp'
mz(object, bySample = FALSE, BPPARAM = bpparam())

## S4 method for signature 'XCMSnExp'
intensity(object, bySample = FALSE, BPPARAM = bpparam())

## S4 method for signature 'XCMSnExp'
spectra(
  object,
  bySample = FALSE,
  adjusted = hasAdjustedRtime(object),
  BPPARAM = bpparam()
)

## S4 method for signature 'XCMSnExp'
processHistory(object, fileIndex, type, msLevel)

## S4 method for signature 'XCMSnExp'
dropChromPeaks(object, keepAdjustedRtime = FALSE)

## S4 method for signature 'XCMSnExp'
dropFeatureDefinitions(object, keepAdjustedRtime = FALSE, dropLastN = -1)

## S4 method for signature 'XCMSnExp'
dropAdjustedRtime(object)

## S4 method for signature 'XCMSnExp'
profMat(
  object,
  method = "bin",
```

```
  step = 0.1,
  baselevel = NULL,
  basespace = NULL,
  mzrange. = NULL,
  fileIndex,
  ...
)

## S4 method for signature 'XCMSnExp,Param'
findChromPeaks(
  object,
  param,
  BPPARAM = bpparam(),
  return.type = "XCMSnExp",
  msLevel = 1L,
  add = FALSE
)

