Package 'Spectra'

June 3, 2023

Title Spectra Infrastructure for Mass Spectrometry Data

Version 1.11.2

Description The Spectra package defines an efficient infrastructure for storing and handling mass spectrometry spectra and functionality to subset, process, visualize and compare spectra data. It provides different implementations (backends) to store mass spectrometry data. These comprise backends tuned for fast data access and processing and backends for very large data sets ensuring a small memory footprint.

Depends R (>= 4.0.0), S4Vectors, BiocParallel, ProtGenerics (>= 1.29.1)

Imports methods, IRanges, MsCoreUtils (>= 1.7.5), graphics, grDevices, stats, tools, utils, fs, BiocGenerics, MetaboCoreUtils

Suggests testthat, knitr (>= 1.1.0), msdata (>= 0.19.3), roxygen2, BiocStyle (>= 2.5.19), mzR (>= 2.19.6), rhdf5 (>= 2.32.0), rmarkdown, vdiffr (>= 1.0.0)

License Artistic-2.0

LazyData false

VignetteBuilder knitr

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

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

biocViews Infrastructure, Proteomics, MassSpectrometry, Metabolomics

RoxygenNote 7.2.3

Roxygen list(markdown=TRUE)

Collate 'hidden_aliases.R' 'AllGenerics.R' 'MsBackend-functions.R' 'MsBackend.R' 'MsBackendCached.R' 'MsBackendDataFrame-functions.R' 'MsBackendDataFrame.R' 'MsBackendHdf5Peaks-functions.R' 'MsBackendHdf5Peaks.R' 'MsBackendMemory-functions.R' 'MsBackendMemory.R' 'MsBackendMzR-functions.R' 'MsBackendMzR.R' 'Spectra-functions.R' 'Spectra.R' 'Spectra-neutralLoss.R' 'countIdentifications.R' 'fft_spectrum.R' 'functions-util.R'

'mz-delta-functions.R' 'peak-list-functions.R'
'peaks-functions.R' 'plotting-functions.R' 'zzz.R'

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

git_branch devel

git_last_commit 3203594

git_last_commit_date 2023-05-17

Date/Publication 2023-06-02

Author RforMassSpectrometry Package Maintainer [cre],
Laurent Gatto [aut] (<https://orcid.org/0000-0002-1520-2268>),
Johannes Rainer [aut] (<https://orcid.org/0000-0002-6977-7147>),
Sebastian Gibb [aut] (<https://orcid.org/0000-0001-7406-4443>),
Jan Stanstrup [ctb] (<https://orcid.org/0000-0003-0541-7369>),
Nir Shahaf [ctb]

Maintainer
RforMassSpectrometry Package Maintainer <maintainer@rformassspectrometry.org>

R topics documented:

`applyProcessing` *The Spectra class to manage and access MS data*

Description

The `Spectra` class encapsules spectral mass spectrometry data and related metadata.

It supports multiple data backends, e.g. in-memory (MsBackendMemory, `MsBackendDataFrame()`), on-disk as mzML (`MsBackendMzR()`) or HDF5 (`MsBackendHdf5Peaks()`).

Usage

```
applyProcessing(object, f = dataStorage(object), BPPARAM = bpparam(), ...)

concatenateSpectra(x, ...)

combineSpectra(
  x,
  f = x$dataStorage,
  p = x$dataStorage,
  FUN = combinePeaks,
  ...,
  BPPARAM = bpparam()
)

joinSpectraData(x, y, by.x = "spectrumId", by.y, suffix.y = ".y")

processingLog(x)

estimatePrecursorIntensity(
  x,
  ppm = 10,
  tolerance = 0,
  method = c("previous", "interpolation"),
  msLevel. = 2L,
  f = dataOrigin(x),
  BPPARAM = bpparam()
)

deisotopeSpectra(
  x,
  substDefinition = isotopicSubstitutionMatrix("HMDB_NEUTRAL"),
  tolerance = 0,
  ppm = 10,
  charge = 1
)

reduceSpectra(x, tolerance = 0, ppm = 10)

## S4 method for signature 'missing'
Spectra(
  object,
  processingQueue = list(),
  metadata = list(),
  ...,
  backend = MsBackendMemory(),
  BPPARAM = bpparam()
)
```

```
## S4 method for signature 'MsBackend'
Spectra(
  object,
  processingQueue = list(),
  metadata = list(),
  ...,
  BPPARAM = bpparam()
)

## S4 method for signature 'character'
Spectra(
  object,
  processingQueue = list(),
  metadata = list(),
  source = MsBackendMzR(),
  backend = source,
  ...,
  BPPARAM = bpparam()
)

## S4 method for signature 'ANY'
Spectra(
  object,
  processingQueue = list(),
  metadata = list(),
  source = MsBackendMemory(),
  backend = source,
  ...,
  BPPARAM = bpparam()
)

## S4 method for signature 'Spectra,MsBackend'
setBackend(object, backend, f = dataStorage(object), ..., BPPARAM = bpparam())

## S4 method for signature 'Spectra'
c(x, ...)

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

## S4 method for signature 'Spectra'
export(object, backend, ...)

## S4 method for signature 'Spectra'
acquisitionNum(object)

## S4 method for signature 'Spectra'
peaksData(object, columns = c("mz", "intensity"), ..., BPPARAM = bpparam())
```

```
## S4 method for signature 'Spectra'
peaksVariables(object)

## S4 method for signature 'Spectra'
centroided(object)

## S4 replacement method for signature 'Spectra'
centroided(object) <- value

## S4 method for signature 'Spectra'
collisionEnergy(object)

## S4 replacement method for signature 'Spectra'
collisionEnergy(object) <- value

## S4 method for signature 'Spectra'
dataOrigin(object)

## S4 replacement method for signature 'Spectra'
dataOrigin(object) <- value

## S4 method for signature 'Spectra'
dataStorage(object)

## S4 method for signature 'Spectra'
dropNaSpectraVariables(object)

## S4 method for signature 'Spectra'
intensity(object, ...)

## S4 method for signature 'Spectra'
ionCount(object)

## S4 method for signature 'Spectra'
isCentroided(object, ...)

## S4 method for signature 'Spectra'
isEmpty(x)

## S4 method for signature 'Spectra'
isolationWindowLowerMz(object)

## S4 replacement method for signature 'Spectra'
isolationWindowLowerMz(object) <- value

## S4 method for signature 'Spectra'
isolationWindowTargetMz(object)
```

```
## S4 replacement method for signature 'Spectra'
isolationWindowTargetMz(object) <- value

## S4 method for signature 'Spectra'
isolationWindowUpperMz(object)

## S4 replacement method for signature 'Spectra'
isolationWindowUpperMz(object) <- value

## S4 method for signature 'Spectra'
containsMz(
  object,
  mz = numeric(),
  tolerance = 0,
  ppm = 20,
  which = c("any", "all"),
  BPPARAM = bpparam()
)

## S4 method for signature 'Spectra'
containsNeutralLoss(
  object,
  neutralLoss = 0,
  tolerance = 0,
  ppm = 20,
  BPPARAM = bpparam()
)

## S4 method for signature 'Spectra'
spectrapply(
  object,
  FUN,
  ...,
  chunkSize = integer(),
  f = factor(),
  BPPARAM = SerialParam()
)

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

## S4 method for signature 'Spectra'
msLevel(object)

## S4 method for signature 'Spectra'
mz(object, ...)
```

```
## S4 method for signature 'Spectra'
lengths(x, use.names = FALSE)

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

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

## S4 method for signature 'Spectra'
precScanNum(object)

## S4 method for signature 'Spectra'
precursorCharge(object)

## S4 method for signature 'Spectra'
precursorIntensity(object)

## S4 method for signature 'Spectra'
precursorMz(object)

## S4 method for signature 'Spectra'
rtime(object)

## S4 replacement method for signature 'Spectra'
rtime(object) <- value

## S4 method for signature 'Spectra'
scanIndex(object)

## S4 method for signature 'Spectra'
selectSpectraVariables(
  object,
  spectraVariables = union(spectraVariables(object), peaksVariables(object))
)

## S4 method for signature 'Spectra'
smoothed(object)

## S4 replacement method for signature 'Spectra'
smoothed(object) <- value

## S4 method for signature 'Spectra'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'Spectra'
spectraData(object) <- value
```

```
## S4 method for signature 'Spectra'
spectraNames(object)

## S4 replacement method for signature 'Spectra'
spectraNames(object) <- value

## S4 method for signature 'Spectra'
spectraVariables(object)

## S4 method for signature 'Spectra'
tic(object, initial = TRUE)

## S4 method for signature 'Spectra'
x$name

## S4 replacement method for signature 'Spectra'
x$name <- value

## S4 method for signature 'Spectra'
x[[i, j, ...]]

## S4 replacement method for signature 'Spectra'
x[[i, j, ...]] <- value

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

## S4 method for signature 'Spectra'
filterAcquisitionNum(
  object,
  n = integer(),
  dataStorage = character(),
  dataOrigin = character()
)

## S4 method for signature 'Spectra'
filterEmptySpectra(object)

## S4 method for signature 'Spectra'
filterDataOrigin(object, dataOrigin = character())

## S4 method for signature 'Spectra'
filterDataStorage(object, dataStorage = character())

## S4 method for signature 'Spectra'
filterFourierTransformArtefacts(
  object,
  halfWindowSize = 0.05,
```

```
  threshold = 0.2,
  keepIsotopes = TRUE,
  maxCharge = 5,
  isotopeTolerance = 0.005
)

## S4 method for signature 'Spectra'
filterIntensity(
  object,
  intensity = c(0, Inf),
  msLevel. = uniqueMsLevels(object),
  ...
)

## S4 method for signature 'Spectra'
filterIsolationWindow(object, mz = numeric())

## S4 method for signature 'Spectra'
filterMsLevel(object, msLevel. = integer())

## S4 method for signature 'Spectra'
filterMzRange(
  object,
  mz = numeric(),
  msLevel. = uniqueMsLevels(object),
  keep = TRUE
)

## S4 method for signature 'Spectra'
filterMzValues(
  object,
  mz = numeric(),
  tolerance = 0,
  ppm = 20,
  msLevel. = uniqueMsLevels(object),
  keep = TRUE
)

## S4 method for signature 'Spectra'
filterPolarity(object, polarity = integer())

## S4 method for signature 'Spectra'
filterPrecursorMz(object, mz = numeric())

## S4 method for signature 'Spectra'
filterPrecursorMzRange(object, mz = numeric())

## S4 method for signature 'Spectra'
```

```
filterPrecursorMzValues(object, mz = numeric(), ppm = 20, tolerance = 0)

## S4 method for signature 'Spectra'
filterPrecursorCharge(object, z = integer())

## S4 method for signature 'Spectra'
filterPrecursorScan(object, acquisitionNum = integer(), f = dataOrigin(object))

## S4 method for signature 'Spectra'
filterRt(object, rt = numeric(), msLevel. = uniqueMsLevels(object))

## S4 method for signature 'Spectra'
reset(object, ...)

## S4 method for signature 'Spectra'
bin(x, binSize = 1L, breaks = NULL, msLevel. = uniqueMsLevels(x), FUN = sum)

## S4 method for signature 'Spectra,Spectra'
compareSpectra(
  x,
  y,
  MAPFUN = joinPeaks,
  tolerance = 0,
  ppm = 20,
  FUN = ndotproduct,
  ...,
  SIMPLIFY = TRUE
)

## S4 method for signature 'Spectra,missing'
compareSpectra(
  x,
  y = NULL,
  MAPFUN = joinPeaks,
  tolerance = 0,
  ppm = 20,
  FUN = ndotproduct,
  ...,
  SIMPLIFY = TRUE
)

## S4 method for signature 'Spectra'
pickPeaks(
  object,
  halfWindowSize = 2L,
  method = c("MAD", "SuperSmoother"),
  snr = 0,
  k = 0L,
```

```
  descending = FALSE,
  threshold = 0,
  msLevel. = uniqueMsLevels(object),
  ...
)

## S4 method for signature 'Spectra'
replaceIntensitiesBelow(
  object,
  threshold = min,
  value = 0,
  msLevel. = uniqueMsLevels(object)
)

## S4 method for signature 'Spectra'
smooth(
  x,
  halfWindowSize = 2L,
  method = c("MovingAverage", "WeightedMovingAverage", "SavitzkyGolay"),
  msLevel. = uniqueMsLevels(x),
  ...
)

## S4 method for signature 'Spectra'
addProcessing(object, FUN, ..., spectraVariables = character())

coreSpectraVariables()

## S4 method for signature 'Spectra'
uniqueMsLevels(object, ...)

## S4 method for signature 'Spectra'
backendBpparam(object, BPPARAM = bpparam())
```

Arguments

object	For Spectra: either a DataFrame or missing. See section on creation of Spectra objects for details. For all other methods a Spectra object.
f	For split: factor defining how to split x. See base::split() for details. For setBackend: factor defining how to split the data for parallelized copying of the spectra data to the new backend. For some backends changing this parameter can lead to errors. For combineSpectra: factor defining the grouping of the spectra that should be combined. For spectrapply: factor how object should be splitted. For estimatePrecursorIntensity and filterPrecursorScan: defining which spectra belong to the same original data file (sample). Defaults to f = dataOrigin(x).
BPPARAM	Parallel setup configuration. See bpparam() for more information. This is passed directly to the backendInitialize() method of the MsBackend.

`...` Additional arguments.

`x` A `Spectra` object.

`p` For `combineSpectra`: `factor` defining how to split the input `Spectra` for parallel processing. Defaults to `x$dataStorage`, i.e., depending on the used backend, per-file parallel processing will be performed.

`FUN` For `addProcessing`: function to be applied to the peak matrix of each spectrum in `object`. For `compareSpectra`: function to compare intensities of peaks between two spectra with each other. For `combineSpectra`: function to combine the (peak matrices) of the spectra. See section *Data manipulations* and examples below for more details. For `bin`: function to aggregate intensity values of peaks falling into the same bin. Defaults to `FUN = sum` thus summing up intensities. For `spectrapply` and `chunkapply`: function to be applied to `Spectra`.

`y` A `Spectra` object. A `DataFrame` for `joinSpectraData()`.

`by.x` A `character(1)` specifying the spectra variable used for merging. Default is `"spectrumId"`.

`by.y` A `character(1)` specifying the column used for merging. Set to `by.x` if missing.

`suffix.y` A `character(1)` specifying the suffix to be used for making the names of columns in the merged spectra variables unique. This suffix will be used to amend `names(y)`, while `spectraVariables(x)` will remain unchanged.

`ppm` For `compareSpectra`, `containsMz`, `deisotopeSpectra`, `filterMzValues` and `reduceSpectra`: `numeric(1)` defining a relative, m/z-dependent, maximal accepted difference between m/z values for peaks to be matched (or grouped).

`tolerance` For `compareSpectra`, `containsMz`, `deisotopeSpectra`, `filterMzValues` and `reduceSpectra`: `numeric(1)` allowing to define a constant maximal accepted difference between m/z values for peaks to be matched (or grouped). For `containsMz` it can also be of length equal `mz` to specify a different tolerance for each m/z value.

`method`

- For `pickPeaks`: `character(1)`, the noise estimators that should be used, currently the the *M*edian *A*bsolute *D*eviation (`method = "MAD"`) and Friedman's Super Smoother (`method = "SuperSmoother"`) are supported.
- For `smooth`: `character(1)`, the smoothing function that should be used, currently, the Moving-Average- (`method = "MovingAverage"`), Weighted-Moving-Average- (`method = "WeightedMovingAverage"`), Savitzky-Golay-Smoothing (`method = "SavitzkyGolay"`) are supported.
- For `estimatePrecursorIntensity`: `character(1)` defining whether the precursor intensity should be estimated on the previous MS1 spectrum (`method = "previous"`, the default) or based on an interpolation on the previous and next MS1 spectrum (`method = "interpolation"`).

`msLevel.` `integer` defining the MS level(s) of the spectra to which the function should be applied (defaults to all MS levels of `object`. For `filterMsLevel`: the MS level to which `object` should be subsetted.

`substDefinition`

For `deisotopeSpectra`: `matrix` or `data.frame` with definitions of isotopic substitutions. Uses by default isotopic substitutions defined from all compounds

in the Human Metabolome Database (HMDB). See `isotopologues()` or `isotopicSubstitutionMatri`
for details.

`charge` For `deisotopeSpectra`: expected charge of the ionized compounds. See `isotopologues()` for details.

`processingQueue`
For `Spectra`: optional `list` of ProcessingStep objects.

`metadata` For `Spectra`: optional `list` with metadata information.

`backend` For `Spectra`: MsBackend to be used as backend. See section on creation of `Spectra` objects for details. For `setBackend`: instance of MsBackend that supports `setBackend` (i.e. for which `supportsSetBackend` returns `TRUE`). Such backends have a parameter `data` in their `backendInitialize` function that support passing the full spectra data to the initialize method. See section on creation of `Spectra` objects for details. For `export`: MsBackend to be used to export the data.

`source` For `Spectra`: instance of MsBackend that can be used to import spectrum data from the provided files. See section *Creation of objects, conversion and changing the backend* for more details.

`drop` For `[`, `split`: not considered.

`columns` For `spectraData` accessor: optional `character` with column names (spectra variables) that should be included in the returned `DataFrame`. By default, all columns are returned. For `peaksData` accessor: optional `character` with requested columns in the individual `matrix` of the returned `list`. Defaults to `c("mz", "value")` but any values returned by `peaksVariables(object)` with `object` being the `Spectra` object are supported.

`value` replacement value for `<-` methods. See individual method description or expected data type.

`mz` For `filterIsolationWindow`: `numeric(1)` with the m/z value to filter the object. For `filterPrecursorMz` and `filterMzRange`: `numeric(2)` defining the lower and upper m/z boundary. For `filterMzValues` and `filterPrecursorMzValues`: `numeric` with the m/z values to match peaks or precursor m/z against.

`which` for `containsMz`: either `"any"` or `"all"` defining whether any (the default) or all provided `mz` have to be present in the spectrum.

`neutralLoss` for `containsNeutralLoss`: `numeric(1)` defining the value which should be subtracted from the spectrum's precursor m/z.

`chunkSize` For `spectrapply`: size of the chunks into which `Spectra` should be split. This parameter overrides parameters `f` and `BPPARAM`.

`use.names` For `lengths`: ignored.

`spectraVariables`
For `selectSpectraVariables`: `character` with the names of the spectra variables to which the backend should be subsetted. For `addProcessing`: `character` with additional spectra variables that should be passed along to the function defined with `FUN`. See function description for details.