## S4 method for signature 'XCMSnExp'
dropFilledChromPeaks(object)

## S4 method for signature 'XCMSnExp'
spectrapply(object, FUN = NULL, BPPARAM = bpparam(), ...)

## S3 method for class 'XCMSnExp'
c(...)

## S4 method for signature 'XCMSnExp'
chromPeakData(object)

## S4 replacement method for signature 'XCMSnExp'
chromPeakData(object) <- value

## S4 method for signature 'XCMSnExp,missing'
plot(x, y, type = c("spectra", "XIC"), peakCol = "#ff000060", ...)
```

Arguments

`object`	For `adjustedRtime`, `featureDefinitions`, `chromPeaks`, `hasAdjustedRtime`, `hasFeatures` and `hasChromPeaks` either a `MsFeatureData` or a `XCMSnExp` object, for all other methods a `XCMSnExp` object.
`mzmin`	for `featureArea`: `function` to be applied to values in the `"mzmin"` column of all chromatographic peaks of a feature to define the lower m/z value of the feature area. Defaults to `median`.
`mzmax`	for `featureArea`: `function` same as `mzmin` but for the `"mzmax"` column.
`rtmin`	for `featureArea`: `function` same as `mzmin` but for the `"rtmin"` column.
`rtmax`	for `featureArea`: `function` same as `mzmin` but for the `"rtmax"` column.

`msLevel`	`integer` specifying the MS level(s) for which identified chromatographic peaks should be returned.
`features`	for `featureArea`: IDs of features for which the area should be extracted.
`method`	The profile matrix generation method. Allowed are `"bin"`, `"binlin"`, `"binlinbase"` and `"intlin"`. See details section for more information.
`step`	numeric(1) representing the m/z bin size.
`baselevel`	numeric(1) representing the base value to which empty elements (i.e. m/z bins without a measured intensity) should be set. Only considered if `method = "binlinbase"`. See `baseValue` parameter of `imputeLinInterpol` for more details.
`basespace`	numeric(1) representing the m/z length after which the signal will drop to the base level. Linear interpolation will be used between consecutive data points falling within `2 * basespace` to each other. Only considered if `method = "binlinbase"`. If not specified, it defaults to `0.075`. Internally this parameter is translated into the `distance` parameter of the `imputeLinInterpol` function by `distance = floor(basespace / step)`. See `distance` parameter of `imputeLinInterpol` for more details.
`mzrange.`	Optional numeric(2) manually specifying the mz value range to be used for binnind. If not provided, the whole mz value range is used.
`fileIndex`	For `processHistory`: optional `integer` specifying the index of the files/samples for which the `ProcessHistory` objects should be retrieved.
`...`	Additional parameters.
`bySample`	logical(1) specifying whether results should be grouped by sample.
`value`	For `adjustedRtime<-`: a `list` (length equal to the number of samples) with numeric vectors representing the adjusted retention times per scan. For `featureDefinitions<-`: a `DataFrame` with peak grouping information. See return value for the `featureDefinitions` method for the expected format. For `chromPeaks<-`: a `matrix` with information on detected peaks. See return value for the `chromPeaks` method for the expected format.
`mz`	optional `numeric(2)` defining the mz range for which chromatographic peaks should be returned.
`rt`	optional `numeric(2)` defining the retention time range for which chromatographic peaks should be returned.
`ppm`	optional `numeric(1)` specifying the ppm by which the `mz` range should be extended. For a value of `ppm = 10`, all peaks within `mz[1] - ppm / 1e6` and `mz[2] + ppm / 1e6` are returned.
`type`	For `processHistory`: restrict returned `ProcessHistory` objects to analysis steps of a certain type. Use the `processHistoryTypes` to list all supported values. For `chromPeaks`: `character` specifying which peaks to return if `rt` or `mz` are defined. For `type = "any"` all chromatographic peaks partially overlapping the range defined by `mz` and/or `rt` are returned, `type = "within"` returns only peaks completely within the region and `type = "apex_within"` peaks for which the peak's apex is within the region.
`isFilledColumn`	`logical(1)` whether a column `"is_filled"` is included in the returned `"matrix"` providing the information if a peak was filled in. Alternatively, this information would be provided by the `chromPeakData` data frame.

`adjusted`	logical(1) whether adjusted or raw (i.e. the original retention times reported in the files) should be returned.
`BPPARAM`	Parameter class for parallel processing. See `bpparam`.
`keepAdjustedRtime`	For `dropFeatureDefinitions,XCMSnExp`: `logical(1)` defining whether eventually present retention time adjustment should not be dropped. By default dropping feature definitions drops retention time adjustment results too.
`dropLastN`	For `dropFeatureDefinitions,XCMSnExp`: `numeric(1)` defining the number of peak grouping related process history steps to remove. By default `dropLastN = -1`, dropping the chromatographic peaks removes all process history steps related to peak grouping. Setting e.g. `dropLastN = 1` will only remove the most recent peak grouping related process history step.
`param`	A `CentWaveParam`, `MatchedFilterParam`, `MassifquantParam`, `MSWParam` or `CentWavePredIsoParam` object with the settings for the chromatographic peak detection algorithm.
`return.type`	Character specifying what type of object the method should return. Can be either `"XCMSnExp"` (default), `"list"` or `"xcmsSet"`.
`add`	For `findChromPeaks`: if newly identified chromatographic peaks should be added to the peak matrix with the already identified chromatographic peaks. By default (`add = FALSE`) previous peak detection results will be removed.
`FUN`	For `spectrapply`: a function that should be applied to each spectrum in the object.
`x`	For `plot`: `XCMSnExp` object.
`y`	For `plot`: not used.
`peakCol`	For `plot`: the color that should be used to indicate identified chromatographic peaks (only in combination with `type = "XIC"` and if chromatographic peaks are present).

Value

For `profMat`: a `list` with a the profile matrix `matrix` (or matrices if `fileIndex` was not specified or if `length(fileIndex) > 1`). See `profile-matrix` for general help and information about the profile matrix.

For `adjustedRtime`: if `bySample = FALSE` a `numeric` vector with the adjusted retention for each spectrum of all files/samples within the object. If `bySample = TRUE` a `list` (length equal to the number of samples) with adjusted retention times grouped by sample. Returns `NULL` if no adjusted retention times are present.