`initial` For `tic`: `logical(1)` whether the initially reported total ion current should be reported, or whether the total ion current should be (re)calculated on the actual data (`initial = FALSE`, same as `ionCount`).

name — For $ and $<-: the name of the spectra variable to return or set.

i — For [: integer, logical or character to subset the object.

j — For [: not supported.

n — for filterAcquisitionNum: integer with the acquisition numbers to filter for.

dataStorage — For filterDataStorage: character to define which spectra to keep. For filterAcquisitionNum: optionally specify if filtering should occur only for spectra of selected dataStorage.

dataOrigin — For filterDataOrigin: character to define which spectra to keep. For filterAcquisitionNum: optionally specify if filtering should occurr only for spectra of selected dataOrigin.

halfWindowSize
- For pickPeaks: integer(1), used in the identification of the mass peaks: a local maximum has to be the maximum in the window from (i - halfWindowSize):(i + halfWindowSize).
- For smooth: integer(1), used in the smoothing algorithm, the window reaches from (i - halfWindowSize):(i + halfWindowSize).
- For filterFourierTransformArtefacts: numeric(1) defining the m/z window left and right of a peak where to remove fourier transform artefacts.

threshold
- For pickPeaks: a double(1) defining the proportion of the maximal peak intensity. Just values above are used for the weighted mean calculation.
- For replaceIntensitiesBelow: a numeric(1) defining the threshold or a function to calculate the threshold for each spectrum on its intensity values. Defaults to threshold = min.
- For filterFourierTransformArtefacts: the relative intensity (to a peak) below which peaks are considered fourier artefacts. Defaults to threshold = 0.2 hence removing peaks that have an intensity below 0.2 times the intensity of the tested peak (within the selected halfWindowSize).

keepIsotopes — For filterFourierTransformArtefacts: whether isotope peaks should not be removed as fourier artefacts.

maxCharge — For filterFourierTransformArtefacts: the maximum charge to be considered for isotopes.

isotopeTolerance — For filterFourierTransformArtefacts: the m/z tolerance to be used to define whether peaks might be isotopes of the current tested peak.

intensity — For filterIntensity: numeric of length 1 or 2 defining either the lower or the lower and upper intensity limit for the filtering, or a function that takes the intensities as input and returns a logical (same length then peaks in the spectrum) whether the peak should be retained or not. Defaults to intensity = c(0, Inf) thus only peaks with NA intensity are removed.

keep — For filterMzValues and filterMzRange: logical(1) whether the matching peaks should be retained (keep = TRUE, the default) or dropped (keep = FALSE').

polarity — for filterPolarity: integer specifying the polarity to to subset object.

z — For filterPrecursorCharge: integer() with the precursor charges to be used as filter.

acquisitionNum	for filterPrecursorScan: integer with the acquisition number of the spectra to which the object should be subsetted.
rt	for filterRt: numeric(2) defining the retention time range to be used to subset/filter object.
binSize	For bin: numeric(1) defining the size for the m/z bins. Defaults to binSize = 1.
breaks	For bin: numeric defining the m/z breakpoints between bins.
MAPFUN	For compareSpectra: function to map/match peaks between the two compared spectra. See joinPeaks() for more information and possible functions.
SIMPLIFY	For compareSpectra whether the result matrix should be *simplified* to a numeric if possible (i.e. if either x or y is of length 1).
snr	For pickPeaks: double(1) defining the *S*ignal-to-*N*oise-*R*atio. The intensity of a local maximum has to be higher than snr * noise to be considered as peak.
k	For pickPeaks: integer(1), number of values left and right of the peak that should be considered in the weighted mean calculation.
descending	For pickPeaks: logical, if TRUE just values between the nearest valleys around the peak centroids are used.

Details

The Spectra class uses by default a lazy data manipulation strategy, i.e. data manipulations such as performed with replaceIntensitiesBelow are not applied immediately to the data, but applied on-the-fly to the spectrum data once it is retrieved. For some backends that allow to write data back to the data storage (such as the MsBackendMemory(), MsBackendDataFrame() and MsBackendHdf5Peaks()) it is possible to apply to queue with the applyProcessing function. See the *Data manipulation and analysis methods* section below for more details.

To apply arbitrary functions to a Spectra use the spectrapply function (or directly chunkapply() for chunk-wise processing). See description of the spectrapply function below for details.

For details on plotting spectra, see plotSpectra().

Clarifications regarding scan/acquisition numbers and indices:

- A spectrumId (or spectrumID) is a vendor specific field in the mzML file that contains some information about the run/spectrum, e.g.: controllerType=0 controllerNumber=1 scan=5281 file=2
- acquisitionNum is a more a less sanitize spectrum id generated from the spectrumId field by mzR (see here).
- scanIndex is the mzR generated sequence number of the spectrum in the raw file (which doesn't have to be the same as the acquisitionNum)

See also this issue.

Value

See individual method description for the return value.

Creation of objects, conversion, changing the backend and export

`Spectra` classes can be created with the `Spectra` constructor function which supports the following formats:

- parameter `object` is a `data.frame` or `DataFrame` containing the spectrum data. The provided `backend` (by default a MsBackendMemory) will be initialized with that data.
- parameter `object` is a MsBackend (assumed to be already initialized).
- parameter `object` is missing, in which case it is supposed that the data is provided by the MsBackend class passed along with the `backend` argument.
- parameter `object` is of type `character` and is expected to be the file names(s) from which spectra should be imported. Parameter `source` allows to define a MsBackend that is able to import the data from the provided source files. The default value for `source` is `MsBackendMzR()` which allows to import spectra data from mzML, mzXML or CDF files.

With `...` additional arguments can be passed to the backend's `backendInitialize()` method. Parameter `backend` allows to specify which MsBackend should be used for data storage.

The backend of a `Spectra` object can be changed with the `setBackend` method that takes an instance of the new backend as second parameter `backend`. A call to `setBackend(sps, backend = MsBackendDataFrame())` would for example change the backend of `sps` to the *in-memory* `MsBackendDataFrame`. Changing to a backend is only supported if that backend has a `data` parameter in its `backendInitialize` method and if `supportsSetBackend` returns `TRUE` for that backend. `setBackend` will transfer the full spectra data from the originating backend as a `DataFrame` to the new backend. Most *read-only* backends do not support `setBackend`. It is for example not possible to change the backend to a *read-only* backend (such as the `MsBackendMzR()` backend).

The definition of the function is: `setBackend(object, backend, ..., f = dataStorage(object), BPPARAM = bpparam())` and its parameters are:

- parameter `object`: the `Spectra` object.
- parameter `backend`: an instance of the new backend, e.g. `[MsBackendMemory()]`.
- parameter `f`: factor allowing to parallelize the change of the backends. By default the process of copying the spectra data from the original to the new backend is performed separately (and in parallel) for each file. Users are advised to use the default setting.
- parameter `...`: optional additional arguments passed to the `backendInitialize()` method of the new `backend`.
- parameter `BPPARAM`: setup for the parallel processing. See `bpparam()` for details.

Data from a `Spectra` object can be **exported** to a file with the `export` function. The actual export of the data has to be performed by the `export` method of the MsBackend class defined with the mandatory parameter `backend`. Note however that not all backend classes support export of data. From the `MsBackend` classes in the `Spectra` package currently only the `MsBackendMzR` backend supports data export (to mzML/mzXML file(s)); see the help page of the MsBackend for information on its arguments or the examples below or the vignette for examples.

The definition of the function is `export(object, backend, ...)` and its parameters are:

- `object`: the `Spectra` object to be exported.
- `backend`: instance of a class extending MsBackend which supports export of the data (i.e. which has a defined `export` method).
- `...`: additional parameters specific for the `MsBackend` passed with parameter `backend`.

Accessing spectra data

- `$`, `$<-`: gets (or sets) a spectra variable for all spectra in `object`. See examples for details.
- `[[`, `[[<-`: access or set/add a single spectrum variable (column) in the backend.
- `acquisitionNum`: returns the acquisition number of each spectrum. Returns an `integer` of length equal to the number of spectra (with `NA_integer_` if not available).
- `centroided`, `centroided<-`: gets or sets the centroiding information of the spectra. `centroided` returns a `logical` vector of length equal to the number of spectra with `TRUE` if a spectrum is centroided, `FALSE` if it is in profile mode and `NA` if it is undefined. See also `isCentroided` for estimating from the spectrum data whether the spectrum is centroided. `value` for `centroided<-` is either a single `logical` or a `logical` of length equal to the number of spectra in `object`.
- `collisionEnergy`, `collisionEnergy<-`: gets or sets the collision energy for all spectra in `object`. `collisionEnergy` returns a `numeric` with length equal to the number of spectra (`NA_real_` if not present/defined), `collisionEnergy<-` takes a `numeric` of length equal to the number of spectra in `object`.
- `coreSpectraVariables`: returns the *core* spectra variables along with their expected data type.
- `dataOrigin`, `dataOrigin<-`: gets or sets the *data origin* for each spectrum. `dataOrigin` returns a `character` vector (same length than `object`) with the origin of the spectra. `dataOrigin<-` expects a `character` vector (same length than `object`) with the replacement values for the data origin of each spectrum.
- `dataStorage`: returns a `character` vector (same length than `object`) with the data storage location of each spectrum.
- `intensity`: gets the intensity values from the spectra. Returns a `NumericList()` of `numeric` vectors (intensity values for each spectrum). The length of the list is equal to the number of `spectra` in `object`.
- `ionCount`: returns a `numeric` with the sum of intensities for each spectrum. If the spectrum is empty (see `isEmpty`), `NA_real_` is returned.
- `isCentroided`: a heuristic approach assessing if the spectra in `object` are in profile or centroided mode. The function takes the `qtl`th quantile top peaks, then calculates the difference between adjacent m/z value and returns `TRUE` if the first quartile is greater than `k`. (See `Spectra:::.isCentroided` for the code.)
- `isEmpty`: checks whether a spectrum in `object` is empty (i.e. does not contain any peaks). Returns a `logical` vector of length equal number of spectra.
- `isolationWindowLowerMz`, `isolationWindowLowerMz<-`: gets or sets the lower m/z boundary of the isolation window.
- `isolationWindowTargetMz`, `isolationWindowTargetMz<-`: gets or sets the target m/z of the isolation window.
- `isolationWindowUpperMz`, `isolationWindowUpperMz<-`: gets or sets the upper m/z boundary of the isolation window.
- `containsMz`: checks for each of the spectra whether they contain mass peaks with an m/z equal to `mz` (given acceptable difference as defined by parameters `tolerance` and `ppm` - see `common()` for details). Parameter `which` allows to define whether any (`which = "any"`, the default) or all (`which = "all"`) of the `mz` have to match. The function returns `NA` if `mz` is of length 0 or is `NA`.

- containsNeutralLoss: checks for each spectrum in object if it has a peak with an m/z value equal to its precursor m/z - neutralLoss (given acceptable difference as defined by parameters tolerance and ppm). Returns NA for MS1 spectra (or spectra without a precursor m/z).
- length: gets the number of spectra in the object.
- lengths: gets the number of peaks (m/z-intensity values) per spectrum. Returns an integer vector (length equal to the number of spectra). For empty spectra, 0 is returned.
- msLevel: gets the spectra's MS level. Returns an integer vector (names being spectrum names, length equal to the number of spectra) with the MS level for each spectrum.
- mz: gets the mass-to-charge ratios (m/z) from the spectra. Returns a NumericList() or length equal to the number of spectra, each element a numeric vector with the m/z values of one spectrum.
- peaksData: gets the *peaks* matrices for all spectra in object. The function returns a SimpleList() of matrices, each matrix with columns "mz" and "intensity" with the m/z and intensity values for all peaks of a spectrum. Optional parameter columns is passed to the backend's peaksData function to allow selection of specific (or additional) peaks variables (columns) that should be extracted (if available). Importantly, it is **not** guaranteed that each backend supports this parameter (while each backend must support extraction of "mz" and "intensity" columns). Parameter columns defaults to c("mz", "intensity") but any value returned from peaksVariables is supported. Note also that it is possible to extract the peaks matrices with as(x, "list") and as(x, "SimpleList") as a list and SimpleList, respectively. Note however that, in contrast to peaksData, as does not support the parameter columns.
- peaksVariables: lists the available variables for mass peaks provided by the backend. Default peak variables are "mz" and "intensity" (which all backends need to support and provide), but some backends might provide additional variables. These variables correspond to the column names of the numeric matrix representing the peak data (returned by peaksData).
- polarity, polarity<-: gets or sets the polarity for each spectrum. polarity returns an integer vector (length equal to the number of spectra), with 0 and 1 representing negative and positive polarities, respectively. polarity<- expects an integer vector of length 1 or equal to the number of spectra.
- precursorCharge, precursorIntensity, precursorMz, precScanNum, precAcquisitionNum: gets the charge (integer), intensity (numeric), m/z (numeric), scan index (integer) and acquisition number (interger) of the precursor for MS level > 2 spectra from the object. Returns a vector of length equal to the number of spectra in object. NA are reported for MS1 spectra of if no precursor information is available.
- rtime, rtime<-: gets or sets the retention times (in seconds) for each spectrum. rtime returns a numeric vector (length equal to the number of spectra) with the retention time for each spectrum. rtime<- expects a numeric vector with length equal to the number of spectra.
- scanIndex: returns an integer vector with the *scan index* for each spectrum. This represents the relative index of the spectrum within each file. Note that this can be different to the acquisitionNum of the spectrum which represents the index of the spectrum during acquisition/measurement (as reported in the mzML file).
- smoothed,smoothed<-: gets or sets whether a spectrum is *smoothed*. smoothed returns a logical vector of length equal to the number of spectra. smoothed<- takes a logical vector of length 1 or equal to the number of spectra in object.

- spectraData: gets general spectrum metadata (annotation, also called header). spectraData returns a DataFrame. Note that this method does by default **not** return m/z or intensity values.
- spectraData<-: **replaces** the full spectra data of the Spectra object with the one provided with value. The use of this function is disencouraged, as replacing spectra data with values that are in a different can break the linkage with the associated m/z and intensity values. If possible, spectra variables (i.e. *columns* of the Spectra) should be replaced individually. The spectraData<- function expects a DataFrame to be passed as value.
- spectraNames, spectraNames<-: gets or sets the spectra names.
- spectraVariables: returns a character vector with the available spectra variables (columns, fields or attributes) available in object.
- tic: gets the total ion current/count (sum of signal of a spectrum) for all spectra in object. By default, the value reported in the original raw data file is returned. For an empty spectrum, 0 is returned.
- uniqueMsLevels: get the unique MS levels available in object. This function is supposed to be more efficient than unique(msLevel(object)).

Data subsetting, filtering and merging

Subsetting and filtering of Spectra objects can be performed with the below listed methods.

- [: subsets the spectra keeping only selected elements (i). The method **always** returns a Spectra object.
- dropNaSpectraVariables: removes spectra variables (i.e. columns in the object's spectraData that contain only missing values (NA). Note that while columns with only NAs are removed, a spectraData call after dropNaSpectraVariables might still show columns containing NA values for *core* spectra variables.
- filterAcquisitionNum: filters the object keeping only spectra matching the provided acquisition numbers (argument n). If dataOrigin or dataStorage is also provided, object is subsetted to the spectra with an acquisition number equal to n **in spectra with matching dataOrigin or dataStorage values** retaining all other spectra. Returns the filtered Spectra.
- filterDataOrigin: filters the object retaining spectra matching the provided dataOrigin. Parameter dataOrigin has to be of type character and needs to match exactly the data origin value of the spectra to subset. Returns the filtered Spectra object (with spectra ordered according to the provided dataOrigin parameter).
- filterDataStorage: filters the object retaining spectra stored in the specified dataStorage. Parameter dataStorage has to be of type character and needs to match exactly the data storage value of the spectra to subset. Returns the filtered Spectra object (with spectra ordered according to the provided dataStorage parameter).
- filterEmptySpectra: removes empty spectra (i.e. spectra without peaks). Returns the filtered Spectra object (with spectra in their original order).
- filterFourierTransformArtefacts: remove (Orbitrap) fast fourier artefact peaks from spectra (see examples below). The function iterates through all intensity ordered peaks in a spectrum and removes all peaks with an m/z within +/- halfWindowSize of the current peak if their intensity is lower than threshold times the current peak's intensity. Additional parameters keepIsotopes, maxCharge and isotopeTolerance allow to avoid removing of

potential [13]C isotope peaks (maxCharge being the maximum charge that should be considered and isotopeTolerance the absolute acceptable tolerance for matching their m/z). See filterFourierTransformArtefacts() for details and background.

- filterIsolationWindow: retains spectra that contain mz in their isolation window m/z range (i.e. with an isolationWindowLowerMz <= mz and isolationWindowUpperMz >= mz. Returns the filtered Spectra object (with spectra in their original order).
- filterMsLevel: filters object by MS level keeping only spectra matching the MS level specified with argument msLevel. Returns the filtered Spectra (with spectra in their original order).
- filterMzRange: filters the object keeping or removing peaks in each spectrum that are within the provided m/z range. Whether peaks are retained or removed can be configured with parameter keep (default keep = TRUE).
- filterMzValues: filters the object keeping **all** peaks in each spectrum that match the provided m/z value(s) (for keep = TRUE, the default) or removing **all** of them (for keep = FALSE). The m/z matching considers also the absolute tolerance and m/z-relative ppm values. tolerance and ppm have to be of length 1.
- filterPolarity: filters the object keeping only spectra matching the provided polarity. Returns the filtered Spectra (with spectra in their original order).
- filterPrecursorMzRange (previously filterPrecursorMz which is now deprecated): retains spectra with a precursor m/z within the provided m/z range. See examples for details on selecting spectra with a precursor m/z for a target m/z accepting a small difference in *ppm*.
- filterPrecursorMzValues: retains spectra with precursor m/z matching any of the provided m/z values (given ppm and tolerance). Spectra with missing precursor m/z value (e.g. MS1 spectra) are dropped.
- filterPrecursorCharge: retains spectra with the defined precursor charge(s).
- filterPrecursorScan: retains parent (e.g. MS1) and children scans (e.g. MS2) of acquisition number acquisitionNum. Returns the filtered Spectra (with spectra in their original order). Parameter f allows to define which spectra belong to the same sample or original data file (defaults to f = dataOrigin(object)).
- filterRt: retains spectra of MS level msLevel with retention times (in seconds) within (>=) rt[1] and (<=) rt[2]. Returns the filtered Spectra (with spectra in their original order).
- reset: restores the data to its original state (as much as possible): removes any processing steps from the lazy processing queue and calls reset on the backend which, depending on the backend, can also undo e.g. data filtering operations. Note that a reset call after applyProcessing will not have any effect. See examples below for more information.
- selectSpectraVariables: reduces the information within the object to the selected spectra variables: all data for variables not specified will be dropped. For mandatory columns (i.e., those listed by coreSpectraVariables(), such as *msLevel*, *rtime* ...) only the values will be dropped but not the variable itself. Additional (or user defined) spectra variables will be completely removed. Returns the filtered Spectra.
- split: splits the Spectra object based on parameter f into a list of Spectra objects.
- joinSpectraData: Individual spectra variables can be directly added with the $<- or [[<- syntax. The joinSpectraData() function allows to merge a DataFrame to the existing spectra data. This function diverges from the merge() method in two main ways:

- The by.x and by.y column names must be of length 1.
- If variable names are shared in x and y, the spectra variables of x are not modified. It's only the y variables that are appended the suffix defined in suffix.y. This is to avoid modifying any core spectra variables that would lead to an invalid object.
- Duplicated Spectra keys (i.e. x[[by.x]]) are not allowed. Duplicated keys in the DataFrame (i.e y[[by.y]]) throw a warning and only the last occurrence is kept. These should be explored and ideally be removed using for QFeatures::reduceDataFrame(), PMS::reducePSMs() or similar functions.

Several Spectra objects can be concatenated into a single object with the c or the concatenateSpectra function. Concatenation will fail if the processing queue of any of the Spectra objects is not empty or if different backends are used in the Spectra objects. The spectra variables of the resulting Spectra object is the union of the spectra variables of the individual Spectra objects.

Data manipulation and analysis methods

Many data manipulation operations, such as those listed in this section, are not applied immediately to the spectra, but added to a *lazy processing/manipulation queue*. Operations stored in this queue are applied on-the-fly to spectra data each time it is accessed. This lazy execution guarantees the same functionality for Spectra objects with any backend, i.e. backends supporting to save changes to spectrum data (MsBackendMemory, MsBackendDataFrame() or MsBackendHdf5Peaks()) as well as read-only backends (such as the MsBackendMzR()). Note that for the former it is possible to apply the processing queue and write the modified peak data back to the data storage with the applyProcessing function.

- addProcessing: adds an arbitrary function that should be applied to the peaks matrix of every spectrum in object. The function (can be passed with parameter FUN) is expected to take a peaks matrix as input and to return a peaks matrix. A peaks matrix is a numeric matrix with two columns, the first containing the m/z values of the peaks and the second the corresponding intensities. The function has to have ... in its definition. Additional arguments can be passed with .... With parameter spectraVariables it is possible to define additional spectra variables from object that should be passed to the function FUN. These will be passed by their name (e.g. specifying spectraVariables = "precursorMz" will pass the spectra's precursor m/z as a parameter named precursorMz to the function. The only exception is the spectra's MS level, these will be passed to the function as a parameter called spectrumMsLevel (i.e. with spectraVariables = "msLevel" the MS levels of each spectrum will be submitted to the function as a parameter called spectrumMsLevel). Examples are provided in the package vignette.
- applyProcessing: for Spectra objects that use a **writeable** backend only: apply all steps from the lazy processing queue to the peak data and write it back to the data storage. Parameter f allows to specify how object should be split for parallel processing. This should either be equal to the dataStorage, or f = rep(1, length(object)) to disable parallel processing alltogether. Other partitionings might result in errors (especially if a MsBackendHdf5Peaks backend is used).
- bin: aggregates individual spectra into discrete (m/z) bins. Binning is performed only on spectra of the specified MS level(s) (parameter msLevel, by default all MS levels of x). The bins can be defined with parameter breaks which by default are equally sized bins, with size being defined by parameter binSize, from the minimal to the maximal m/z of all spectra (of MS level msLevel) within x. The same bins are used for all spectra in x. All intensity values

for peaks falling into the same bin are aggregated using the function provided with parameter FUN (defaults to FUN = sum, i.e. all intensities are summed up). Note that the binning operation is applied to the peak data on-the-fly upon data access and it is possible to *revert* the operation with the reset function (see description of reset above).

- combineSpectra: combine sets of spectra into a single spectrum per set. For each spectrum group (set), spectra variables from the first spectrum are used and the peak matrices are combined using the function specified with FUN, which defaults to combinePeaks(). Please refer to the combinePeaks() help page for details and options of the actual combination of peaks across the sets of spectra and to the package vignette for examples and alternative ways to aggregate spectra. The sets of spectra can be specified with parameter f. In addition it is possible to define, with parameter p if and how to split the input data for parallel processing. This defaults to p = x$dataStorage and hence a per-file parallel processing is applied for Spectra with file-based backends (such as the MsBackendMzR()). Prior combination of the spectra all processings queued in the lazy evaluation queue are applied. Be aware that calling combineSpectra on a Spectra object with certain backends that allow modifications might **overwrite** the original data. This does not happen with a MsBackendMemory or MsBackendDataFrame backend, but with a MsBackendHdf5Peaks backend the m/z and intensity values in the original hdf5 file(s) will be overwritten. The function returns a Spectra of length equal to the unique levels of f.
- compareSpectra: compare each spectrum in x with each spectrum in y using the function provided with FUN (defaults to ndotproduct()). If y is missing, each spectrum in x is compared with each other spectrum in x. The matching/mapping of peaks between the compared spectra is done with the MAPFUN function. The default joinPeaks() matches peaks of both spectra and allows to keep all peaks from the first spectrum (type = "left"), from the second (type = "right"), from both (type = "outer") and to keep only matching peaks (type = "inner"); see joinPeaks() for more information and examples). The MAPFUN function should have parameters x, y, xPrecursorMz and yPrecursorMz as these values are passed to the function. In addition to joinPeaks() also joinPeaksGnps() is supported for GNPS-like similarity score calculations. Note that joinPeaksGnps should only be used in combination with FUN = MsCoreUtils::gnps (see joinPeaksGnps() for more information and details). FUN is supposed to be a function to compare intensities of (matched) peaks of the two spectra that are compared. The function needs to take two matrices with columns "mz" and "intensity" as input and is supposed to return a single numeric as result. In addition to the two peak matrices the spectra's precursor m/z values are passed to the function as parameters xPrecursorMz (precursor m/z of the x peak matrix) and yPrecursorMz (precursor m/z of the y peak matrix). Additional parameters to functions FUN and MAPFUN can be passed with .... The function returns a matrix with the results of FUN for each comparison, number of rows equal to length(x) and number of columns equal length(y) (i.e. element in row 2 and column 3 is the result from the comparison of x[2] with y[3]). If SIMPLIFY = TRUE the matrix is *simplified* to a numeric if length of x or y is one.
- deisotopeSpectra: *deisotope* each spectrum keeping only the monoisotopic peak for groups of isotopologues. Isotopologues are estimated using the isotopologues() function from the *MetaboCoreUtils* package. Note that the default parameters for isotope prediction/detection have been determined using data from the Human Metabolome Database (HMDB) and isotopes for elements other than CHNOPS might not be detected. See parameter substDefinition in the documentation of isotopologues() for more information.
- estimatePrecursorIntensity: define the precursor intensities for MS2 spectra using the intensity of the matching MS1 peak from the closest MS1 spectrum (i.e. the last MS1 spectrum

measured before the respective MS2 spectrum). With `method = "interpolation"` it is also possible to calculate the precursor intensity based on an interpolation of intensity values (and retention times) of the matching MS1 peaks from the previous and next MS1 spectrum. See below for an example.

- `filterIntensity`: filters each spectrum keeping only peaks with intensities that are within the provided range or match the criteria of the provided function. For the former, parameter `intensity` has to be a `numeric` defining the intensity range, for the latter a `function` that takes the intensity values of the spectrum and returns a `logical` whether the peak should be retained or not (see examples below for details) - additional parameters to the function can be passed with `...`. To remove only peaks with intensities below a certain threshold, say 100, use `intensity = c(100, Inf)`. Note: also a single value can be passed with the `intensity` parameter in which case an upper limit of `Inf` is used. Note that this function removes also peaks with missing intensities (i.e. an intensity of `NA`). Parameter `msLevel.` allows to restrict the filtering to spectra of the specified MS level(s).
- `neutralLoss`: calculate neutral loss spectra for fragment spectra. See `neutralLoss()` for detailed documentation.
- `processingLog`: returns a `character` vector with the processing log messages.
- `reduceSpectra`: for groups of peaks within highly similar m/z values within each spectrum (given `ppm` and `tolerance`), this function keeps only the peak with the highest intensity removing all other peaks hence *reducing* each spectrum to the highest intensity peaks per *peak group*. Peak groups are defined using the `group()` function from the *MsCoreUtils* package.
- `spectrapply`: apply a given function to each individual spectrum or sets of a `Spectra` object. By default, the `Spectra` is split into individual spectra (i.e. `Spectra` of length 1) and the function `FUN` is applied to each of them. An alternative splitting can be defined with parameter `f`. Parameters for `FUN` can be passed using `...`. The returned result and its order depend on the function `FUN` and how `object` is split (hence on `f`, if provided). Parallel processing is supported and can be configured with parameter `BPPARAM`, is however only suggested for computational intense `FUN`. As an alternative to the (eventual parallel) processing of the full `Spectra`, `spectrapply` supports also a chunk-wise processing. For this, parameter `chunkSize` needs to be specified. `object` is then split into chunks of size `chunkSize` which are then (stepwise) processed by `FUN`. This guarantees a lower memory demand (especially for on-disk backends) since only the data for one chunk needs to be loaded into memory in each iteration. Note that by specifying `chunkSize`, parameters `f` and `BPPARAM` will be ignored. See also `chunkapply()` or examples below for details on chunk-wise processing.
- `smooth`: smooth individual spectra using a moving window-based approach (window size = `2 * halfWindowSize`). Currently, the Moving-Average- (`method = "MovingAverage"`), Weighted-Moving-Average- (`method = "WeightedMovingAverage"`), weights depending on the distance of the center and calculated `1/2^(-halfWindowSize:halfWindowSize)`) and Savitzky-Golay-Smoothing (`method = "SavitzkyGolay"`) are supported. For details how to choose the correct `halfWindowSize` please see `MsCoreUtils::smooth()`.
- `pickPeaks`: picks peaks on individual spectra using a moving window-based approach (window size = `2 * halfWindowSize`). For noisy spectra there are currently two different noise estimators available, the *M*edian *A*bsolute *D*eviation (`method = "MAD"`) and Friedman's Super Smoother (`method = "SuperSmoother"`), as implemented in the `MsCoreUtils::noise()`. The method supports also to optionally *refine* the m/z value of the identified centroids by considering data points that belong (most likely) to the same mass peak. Therefore the m/z value is calculated as an intensity weighted average of the m/z values within the peak region. The

peak region is defined as the m/z values (and their respective intensities) of the `2 * k` closest signals to the centroid or the closest valleys (`descending = TRUE`) in the `2 * k` region. For the latter the `k` has to be chosen general larger. See `MsCoreUtils::refineCentroids()` for details. If the ratio of the signal to the highest intensity of the peak is below `threshold` it will be ignored for the weighted average.

- `replaceIntensitiesBelow`: replaces intensities below a specified threshold with the provided `value`. Parameter `threshold` can be either a single numeric value or a function which is applied to all non-`NA` intensities of each spectrum to determine a threshold value for each spectrum. The default is `threshold = min` which replaces all values which are `<=` the minimum intensity in a spectrum with `value` (the default for `value` is `0`). Note that the function specified with `threshold` is expected to have a parameter `na.rm` since `na.rm = TRUE` will be passed to the function. If the spectrum is in profile mode, ranges of successive non-0 peaks `<= threshold` are set to 0. Parameter `msLevel.` allows to apply this to only spectra of certain MS level(s).

Parallel processing

Some `Spectra` functions have build-in parallel processing that can be configured by passing the parallel processing setup with the `BPPARAM` function argument (which defaults to `BPPARAM = bpparam()`, thus uses the default set up). Most functions have an additional parameter `f` that allows to define how `Spectra` will be split to perform parallel processing. This parameter `f` defaults to `f = dataStorage(object)` and hence parallel processing is performed *by file* (if a file-based, on-disk backend such as `MsBackendMzR` is used). Some `MsBackend` classes might however not support parallel processing. The `backendBpparam` function allows to evaluate wheter a `Spectra` (respectively its `MsBackend`) supports a certain parallel processing setup. Calling `backendBpparam(sps, BPPARAM = MulticoreParam(3))` on a `Spectra` object `sps` would return `SerialParam()` in case the backend of the `Spectra` object does not support parallel processing. All functions listed below use this same function to eventually disable parallel processing to avoid failure of a function call.

The functions with build-in parallel processing capabilities are:

- `applyProcessing`.
- `combineSpectra`.
- `containsMz` (does not provide a parameter `f`, but performs parallel processing separate for `dataStorage`).
- `containsNeutralLoss` (same as `containsMz`).
- `estimatePrecursorIntensity`.
- `setBackend`.
- `Spectra` (that passes the `BPPARAM` to the `backendInitialize` of the used `MsBackend`).
- `spectrapply`.

Author(s)

Nir Shahaf, Johannes Rainer

Sebastian Gibb, Johannes Rainer, Laurent Gatto

Examples

```
## Create a Spectra providing a `DataFrame` containing the spectrum data.

spd <- DataFrame(msLevel = c(1L, 2L), rtime = c(1.1, 1.2))
spd$mz <- list(c(100, 103.2, 104.3, 106.5), c(45.6, 120.4, 190.2))
spd$intensity <- list(c(200, 400, 34.2, 17), c(12.3, 15.2, 6.8))

data <- Spectra(spd)
data

## Get the number of spectra
length(data)

## Get the number of peaks per spectrum
lengths(data)

## Create a Spectra from mzML files and use the `MsBackendMzR` on-disk
## backend.
sciex_file <- dir(system.file("sciex", package = "msdata"),
    full.names = TRUE)
sciex <- Spectra(sciex_file, backend = MsBackendMzR())
sciex

## The MS data is on disk and will be read into memory on-demand. We can
## however change the backend to a MsBackendMemory backend which will
## keep all of the data in memory.
sciex_im <- setBackend(sciex, MsBackendMemory())
sciex_im

## The `MsBackendMemory()` supports the `setBackend` method:
supportsSetBackend(MsBackendMemory())

## Thus, it is possible to change to that backend with `setBackend`. Most
## read-only backends however don't support that, such as the
## `MsBackendMzR` and `setBackend` would fail to change to that backend.
supportsSetBackend(MsBackendMzR())

## The on-disk object `sciex` is light-weight, because it does not keep the
## MS peak data in memory. The `sciex_im` object in contrast keeps all the
## data in memory and its size is thus much larger.
object.size(sciex)
object.size(sciex_im)

## The spectra variable `dataStorage` returns for each spectrum the location
## where the data is stored. For in-memory objects:
head(dataStorage(sciex_im))

## While objects that use an on-disk backend will list the files where the
## data is stored.
head(dataStorage(sciex))

## The spectra variable `dataOrigin` returns for each spectrum the *origin*
```

```
## of the data. If the data is read from e.g. mzML files, this will be the
## original mzML file name:
head(dataOrigin(sciex))
head(dataOrigin(sciex_im))

## ---- ACCESSING AND ADDING DATA ----

## Get the MS level for each spectrum.
msLevel(data)

## Alternatively, we could also use $ to access a specific spectra variable.
## This could also be used to add additional spectra variables to the
## object (see further below).
data$msLevel

## Get the intensity and m/z values.
intensity(data)
mz(data)

## Determine whether one of the spectra has a specific m/z value
containsMz(data, mz = 120.4)

## Accessing spectra variables works for all backends:
intensity(sciex)
intensity(sciex_im)

## Get the m/z for the first spectrum.
mz(data)[[1]]

## Get the peak data (m/z and intensity values).
pks <- peaksData(data)
pks
pks[[1]]
pks[[2]]

## Note that we could get the same resulb by coercing the `Spectra` to
## a `list` or `SimpleList`:
as(data, "list")
as(data, "SimpleList")

## List all available spectra variables (i.e. spectrum data and metadata).
spectraVariables(data)

## For all *core* spectrum variables accessor functions are available. These
## return NA if the variable was not set.
centroided(data)
dataStorage(data)
rtime(data)
precursorMz(data)

## The core spectra variables are:
coreSpectraVariables()
```

```
## Add an additional metadata column.
data$spectrum_id <- c("sp_1", "sp_2")

## List spectra variables, "spectrum_id" is now also listed
spectraVariables(data)

## Get the values for the new spectra variable
data$spectrum_id

## Extract specific spectra variables.
spectraData(data, columns = c("spectrum_id", "msLevel"))

## Drop spectra variable data and/or columns.
res <- selectSpectraVariables(data, c("mz", "intensity"))

## This removed the additional columns "spectrum_id" and deleted all values
## for all spectra variables, except "mz" and "intensity".
spectraData(res)

## Compared to the data before selectSpectraVariables.
spectraData(data)


## ---- SUBSETTING, FILTERING AND COMBINING

## Subset to all MS2 spectra.
data[msLevel(data) == 2]

## Same with the filterMsLevel function
filterMsLevel(data, 2)

## Below we combine the `data` and `sciex_im` objects into a single one.
data_comb <- c(data, sciex_im)

## The combined Spectra contains a union of all spectra variables:
head(data_comb$spectrum_id)
head(data_comb$rtime)
head(data_comb$dataStorage)
head(data_comb$dataOrigin)

## Filter a Spectra for a target precursor m/z with a tolerance of 10ppm
spd$precursorMz <- c(323.4, 543.2302)
data_filt <- Spectra(spd)
filterPrecursorMzRange(data_filt, mz = 543.23 + ppm(c(-543.23, 543.23), 10))

## Filter a Spectra keeping only peaks matching certain m/z values
sps_sub <- filterMzValues(data, mz = c(103, 104), tolerance = 0.3)
mz(sps_sub)

## This function can also be used to remove specific peaks from a spectrum
## by setting `keep = FALSE`.
sps_sub <- filterMzValues(data, mz = c(103, 104),
    tolerance = 0.3, keep = FALSE)
```

```
mz(sps_sub)

## Note that `filterMzValues` keeps or removes all peaks with a matching
## m/z given the provided `ppm` and `tolerance` parameters.

## Filter a Spectra keeping only peaks within a m/z range
sps_sub <- filterMzRange(data, mz = c(100, 300))
mz(sps_sub)

## Remove empty spectra variables
sciex_noNA <- dropNaSpectraVariables(sciex)

## Available spectra variables before and after dropNaSpectraVariables
spectraVariables(sciex)
spectraVariables(sciex_noNA)


## Adding new spectra variables
sciex1 <- filterDataOrigin(sciex, dataOrigin(sciex)[1])
spv <- DataFrame(spectrumId = sciex1$spectrumId[3:12], ## used for merging
                 var1 = rnorm(10),
                 var2 = sample(letters, 10))
spv

sciex2 <- joinSpectraData(sciex1, spv, by.y = "spectrumId")

spectraVariables(sciex2)
spectraData(sciex2)[1:13, c("spectrumId", "var1", "var2")]

## Removing fourier transform artefacts seen in Orbitra data.

## Loading an Orbitrap spectrum with artefacts.
data(fft_spectrum)
plotSpectra(fft_spectrum, xlim = c(264.5, 265.5))
plotSpectra(fft_spectrum, xlim = c(264.5, 265.5), ylim = c(0, 5e6))

fft_spectrum <- filterFourierTransformArtefacts(fft_spectrum)
fft_spectrum
plotSpectra(fft_spectrum, xlim = c(264.5, 265.5), ylim = c(0, 5e6))

## Using a few examples peaks in your data you can optimize the parameters
fft_spectrum_filtered <- filterFourierTransformArtefacts(fft_spectrum,
                                                         halfWindowSize = 0.2,
                                                         threshold = 0.005,
                                                         keepIsotopes = TRUE,
                                                         maxCharge = 5,
                                                         isotopeTolerance = 0.005
                                                         )

fft_spectrum_filtered
length(mz(fft_spectrum_filtered)[[1]])
plotSpectra(fft_spectrum_filtered, xlim = c(264.5, 265.5), ylim = c(0, 5e6))
```

```
## ---- DATA MANIPULATIONS AND OTHER OPERATIONS ----

## Set the data to be centroided
centroided(data) <- TRUE

## Replace peak intensities below 40 with 3.
res <- replaceIntensitiesBelow(data, threshold = 40, value = 3)
res

## Get the intensities of the first and second spectrum.
intensity(res)[[1]]
intensity(res)[[2]]

## Remove all peaks with an intensity below 40.
res <- filterIntensity(res, intensity = c(40, Inf))

## Get the intensities of the first and second spectrum.
intensity(res)[[1]]
intensity(res)[[2]]

## Lengths of spectra is now different
lengths(mz(res))
lengths(mz(data))

## In addition it is possible to pass a function to `filterIntensity`: in
## the example below we want to keep only peaks that have an intensity which
## is larger than one third of the maximal peak intensity in that spectrum.
keep_peaks <- function(x, prop = 3) {
    x > max(x, na.rm = TRUE) / prop
}
res2 <- filterIntensity(data, intensity = keep_peaks)
intensity(res2)[[1L]]
intensity(data)[[1L]]

## We can also change the proportion by simply passing the `prop` parameter
## to the function. To keep only peaks that have an intensity which is
## larger than half of the maximum intensity:
res2 <- filterIntensity(data, intensity = keep_peaks, prop = 2)
intensity(res2)[[1L]]
intensity(data)[[1L]]

## Since data manipulation operations are by default not directly applied to
## the data but only added to the internal lazy evaluation queue, it is also
## possible to remove these data manipulations with the `reset` function:
res_rest <- reset(res)
res_rest
lengths(mz(res_rest))
lengths(mz(res))
lengths(mz(data))

## `reset` after a `applyProcessing` can not restore the data, because the
## data in the backend was changed. Similarly, `reset` after any filter
## operations can not restore data for a `Spectra` with a
```

```
## `MsBackendMemory` or `MsBackendDataFrame`.
res_2 <- applyProcessing(res)
res_rest <- reset(res_2)
lengths(mz(res))
lengths(mz(res_rest))


## Compare spectra: comparing spectra 2 and 3 against spectra 10:20 using
## the normalized dotproduct method.
res <- compareSpectra(sciex_im[2:3], sciex_im[10:20])
## first row contains comparisons of spectrum 2 with spectra 10 to 20 and
## the second row comparisons of spectrum 3 with spectra 10 to 20
res

## To use a simple Pearson correlation instead we can define a function
## that takes the two peak matrices and calculates the correlation for
## their second columns (containing the intensity values).
correlateSpectra <- function(x, y, use = "pairwise.complete.obs", ...) {
    cor(x[, 2], y[, 2], use = use)
}
res <- compareSpectra(sciex_im[2:3], sciex_im[10:20],
    FUN = correlateSpectra)
res

## Use compareSpectra to determine the number of common (matching) peaks
## with a ppm of 10:
## type = "inner" uses a *inner join* to match peaks, i.e. keeps only
## peaks that can be mapped betwen both spectra. The provided FUN returns
## simply the number of matching peaks.
compareSpectra(sciex_im[2:3], sciex_im[10:20], ppm = 10, type = "inner",
    FUN = function(x, y, ...) nrow(x))

## Apply an arbitrary function to each spectrum in a Spectra.
## In the example below we calculate the mean intensity for each spectrum
## in a subset of the sciex_im data. Note that we can access all variables
## of each individual spectrum either with the `$` operator or the
## corresponding method.
res <- spectrapply(sciex_im[1:20], FUN = function(x) mean(x$intensity[[1]]))
head(res)

## It is however important to note that dedicated methods to access the
## data (such as `intensity`) are much more efficient than using `lapply`:
res <- lapply(intensity(sciex_im[1:20]), mean)
head(res)

## As an alternative, applying a function `FUN` to a `Spectra` can be
## performed *chunk-wise*. The advantage of this is, that only the data for
## one chunk at a time needs to be loaded into memory reducing the memory
## demand. This type of processing can be performed by specifying the size
## of the chunks (i.e. number of spectra per chunk) with the `chunkSize`
## parameter
spectrapply(sciex_im[1:20], lengths, chunkSize = 5L)
```

```
## Calculating the precursor intensity for MS2 spectra:
##
## Some MS instrument manufacturer don't report the precursor intensities
## for MS2 spectra. The `estimatePrecursorIntensity` function can be used
## in these cases to calculate the precursor intensity on MS1 data. Below
## we load an mzML file from a vendor providing precursor intensities and
## compare the estimated and reported precursor intensities.
tmt <- Spectra(msdata::proteomics(full.names = TRUE)[5],
    backend = MsBackendMzR())
pmi <- estimatePrecursorIntensity(tmt)
plot(pmi, precursorIntensity(tmt))

## We can also replace the original precursor intensity values with the
## newly calculated ones
tmt$precursorIntensity <- pmi

## ---- DATA EXPORT ----

## Some `MsBackend` classes provide an `export` method to export the data to
## the file format supported by the backend. The `MsBackendMzR` for example
## allows to export MS data to mzML or mzXML file(s), the `MsBackendMgf`
## (defined in the MsBackendMgf R package) would allow to export the data
## in mgf file format. Below we export the MS data in `data`. We
## call the `export` method on this object, specify the backend that should
## be used to export the data (and which also defines the output format) and
## provide a file name.
fl <- tempfile()
export(data, MsBackendMzR(), file = fl)

## This exported our data in mzML format. Below we read the first 6 lines
## from that file.
readLines(fl, n = 6)

## If only a single file name is provided, all spectra are exported to that
## file. To export data with the `MsBackendMzR` backend to different files, a
## file name for each individual spectrum has to be provided.
## Below we export each spectrum to its own file.
fls <- c(tempfile(), tempfile())
export(data, MsBackendMzR(), file = fls)

## Reading the data from the first file
res <- Spectra(backendInitialize(MsBackendMzR(), fls[1]))

mz(res)
mz(data)
```