For `featureDefinitions`: a `DataFrame` with peak grouping information, each row corresponding to one mz-rt feature (grouped peaks within and across samples) and columns `"mzmed"` (median mz value), `"mzmin"` (minimal mz value), `"mzmax"` (maximum mz value), `"rtmed"` (median retention time), `"rtmin"` (minimal retention time), `"rtmax"` (maximal retention time) and `"peakidx"`. Column `"peakidx"` contains a `list` with indices of chromatographic peaks (rows) in the matrix returned by the `chromPeaks` method that belong to that feature group. The method returns `NULL` if no feature definitions are present. `featureDefinitions` supports also parameters `mz`, `rt`, `ppm` and `type` to return only features within certain ranges (see description of `chromPeaks` for details).

For `chromPeaks`: if `bySample = FALSE` a `matrix` (each row being a chromatographic peak, rownames representing unique IDs of the peaks) with at least the following columns: `"mz"` (intensity-weighted mean of mz values of the peak across scans/retention times), `"mzmin"` (minimal mz value), `"mzmax"` (maximal mz value), `"rt"` (retention time of the peak apex), `"rtmin"` (minimal retention time), `"rtmax"` (maximal retention time), `"into"` (integrated, original, intensity of the peak), `"maxo"` (maximum intentity of the peak), `"sample"` (sample index in which the peak was identified) and Depending on the employed peak detection algorithm and the `verboseColumns` parameter of it, additional columns might be returned. If parameter `isFilledColumn` was set to `TRUE` a column named `"is_filled"` is also returned. For `bySample = TRUE` the chromatographic peaks are returned as a `list` of matrices, each containing the chromatographic peaks of a specific sample. For samples in which no peaks were detected a matrix with 0 rows is returned.

For `rtime`: if `bySample = FALSE` a numeric vector with the retention times of each scan, if `bySample = TRUE` a `list` of numeric vectors with the retention times per sample.

For `mz`: if `bySample = FALSE` a `list` with the mz values (numeric vectors) of each scan. If `bySample = TRUE` a `list` with the mz values per sample.

For `intensity`: if `bySample = FALSE` a `list` with the intensity values (numeric vectors) of each scan. If `bySample = TRUE` a `list` with the intensity values per sample.

For `spectra`: if `bySample = FALSE` a `list` with `Spectrum` objects. If `bySample = TRUE` the result is grouped by sample, i.e. as a `list` of `lists`, each element in the *outer* `list` being the `list` of spectra of the specific file.

For `processHistory`: a `list` of `ProcessHistory` objects providing the details of the individual data processing steps that have been performed.

Slots

`.processHistory` `list` with `XProcessHistory` objects tracking all individual analysis steps that have been performed.

`msFeatureData` `MsFeatureData` class extending `environment` and containing the results from a chromatographic peak detection (element `"chromPeaks"`), peak grouping (element `"featureDefinitions"`) and retention time correction (element `"adjustedRtime"`) steps. This object should not be manipulated directly.

Chromatographic peak data

Chromatographic peak data is added to an `XCMSnExp` object by the `findChromPeaks` function. Functions to access chromatographic peak data are:

- `hasChromPeaks` whether chromatographic peak data is available, see below for help of the function.
- `chromPeaks` access chromatographic peaks (see below for help).
- `dropChromPeaks` remove chromatographic peaks (see below for help).
- `dropFilledChromPeaks` remove filled-in peaks (see below for help).
- `fillChromPeaks` fill-in missing peaks (see respective help page).
- `plotChromPeaks` plot identified peaks for a file (see respective help page).
- `plotChromPeakImage` plot distribution of peaks along the retention time axis (see respective help page).

- `highlightChromPeaks` add chromatographic peaks to an existing plot of a `Chromatogram` (see respective help page).

Adjusted retention times

Adjusted retention times are stored in an `XCMSnExp` object besides the original, raw, retention times, allowing to switch between raw and adjusted times. It is also possible to replace the raw retention times with the adjusted ones with the `applyAdjustedRtime`. The adjusted retention times are added to an `XCMSnExp` by the `adjustRtime` function. All functions related to the access of adjusted retention times are:

- `hasAdjustedRtime` whether adjusted retention times are available (see below for help).
- `dropAdjustedRtime` remove adjusted retention times (see below for help).
- `applyAdjustedRtime` replace the raw retention times with the adjusted ones (see respective help page).
- `plotAdjustedRtime` plot differences between adjusted and raw retention times (see respective help page).

Correspondence results, features

The correspondence analysis (`groupChromPeaks`) adds the feature definitions to an `XCMSnExp` object. All functions related to these are listed below:

- `hasFeatures` whether correspondence results are available (see below for help).
- `featureDefinitions` access the definitions of the features (see below for help).
- `dropFeatureDefinitions` remove correspondence results (see below for help).
- `featureValues` access values for features (see respective help page).
- `featureSummary` perform a simple summary of the defined features (see respective help page).
- `overlappingFeatures` identify features that are overlapping or close in the m/z - rt space (see respective help page).
- `quantify` extract feature intensities and put them, along with feature definitions and phenodata information, into a `SummarizedExperiment`. See help page for details.

Note

The `"chromPeaks"` element in the `msFeatureData` slot is equivalent to the `@peaks` slot of the `xcmsSet` object, the `"featureDefinitions"` contains information from the `@groups` and `@groupidx` slots from an `xcmsSet` object.

Author(s)

Johannes Rainer

See Also

`xcmsSet` for the old implementation. `OnDiskMSnExp`, `MSnExp` and `pSet` for a complete list of inherited methods.

`findChromPeaks` for available peak detection methods returning a `XCMSnExp` object as a result.

`groupChromPeaks` for available peak grouping methods and `featureDefinitions` for the method to extract the feature definitions representing the peak grouping results. `adjustRtime` for retention time adjustment methods.

`chromatogram` to extract MS data as `Chromatogram` objects.

`as` (`as(x, "data.frame")`) in the `MSnbase` package for the method to extract MS data as `data.frame`s.

`featureSummary` to calculate basic feature summaries.

`featureChromatograms` to extract chromatograms for each feature.

`chromPeakSpectra` to extract MS2 spectra with the m/z of the precursor ion within the m/z range of a peak and a retention time within its retention time range.

`featureSpectra` to extract MS2 spectra associated with identified features.

`fillChromPeaks` for the method to fill-in eventually missing chromatographic peaks for a feature in some samples.

Examples

```
## Load a test data set with detected peaks
data(faahko_sub)
## Update the path to the files for the local system
dirname(faahko_sub) <- system.file("cdf/KO", package = "faahKO")

## Disable parallel processing for this example
register(SerialParam())

## The results from the peak detection are now stored in the XCMSnExp
## object
faahko_sub

## The detected peaks can be accessed with the chromPeaks method.
head(chromPeaks(faahko_sub))

## The settings of the chromatographic peak detection can be accessed with
## the processHistory method
processHistory(faahko_sub)

## Also the parameter class for the peak detection can be accessed
processParam(processHistory(faahko_sub)[[1]])

## The XCMSnExp inherits all methods from the pSet and OnDiskMSnExp classes
## defined in Bioconductor's MSnbase package. To access the (raw) retention
## time for each spectrum we can use the rtime method. Setting bySample = TRUE
## would cause the retention times to be grouped by sample
head(rtime(faahko_sub))

## Similarly it is possible to extract the mz values or the intensity values
```

```
## using the mz and intensity method, respectively, also with the option to
## return the results grouped by sample instead of the default, which is
## grouped by spectrum. Finally, to extract all of the data we can use the
## spectra method which returns Spectrum objects containing all raw data.
## Note that all these methods read the information from the original input
## files and subsequently apply eventual data processing steps to them.
mzs <- mz(faahko_sub, bySample = TRUE)
length(mzs)
lengths(mzs)

## The full data could also be read using the spectra data, which returns
## a list of Spectrum object containing the mz, intensity and rt values.
## spctr <- spectra(faahko_sub)
## To get all spectra of the first file we can split them by file
## head(split(spctr, fromFile(faahko_sub))[[1]])

############
## Filtering
##
## XCMSnExp objects can be filtered by file, retention time, mz values or
## MS level. For some of these filter preprocessing results (mostly
## retention time correction and peak grouping results) will be dropped.
## Below we filter the XCMSnExp object by file to extract the results for
## only the second file.
xod_2 <- filterFile(faahko_sub, file = 2)
xod_2

## Now the objects contains only the idenfified peaks for the second file
head(chromPeaks(xod_2))

##########
## Coercing to an xcmsSet object
##
## We can also coerce the XCMSnExp object into an xcmsSet object:
xs <- as(faahko_sub, "xcmsSet")
head(peaks(xs))
```

`xcmsPeaks-class` *A matrix of peaks*

Description

A matrix of peak information. The actual columns depend on how it is generated (i.e. the `findPeaks` method).

Objects from the Class

Objects can be created by calls of the form `new("xcmsPeaks", ...)`.

Slots

`.Data`: The matrix holding the peak information

Extends

Class "`matrix`", from data part. Class "`array`", by class "matrix", distance 2. Class "`structure`", by class "matrix", distance 3. Class "`vector`", by class "matrix", distance 4, with explicit coerce.

Methods

None yet. Some utilities for working with peak data would be nice.

Author(s)

Michael Lawrence

See Also

`findPeaks` for detecting peaks in an `xcmsRaw`.

`xcmsRaw` *Constructor for xcmsRaw objects which reads NetCDF/mzXML files*

Description

This function handles the task of reading a NetCDF/mzXML file containing LC/MS or GC/MS data into a new `xcmsRaw` object. It also transforms the data into profile (maxrix) mode for efficient plotting and data exploration.

Usage

```
xcmsRaw(filename, profstep = 1, profmethod = "bin", profparam =
list(), includeMSn=FALSE, mslevel=NULL, scanrange=NULL)

deepCopy(object)
```

Arguments

`filename`	path name of the NetCDF or mzXML file to read
`profstep`	step size (in m/z) to use for profile generation
`profmethod`	method to use for profile generation. See `profile-matrix` for details and supported values.
`profparam`	extra parameters to use for profile generation
`includeMSn`	only for XML file formats: also read MS^n (Tandem-MS of Ion-/Orbi- Trap spectra)

`mslevel`	move data from mslevel into normal MS1 slots, e.g. for peak picking and visualisation
`scanrange`	scan range to read
`object`	An xcmsRaw object

Details

See `profile-matrix` for details on profile matrix generation methods and settings.

The scanrange to import can be restricted, otherwise all MS1 data is read. If `profstep` is set to 0, no profile matrix is generated. Unless `includeMSn = TRUE` only first level MS data is read, not MS/MS, etc.

deepCopy(xraw) will create a copy of the xcmsRaw object with its own copy of mz and intensity data in `xraw@env`.

Value

A `xcmsRaw` object.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

References

NetCDF file format: `https://www.unidata.ucar.edu/software/netcdf/` `http://www.astm.org/Standards/E2077.htm` `http://www.astm.org/Standards/E2078.htm`

mzXML file format: `http://sashimi.sourceforge.net/software_glossolalia.html`

PSI-MS working group who developed mzData and mzML file formats: `http://www.psidev.info/index.php?q=node/80`

Parser used for XML file formats: `http://tools.proteomecenter.org/wiki/index.php?title=Software:RAMP`

See Also

`xcmsRaw-class`, `profStep`, `profMethod` `xcmsFragments`

Examples

```
## Not run:
    library(xcms)
    library(faahKO)
    cdfpath <- system.file("cdf", package = "faahKO")
    cdffiles <- list.files(cdfpath, recursive = TRUE, full.names = TRUE)
    xr<-xcmsRaw(cdffiles[1])
    xr
    ##This gives some information about the file
    names(attributes(xr))
    ## Lets have a look at the structure of the object
```

```
    str(xr)
    ##same but with a preview of each slot in the object
    ##SO... lets have a look at how this works
    head(xr@scanindex)
    ##[1]    0  429  860 1291 1718 2140
    xr@env$mz[425:430]
    ##[1] 596.3 597.0 597.3 598.1 599.3 200.1
    ##We can see that the 429 index is the last mz of scan 1 therefore...