---

chunkapply *Apply a function stepwise to chunks of data*

Description

chunkapply splits x into chunks and applies the function FUN stepwise to each of these chunks. Depending on the object it is called, this function might reduce memory demand considerably, if for example only the full data for a single chunk needs to be loaded into memory at a time (e.g., for Spectra objects with on-disk or similar backends).

Usage

```
chunkapply(x, FUN, ..., chunkSize = 1000L, chunks = factor())
```

Arguments

x	object to which FUN should be applied. Can be any object that supports split.
FUN	the function to apply to x.
...	additional parameters to FUN.
chunkSize	integer(1) defining the size of each chunk into which x should be splitted.
chunks	optional factor or length equal to length(x) defining the chunks into which x should be splitted.

Value

Depending on FUN, but in most cases a vector/result object of length equal to length(x).

Author(s)

Johannes Rainer

Examples

```
## Apply a function (`sqrt`) to each element in `x`, processed in chunks of
## size 200.
x <- rnorm(n = 1000, mean = 500)
res <- chunkapply(x, sqrt, chunkSize = 200)
length(res)
head(res)

## For such a calculation the vectorized `sqrt` would however be recommended
system.time(sqrt(x))
system.time(chunkapply(x, sqrt, chunkSize = 200))

## Simple example splitting a numeric vector into chunks of 200 and
## aggregating the values within the chunk using the `mean`. Due to the
## `unsplit` the result has the same length than the input with the mean
## value repeated.
x <- 1:1000
res <- chunkapply(x, mean, chunkSize = 200)
length(res)
head(res)
```

combinePeaks *Combine peaks with similar m/z across spectra*

Description

combinePeaks aggregates provided peak matrices into a single peak matrix. Peaks are grouped by their m/z values with the group() function from the MsCoreUtils package. In brief, all peaks in all provided spectra are first ordered by their m/z and consecutively grouped into one group if the (pairwise) difference between them is smaller than specified with parameter tolerance and ppm (see group() for grouping details and examples).

The m/z and intensity values for the resulting peak matrix are calculated using the mzFun and intensityFun on the grouped m/z and intensity values.

Note that only the grouped m/z and intensity values are used in the aggregation functions (mzFun and intensityFun) but not the number of spectra.

The function supports also different strategies for peak combinations which can be specified with the peaks parameter:

- peaks = "union" (default): report all peaks from all input spectra.
- peaks = "intersect": keep only peaks in the resulting peak matrix that are present in >= minProp proportion of input spectra. This would generate a *consensus* or *representative* spectra from a set of e.g. fragment spectra measured from the same precursor ion.

As a special case it is possible to report only peaks in the resulting matrix from peak groups that contain a peak from one of the input spectra, which can be specified with parameter main. Thus, if e.g. main = 2 is specified, only (grouped) peaks that have a peak in the second input matrix are returned.

Setting timeDomain to TRUE causes grouping to be performed on the square root of the m/z values (assuming a TOF instrument was used to create the data).

Usage

```
combinePeaks(
  x,
  intensityFun = base::mean,
  mzFun = base::mean,
  weighted = FALSE,
  tolerance = 0,
  ppm = 0,
  timeDomain = FALSE,
  peaks = c("union", "intersect"),
  main = integer(),
  minProp = 0.5,
  ...
)
```

Arguments

x	list of peak matrices.
intensityFun	function to be used to combine intensity values for matching peaks. By default the mean intensity value is returned.
mzFun	function to be used to combine m/z values for matching peaks. By default the mean m/z value is returned.
weighted	logical(1) defining whether m/z values for matching peaks should be calculated by an intensity-weighted average of the individuak m/z values. This overrides parameter mzFun.
tolerance	numeric(1) defining the (absolute) maximal accepted difference between mass peaks to group them into the same final peak.
ppm	numeric(1) defining the m/z-relative maximal accepted difference between mass peaks (expressed in parts-per-million) to group them into the same final peak.
timeDomain	logical(1) whether grouping of mass peaks is performed on the m/z values (timeDomain = FALSE) or on sqrt(mz) (timeDomain = TRUE).
peaks	character(1) specifying how peaks should be combined. Can be either "peaks = "union" (default) or peaks = "intersect". See function description for details.
main	optional integer(1) to force the resulting peak list to contain only peaks that are present in the specified input spectrum. See description for details.
minProp	numeric(1) for 'peaks = "intersect": the minimal required proportion of input spectra (peak matrices) a mass peak has to be present to be included in the consensus peak matrix.
...	additional parameters to the mzFun and intensityFun functions.

Details

For general merging of spectra, the tolerance and/or ppm should be manually specified based on the precision of the MS instrument. Peaks from spectra with a difference in their m/z being smaller than tolerance or smaller than ppm of their m/z are grouped into the same final peak.

Some details for the combination of consecutive spectra of an LC-MS run:

The m/z values of the same ion in consecutive scans (spectra) of a LC-MS run will not be identical. Assuming that this random variation is much smaller than the resolution of the MS instrument (i.e. the difference between m/z values within each single spectrum), m/z value groups are defined across the spectra and those containing m/z values of the main spectrum are retained. Intensities and m/z values falling within each of these m/z groups are aggregated using the intensityFun and mzFun, respectively. It is highly likely that all QTOF profile data is collected with a timing circuit that collects data points with regular intervals of time that are then later converted into m/z values based on the relationship t = k * sqrt(m/z). The m/z scale is thus non-linear and the m/z scattering (which is in fact caused by small variations in the time circuit) will thus be different in the lower and upper m/z scale. m/z-intensity pairs from consecutive scans to be combined are therefore defined by default on the square root of the m/z values. With timeDomain = FALSE, the actual m/z values will be used.

Value

Peaks `matrix` with m/z and intensity values representing the aggregated values across the provided peak matrices.

Author(s)

Johannes Rainer

Examples

```
set.seed(123)
mzs <- seq(1, 20, 0.1)
ints1 <- abs(rnorm(length(mzs), 10))
ints1[11:20] <- c(15, 30, 90, 200, 500, 300, 100, 70, 40, 20) # add peak
ints2 <- abs(rnorm(length(mzs), 10))
ints2[11:20] <- c(15, 30, 60, 120, 300, 200, 90, 60, 30, 23)
ints3 <- abs(rnorm(length(mzs), 10))
ints3[11:20] <- c(13, 20, 50, 100, 200, 100, 80, 40, 30, 20)

## Create the peaks matrices
p1 <- cbind(mz = mzs + rnorm(length(mzs), sd = 0.01),
    intensity = ints1)
p2 <- cbind(mz = mzs + rnorm(length(mzs), sd = 0.01),
    intensity = ints2)
p3 <- cbind(mz = mzs + rnorm(length(mzs), sd = 0.009),
    intensity = ints3)

## Combine the spectra. With `tolerance = 0` and `ppm = 0` only peaks with
## **identical** m/z are combined. The result will be a single spectrum
## containing the *union* of mass peaks from the individual input spectra.
p <- combinePeaks(list(p1, p2, p3))

## Plot the spectra before and after combining
par(mfrow = c(2, 1), mar = c(4.3, 4, 1, 1))
plot(p1[, 1], p1[, 2], xlim = range(mzs[5:25]), type = "h", col = "red")
points(p2[, 1], p2[, 2], type = "h", col = "green")
points(p3[, 1], p3[, 2], type = "h", col = "blue")

plot(p[, 1], p[, 2], xlim = range(mzs[5:25]), type = "h",
    col = "black")
## The peaks were not merged, because their m/z differs too much.

## Combine spectra with `tolerance = 0.05`. This will merge all triplets.
p <- combinePeaks(list(p1, p2, p3), tolerance = 0.05)

## Plot the spectra before and after combining
par(mfrow = c(2, 1), mar = c(4.3, 4, 1, 1))
plot(p1[, 1], p1[, 2], xlim = range(mzs[5:25]), type = "h", col = "red")
points(p2[, 1], p2[, 2], type = "h", col = "green")
points(p3[, 1], p3[, 2], type = "h", col = "blue")

plot(p[, 1], p[, 2], xlim = range(mzs[5:25]), type = "h",
```

```
    col = "black")

## With `intensityFun = max` the maximal intensity per peak is reported.
p <- combinePeaks(list(p1, p2, p3), tolerance = 0.05,
    intensityFun = max)

## Create *consensus*/representative spectrum from a set of spectra

p1 <- cbind(mz = c(12, 45, 64, 70), intensity = c(10, 20, 30, 40))
p2 <- cbind(mz = c(17, 45.1, 63.9, 70.2), intensity = c(11, 21, 31, 41))
p3 <- cbind(mz = c(12.1, 44.9, 63), intensity = c(12, 22, 32))

## No mass peaks identical thus consensus peaks are empty
combinePeaks(list(p1, p2, p3), peaks = "intersect")

## Reducing the minProp to 0.2. The consensus spectrum will contain all
## peaks
combinePeaks(list(p1, p2, p3), peaks = "intersect", minProp = 0.2)

## With a tolerance of 0.1 mass peaks can be matched across spectra
combinePeaks(list(p1, p2, p3), peaks = "intersect", tolerance = 0.1)

## Report the minimal m/z and intensity
combinePeaks(list(p1, p2, p3), peaks = "intersect", tolerance = 0.1,
    intensityFun = min, mzFun = min)
```

`countIdentifications` *Count the number of identifications per scan*

Description

The function takes a `Spectra` object containing identification results as input. It then counts the number of identifications each scan (or their descendants) has lead to - this is either 0 or 1 for MS2 scans, or, for MS1 scans, the number of MS2 scans originating from any MS1 peak that lead to an identification.

This function can be used to generate id-annotated total ion chromatograms, as can illustrated here.

Usage

```
countIdentifications(
  object,
  identification = "sequence",
  f = dataStorage(object),
  BPPARAM = bpparam()
)
```

Arguments

object	An instance of class Spectra() that contains identification data, as defined by the sequence argument.
identification	character(1) with the name of the spectra variable that defines whether a scan lead to an identification (typically containing the idenfified peptides sequence in proteomics). The absence of identification is encode by an NA. Default is "sequence".
f	A factor defining how to split object for parallelized processing. Default is dataOrigin(x), i.e. each raw data files is processed in parallel.
BPPARAM	Parallel setup configuration. See BiocParallel::bpparam() for details.

Details

The computed number of identifications is stored in a new spectra variables named "countIdentifications". If it already exists, the function throws a message and returns the object unchanged. To force the recomputation of the "countIdentifications" variable, users should either delete or rename it.

Value

An updated Spectra() object that now contains an integer spectra variable countIdentifications with the number of identification for each scan.

Author(s)

Laurent Gatto

Examples

```
spdf <- new("DFrame", rownames = NULL, nrows = 86L,
   listData = list(
       msLevel = c(1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,
                   2L, 2L, 2L, 2L, 2L, 2L, 2L, 1L, 2L, 2L, 2L, 2L,
                   2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,
                   2L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,
                   2L, 2L, 2L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,
                   2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 1L, 2L,
                   2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,
                   2L, 2L),
       acquisitionNum = 8975:9060,
       precScanNum = c(NA, 8956L, 8956L, 8956L, 8956L, 8956L, 8956L,
                       8956L, 8956L, 8956L, 8956L, 8956L, 8956L,
                       8956L, 8956L, 8956L, 8956L, 8956L, 8956L, NA,
                       8975L, 8975L, 8975L, 8975L, 8975L, 8975L,
                       8975L, 8975L, 8975L, 8975L, 8975L, 8975L,
                       8975L, 8975L, 8975L, 8975L, 8975L, NA, 8994L,
                       8994L, 8994L, 8994L, 8994L, 8994L, 8994L,
                       8994L, 8994L, 8994L, 8994L, 8994L, 8994L, NA,
                       9012L, 9012L, 9012L, 9012L, 9012L, 9012L,
                       9012L, 9012L, 9012L, 9012L, 9012L, 9012L,
                       9012L, 9012L, 9012L, 9012L, 9012L, 9012L, NA,
```

```
                        9026L, 9026L, 9026L, 9026L, 9026L, 9026L,
                        9026L, 9026L, 9026L, 9026L, 9026L, 9026L,
                        9026L, 9026L, 9026L),
        sequence = c(NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
                     NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
                     "LSEHATAPTR", NA, NA, NA, NA, NA, NA, NA,
                     "EGSDATGDGTK", NA, NA, "NEDEDSPNK", NA, NA, NA,
                     NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
                     NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
                     NA, NA, NA, NA, NA, NA, NA, NA, NA, "GLTLAQGGVK",
                     NA, NA, NA, NA, "STLPDADRER", NA, NA, NA, NA, NA,
                     NA, NA, NA)),
   elementType = "ANY", elementMetadata = NULL, metadata = list())

sp <- Spectra(spdf)

## We have in this data 5 MS1 and 81 MS2 scans
table(msLevel(sp))

## The acquisition number of the MS1 scans
acquisitionNum(filterMsLevel(sp, 1))

## And the number of MS2 scans with precursor ions selected
## from MS1 scans (those in the data and others)
table(precScanNum(sp))

## Count number of sequences/identifications per scan
sp <- countIdentifications(sp)

## MS2 scans either lead to an identification (5 instances) or none
## (76). Among the five MS1 scans in the experiment, 3 lead to MS2
## scans being matched to no peptides and two MS1 scans produced two
## and three PSMs respectively.
table(sp$countIdentifications, sp$msLevel)
```

`filterFourierTransformArtefacts` *Fast fourier transform artefact filter*

Description

The `filterFourierTransformArtefacts` function removes (Orbitrap) fast fourier artefact peaks from spectra. Such artefacts (also referred to as *rippples*) seem to be related to the *ringing* phenomenon and are frequently seen in Orbitrap data as small random mass peaks ~ 0.01 Da from a main peak with a very large intensity. See also here for more details and information. The data set `fft_spectrum` represents a `Spectra()` object with a single Orbitrap spectrum with such artefacts (see examples below).

See also `Spectra()` (section *Data subsetting, filtering and merging) for the definition of the function.

Details

The current implementation iterates through all intensity ordered peaks in a spectrum and removes all peaks with an m/z within +/- `halfWindowSize` of the current peak if their intensity is lower than `threshold` times the current peak's intensity. Additional parameters `keepIsotopes`, `maxCharge` and `isotopeTolerance` allow to avoid removing of potential `[13]C` isotope peaks (`maxCharge` being the maximum charge that should be considered and `isotopeTolerance` the absolute acceptable tolerance for matching their m/z).

Author(s)

Jan Stanstrup, Johannes Rainer

Examples

```
library(Spectra)
data(fft_spectrum)

plotSpectra(fft_spectrum)

## Focus on an artefact
plotSpectra(fft_spectrum, xlim = c(264.5, 265.5))
plotSpectra(fft_spectrum, xlim = c(264.5, 265.5), ylim = c(0, 5e6))

fft_spectrum <- filterFourierTransformArtefacts(fft_spectrum)
fft_spectrum
plotSpectra(fft_spectrum, xlim = c(264.5, 265.5), ylim = c(0, 5e6))

## R code to download/extract the data.

## Not run:
library(Spectra)
# get orbitrap data
download.file("https://www.ebi.ac.uk/metabolights/ws/studies/MTBLS469/download/4cc5d820-dc5d-4766-8112-7a05f74
data <- Spectra("AV_01_v2_male_arm1_juice.mzXML")
extracted_spectrum <- data[195]

## End(Not run)
```

`hidden_aliases` *Internal page for hidden aliases*

Description

For S4 methods that require a documentation entry but only clutter the index.