    mz.scan1<-xr@env$mz[(1+xr@scanindex[1]):xr@scanindex[2]]
    intensity.scan1<-xr@env$intensity[(1+xr@scanindex[1]):xr@scanindex[2]]
    plot(mz.scan1, intensity.scan1, type="h",
         main=paste("Scan 1 of file", basename(cdffiles[1]), sep=""))
    ##the easier way :p
    scan1<-getScan(xr, 1)
    head(scan1)
    plotScan(xr, 1)

## End(Not run)
```

`xcmsRaw-class` *Class xcmsRaw, a class for handling raw data*

Description

This class handles processing and visualization of the raw data from a single LC/MS or GS/MS run. It includes methods for producing a standard suite of plots including individual spectra, multi-scan average spectra, TIC, and EIC. It will also produce a feature list of significant peaks using matched filtration.

Objects from the Class

Objects can be created with the `xcmsRaw` constructor which reads data from a NetCDF file into a new object.

Slots

`acquisitionNum`**:** Numeric representing the acquisition number of the individual scans/spectra. Length of `acquisitionNum` is equal to the number of spectra/scans in the object and hence equal to the `scantime` slot. Note however that this information is only available in mzML files.

`env`**:** environment with three variables: `mz` - concatenated m/z values for all scans, `intensity` - corresponding signal intensity for each m/z value, and `profile` - matrix represention of the intensity values with columns representing scans and rows representing equally spaced m/z values. The profile matrix should be extracted with the `profMat` method.

`filepath`**:** Path to the raw data file `gradient`**:** matrix with first row, `time`, containing the time point for interpolation and successive columns representing solvent fractions at each point

`msnAcquisitionNum`**:** for each scan a unique acquisition number as reported via "spectrum id" (mzData) or "<scan num=...>" and "<scanOrigin num=...>" (mzXML)

`msnCollisionEnergy`**:** "CollisionEnergy" (mzData) or "collisionEnergy" (mzXML)

`msnLevel`**:** for each scan the "msLevel" (both mzData and mzXML)

`msnPrecursorCharge`**:** "ChargeState" (mzData) and "precursorCharge" (mzXML)

`msnPrecursorIntensity`**:** "Intensity" (mzData) or "precursorIntensity" (mzXML)

`msnPrecursorMz`**:** "MassToChargeRatio" (mzData) or "precursorMz" (mzXML)

`msnPrecursorScan`**:** "spectrumRef" (both mzData and mzXML)

`msnRt`**:** Retention time of the scan

`msnScanindex`**:** msnScanindex

`mzrange`**:** numeric vector of length 2 with minimum and maximum m/z values represented in the profile matrix

`polarity`**:** polarity

`profmethod`**:** characer value with name of method used for generating the profile matrix.

`profparam`**:** list to store additional profile matrix generation settings. Use the `profinfo` method to extract all profile matrix creation relevant information.

`scanindex`**:** integer vector with starting positions of each scan in the `mz` and `intensity` variables (note that index values are based off a 0 initial position instead of 1).

`scantime`**:** numeric vector with acquisition time (in seconds) for each scan.

`tic`**:** numeric vector with total ion count (intensity) for each scan

`mslevel`**:** Numeric representing the MS level that is present in MS1 slot. This slot should be accessed through its getter method `mslevel`.

`scanrange`**:** Numeric of length 2 specifying the scan range (or `NULL` for the full range). This slot should be accessed through its getter method `scanrange`. Note that the `scanrange` will always be 1 to the number of scans within the `xcmsRaw` object, which does not necessarily have to match to the scan index in the original mzML file (e.g. if the original data was subsetted). The `acquisitionNum` information can be used to track the original *position* of each scan in the mzML file.

Methods

findPeaks `signature(object = "xcmsRaw")`: feature detection using matched filtration in the chromatographic time domain

getEIC `signature(object = "xcmsRaw")`: get extracted ion chromatograms in specified m/z ranges. This will return the total ion chromatogram (TIC) if the m/z range corresponds to the full m/z range (i.e. sum of all signals per retention time across all m/z).

getPeaks `signature(object = "xcmsRaw")`: get data for peaks in specified m/z and time ranges

getScan `signature(object = "xcmsRaw")`: get m/z and intensity values for a single mass scan

getSpec `signature(object = "xcmsRaw")`: get average m/z and intensity values for multiple mass scans

image `signature(x = "xcmsRaw")`: get data for peaks in specified m/z and time ranges

levelplot Create an image of the raw (profile) data m/z against retention time, with the intensity color coded.

mslevel Getter method for the `mslevel` slot.

plotChrom `signature(object = "xcmsRaw")`: plot a chromatogram from profile data

plotRaw `signature(object = "xcmsRaw")`: plot locations of raw intensity data points

plotScan `signature(object = "xcmsRaw")`: plot a mass spectrum of an individual scan from the raw data

plotSpec `signature(object = "xcmsRaw")`: plot a mass spectrum from profile data

plotSurf `signature(object = "xcmsRaw")`: experimental method for plotting 3D surface of profile data with `rgl`.

plotTIC `signature(object = "xcmsRaw")`: plot total ion count chromatogram

profinfo `signature(object = "xcmsRaw")`: returns a list containing the profile generation method and step (profile m/z step size) and eventual additional parameters to the profile function.

profMedFilt `signature(object = "xcmsRaw")`: median filter profile data in time and m/z dimensions

profMethod<- `signature(object = "xcmsRaw")`: change the method of generating the `profile` matrix

profMethod `signature(object = "xcmsRaw")`: get the method of generating the `profile` matrix

profMz `signature(object = "xcmsRaw")`: get vector of m/z values for each row of the `profile` matrix

profRange `signature(object = "xcmsRaw")`: interpret flexible ways of specifying subsets of the `profile` matrix

profStep<- `signature(object = "xcmsRaw")`: change the m/z step used for generating the `profile` matrix

profStep `signature(object = "xcmsRaw")`: get the m/z step used for generating the `profile` matrix

revMz `signature(object = "xcmsRaw")`: reverse the order of the data points for each scan

scanrange Getter method for the `scanrange` slot. See slot description above for more information.

sortMz `signature(object = "xcmsRaw")`: sort the data points by increasing m/z for each scan

stitch `signature(object = "xcmsRaw")`: Raw data correction for lock mass calibration gaps.

findmzROI `signature(object = "xcmsRaw")`: internal function to identify regions of interest in the raw data as part of the first step of centWave-based peak detection.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`, Johannes Rainer `<johannes.rainer@eurac.edu>`

See Also

`xcmsRaw`, `subset-xcmsRaw` for subsetting by spectra.

`xcmsSet` *Constructor for xcmsSet objects which finds peaks in NetCDF/mzXML files*

Description

This function handles the construction of xcmsSet objects. It finds peaks in batch mode and pre-sorts files from subdirectories into different classes suitable for grouping.

Usage

```
xcmsSet(files = NULL, snames = NULL, sclass = NULL, phenoData = NULL,
        profmethod = "bin", profparam = list(),
        polarity = NULL, lockMassFreq=FALSE,
mslevel=NULL, nSlaves=0, progressCallback=NULL,
        scanrange = NULL, BPPARAM = bpparam(),
        stopOnError = TRUE, ...)
```

Arguments

`files`	path names of the NetCDF/mzXML files to read
`snames`	sample names. By default the file name without extension is used.
`sclass`	sample classes.
`phenoData`	`data.frame` or `AnnotatedDataFrame` defining the sample names and classes and other sample related properties. If not provided, the argument `sclass` or the subdirectories in which the samples are stored will be used to specify sample grouping.
`profmethod`	Method to use for profile generation. Supported values are `"bin"`, `"binlin"`, `"binlinbase"` and `"intlin"` (for methods `profBin`, `profBinLin`, `profBinLinBase` and `profIntLin`, respectively). See help on `profBin` for a complete list of available methods and their supported parameters.
`profparam`	parameters to use for profile generation.
`polarity`	filter raw data for positive/negative scans
`lockMassFreq`	Performs correction for Waters LockMass function
`mslevel`	perform peak picking on data of given mslevel
`nSlaves`	*DEPRECATED*, use `BPPARAM` argument instead.
`progressCallback`	function to be called, when progressInfo changes (useful for GUIs)
`scanrange`	scan range to read
`BPPARAM`	a `BiocParallel` parameter object to control how and if parallel processing should be performed. Such objects can be created by the `SerialParam`, `MulticoreParam` or `SnowParam` functions.

`stopOnError`	Logical specifying whether the feature detection call should stop on the first encountered error (the default), or whether feature detection is performed in all files regardless eventual failures for individual files in which case all errors are reported as `warnings`.
`...`	further arguments to the `findPeaks` method of the `xcmsRaw` class

Details

The default values of the `files`, `snames`, `sclass`, and `phenoData` arguments cause the function to recursively search for readable files. The filename without extention is used for the sample name. The subdirectory path is used for the sample class. If the files contain both positive and negative spectra, the polarity can be selected explicitly. The default (NULL) is to read all scans.

If `phenoData` is provided, it is stored to the `phenoData` slot of the returned `xcmsSet` class. If that `data.frame` contains a column named “class”, its content will be returned by the `sampclass` method and thus be used for the group/class assignment of the individual files (e.g. for peak grouping etc.). For more details see the help of the `xcmsSet-class`.

The step size (in m/z) to use for profile generation can be submitted either using the `profparam` argument (e.g. `profparam=list(step=0.1)`) or by submitting `step=0.1`. By specifying a value of `0` the profile matrix generation can be skipped.

The feature/peak detection algorithm can be specified with the `method` argument which defaults to the `"matchFilter"` method (`findPeaks.matchedFilter`). Possible values are returned by `getOption("BioC")$xcms$findPeaks.methods`.

The lock mass correction allows for the lock mass scan to be added back in with the last working scan. This correction gives better reproducibility between sample sets.

Value

A `xcmsSet` object.

Note

The arguments `profmethod` and `profparam` have no influence on the feature/peak detection. The step size parameter `step` for the profile generation in the `findPeaks.matchedFilter` peak detection algorithm can be passed using the `...`.

Author(s)

Colin A. Smith, `<csmith@scripps.edu>`

See Also

`xcmsSet-class`, `findPeaks`, `profStep`, `profMethod`, `profBin`

`xcmsSet-class` *Class xcmsSet, a class for preprocessing peak data*

Description

This class transforms a set of peaks from multiple LC/MS or GC/MS samples into a matrix of preprocessed data. It groups the peaks and does nonlinear retention time correction without internal standards. It fills in missing peak values from raw data. Lastly, it generates extracted ion chromatograms for ions of interest.

Details

The `phenoData` slot (and `phenoData` parameter in the `xcmsSet` function) is intended to contain a `data.frame` describing all experimental factors, i.e. the samples along with their properties. If this `data.frame` contains a column named “class”, this will be returned by the `sampclass` method and will thus be used by all methods to determine the sample grouping/class assignment (e.g. to define the colors in various plots or for the `group` method).