Usage

```
backendMerge(object, ...)

## S4 method for signature 'numeric'
bin(
  x,
  y,
  size = 1,
  breaks = seq(floor(min(y)), ceiling(max(y)), by = size),
  FUN = max,
  returnMids = TRUE,
  .check = TRUE
)

containsMz(object, ...)

containsNeutralLoss(object, ...)

dropNaSpectraVariables(object, ...)

export(object, ...)

filterMzRange(object, ...)

filterMzValues(object, ...)

filterPrecursorMzValues(object, ...)

filterPrecursorMzRange(object, ...)

isReadOnly(object, ...)

peaksData(object, ...)

peaksData(object) <- value

peaksVariables(object, ...)

pickPeaks(object, ...)

replaceIntensitiesBelow(object, threshold = min, ...)

reset(object, ...)

selectSpectraVariables(object, ...)

setBackend(object, backend, ...)
```

```
spectrapply(object, ...)

## S4 method for signature 'MsBackendDataFrame'
show(object)

## S4 method for signature 'MsBackendDataFrame'
backendMerge(object, ...)

## S4 method for signature 'MsBackendDataFrame'
acquisitionNum(object)

## S4 method for signature 'MsBackendDataFrame'
peaksData(object, columns = peaksVariables(object))

## S4 method for signature 'MsBackendDataFrame'
centroided(object)

## S4 replacement method for signature 'MsBackendDataFrame'
centroided(object) <- value

## S4 method for signature 'MsBackendDataFrame'
collisionEnergy(object)

## S4 replacement method for signature 'MsBackendDataFrame'
collisionEnergy(object) <- value

## S4 method for signature 'MsBackendDataFrame'
dataOrigin(object)

## S4 replacement method for signature 'MsBackendDataFrame'
dataOrigin(object) <- value

## S4 method for signature 'MsBackendDataFrame'
dataStorage(object)

## S4 replacement method for signature 'MsBackendDataFrame'
dataStorage(object) <- value

## S4 method for signature 'MsBackendDataFrame'
intensity(object)

## S4 replacement method for signature 'MsBackendDataFrame'
intensity(object) <- value

## S4 method for signature 'MsBackendDataFrame'
isEmpty(x)

## S4 method for signature 'MsBackendDataFrame'
```

```
isolationWindowLowerMz(object)

## S4 replacement method for signature 'MsBackendDataFrame'
isolationWindowLowerMz(object) <- value

## S4 method for signature 'MsBackendDataFrame'
isolationWindowTargetMz(object)

## S4 replacement method for signature 'MsBackendDataFrame'
isolationWindowTargetMz(object) <- value

## S4 method for signature 'MsBackendDataFrame'
isolationWindowUpperMz(object)

## S4 replacement method for signature 'MsBackendDataFrame'
isolationWindowUpperMz(object) <- value

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

## S4 method for signature 'MsBackendDataFrame'
lengths(x, use.names = FALSE)

## S4 method for signature 'MsBackendDataFrame'
msLevel(object, ...)

## S4 replacement method for signature 'MsBackendDataFrame'
msLevel(object) <- value

## S4 method for signature 'MsBackendDataFrame'
mz(object)

## S4 replacement method for signature 'MsBackendDataFrame'
mz(object) <- value

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

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

## S4 method for signature 'MsBackendDataFrame'
precScanNum(object)

## S4 method for signature 'MsBackendDataFrame'
precursorCharge(object)

## S4 method for signature 'MsBackendDataFrame'
```

```
precursorIntensity(object)

## S4 method for signature 'MsBackendDataFrame'
precursorMz(object)

## S4 replacement method for signature 'MsBackendDataFrame'
peaksData(object) <- value

## S4 method for signature 'MsBackendDataFrame'
rtime(object)

## S4 replacement method for signature 'MsBackendDataFrame'
rtime(object) <- value

## S4 method for signature 'MsBackendDataFrame'
scanIndex(object)

## S4 method for signature 'MsBackendDataFrame'
selectSpectraVariables(object, spectraVariables = spectraVariables(object))

## S4 method for signature 'MsBackendDataFrame'
smoothed(object)

## S4 replacement method for signature 'MsBackendDataFrame'
smoothed(object) <- value

## S4 method for signature 'MsBackendDataFrame'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'MsBackendDataFrame'
spectraData(object) <- value

## S4 method for signature 'MsBackendDataFrame'
spectraNames(object)

## S4 replacement method for signature 'MsBackendDataFrame'
spectraNames(object) <- value

## S4 method for signature 'MsBackendDataFrame'
spectraVariables(object)

## S4 method for signature 'MsBackendDataFrame'
tic(object, initial = TRUE)

## S4 method for signature 'MsBackendDataFrame'
x$name

## S4 replacement method for signature 'MsBackendDataFrame'
```

```
x$name <- value

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

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

## S4 method for signature 'MsBackendDataFrame'
filterAcquisitionNum(
  object,
  n = integer(),
  dataStorage = character(),
  dataOrigin = character()
)

## S4 method for signature 'MsBackendHdf5Peaks'
backendInitialize(
  object,
  files = character(),
  data = DataFrame(),
  hdf5path = character(),
  ...,
  BPPARAM = bpparam()
)

## S4 method for signature 'MsBackendHdf5Peaks'
show(object)

## S4 method for signature 'MsBackendHdf5Peaks'
peaksData(object, columns = peaksVariables(object))

## S4 method for signature 'MsBackendHdf5Peaks'
intensity(object)

## S4 replacement method for signature 'MsBackendHdf5Peaks'
intensity(object) <- value

## S4 method for signature 'MsBackendHdf5Peaks'
ionCount(object)

## S4 method for signature 'MsBackendHdf5Peaks'
isCentroided(object, ...)

## S4 method for signature 'MsBackendHdf5Peaks'
isEmpty(x)

## S4 method for signature 'MsBackendHdf5Peaks'
```

```
lengths(x, use.names = FALSE)

## S4 method for signature 'MsBackendHdf5Peaks'
mz(object)

## S4 replacement method for signature 'MsBackendHdf5Peaks'
mz(object) <- value

## S4 replacement method for signature 'MsBackendHdf5Peaks'
peaksData(object) <- value

## S4 method for signature 'MsBackendHdf5Peaks'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'MsBackendHdf5Peaks'
spectraData(object) <- value

## S4 replacement method for signature 'MsBackendHdf5Peaks'
x$name <- value

## S4 method for signature 'MsBackendHdf5Peaks'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'MsBackendHdf5Peaks'
backendMerge(object, ...)

## S4 method for signature 'MsBackendMemory'
show(object)

## S4 method for signature 'MsBackendMemory'
backendMerge(object, ...)

## S4 method for signature 'MsBackendMemory'
acquisitionNum(object)

## S4 method for signature 'MsBackendMemory'
centroided(object)

## S4 replacement method for signature 'MsBackendMemory'
centroided(object) <- value

## S4 method for signature 'MsBackendMemory'
collisionEnergy(object)

## S4 replacement method for signature 'MsBackendMemory'
collisionEnergy(object) <- value

## S4 method for signature 'MsBackendMemory'
```

```
dataOrigin(object)

## S4 replacement method for signature 'MsBackendMemory'
dataOrigin(object) <- value

## S4 method for signature 'MsBackendMemory'
dataStorage(object)

## S4 replacement method for signature 'MsBackendMemory'
dataStorage(object) <- value

## S4 method for signature 'MsBackendMemory'
intensity(object)

## S4 replacement method for signature 'MsBackendMemory'
intensity(object) <- value

## S4 method for signature 'MsBackendMemory'
ionCount(object)

## S4 method for signature 'MsBackendMemory'
isEmpty(x)

## S4 method for signature 'MsBackendMemory'
isolationWindowLowerMz(object)

## S4 replacement method for signature 'MsBackendMemory'
isolationWindowLowerMz(object) <- value

## S4 method for signature 'MsBackendMemory'
isolationWindowTargetMz(object)

## S4 replacement method for signature 'MsBackendMemory'
isolationWindowTargetMz(object) <- value

## S4 method for signature 'MsBackendMemory'
isolationWindowUpperMz(object)

## S4 replacement method for signature 'MsBackendMemory'
isolationWindowUpperMz(object) <- value

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

## S4 method for signature 'MsBackendMemory'
lengths(x, use.names = FALSE)

## S4 method for signature 'MsBackendMemory'
```

```
msLevel(object, ...)

## S4 replacement method for signature 'MsBackendMemory'
msLevel(object) <- value

## S4 method for signature 'MsBackendMemory'
mz(object)

## S4 replacement method for signature 'MsBackendMemory'
mz(object) <- value

## S4 method for signature 'MsBackendMemory'
peaksData(object, columns = c("mz", "intensity"))

## S4 replacement method for signature 'MsBackendMemory'
peaksData(object) <- value

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

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

## S4 method for signature 'MsBackendMemory'
precScanNum(object)

## S4 method for signature 'MsBackendMemory'
precursorCharge(object)

## S4 method for signature 'MsBackendMemory'
precursorIntensity(object)

## S4 method for signature 'MsBackendMemory'
precursorMz(object)

## S4 method for signature 'MsBackendMemory'
rtime(object)

## S4 replacement method for signature 'MsBackendMemory'
rtime(object) <- value

## S4 method for signature 'MsBackendMemory'
scanIndex(object)

## S4 method for signature 'MsBackendMemory'
selectSpectraVariables(object, spectraVariables = spectraVariables(object))

## S4 method for signature 'MsBackendMemory'
```

```
smoothed(object)

## S4 replacement method for signature 'MsBackendMemory'
smoothed(object) <- value

## S4 method for signature 'MsBackendMemory'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'MsBackendMemory'
spectraData(object) <- value

## S4 method for signature 'MsBackendMemory'
spectraNames(object)

## S4 replacement method for signature 'MsBackendMemory'
spectraNames(object) <- value

## S4 method for signature 'MsBackendMemory'
spectraVariables(object)

## S4 method for signature 'MsBackendMemory'
peaksVariables(object)

## S4 method for signature 'MsBackendMemory'
tic(object, initial = TRUE)

## S4 method for signature 'MsBackendMemory'
x$name

## S4 replacement method for signature 'MsBackendMemory'
x$name <- value

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

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

## S4 method for signature 'MsBackendMemory'
filterAcquisitionNum(
  object,
  n = integer(),
  dataStorage = character(),
  dataOrigin = character()
)

## S4 method for signature 'MsBackendMzR'
backendInitialize(object, files, ..., BPPARAM = bpparam())
```

```
## S4 method for signature 'MsBackendMzR'
show(object)

## S4 method for signature 'MsBackendMzR'
peaksData(object, columns = peaksVariables(object))

## S4 method for signature 'MsBackendMzR'
intensity(object)

## S4 replacement method for signature 'MsBackendMzR'
intensity(object) <- value

## S4 method for signature 'MsBackendMzR'
ionCount(object)

## S4 method for signature 'MsBackendMzR'
isCentroided(object, ...)

## S4 method for signature 'MsBackendMzR'
isEmpty(x)

## S4 method for signature 'MsBackendMzR'
lengths(x, use.names = FALSE)

## S4 method for signature 'MsBackendMzR'
mz(object)

## S4 replacement method for signature 'MsBackendMzR'
mz(object) <- value

## S4 method for signature 'MsBackendMzR'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'MsBackendMzR'
spectraData(object) <- value

## S4 method for signature 'MsBackendMzR'
spectraNames(object)

## S4 replacement method for signature 'MsBackendMzR'
spectraNames(object) <- value

## S4 method for signature 'MsBackendMzR'
spectraVariables(object)

## S4 replacement method for signature 'MsBackendMzR'
x$name <- value
```

```
## S4 method for signature 'MsBackendMzR'
export(
  object,
  x,
  file = tempfile(),
  format = c("mzML", "mzXML"),
  copy = FALSE,
  BPPARAM = bpparam()
)

## S4 method for signature 'Spectra'
show(object)
```

Value

Not applicable

Note

: this replaces **all** the data in the backend.

`joinPeaks` *Join (map) peaks of two spectra*

Description

These functions map peaks from two spectra with each other if the difference between their m/z values is smaller than defined with parameters `tolerance` and `ppm`. All functions take two matrices

- `joinPeaks`: maps peaks from two spectra allowing to specify the type of *join* that should be performed: `type = "outer"` each peak in `x` will be matched with each peak in `y`, for peaks that do not match any peak in the other spectra an `NA` intensity is returned. With `type = "left"` all peaks from the left spectrum (`x`) will be matched with peaks in `y`. Peaks in `y` that do not match any peak in `x` are omitted. `type = "right"` is the same as `type = "left"` only for `y`. Only peaks that can be matched between `x` and `y` are returned by `type = "inner"`, i.e. only peaks present in both spectra are reported.
- `joinPeaksGnps` matches/maps peaks between spectra with the same approach used in GNPS: peaks are considered matching if a) the difference in their m/z values is smaller than defined by `tolerance` and `ppm` (this is the same as `joinPeaks`) **and** b) the difference of their m/z *adjusted* for the difference of the spectras' precursor is smaller than defined by `tolerance` and `ppm`. Based on this definition, peaks in `x` can match up to two peaks in `y` hence peaks in the returned matrices might be reported multiple times. Note that if one of `xPrecursorMz` or `yPrecursorMz` are `NA` or if both are the same, the results are the same as with `joinPeaks()`. To calculate GNPS similarity scores, `gnps()` should be called on the aligned peak matrices (i.e. `compareSpectra` should be called with `MAPFUN = joinPeaksGnps` and `FUN = MsCoreUtils::gnps`).

Usage

```
joinPeaks(x, y, type = "outer", tolerance = 0, ppm = 10, ...)

joinPeaksGnps(
  x,
  y,
  xPrecursorMz = NA_real_,
  yPrecursorMz = NA_real_,
  tolerance = 0,
  ppm = 0,
  type = "outer",
  ...
)
```

Arguments

x	matrix with two columns "mz" and "intensity" containing the m/z and intensity values of the mass peaks of a spectrum.
y	matrix with two columns "mz" and "intensity" containing the m/z and intensity values of the mass peaks of a spectrum.
type	For joinPeaks and joinPeaksGnps: character(1) specifying the type of join that should be performed. See function description for details.
tolerance	numeric(1) defining a constant maximal accepted difference between m/z values of peaks from the two spectra to be matched/mapped.
ppm	numeric(1) defining a relative, m/z-dependent, maximal accepted difference between m/z values of peaks from the two spectra to be matched/mapped.
...	optional parameters passed to the MsCoreUtils::join() function.
xPrecursorMz	for joinPeaksGnps: numeric(1) with the precursor m/z of the spectrum x.
yPrecursorMz	for joinPeaksGnps: numeric(1) with the precursor m/z of the spectrum y.

Value

All functions return a list of elements "x" and "y" each being a two column matrix with m/z (first column) and intensity values (second column). The two matrices contain the matched peaks between input matrices x and y and hence have the same number of rows. Peaks present in x but not in the y input matrix have m/z and intensity values of NA in the result matrix for y (and *vice versa*).

Implementation notes

A mapping function must take two numeric matrices x and y as input and must return list with two elements named "x" and "y" that represent the aligned input matrices. The function should also have ... in its definition. Parameters ppm and tolerance are suggested but not required.

Author(s)

Johannes Rainer, Michael Witting

See Also

gnps()

Examples

```
x <- cbind(c(31.34, 50.14, 60.3, 120.9, 230, 514.13, 874.1),
    1:7)
y <- cbind(c(12, 31.35, 70.3, 120.9 + ppm(120.9, 5),
    230 + ppm(230, 10), 315, 514.14, 901, 1202),
    1:9)

## No peaks with identical m/z
joinPeaks(x, y, ppm = 0, type = "inner")

## With ppm 10 two peaks are overlapping
joinPeaks(x, y, ppm = 10, type = "inner")

## Outer join: contain all peaks from x and y
joinPeaks(x, y, ppm = 10, type = "outer")

## Left join: keep all peaks from x and those from y that match
joinPeaks(x, y, ppm = 10, type = "left")

## Right join: keep all peaks from y and those from x that match. Using
## a constant tolerance of 0.01
joinPeaks(x, y, tolerance = 0.01, type = "right")

## GNPS-like peak matching

## Define spectra
x <- cbind(mz = c(10, 36, 63, 91, 93), intensity = c(14, 15, 999, 650, 1))
y <- cbind(mz = c(10, 12, 50, 63, 105), intensity = c(35, 5, 16, 999, 450))
## The precursor m/z
pmz_x <- 91
pmz_y <- 105

## Plain joinPeaks identifies only 2 matching peaks: 1 and 5
joinPeaks(x, y)

## joinPeaksGnps finds 4 matches
joinPeaksGnps(x, y, pmz_x, pmz_y)

## with one of the two precursor m/z being NA, the result are the same as
## with joinPeaks (with type = "left").
joinPeaksGnps(x, y, pmz_x, yPrecursorMz = NA)
```

MsBackend *Mass spectrometry data backends*

Description

Note that the classes described here are not meant to be used directly by the end-users and the material in this man page is aimed at package developers.

`MsBackend` is a virtual class that defines what each different backend needs to provide. `MsBackend` objects provide access to mass spectrometry data. Such backends can be classified into *in-memory* or *on-disk* backends, depending on where the data, i.e spectra (m/z and intensities) and spectra annotation (MS level, charge, polarity, ...) are stored.

Typically, in-memory backends keep all data in memory ensuring fast data access, while on-disk backends store (parts of) their data on disk and retrieve it on demand.

The *Backend functions and implementation notes for new backend classes* section documents the API that a backend must implement.

Currently available backends are:

- `MsBackendMemory` and `MsBackendDataFrame`: store all data in memory. The `MsBackendMemory` is optimized for accessing and processing the peak data (i.e. the numerical matrices with the m/z and intensity values) while the `MsBackendDataFrame` keeps all data in a `DataFrame`.
- `MsBackendMzR`: stores the m/z and intensities on-disk in raw data files (typically `mzML` or `mzXML`) and the spectra annotation information (header) in memory in a `DataFrame`. This backend requires the `mzR` package.
- `MsBackendHdf5Peaks`: stores the m/z and intensities on-disk in custom hdf5 data files and the remaining spectra variables in memory (in a `DataFrame`). This backend requires the `rhdf5` package.

See below for more details about individual backends.