The `sampclass<-` method adds or replaces the “class” column in the `phenoData` slot. If a `data.frame` is submitted to this method, the interaction of its columns will be stored into the “class” column.

Also, similar to other classes in Bioconductor, the `$` method can be used to directly access all columns in the `phenoData` slot (e.g. use `xset$name` on a `xcmsSet` object called “xset” to extract the values from a column named “name” in the `phenoData` slot).

Objects from the Class

Objects can be created with the `xcmsSet` constructor which gathers peaks from a set NetCDF files. Objects can also be created by calls of the form `new("xcmsSet", ...)`.

Slots

peaks `matrix` containing peak data.

filled A vector with peak indices of peaks which have been added by a `fillPeaks` method.

groups Matrix containing statistics about peak groups.

groupidx List containing indices of peaks in each group.

phenoData A `data.frame` containing the experimental design factors.

rt `list` containing two lists, `raw` and `corrected`, each containing retention times for every scan of every sample.

filepaths Character vector with absolute path name of each NetCDF file.

profinfo `list` containing the values `method` - profile generation method, and `step` - profile m/z step size and eventual additional parameters to the profile function.

dataCorrection `logical` vector filled if the waters Lock mass correction parameter is used.

polarity A string ("positive" or "negative" or NULL) describing whether only positive or negative scans have been used reading the raw data.

progressInfo Progress informations for some xcms functions (for GUI).

progressCallback Function to be called, when progressInfo changes (for GUI).

mslevel Numeric representing the MS level on which the peak picking was performed (by default on MS1). This slot should be accessed through its getter method `mslevel`.

scanrange Numeric of length 2 specifying the scan range (or `NULL` for the full range). This slot should be accessed through its getter method `scanrange`. The scan range provided in this slot represents the scans to which the whole raw data is subsetted.

.processHistory Internal slot to be used to keep track of performed processing steps. This slot should not be directly accessed by the user.

Methods

c `signature("xcmsSet")`: combine objects together

filepaths<- `signature(object = "xcmsSet")`: set `filepaths` slot

filepaths `signature(object = "xcmsSet")`: get `filepaths` slot

diffreport `signature(object = "xcmsSet")`: create report of differentially regulated ions including EICs

fillPeaks `signature(object = "xcmsSet")`: fill in peak data for groups with missing peaks

getEIC `signature(object = "xcmsSet")`: get list of EICs for each sample in the set

getXcmsRaw `signature(object = "xcmsSet", sampleidx = 1,profmethod = profMethod(object), profstep = profStep(object),profparam=profinfo(object), mslevel = NULL, scanrange = NULL,rt=c("corrected", "raw"), BPPARAM = bpparam())`: read the raw data for one or more files in the `xcmsSet` and return it. The default parameters will apply all settings used in the original `xcmsSet` call to generate the `xcmsSet` object to be applied also to the raw data. Parameter `sampleidx` allows to specify which raw file(s) should be loaded. Argument `BPPARAM` allows to setup parallel processing.

groupidx<- `signature(object = "xcmsSet")`: set `groupidx` slot

groupidx `signature(object = "xcmsSet")`: get `groupidx` slot

groupnames `signature(object = "xcmsSet")`: get textual names for peak groups

groups<- `signature(object = "xcmsSet")`: set `groups` slot

groups `signature(object = "xcmsSet")`: get `groups` slot

groupval `signature(object = "xcmsSet")`: get matrix of values from peak data with a row for each peak group

group `signature(object = "xcmsSet")`: find groups of peaks across samples that share similar m/z and retention times

mslevel Getter method for the `mslevel` slot.

peaks<- `signature(object = "xcmsSet")`: set `peaks` slot

peaks `signature(object = "xcmsSet")`: get `peaks` slot

plotrt `signature(object = "xcmsSet")`: plot retention time deviation profiles

profinfo<- `signature(object = "xcmsSet")`: set `profinfo` slot

profinfo `signature(object = "xcmsSet")`: get `profinfo` slot

profMethod `signature(object = "xcmsSet")`: extract the method used to generate the profile matrix.

profStep `signature(object = "xcmsSet")`: extract the profile step used for the generation of the profile matrix.

retcor `signature(object = "xcmsSet")`: use initial grouping of peaks to do nonlinear loess retention time correction