Usage

```
## S4 method for signature 'MsBackend'
backendBpparam(object, BPPARAM = bpparam())

## S4 method for signature 'MsBackend'
backendInitialize(object, ...)

## S4 method for signature 'list'
backendMerge(object, ...)

## S4 method for signature 'MsBackend'
backendMerge(object, ...)

## S4 method for signature 'MsBackend'
export(object, ...)

## S4 method for signature 'MsBackend'
acquisitionNum(object)

## S4 method for signature 'MsBackend'
peaksData(object, columns = c("mz", "intensity"))
```

```
## S4 method for signature 'MsBackend'
peaksVariables(object)

## S4 method for signature 'MsBackend'
centroided(object)

## S4 replacement method for signature 'MsBackend'
centroided(object) <- value

## S4 method for signature 'MsBackend'
collisionEnergy(object)

## S4 replacement method for signature 'MsBackend'
collisionEnergy(object) <- value

## S4 method for signature 'MsBackend'
dataOrigin(object)

## S4 replacement method for signature 'MsBackend'
dataOrigin(object) <- value

## S4 method for signature 'MsBackend'
dataStorage(object)

## S4 replacement method for signature 'MsBackend'
dataStorage(object) <- value

## S4 method for signature 'MsBackend'
dropNaSpectraVariables(object)

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

## S4 method for signature 'MsBackend'
filterDataOrigin(object, dataOrigin = character())

## S4 method for signature 'MsBackend'
filterDataStorage(object, dataStorage = character())

## S4 method for signature 'MsBackend'
filterEmptySpectra(object, ...)

## S4 method for signature 'MsBackend'
filterIsolationWindow(object, mz = numeric(), ...)

## S4 method for signature 'MsBackend'
filterMsLevel(object, msLevel = integer())
```

```
## S4 method for signature 'MsBackend'
filterPolarity(object, polarity = integer())

## S4 method for signature 'MsBackend'
filterPrecursorMzRange(object, mz = numeric())

## S4 method for signature 'MsBackend'
filterPrecursorMz(object, mz = numeric())

## S4 method for signature 'MsBackend'
filterPrecursorMzValues(object, mz = numeric(), ppm = 20, tolerance = 0)

## S4 method for signature 'MsBackend'
filterPrecursorCharge(object, z = integer())

## S4 method for signature 'MsBackend'
filterPrecursorScan(object, acquisitionNum = integer(), f = dataOrigin(object))

## S4 method for signature 'MsBackend'
filterRt(object, rt = numeric(), msLevel. = uniqueMsLevels(object))

## S4 method for signature 'MsBackend'
intensity(object)

## S4 replacement method for signature 'MsBackend'
intensity(object) <- value

## S4 method for signature 'MsBackend'
ionCount(object)

## S4 method for signature 'MsBackend'
isCentroided(object, ...)

## S4 method for signature 'MsBackend'
isEmpty(x)

## S4 method for signature 'MsBackend'
isolationWindowLowerMz(object)

## S4 replacement method for signature 'MsBackend'
isolationWindowLowerMz(object) <- value

## S4 method for signature 'MsBackend'
isolationWindowTargetMz(object)

## S4 replacement method for signature 'MsBackend'
isolationWindowTargetMz(object) <- value
```

```
## S4 method for signature 'MsBackend'
isolationWindowUpperMz(object)

## S4 replacement method for signature 'MsBackend'
isolationWindowUpperMz(object) <- value

## S4 method for signature 'MsBackend'
isReadOnly(object)

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

## S4 method for signature 'MsBackend'
msLevel(object)

## S4 method for signature 'MsBackend'
mz(object)

## S4 replacement method for signature 'MsBackend'
mz(object) <- value

## S4 method for signature 'MsBackend'
lengths(x, use.names = FALSE)

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

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

## S4 method for signature 'MsBackend'
precScanNum(object)

## S4 method for signature 'MsBackend'
precursorCharge(object)

## S4 method for signature 'MsBackend'
precursorIntensity(object)

## S4 method for signature 'MsBackend'
precursorMz(object)

## S4 replacement method for signature 'MsBackend'
peaksData(object) <- value

## S4 method for signature 'MsBackend'
reset(object)
```

```
## S4 method for signature 'MsBackend'
rtime(object)

## S4 replacement method for signature 'MsBackend'
rtime(object) <- value

## S4 method for signature 'MsBackend'
scanIndex(object)

## S4 method for signature 'MsBackend'
selectSpectraVariables(object, spectraVariables = spectraVariables(object))

## S4 method for signature 'MsBackend'
smoothed(object)

## S4 replacement method for signature 'MsBackend'
smoothed(object) <- value

## S4 method for signature 'MsBackend'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'MsBackend'
spectraData(object) <- value

## S4 method for signature 'MsBackend'
spectraNames(object)

## S4 replacement method for signature 'MsBackend'
spectraNames(object) <- value

## S4 method for signature 'MsBackend'
spectraVariables(object)

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

## S4 method for signature 'MsBackend'
supportsSetBackend(object, ...)

## S4 method for signature 'MsBackend'
tic(object, initial = TRUE)

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

## S4 method for signature 'MsBackend'
x$name
```

```
## S4 replacement method for signature 'MsBackend'
x$name <- value

## S4 method for signature 'MsBackend'
x[[i, j, ...]]

## S4 replacement method for signature 'MsBackend'
x[[i, j, ...]] <- value

## S4 method for signature 'MsBackend'
uniqueMsLevels(object, ...)

MsBackendDataFrame()

## S4 method for signature 'MsBackendDataFrame'
backendInitialize(object, data, ...)

MsBackendHdf5Peaks()

MsBackendMemory()

## S4 method for signature 'MsBackendMemory'
backendInitialize(object, data, peaksVariables = c("mz", "intensity"), ...)

MsBackendMzR()
```

Arguments

object	Object extending MsBackend.
BPPARAM	for backendBpparam: parameter object from the BiocParallel package defining the parallel processing setup. Defaults to BPPARAM = bpparam(). See bpparam() for more information.
...	Additional arguments.
columns	For spectraData accessor: optional character with column names (spectra variables) that should be included in the returned DataFrame. By default, all columns are returned. For peaksData accessor: optional character with requested columns in the individual matrix of the returned list. Defaults to peaksVariables(object) and depends on what *peaks variables* the backend provides.
value	replacement value for <- methods. See individual method description or expected data type.
n	for filterAcquisitionNum: integer with the acquisition numbers to filter for.
file	For filterFile: index or name of the file(s) to which the data should be subsetted. For export: character of length 1 or equal to the number of spectra.
dataOrigin	For filterDataOrigin: character to define which spectra to keep. For filterAcquisitionNum: optionally specify if filtering should occurr only for spectra of selected dataOrigin.

dataStorage For filterDataStorage: character to define which spectra to keep. For filterAcquisitionNum: optionally specify if filtering should occur only for spectra of selected dataStorage.

mz For filterIsolationWindow: numeric(1) with the m/z value to filter the object. For filterPrecursorMzRange: numeric(2) with the lower and upper m/z boundary. For filterPrecursorMzValues: numeric with the m/z value(s) to filter the object.

msLevel integer defining the MS level of the spectra to which the function should be applied. For filterMsLevel: the MS level to which object should be subsetted.

polarity For filterPolarity: integer specifying the polarity to to subset object.

ppm For filterPrecursorMzValues: numeric(1) with the m/z-relative maximal acceptable difference for a m/z to be considered matching. See closest() for details.

tolerance For filterPrecursorMzValues: numeric(1) with the maximal absolute acceptable difference for a m/z value to be considered matching. See closest() for details.

z For filterPrecursorCharge: integer() with the precursor charges to be used as filter.

acquisitionNum for filterPrecursorScan: integer with the acquisition number of the spectra to which the object should be subsetted.

f factor defining the grouping to split x. See split(). For filterPrecursorScan: factor defining from which original data files the spectra derive to avoid selecting spectra from different samples/files. Defaults to f = dataOrigin(object).

rt for filterRt: numeric(2) defining the retention time range to be used to subset/filter object.

msLevel. same as msLevel above.

x Object extending MsBackend.

use.names For lengths: whether spectrum names should be used.

spectraVariables
For selectSpectraVariables: character with the names of the spectra variables to which the backend should be subsetted.

drop For [: not considered.

initial For tic: logical(1) whether the initially reported total ion current should be reported, or whether the total ion current should be (re)calculated on the actual data (initial = FALSE).

i For [: integer, logical or character to subset the object.

j For [: not supported.

name For $ and $<-: the name of the spectra variable to return or set.

data For backendInitialize: DataFrame with spectrum metadata/data. This parameter can be empty for MsBackendMzR backends but needs to be provided for MsBackendDataFrame backends.

peaksVariables For backendInitialize for MsBackendMemory: character specifying which of the columns of the provided data contain *peaks variables* (i.e. information for individual mass peaks). Defaults to peaksVariables = c("mz", "intensity"). "mz" and "intensity" should **always** be specified.

Value

See documentation of respective function.

Implementation notes

Backends extending MsBackend **must** implement all of its methods (listed above). Developers of new MsBackends should follow the MsBackendDataFrame implementation. To ensure a new implementation being conform with the MsBackend definition, developers should included test suites provided by this package in their unit test setup. For that a variable be should be created in the package's "testthat.R" file that represents a (initialized) instance of the developed backend. Then the path to the test suites should be defined with test_suite <- system.file("test_backends", "test_MsBackend", package = "Spectra") followed by test_dir(test_suite) to run all test files in that directory. Individual unit test files could be run with test_file(file.path(test_suite, "test_spectra_variables.R"), stop_on_failure = TRUE) (note that without stop_on_failure = TRUE tests would fail silently) . Adding this code to the packages "testthat.R" file ensures that all tests checking the validity of an MsBackend instance defined in the Spectra package are also run on the newly develped backend class.

The MsBackend defines the following slots:

- @readonly: logical(1) whether the backend supports writing/replacing of m/z or intensity values.

Backends extending MsBackend **must** implement all of its methods (listed above). Developers of new MsBackends should follow the MsBackendDataFrame implementation.

The MsBackendCached() backend provides a caching mechanism to allow *read only* backends to add or change spectra variables. This backend shouldn't be used on its own, but is meant to be extended. See MsBackendCached() for details.

The MsBackend defines the following slots:

- @readonly: logical(1) whether the backend supports writing/replacing of m/z or intensity values.

Backend functions

New backend classes **must** extend the base MsBackend class will have to implement some of the following methods (see the MsBackend vignette for detailed description and examples):

- [: subset the backend. Only subsetting by element (*row*/i) is allowed. Parameter i should support integer indices and logical and should throw an error if i is out of bounds. The MsCoreUtils::i2index could be used to check the input i. For i = integer() an empty backend should be returned.
- $, $<-: access or set/add a single spectrum variable (column) in the backend. Using a value of NULL should allow deleting the specified spectra variable. An error should be thrown if the spectra variable is not available.

- [[, [[<-: access or set/add a single spectrum variable (column) in the backend. The default implementation uses $, thus these methods don't have to be implemented for new classes extending MsBackend.
- acquisitionNum: returns the acquisition number of each spectrum. Returns an integer of length equal to the number of spectra (with NA_integer_ if not available).
- backendBpparam: return the parallel processing setup supported by the backend class. This function can be used by any higher level function to evaluate whether the provided parallel processing setup (or the default one returned by bpparam()) is supported by the backend. Backends not supporting parallel processing (e.g. because they contain a connection to a database that can not be shared across processes) should extend this method to return only SerialParam() and hence disable parallel processing for (most) methods and functions.
- backendInitialize: initialises the backend. This method is supposed to be called rights after creating an instance of the backend class and should prepare the backend (e.g. set the data for the memory backend or read the spectra header data for the MsBackendMzR backend). Parameters can be defined freely for each backend, depending on what is needed to initialize the backend. It is however suggested to also support a parameter data that can be used to submit the full spectra data as a DataFrame to the backend. This would allow the backend to be also usable for the setBackend() function from Spectra. Note that eventually (for *read-only* backends) also the supportsSetBackend method would need to be implemented to return TRUE. The backendInitialize method has also to ensure to correctly set spectra variable dataStorage.
- backendMerge: merges (combines) MsBackend objects into a single instance. All objects to be merged have to be of the same type (e.g. MsBackendDataFrame()).
- dataOrigin: gets a character of length equal to the number of spectra in object with the *data origin* of each spectrum. This could e.g. be the mzML file from which the data was read.
- dataStorage: gets a character of length equal to the number of spectra in object with the data storage of each spectrum. Note that a dataStorage of NA_character_ is not supported.
- dropNaSpectraVariables: removes spectra variables (i.e. columns in the object's spectraData that contain only missing values (NA). Note that while columns with only NAs are removed, a spectraData call after dropNaSpectraVariables might still show columns containing NA values for *core* spectra variables.
- centroided, centroided<-: gets or sets the centroiding information of the spectra. centroided returns a logical vector of length equal to the number of spectra with TRUE if a spectrum is centroided, FALSE if it is in profile mode and NA if it is undefined. See also isCentroided for estimating from the spectrum data whether the spectrum is centroided. value for centroided<- is either a single logical or a logical of length equal to the number of spectra in object.
- collisionEnergy, collisionEnergy<-: gets or sets the collision energy for all spectra in object. collisionEnergy returns a numeric with length equal to the number of spectra (NA_real_ if not present/defined), collisionEnergy<- takes a numeric of length equal to the number of spectra in object.
- export: exports data from a Spectra class to a file. This method is called by the export,Spectra method that passes itself as a second argument to the function. The export,MsBackend implementation is thus expected to take a Spectra class as second argument from which all data is exported. Taking data from a Spectra class ensures that also all eventual data manipulations (cached in the Spectra's lazy evaluation queue) are applied prior to export - this would not be possible with only a MsBackend class. An example implementation is the export method for

the MsBackendMzR backend that supports export of the data in *mzML* or *mzXML* format. See the documentation for the MsBackendMzR class below for more information.

- filterAcquisitionNum: filters the object keeping only spectra matching the provided acquisition numbers (argument n). If dataOrigin or dataStorage is also provided, object is subsetted to the spectra with an acquisition number equal to n **in spectra with matching dataOrigin or dataStorage values** retaining all other spectra.
- filterDataOrigin: filters the object retaining spectra matching the provided dataOrigin. Parameter dataOrigin has to be of type character and needs to match exactly the data origin value of the spectra to subset. filterDataOrigin should return the data ordered by the provided dataOrigin parameter, i.e. if dataOrigin = c("2", "1") was provided, the spectra in the resulting object should be ordered accordingly (first spectra from data origin "2" and then from "1"). Implementation of this method is optional since a default implementation for MsBackend is available.
- filterDataStorage: filters the object retaining spectra matching the provided dataStorage. Parameter dataStorage has to be of type character and needs to match exactly the data storage value of the spectra to subset. filterDataStorage should return the data ordered by the provided dataStorage parameter, i.e. if dataStorage = c("2", "1") was provided, the spectra in the resulting object should be ordered accordingly (first spectra from data storage "2" and then from "1"). Implementation of this method is optional since a default implementation for MsBackend is available.
- filterEmptySpectra: removes empty spectra (i.e. spectra without peaks). Implementation of this method is optional since a default implementation for MsBackend is available.
- filterFile: retains data of files matching the file index or file name provided with parameter file.
- filterIsolationWindow: retains spectra that contain mz in their isolation window m/z range (i.e. with an isolationWindowLowerMz <= mz and isolationWindowUpperMz >= mz. Implementation of this method is optional since a default implementation for MsBackend is available.
- filterMsLevel: retains spectra of MS level msLevel. Implementation of this method is optional since a default implementation for MsBackend is available.
- filterPolarity: retains spectra of polarity polarity. Implementation of this method is optional since a default implementation for MsBackend is available.
- filterPrecursorMzRange (previously filterPrecursorMz): retains spectra with a precursor m/z within the provided m/z range. Implementation of this method is optional since a default implementation for MsBackend is available.
- filterPrecursorMzValues: retains spectra with a precursor m/z matching any of the provided m/z values (given ppm and tolerance). Implementation of this method is optional since a default implementation for MsBackend is available.
- filterPrecursorCharge: retains spectra with the defined precursor charge(s). Implementation of this method is optional since a default implementation for MsBackend is available.
- filterPrecursorScan: retains parent (e.g. MS1) and children scans (e.g. MS2) of acquisition number acquisitionNum. Parameter f is supposed to define the origin of the spectra (i.e. the original data file) to ensure related spectra from the same file/sample are selected and retained. Implementation of this method is optional since a default implementation for MsBackend is available.

- filterRt: retains spectra of MS level msLevel with retention times within (>=) rt[1] and (<=) rt[2]. Implementation of this method is optional since a default implementation for MsBackend is available.
- intensity: gets the intensity values from the spectra. Returns a NumericList() of numeric vectors (intensity values for each spectrum). The length of the list is equal to the number of spectra in object.
- intensity<-: replaces the intensity values. value has to be a list (or NumericList()) of length equal to the number of spectra and the number of values within each list element identical to the number of peaks in each spectrum (i.e. the lengths(x)). Note that just writeable backends support this method.
- ionCount: returns a numeric with the sum of intensities for each spectrum. If the spectrum is empty (see isEmpty), NA_real_ is returned.
- isCentroided: a heuristic approach assessing if the spectra in object are in profile or centroided mode. The function takes the qtl th quantile top peaks, then calculates the difference between adjacent m/z value and returns TRUE if the first quartile is greater than k. (See Spectra:::.peaks_is_centroided for the code.)
- isEmpty: checks whether a spectrum in object is empty (i.e. does not contain any peaks). Returns a logical vector of length equal number of spectra.
- isolationWindowLowerMz, isolationWindowLowerMz<-: gets or sets the lower m/z boundary of the isolation window.
- isolationWindowTargetMz, isolationWindowTargetMz<-: gets or sets the target m/z of the isolation window.
- isolationWindowUpperMz, isolationWindowUpperMz<-: gets or sets the upper m/z boundary of the isolation window.
- isReadOnly: returns a logical(1) whether the backend is *read only* or does allow also to write/update data.
- length: returns the number of spectra in the object.
- lengths: gets the number of peaks (m/z-intensity values) per spectrum. Returns an integer vector (length equal to the number of spectra). For empty spectra, 0 is returned.
- msLevel: gets the spectra's MS level. Returns an integer vector (of length equal to the number of spectra) with the MS level for each spectrum (or NA_integer_ if not available).
- mz: gets the mass-to-charge ratios (m/z) from the spectra. Returns a NumericList() or length equal to the number of spectra, each element a numeric vector with the m/z values of one spectrum.
- mz<-: replaces the m/z values. value has to be a list of length equal to the number of spectra and the number of values within each list element identical to the number of peaks in each spectrum (i.e. the lengths(x)). Note that just writeable backends support this method.
- polarity, polarity<-: gets or sets the polarity for each spectrum. polarity returns an integer vector (length equal to the number of spectra), with 0 and 1 representing negative and positive polarities, respectively. polarity<- expects an integer vector of length 1 or equal to the number of spectra.
- precursorCharge, precursorIntensity, precursorMz, precScanNum, precAcquisitionNum: get the charge (integer), intensity (numeric), m/z (numeric), scan index (integer) and acquisition number (interger) of the precursor for MS level 2 and above spectra from the

object. Returns a vector of length equal to the number of spectra in `object`. `NA` are reported for MS1 spectra of if no precursor information is available.

- `peaksData` returns a `list` with the spectras' peak data, i.e. numeric `matrix` with peak values. The length of the list is equal to the number of spectra in `object`. Each element of the list is a `numeric` `matrix` with columns depending on the provided `columns` parameter (by default `"mz"` and `"intensity"`, but depends on the backend's available `peaksVariables`). For an empty spectrum, a `matrix` with 0 rows and columns according to `columns` is returned. The optional parameter `columns`, if supported by the backend, allows to define which peak variables should be returned in the `numeric` peak `matrix`. As a default `c("mz", "intensity")` should be used.
- `peaksData<-` replaces the peak data (m/z and intensity values) of the backend. This method expects a `list` of `matrix` objects with columns `"mz"` and `"intensity"` that has the same length as the number of spectra in the backend. Note that just writeable backends support this method.
- `peaksVariables`: lists the available variables for mass peaks. Default peak variables are `"mz"` and `"intensity"` (which all backends need to support and provide), but some backends might provide additional variables. These variables correspond to the column names of the `numeric` `matrix` representing the peak data (returned by `peaksData`).
- `reset` a backend (if supported). This method will be called on the backend by the `reset,Spectra` method that is supposed to restore the data to its original state (see `reset,Spectra` for more details). The function returns the *reset* backend. The default implementation for `MsBackend` returns the backend as-is.
- `rtime`, `rtime<-`: gets or sets the retention times for each spectrum (in seconds). `rtime` returns a `numeric` vector (length equal to the number of spectra) with the retention time for each spectrum. `rtime<-` expects a numeric vector with length equal to the number of spectra.
- `scanIndex`: returns an `integer` vector with the *scan index* for each spectrum. This represents the relative index of the spectrum within each file. Note that this can be different to the `acquisitionNum` of the spectrum which is the index of the spectrum as reported in the mzML file.
- `selectSpectraVariables`: reduces the information within the backend to the selected spectra variables. It is suggested to **not** remove values for the `"dataStorage"` variable, since this might be required for some backends to work properly (such as the `MsBackendMzR`).
- `smoothed`,`smoothed<-`: gets or sets whether a spectrum is *smoothed*. `smoothed` returns a `logical` vector of length equal to the number of spectra. `smoothed<-` takes a `logical` vector of length 1 or equal to the number of spectra in `object`.
- `spectraData`, `spectraData<-`: gets or sets general spectrum metadata (annotation, also called header). `spectraData` returns a `DataFrame`, `spectraData<-` expects a `DataFrame` with the same number of rows as there are spectra in `object`. Note that `spectraData` has to return the full data, i.e. also the m/z and intensity values (as a `list` or `SimpleList` in columns `"mz"` and `"intensity"`.
- `spectraNames`: returns a `character` vector with the names of the spectra in `object` or `NULL` if not set. `spectraNames<-` allows to set spectra names (if the object is not read-only).
- `spectraVariables`: returns a `character` vector with the available spectra variables (columns, fields or attributes) available in `object`. This should return **all** spectra variables which are present in `object`, also `"mz"` and `"intensity"` (which are by default not returned by the `spectraVariables,Spectra` method).

- split: splits the backend into a list of backends (depending on parameter f). The default method for MsBackend uses split.default(), thus backends extending MsBackend don't necessarily need to implement this method.
- supportsSetBackend: whether a MsBackend supports the Spectra setBackend function. For a MsBackend to support setBackend it needs to have a parameter called data in its backendInitialize method that support receiving all spectra data as a DataFrame from another backend and to initialize the backend with this data. In general *read-only* backends do not support setBackend hence, the default implementation of supportsSetBackend returns !isReadOnly(object). If a read-only backend would support the setBackend and being initialized with a DataFrame an implementation of this method for that backend could be defined that returns TRUE (see also the MsBackend vignette for details and examples).
- tic: gets the total ion current/count (sum of signal of a spectrum) for all spectra in object. By default, the value reported in the original raw data file is returned. For an empty spectrum, NA_real_ is returned.
- uniqueMsLevels: gets the unique MS levels of all spectra in object. The default implementation calls unique(msLevel(object)) but more efficient implementations could be defined for specific backends.

Subsetting and merging backend classes

Backend classes must support (implement) the [method to subset the object. This method should only support subsetting by spectra (rows, i) and has to return a MsBackend class.

Backends extending MsBackend should also implement the backendMerge method to support combining backend instances (only backend classes of the same type should be merged). Merging should follow the following rules:

- The whole spectrum data of the various objects should be merged. The resulting merged object should contain the union of the individual objects' spectra variables (columns/fields), with eventually missing variables in one object being filled with NA.

In-memory data backends

MsBackendMemory and MsBackendDataFrame:

The MsBackendMemory and MsBackendDataFrame objects keep all MS data in memory are thus ideal for fast data processing. Due to their large memory footprint they are however not suited for large scale experiments. The two backends store the data different. The MsBackendDataFrame stores all data in a DataFrame and thus supports also S4-classes as spectra variables. Also, sepratate access to m/z or intensity values (i.e. using the mz and intensity methods) is faster for the MsBackendDataFrame. The MsBackendMemory on the other hand, due to the way the data is organized internally, provides much faster access to the full peak data (i.e. the numerical matrices of m/z and intensity values). Also subsetting and access to any spectra variable (except "mz" and "intensity" is fastest for the MsBackendMemory. Finally, the MsBackendMemory supports also arbitrary peak annotations while the MsBackendDataFrame does not have support for such additional peak variables.

Thus, for most use cases, the MsBackendMemory provides a higher performance and flexibility than the MsBackendDataFrame and should thus be preferred. See also issue 246 for a performance comparison.

New objects can be created with the MsBackendMemory() and MsBackendDataFrame() function, respectively. The backend can be subsequently initialized with the backendInitialize method, taking a DataFrame (or data.frame) with the MS data as first parameter data. backendInitialize for MsBackendMemory has a second parameter peaksVariables (default peaksVariables = c("mz", "intensity") that allows to specify which of the columns in the provided data frame should be considered as a *peaks variable* (i.e. information of an individual mass peak) rather than a *spectra variable* (i.e. information of an individual spectrum). Note that it is important to also include "mz" and "intensity" in peaksVariables as these would otherwise be considered to be spectra variables! Also, while it is possible to change the values of existing peaks variables using the $<- method, this method does **not** allow to add new peaks variables to an existing MsBackendMemory. New peaks variables should be added using the backendInitialize method.

Suggested columns of this DataFrame are:

- "msLevel": integer with MS levels of the spectra.
- "rt": numeric with retention times of the spectra.
- "acquisitionNum": integer with the acquisition number of the spectrum.
- "scanIndex": integer with the index of the scan/spectrum within the *mzML/mzXML/CDF* file.
- "dataOrigin": character defining the *data origin*.
- "dataStorage": character indicating grouping of spectra in different e.g. input files. Note that missing values are not supported.
- "centroided": logical whether the spectrum is centroided.
- "smoothed": logical whether the spectrum was smoothed.
- "polarity": integer with the polarity information of the spectra.
- "precScanNum": integer specifying the index of the (MS1) spectrum containing the precursor of a (MS2) spectrum.
- "precursorMz": numeric with the m/z value of the precursor.
- "precursorIntensity": numeric with the intensity value of the precursor.
- "precursorCharge": integer with the charge of the precursor.
- "collisionEnergy": numeric with the collision energy.
- "mz": NumericList() of numeric vectors representing the m/z values for each spectrum.
- "intensity": NumericList() of numeric vectors representing the intensity values for each spectrum.

Additional columns are allowed too.

For the MsBackendMemory, any column in the provided data.frame which contains a list of vectors each with length equal to the number of peaks for a spectrum will be used as additional *peak variable* (see examples below for details).

The MsBackendDataFrame ignores parameter columns of the peaksData function and returns **always** m/z and intensity values.

MsBackendMzR, on-disk MS data backend

The MsBackendMzR keeps only a limited amount of data in memory, while the spectra data (m/z and intensity values) are fetched from the raw files on-demand. This backend uses the mzR package for data import and retrieval and hence requires that package to be installed. Also, it can only be used to import and represent data stored in *mzML*, *mzXML* and *CDF* files.

The MsBackendMzR backend extends the MsBackendDataFrame backend using its DataFrame to keep spectra variables (except m/z and intensity) in memory.

New objects can be created with the MsBackendMzR() function which can be subsequently filled with data by calling backendInitialize passing the file names of the input data files with argument files.

This backend provides an export method to export data from a Spectra in *mzML* or *mzXML* format. The definition of the function is:

```
export(object, x, file = tempfile(), format = c("mzML", "mzXML"), copy = FALSE)
```

The parameters are:

- object: an instance of the MsBackendMzR class.
- x: the Spectra object to be exported.
- file: character with the (full) output file name(s). Should be of length 1 or equal length(x). If a single file is specified, all spectra are exported to that file. Alternatively it is possible to specify for each spectrum in x the name of the file to which it should be exported (and hence file has to be of length equal length(x)).
- format: character(1), either "mzML" or "mzXML" defining the output file format.
- copy: logical(1) whether general file information should be copied from the original MS data files. This only works if x uses a MsBackendMzR backend and if dataOrigin(x) contains the original MS data file names.
- BPPARAM: parallel processing settings.

See examples in Spectra or the vignette for more details and examples.

The MsBackendMzR ignores parameter columns of the peaksData function and returns **always** m/z and intensity values.

MsBackendHdf5Peaks, on-disk MS data backend

The MsBackendHdf5Peaks keeps, similar to the MsBackendMzR, peak data (i.e. m/z and intensity values) in custom data files (in HDF5 format) on disk while the remaining spectra variables are kept in memory. This backend supports updating and writing of manipulated peak data to the data files.

New objects can be created with the MsBackendHdf5Peaks() function which can be subsequently filled with data by calling the object's backendInitialize method passing the desired file names of the HDF5 data files along with the spectra variables in form of a DataFrame (see MsBackendDataFrame for the expected format). An optional parameter hdf5path allows to specify the folder where the HDF5 data files should be stored to. If provided, this is added as the path to the submitted file names (parameter files).

By default backendInitialize will store all peak data into a single HDF5 file which name has to be provided with the parameter files. To store peak data across several HDF5 files data has to contain a column "dataStorage" that defines the grouping of spectra/peaks into files: peaks for

spectra with the same value in "dataStorage" are saved into the same HDF5 file. If parameter files is omitted, the value in dataStorage is used as file name (replacing any file ending with ".h5". To specify the file names, files' length has to match the number of unique elements in "dataStorage".

For details see examples on the Spectra() help page.

The MsBackendHdf5Peaks ignores parameter columns of the peaksData function and returns **always** m/z and intensity values.

Author(s)

Johannes Rainer, Sebastian Gibb, Laurent Gatto

Examples

```
## The MsBackend class is a virtual class and can not be instantiated
## directly. Below we define a new backend class extending this virtual
## class
MsBackendDummy <- setClass("MsBackendDummy", contains = "MsBackend")
MsBackendDummy()

## This class inherits now all methods from `MsBackend`, all of which
## however throw an error. These methods would have to be implemented
## for the new backend class.
try(mz(MsBackendDummy()))

## See `MsBackendDataFrame` as a reference implementation for a backend
## class (in the *R/MsBackendDataFrame.R* file).

## MsBackendDataFrame
##
## The `MsBackendDataFrame` uses a `S4Vectors::DataFrame` to store all MS
## data. Below we create such a backend by passing a `DataFrame` with all
## data to it.
data <- DataFrame(msLevel = c(1L, 2L, 1L), scanIndex = 1:3)
data$mz <- list(c(1.1, 1.2, 1.3), c(1.4, 54.2, 56.4, 122.1), c(15.3, 23.2))
data$intensity <- list(c(3, 2, 3), c(45, 100, 12.2, 1), c(123, 12324.2))

## Backends are supposed to be created with their specific constructor
## function
be <- MsBackendDataFrame()

be

## The `backendInitialize` method initializes the backend filling it with
## data. This method can take any parameters needed for the backend to
## get loaded with the data (e.g. a file name from which to load the data,
## a database connection or, in this case, a data frame containing the data).
be <- backendInitialize(be, data)

be
```

```
## Data can be accessed with the accessor methods
msLevel(be)

mz(be)

## Even if no data was provided for all spectra variables, its accessor
## methods are supposed to return a value.
precursorMz(be)

## The `peaksData` method is supposed to return the peaks of the spectra as
## a `list`.
peaksData(be)

## List available peaks variables
peaksVariables(be)

## Use columns to extract specific peaks variables. Below we extract m/z and
## intensity values, but in reversed order to the default.
peaksData(be, columns = c("intensity", "mz"))

## List available spectra variables (i.e. spectrum metadata)
spectraVariables(be)

## Extract precursor m/z, rtime, MS level spectra variables
spectraData(be, c("precursorMz", "rtime", "msLevel"))

## MsBackendMemory
##
## The `MsBackendMemory` uses a more efficient internal data organization
## and allows also adding arbitrary additional peaks variables (annotations)
## Below we thus add a column "peak_ann" with arbitrary names/ids for each
## peak and add the name of this column to the `peaksVariables` parameter
## of the `backendInitialize` method (in addition to `"mz"` and
## `"intensity"` that should **always** be specified.
data$peak_ann <- list(c("a", "", "d"), c("", "d", "e", "f"), c("h", "i"))
be <- backendInitialize(MsBackendMemory(), data,
    peaksVariables = c("mz", "intensity", "peak_ann"))
be

spectraVariables(be)

## peak_ann is also listed as a peaks variable
peaksVariables(be)

## The additional peaks variable can be accessed using the peaksData
## function
peaksData(be, "peak_ann")

## The $<- method can be used to replace values of an existing peaks
## variable. It is important that the number of elements matches the
## number of peaks per spectrum.
be$peak_ann <- list(1:3, 1:4, 1:2)
```

```
## A peaks variable can again be removed by setting it to NULL
be$peak_ann <- NULL

peaksVariables(be)
```

MsBackendCached *Base MsBackend class providing data caching mechanism*

Description

The MsBackendCached class is a rudimentary implementation of the MsBackend providing a simple mechanism to cache spectra data locally. This class is thought to be used as a base class for other MsBackend implementations to reuse its caching mechanism and avoid having to re-implement commonly used methods. This class is thus not thought to be used directly by a user.

The MsBackendCached caching mechanism allows MsBackend instances to add or replace spectra variables even if the backend used by them does not allow to alter values (e.g. if a SQL database is used as a backend). Any replacement operation with $<- will add the specified values to a local data.frame within the MsBackendCached class that allows to *cache* these values (increasing obviously the memory demand of the object).

Any data accessor functions of the extending MsBackend class (such as $ or msLevel or spectraData) should first use callNextMethod to call the respective accessor of MsBackendCached that will evaluate if the requested spectra variable(s) are in the local cache and return these. If the requested spectra variables are neither in the local cache, nor listed in the @spectraVariables slot (which defines all spectra variables that can be requested from the extending MsBackend class) but are *core spectra variables* then missing values of the correct data type are returned.

Usage

```
MsBackendCached()

## S4 method for signature 'MsBackendCached'
backendInitialize(
  object,
  data = data.frame(),
  nspectra = 0L,
  spectraVariables = character(),
  ...
)

## S4 method for signature 'MsBackendCached'
dataStorage(object)

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

## S4 method for signature 'MsBackendCached'
spectraVariables(object)
```

```
## S4 method for signature 'MsBackendCached'
spectraData(object, columns = spectraVariables(object))

## S4 replacement method for signature 'MsBackendCached'
spectraData(object) <- value

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

## S4 method for signature 'MsBackendCached'
x$name

## S4 replacement method for signature 'MsBackendCached'
x$name <- value

## S4 method for signature 'MsBackendCached'
selectSpectraVariables(object, spectraVariables = spectraVariables(object))

## S4 method for signature 'MsBackendCached'
show(object)

## S4 method for signature 'MsBackendCached'
acquisitionNum(object)

## S4 method for signature 'MsBackendCached'
centroided(object)

## S4 replacement method for signature 'MsBackendCached'
centroided(object) <- value

## S4 method for signature 'MsBackendCached'
collisionEnergy(object)

## S4 replacement method for signature 'MsBackendCached'
collisionEnergy(object) <- value

## S4 method for signature 'MsBackendCached'
dataOrigin(object)

## S4 replacement method for signature 'MsBackendCached'
dataOrigin(object) <- value

## S4 method for signature 'MsBackendCached'
msLevel(object)

## S4 method for signature 'MsBackendCached'
intensity(object)
```

```
## S4 method for signature 'MsBackendCached'
ionCount(object)

## S4 method for signature 'MsBackendCached'
isEmpty(x)

## S4 method for signature 'MsBackendCached'
isolationWindowLowerMz(object)

## S4 replacement method for signature 'MsBackendCached'
isolationWindowLowerMz(object) <- value

## S4 method for signature 'MsBackendCached'
isolationWindowTargetMz(object)

## S4 replacement method for signature 'MsBackendCached'
isolationWindowTargetMz(object) <- value

## S4 method for signature 'MsBackendCached'
isolationWindowUpperMz(object)

## S4 replacement method for signature 'MsBackendCached'
isolationWindowUpperMz(object) <- value

## S4 method for signature 'MsBackendCached'
lengths(x, use.names = FALSE)

## S4 method for signature 'MsBackendCached'
mz(object)

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

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

## S4 method for signature 'MsBackendCached'
precursorCharge(object)

## S4 method for signature 'MsBackendCached'
precursorIntensity(object)

## S4 method for signature 'MsBackendCached'
precursorMz(object)

## S4 method for signature 'MsBackendCached'
rtime(object)
```

```
## S4 replacement method for signature 'MsBackendCached'
rtime(object) <- value

## S4 method for signature 'MsBackendCached'
scanIndex(object)

## S4 method for signature 'MsBackendCached'
smoothed(object)

## S4 replacement method for signature 'MsBackendCached'
smoothed(object) <- value
```

Arguments

object	A MsBackendCached object.
data	For backendInitialize: (optional) data.frame with cached values. The number of rows (and their order) has to match the number of spectra.
nspectra	For backendInitialize: integer with the number of spectra.
spectraVariables	For backendInitialize: character with the names of the spectra variables that are provided by the extending backend. For selectSpectraVariables: character specifying the spectra variables to keep.
...	ignored
x	A MsBackendCached object.
columns	For spectraData: character with the names of the spectra variables to retrieve.
value	replacement value for <- methods. See individual method description or expected data type.
i	For [: integer with the indices to subset the object.
j	For [: ignored.
drop	For [: not considered.
name	For $<-: the name of the spectra variable to set.
use.names	For lengths: whether spectrum names should be used.

Value

See documentation of respective function.

Implementation notes

Classes extending the MsBackendCached need to

- call the backendInitialize method of this class in their own backendInitialize method and set at least the number of spectra with the nspectra parameter and the spectraVariables that are available to the (extending) backend class.

- implement the spectraData method that also calls the spectraData method from MsBackendCached to also retrieve cached values (e.g. using res <- callNextMethod() at the beginning of the spectraData function). The spectraData,MsBackendCached method will return NULL if the selected spectra variables were not cached and are not *core spectra variables* not being provided by the extending backend. Thus, the extending backend can then proceed to retrieve the respective values from its own backend/data storage.
- implement eventually the [method that calls in addition the [from the MsBackendCached.

All other methods accessing or setting spectra variables don't need to be implemented by the extending backend class (the default implementations of the MsBackendCached will then be used instead; these ensure that cached values are returned first). Spectra variables can be modified or added using the $<- method of the MsBackendCached. Replacing or adding multiple variables using the spectraData<- is not supported by MsBackendCached. The extending backend might however implement such a method that internally uses $<- to add/replace single variables.

The MsBackendCached has the following slots:

- nspectra: integer(1) defining the number of spectra of the backend. This variable needs to be set and must match the number of rows of localData and the actual number of spectra in the (extending) backend.
- localData: data.frame with the cached local data. Any replacement operation with $<- will set/add a column with the respective values.
- spectraVariables: character defining the spectra variables that are provided by the extending MsBackend class (e.g. all spectra variables that can be retrieved from the data base or original data files).

Available methods

- acquisitionNum: returns the acquisition number of each spectrum. Returns an integer of length equal to the number of spectra (with NA_integer_ if not available).
- backendInitialize: *initializes* the backend. The method takes parameters data (data.frame with cached data), nspectra (integer defining the number of spectra) and spectraVariables (character with the spectra variables that are provided by the extending backend.
- centroided, centroided<-: gets or sets the centroiding information of the spectra. centroided returns a logical vector of length equal to the number of spectra with TRUE if a spectrum is centroided, FALSE if it is in profile mode and NA if it is undefined. See also isCentroided for estimating from the spectrum data whether the spectrum is centroided. value for centroided<- is either a single logical or a logical of length equal to the number of spectra in object.
- collisionEnergy, collisionEnergy<-: gets or sets the collision energy for all spectra in object. collisionEnergy returns a numeric with length equal to the number of spectra (NA_real_ if not present/defined), collisionEnergy<- takes a numeric of length equal to the number of spectra in object.
- dataOrigin: gets a character of length equal to the number of spectra in object with the *data origin* of each spectrum. This could e.g. be the mzML file from which the data was read.
- intensity: gets the intensity values from the spectra. Returns a NumericList() of numeric vectors (intensity values for each spectrum). The length of the list is equal to the number of spectra in object.