sampclass<- `signature(object = "xcmsSet")`: Replaces the column “class” in the `phenoData` slot. See details for more information.

sampclass `signature(object = "xcmsSet")`: Returns the content of the column “class” from the `phenoData` slot or, if not present, the interaction of the experimental design factors (i.e. of the `phenoData` `data.frame`). See details for more information.

phenoData<- `signature(object = "xcmsSet")`: set the `phenoData` slot

phenoData `signature(object = "xcmsSet")`: get the `phenoData` slot

progressCallback<- `signature(object = "xcmsSet")`: set the `progressCallback` slot

progressCallback `signature(object = "xcmsSet")`: get the `progressCallback` slot

scanrange Getter method for the `scanrange` slot. See scanrange slot description above for more details.

sampnames<- `signature(object = "xcmsSet")`: set rownames in the `phenoData` slot

sampnames `signature(object = "xcmsSet")`: get rownames in the `phenoData` slot

split `signature("xcmsSet")`: divide the xcmsSet into a list of xcmsSet objects depending on the provided factor. Note that only peak data will be preserved, i.e. eventual peak grouping information will be lost.

`object$name`**,** `object$name<-value` Access and set `name` column in `phenoData`

`object[, i]` Conducts subsetting of a `xcmsSet` instance. Only subsetting on columns, i.e. samples, is supported. Subsetting is performed on all slots, also on `groups` and `groupidx`. Parameter `i` can be an integer vector, a logical vector or a character vector of sample names (matching `sampnames`).

Author(s)

Colin A. Smith, <`csmith@scripps.edu`>, Johannes Rainer <`johannes.rainer@eurac.edu`>

See Also

`xcmsSet`

`xcmsSource-class` *Virtual class for raw data sources*

Description

This virtual class provides an implementation-independent way to load mass spectrometer data from various sources for use in an `xcmsRaw` object. Subclasses can be defined to enable data to be loaded from user-specified sources. The virtual class `xcmsFileSource` is included out of the box which contains a file name as a character string.

When implementing child classes of `xcmsSource`, a corresponding `loadRaw-methods` method must be provided which accepts the `xcmsSource` child class and returns a list in the format described in `loadRaw-methods`.

Objects from the Class

A virtual Class: No objects may be created from it.

Author(s)

Daniel Hackney, <`dan@haxney.org`>

See Also

`xcmsSource-methods` for creating `xcmsSource` objects in various ways.

`xcmsSource-methods` *Create an* `xcmsSource` *object in a flexible way*

Description

Users can define alternate means of reading data for `xcmsRaw` objects by creating new implementations of this method.

Methods

`signature(object = "xcmsSource")` Pass the object through unmodified.

Author(s)

Daniel Hackney, <`dan@haxney.org`>

See Also

`xcmsSource`

`xdata` *LC-MS preprocessing result test data*

Description

The ‘xdata‘ variable represent the results from a ‘xcms‘-based pre-processing of an LC-MS untargeted metabolomics data set. The raw data files are provided in the ‘faahKO‘ package. The pre-processing of this data set is described in detail in the *xcms* vignette of the ‘xcms‘ package.

`[,xcmsRaw,logicalOrNumeric,missing,missing-method` *Subset an xcmsRaw object by scans*

Description

Subset an `xcmsRaw` object by scans. The returned `xcmsRaw` object contains values for all scans specified with argument `i`. Note that the `scanrange` slot of the returned `xcmsRaw` will be `c(1, length(object@scantime))` and hence not `range(i)`.

Usage

```
## S4 method for signature 'xcmsRaw,logicalOrNumeric,missing,missing'
x[i, j, drop]
```

Arguments

`x`	The `xcmsRaw` object that should be sub-setted.
`i`	Integer or logical vector specifying the scans/spectra to which `x` should be sub-setted.
`j`	Not supported.
`drop`	Not supported.

Details

Only subsetting by scan index in increasing order or by a logical vector are supported. If not ordered, argument `i` is sorted automatically. Indices which are larger than the total number of scans are discarded.

Value

The sub-setted `xcmsRaw` object.

Author(s)

Johannes Rainer

See Also

`split.xcmsRaw`

Examples

```
## Load a test file
file <- system.file('cdf/KO/ko15.CDF', package = "faahKO")
xraw <- xcmsRaw(file, profstep = 0)
## The number of scans/spectra:
length(xraw@scantime)

## Subset the object to scans with a scan time from 3500 to 4000.
xsub <- xraw[xraw@scantime >= 3500 & xraw@scantime <= 4000]
range(xsub@scantime)
## The number of scans:
length(xsub@scantime)
## The number of values of the subset:
length(xsub@env$mz)
```

Index