- ionCount: returns a numeric with the sum of intensities for each spectrum. If the spectrum is empty (see isEmpty), NA_real_ is returned.
- isEmpty: checks whether a spectrum in object is empty (i.e. does not contain any peaks). Returns a logical vector of length equal number of spectra.
- isolationWindowLowerMz, isolationWindowLowerMz<-: gets or sets the lower m/z boundary of the isolation window.
- isolationWindowTargetMz, isolationWindowTargetMz<-: gets or sets the target m/z of the isolation window.
- isolationWindowUpperMz, isolationWindowUpperMz<-: gets or sets the upper m/z boundary of the isolation window.
- length: returns the number of spectra (i.e. the @nspectra).
- lengths: gets the number of peaks (m/z-intensity values) per spectrum. Returns an integer vector (length equal to the number of spectra). For empty spectra, 0 is returned.
- msLevel: gets the spectra's MS level. Returns an integer vector (of length equal to the number of spectra) with the MS level for each spectrum (or NA_integer_ if not available).
- mz: gets the mass-to-charge ratios (m/z) from the spectra. Returns a NumericList() or length equal to the number of spectra, each element a numeric vector with the m/z values of one spectrum.
- polarity, polarity<-: gets or sets the polarity for each spectrum. polarity returns an integer vector (length equal to the number of spectra), with 0 and 1 representing negative and positive polarities, respectively. polarity<- expects an integer vector of length 1 or equal to the number of spectra.
- precursorCharge, precursorIntensity, precursorMz, precScanNum, precAcquisitionNum: get the charge (integer), intensity (numeric), m/z (numeric), scan index (integer) and acquisition number (interger) of the precursor for MS level 2 and above spectra from the object. Returns a vector of length equal to the number of spectra in object. NA are reported for MS1 spectra of if no precursor information is available.
- rtime, rtime<-: gets or sets the retention times for each spectrum (in seconds). rtime returns a numeric vector (length equal to the number of spectra) with the retention time for each spectrum. rtime<- expects a numeric vector with length equal to the number of spectra.
- scanIndex: returns an integer vector with the *scan index* for each spectrum. This represents the relative index of the spectrum within each file. Note that this can be different to the acquisitionNum of the spectrum which is the index of the spectrum as reported in the mzML file.
- selectSpectraVariables: subset the object to specified spectra variables. This will eventually remove spectra variables listed in @spectraVariables and will also drop columns from the local cache if not among spectraVariables.
- smoothed,smoothed<-: gets or sets whether a spectrum is *smoothed*. smoothed returns a logical vector of length equal to the number of spectra. smoothed<- takes a logical vector of length 1 or equal to the number of spectra in object.
- spectraVariables: returns the available spectra variables, i.e. the unique set of *core spectra variables*, cached spectra variables and spectra variables defined in the @spectraVariables slot (i.e. spectra variables thought to be provided by the extending MsBackend instance).

- `spectraData`: returns a `DataFrame` with cached spectra variablers or initialized *core spectra variables*. Parameter `spectraVariables` allows to specify the variables to retrieve. The function returns `NULL` if the requested variables are not cached and are not provided by the extending backend. Note that this method **only** returns cached spectra variables or core spectra variables **not** provided by the extending backend. It is the responsibility of the extending backend to add/provide these.
- `[`: subsets the cached data. Parameter `i` needs to be an `integer` vector.
- `$`, `$<-`: access or set/add a single spectrum variable (column) in the backend.

Author(s)

Johannes Rainer

See Also

MsBackend for the documentation of MS backends.

`neutralLoss` *Calculate Neutral Loss Spectra*

Description

This help page lists functions that convert MS/MS spectra to neutral loss spectra. The main function for this is `neutralLoss` and the specific algorithm to be used is defined (and configured) with dedicated *parameter* objects (paramer `param` of the `neutralLoss` function).

The parameter objects for the different algorithms are:

- `PrecursorMzParam`: calculates neutral loss spectra as in Aisporna *et al.* 2022 by subtracting the (fragment's) peak m/z value from the precursor m/z value of each spectrum (precursor m/z - fragment m/z). Parameter `msLevel` allows to restrict calculation of neutral loss spectra to specified MS level(s). Spectra from other MS level(s) are returned as-is. Parameter `filterPeaks` allows to remove certain peaks from the neutral loss spectra. By default (`filterPeaks = "none"`) no filtering takes place. With `filterPeaks = "removePrecursor"` all fragment peaks with an m/z value matching the precursor m/z (considering also `ppm` and `tolerance` are removed. With `filterPeaks = "abovePrecursor"`, all fragment peaks with an m/z larger than the precursor m/z (m/z > precursor m/z - `tolerance` - `ppm` of the precursor m/z) are removed (thus removing also in most cases the fragment peaks representing the precursor). Finally, with `filterPeaks = "belowPrecursor"` all fragment peaks with an m/z smaller than the precursor m/z (m/z < precursor m/z + `tolerance` + `ppm` of the precursor m/z) are removed. Also in this case the precursor fragment peak is (depending on the values of `ppm` and `tolerance`) removed.

Usage

```
neutralLoss(object, param, ...)

PrecursorMzParam(
  filterPeaks = c("none", "abovePrecursor", "belowPrecursor", "removePrecursor"),
  msLevel = c(2L, NA_integer_),
  ppm = 10,
  tolerance = 0
)

## S4 method for signature 'Spectra,PrecursorMzParam'
neutralLoss(object, param, ...)
```

Arguments

object	Spectra() object with the fragment spectra for which neutral loss spectra should be calculated.
param	One of the *parameter* objects discussed below.
...	Currently ignored.
filterPeaks	For PrecursorMzParam: character(1) or function defining if and how fragment peaks should be filtered before calculation. Pre-defined options are: "none" (keep all peaks), "abovePrecursor" (removes all fragment peaks with an m/z >= precursor m/z), "belowPrecursor" (removes all fragment peaks with an m/z <= precursor m/z). In addition, it is possible to pass a custom function with this parameter with arguments x (two column peak matrix) and precursorMz (the precursor m/z) that returns the sub-setted two column peak matrix.
msLevel	integer defining for which MS level(s) the neutral loss spectra should be calculated. Defaults to msLevel = c(2L, NA) thus, neutral loss spectra will be calculated for all spectra with MS level equal to 2 or with missing/undefined MS level. All spectra with a MS level different than msLevel will be returned unchanged.
ppm	numeric(1) with m/z-relative acceptable difference in m/z values to filter peaks. Defaults to ppm = 10. See function description for details.
tolerance	numeric(1) with absolute acceptable difference in m/z values to filter peaks. Defaults to tolerance = 0. See function description for details.

Value

A Spectra() object with calculated neutral loss spectra.

Note

By definition, mass peaks in a Spectra object need to be ordered by their m/z value (in increasing order). Thus, the order of the peaks in the calculated neutral loss spectra might not be the same than in the original Spectra object.

Note also that for spectra with a missing precursor m/z empty spectra are returned (i.e. spectra without peaks) since it is not possible to calcualte the neutral loss spectra.

Author(s)

Johannes Rainer

References

Aisporna A, Benton PH, Chen A, Derks RJE, Galano JM, Giera M and Siuzdak G (2022). Neutral Loss Mass Spectral Data Enhances Molecular Similarity Analysis in METLIN. Journal of the American Society for Mass Spectrometry. doi:10.1021/jasms.1c00343

Examples

```
## Create a simple example Spectra object with some MS1, MS2 and MS3 spectra.
DF <- DataFrame(msLevel = c(1L, 2L, 3L, 1L, 2L, 3L),
                precursorMz = c(NA, 40, 20, NA, 300, 200))
DF$mz <- IRanges::NumericList(
                      c(3, 12, 14, 15, 16, 200),
                      c(13, 23, 39, 86),
                      c(5, 7, 20, 34, 50),
                      c(5, 7, 9, 20, 100),
                      c(15, 53, 299, 300),
                      c(34, 56, 100, 200, 204, 309)
                  , compress = FALSE)
DF$intensity <- IRanges::NumericList(1:6, 1:4, 1:5, 1:5, 1:4, 1:6,
                                     compress = FALSE)
sps <- Spectra(DF, backend = MsBackendDataFrame())

## Calculate neutral loss spectra for all MS2 spectra, keeping MS1 and MS3
## spectra unchanged.
sps_nl <- neutralLoss(sps, PrecursorMzParam(msLevel = 2L))
mz(sps)
mz(sps_nl)

## Calculate neutral loss spectra for MS2 and MS3 spectra, removing peaks
## with an m/z >= precursorMz
sps_nl <- neutralLoss(sps, PrecursorMzParam(
    filterPeaks = "abovePrecursor", msLevel = 2:3))
mz(sps_nl)
## This removed also the peak with m/z 39 from the second spectrum

## Removing all fragment peaks matching the precursor m/z with a tolerance
## of 1 and ppm 10
sps_nl <- neutralLoss(sps, PrecursorMzParam(
    filterPeaks = "removePrecursor", tolerance = 1, ppm = 10, msLevel = 2:3))
mz(sps_nl)

## Empty spectra are returned for MS 2 spectra with undefined precursor m/z.
sps$precursorMz <- NA_real_
sps_nl <- neutralLoss(sps, PrecursorMzParam())
mz(sps_nl)
```

plotMzDelta *MZ delta Quality Control*

Description

The M/Z delta plot illustrates the suitability of MS2 spectra for identification by plotting the M/Z differences of the most intense peaks. The resulting histogram should optimally show modes at amino acid residu masses. The plots have been described in Foster et al. 2011.

Only a certain percentage of most intense MS2 peaks are taken into account to use the most significant signal. Default value is 20% (see `percentage` argument). The difference between peaks is then computed for all individual spectra and their distribution is plotted as a histogram. Delta M/Z between 40 and 200 are plotted by default, to encompass the residue masses of all amino acids and several common contaminants, although this can be changes with the `mzRange` argument.

In addition to the processing described above, isobaric reporter tag peaks and the precursor peak can also be removed from the MS2 spectrum, to avoid interence with the fragment peaks.

Note that figures in Foster et al. 2011 have been produced and optimised for centroided data. While running the function on profile mode is likely fine, it is recommended to use centroided data.

A `ggplot2` based function called `ggMzDeltaPlot()` to visualise the M/Z delta distributions is available at https://gist.github.com/lgatto/c72b1ff5a4116118dbb34d9d2bc3470a.

Usage

```
computeMzDeltas(
  object,
  percentage = 0.2,
  mzRange = c(40, 200),
  BPPARAM = BiocParallel::bpparam()
)

plotMzDelta(x, aaLabels = TRUE)
```

Arguments

`object`	An instance of class `Spectra()`.
`percentage`	`numeric(1)` between 0 and 1 indicating the percentage of the most intense peaks in each MS2 spectrum to include in the calculation. Default is 0.2.
`mzRange`	`numeric(2)` with the upper and lower M/Z to be used to the MZ deltas. Default is `c(40, 200)`.
`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation. Default is to use `BiocParallel::bpparam()`. See `?BiocParallel::bpparam` for details.
`x`	A list of M/Z delta values, as returned by `computeMzDeltas()`.
`aaLabels`	`logical(1)` defining whether the amino acids should be labelled on the histogram. Default is `TRUE`.

Value

`computeMzDeltas()` returns a `list` of numeric vectors. `plotMzDelta()` is used to visualise of M/Z delta distributions.

Author(s)

Laurent Gatto with contributions (to MSnbase) of Guangchuang Yu.

References

Foster JM, Degroeve S, Gatto L, Visser M, Wang R, Griss J, et al. A posteriori quality control for the curation and reuse of public proteomics data. Proteomics. 2011;11: 2182-2194. http://dx.doi.org/10.1002/pmic.201000602

Examples

```
library(msdata)
f <- proteomics(pattern = "TMT.+20141210.mzML.gz", full.names = TRUE)
sp <- Spectra(f)

d <- computeMzDeltas(sp[1:1000])
plotMzDelta(d)
```

`spectra-plotting` *Plotting Spectra*

Description

`Spectra()` can be plotted with one of the following functions

- `plotSpectra`: plots each spectrum in its separate plot by splitting the plot area into as many panels as there are spectra.
- `plotSpectraOverlay`: plots all spectra in `x` **into the same** plot (as an overlay).
- `plotSpectraMirror`: plots a pair of spectra as a *mirror plot*. Parameters `x` and `y` both have to be a `Spectra` of length 1. Matching peaks (considering `ppm` and `tolerance`) are highlighted. See `common()` for details on peak matching. Parameters `matchCol`, `matchLty`, `matchLwd` and `matchPch` allow to customize how matching peaks are indicated.

Usage

```
plotSpectra(
  x,
  xlab = "m/z",
  ylab = "intensity",
  type = "h",
  xlim = numeric(),
  ylim = numeric(),
```

```
  main = character(),
  col = "#00000080",
  labels = character(),
  labelCex = 1,
  labelSrt = 0,
  labelAdj = NULL,
  labelPos = NULL,
  labelOffset = 0.5,
  labelCol = "#00000080",
  asp = 1,
  ...
)

plotSpectraOverlay(
  x,
  xlab = "m/z",
  ylab = "intensity",
  type = "h",
  xlim = numeric(),
  ylim = numeric(),
  main = paste(length(x), "spectra"),
  col = "#00000080",
  labels = character(),
  labelCex = 1,
  labelSrt = 0,
  labelAdj = NULL,
  labelPos = NULL,
  labelOffset = 0.5,
  labelCol = "#00000080",
  axes = TRUE,
  frame.plot = axes,
  ...
)

## S4 method for signature 'Spectra'
plotSpectraMirror(
  x,
  y,
  xlab = "m/z",
  ylab = "intensity",
  type = "h",
  xlim = numeric(),
  ylim = numeric(),
  main = character(),
  col = "#00000080",
  labels = character(),
  labelCex = 1,
  labelSrt = 0,
```

```
  labelAdj = NULL,
  labelPos = NULL,
  labelOffset = 0.5,
  labelCol = "#00000080",
  axes = TRUE,
  frame.plot = axes,
  ppm = 20,
  tolerance = 0,
  matchCol = "#80B1D3",
  matchLwd = 1,
  matchLty = 1,
  matchPch = 16,
  ...
)
```

Arguments

x	a Spectra() object. For plotSpectraMirror it has to be an object of length 2.
xlab	character(1) with the label for the x-axis (by default xlab = "m/z").
ylab	character(1) with the label for the y-axis (by default ylab = "intensity").
type	character(1) specifying the type of plot. See plot.default() for details. Defaults to type = "h" which draws each peak as a line.
xlim	numeric(2) defining the x-axis limits. The range of m/z values are used by default.
ylim	numeric(2) defining the y-axis limits. The range of intensity values are used by default.
main	character(1) with the title for the plot. By default the spectrum's MS level and retention time (in seconds) is used.
col	color to be used to draw the peaks. Should be either of length 1, or equal to the number of spectra (to plot each spectrum in a different color) or be a list with colors for each individual peak in each spectrum.
labels	allows to specify a label for each peak. Can be a character with length equal to the number of peaks, or, ideally, a function that uses one of the Spectra's variables (see examples below). plotSpectraMirror supports only labels of type *function*.
labelCex	numeric(1) giving the amount by which the text should be magnified relative to the default. See parameter cex in par().
labelSrt	numeric(1) defining the rotation of the label. See parameter srt in text().
labelAdj	see parameter adj in text().
labelPos	see parameter pos in text().
labelOffset	see parameter offset in text().
labelCol	color for the label(s).
asp	for plotSpectra: the target ratio (columns / rows) when plotting mutliple spectra (e.g. for 20 spectra use asp = 4/5 for 4 columns and 5 rows or asp = 5/4 for 5 columns and 4 rows; see grDevices::n2mfrow() for details).

...	additional parameters to be passed to the plot.default() function.
axes	logical(1) whether (x and y) axes should be drawn.
frame.plot	logical(1) whether a box should be drawn around the plotting area.
y	for plotSpectraMirror: Spectra object of length 1 against which x should be plotted against.
ppm	for plotSpectraMirror: m/z relative acceptable difference (in ppm) for peaks to be considered matching (see common() for more details).
tolerance	for plotSpectraMirror: absolute acceptable difference of m/z values for peaks to be considered matching (see common() for more details).
matchCol	for plotSpectraMirror: color for matching peaks.
matchLwd	for plotSpectraMirror: line width (lwd) to draw matching peaks. See par() for more details.
matchLty	for plotSpectraMirror: line type (lty) to draw matching peaks. See par() for more details.
matchPch	for plotSpectraMirror: point character (pch) to label matching peaks. Defaults to matchPch = 16, set to matchPch = NA to disable. See par() for more details.

Value

These functions create a plot.

Author(s)

Johannes Rainer, Sebastian Gibb, Laurent Gatto

Examples

```
ints <- list(c(4.3412, 12, 8, 34, 23.4),
    c(8, 25, 16, 32))
mzs <- list(c(13.453421, 43.433122, 46.6653553, 129.111212, 322.24432),
    c(13.452, 43.5122, 129.112, 322.245))

df <- DataFrame(msLevel = c(1L, 1L), rtime = c(123.12, 124))
df$mz <- mzs
df$intensity <- ints
sp <- Spectra(df)

#### ------------------------------------------- ####
##                  plotSpectra                    ##

## Plot one spectrum.
plotSpectra(sp[1])

## Plot both spectra.
plotSpectra(sp)

## Define a color for each peak in each spectrum.
```

```
plotSpectra(sp, col = list(c(1, 2, 3, 4, 5), 1:4))

## Color peaks from each spectrum in different colors.
plotSpectra(sp, col = c("green", "blue"))

## Label each peak with its m/z.
plotSpectra(sp, labels = function(z) format(unlist(mz(z)), digits = 4))

## Rotate the labels.
plotSpectra(sp, labels = function(z) format(unlist(mz(z)), digits = 4),
    labelPos = 2, labelOffset = 0.1, labelSrt = -30)

## Add a custom annotation for each peak.
sp$label <- list(c("", "A", "B", "C", "D"),
    c("Frodo", "Bilbo", "Peregrin", "Samwise"))
## Plot each peak in a different color
plotSpectra(sp, labels = function(z) unlist(z$label),
    col = list(1:5, 1:4))

## Plot a single spectrum specifying the label.
plotSpectra(sp[2], labels = c("A", "B", "C", "D"))


#### ------------------------------------------- ####
##                plotSpectraOverlay                ##

## Plot both spectra overlaying.
plotSpectraOverlay(sp)

## Use a different color for each spectrum.
plotSpectraOverlay(sp, col = c("#ff000080", "#0000ff80"))

## Label also the peaks with their m/z if their intensity is above 15.
plotSpectraOverlay(sp, col = c("#ff000080", "#0000ff80"),
    labels = function(z) {
        lbls <- format(mz(z)[[1L]], digits = 4)
        lbls[intensity(z)[[1L]] <= 15] <- ""
        lbls
    })
abline(h = 15, lty = 2)

## Use different asp values
plotSpectra(sp, asp = 1/2)
plotSpectra(sp, asp = 2/1)

#### ------------------------------------------- ####
##                plotSpectraMirror                 ##

## Plot two spectra against each other.
plotSpectraMirror(sp[1], sp[2])

## Label the peaks with their m/z
plotSpectraMirror(sp[1], sp[2],
```

```
    labels = function(z) format(mz(z)[[1L]], digits = 3),
    labelSrt = -30, labelPos = 2, labelOffset = 0.2)
grid()

## The same plot with a tolerance of 0.1 and using a different color to
## highlight matching peaks
plotSpectraMirror(sp[1], sp[2],
    labels = function(z) format(mz(z)[[1L]], digits = 3),
    labelSrt = -30, labelPos = 2, labelOffset = 0.2, tolerance = 0.1,
    matchCol = "#ff000080", matchLwd = 2)
grid()
```

`spectraVariableMapping` *Mapping between spectra variables and data file fields*

Description

The `spectraVariableMapping` function provides the mapping between *spectra variables* of a `Spectra()` object with data fields from a data file. Such name mapping is expected to enable an easier import of data files with specific *dialects*, e.g. files in MGF format that use a different naming convention for core spectra variables.

`MsBackend()` implementations are expected to implement this function (if needed) to enable import of data from file formats with non-standardized data fields.

Usage

```
spectraVariableMapping(object, ...)
```

Arguments

`object`	An instance of an object extending `MsBackend()`.
`...`	Optional parameters.

Value

A named `character` with names being spectra variable names (use `spectraVariables()` for a list of supported names) and values being the data field names.

Author(s)

Johannes Rainer

Index