Package: ScanCentricPeakCharacterization (via r-universe)

September 12, 2024

Title Functionality for Characterizing Peaks in Mass Spectrometry in a Scan-Centric Manner

Version 0.3.65

Date 2024-03-28

Description Provides a functions and classes for detecting, characterizing, and integrating peaks in a scan-centric manner from direct-injection mass spectrometry data.

Depends R (>= 3.5.0)

Imports cowplot, dplyr, readxl, MSnbase, debugme, assertthat, jsonlite, purrr, R6, tibble, XML, forcats, pracma, R.utils, IRanges, S4Vectors, magrittr, tidyr, lubridate, stats, stringr, digest, knitrProgressBar

License file LICENSE

LazyData true

RoxygenNote 7.2.1

Suggests knitr, rmarkdown, testthat (>= 3.0.0), furrr, ggplot2, patchwork, ggforce, logger

Encoding UTF-8

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

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

RemoteUrl https://github.com/MoseleyBioinformaticsLab/ScanCentricPeakCharacterization

RemoteRef *release

RemoteSha 2373d738beb5fb8a085a6beccb4d22873d860458

Contents

.get_scan_polarity . . . 3
.remove_attrs . . . 4
.to_data_frame . . . 4
add_scan_info . . . 4
add_to_zip . . . 5
area_sum_points . . . 6
characterize_peaks . . . 6
check_frequency_r2 . . . 7
check_ranges_convert_to_regions . . . 7
check_zip_file . . . 8
choose_frequency_model_builtin . . . 8
convert_mz_frequency . . . 9
correct_mean . . . 10
correct_peak . . . 10
correct_peak_sd_height . . . 11
correct_variance . . . 12
create_frequency_regions . . . 12
create_value . . . 13
default_correct_offset_function . . . 14
default_offset_predict_function . . . 14
disable_logging . . . 15
enable_logging . . . 15
extract . . . 16
extract_raw_method . . . 16
filter_scans_builtin . . . 17
find_signal_regions . . . 17
frequency_points_to_frequency_regions . . . 18
get_fitted_peak_info . . . 19
get_mzml_header . . . 19
get_mzml_metadata . . . 20
get_raw_ms_metadata . . . 20
import_json . . . 21
import_sc_mzml . . . 21
indicate_standards_contaminents . . . 22
initialize_metadata_from_mzml . . . 23
initialize_zip_metadata . . . 23
integrate_model . . . 24
integrate_sides . . . 24
integration_based_area . . . 25
json_2_peak_list . . . 25
json_mzML_2_df . . . 26
lists_2_json . . . 26
load_metadata . . . 27
loess_to_df . . . 27
log_memory . . . 28
log_message . . . 28

log_with_min . . . 28
meta_export_json . . . 29
model_peak_center_intensity . . . 29
mzml_to_zip . . . 30
mz_frequency_interpolation . . . 31
mz_scans_to_frequency . . . 31
parabolic_fit . . . 32
peak_list_2_json . . . 33
predicted_frequency_r2 . . . 33
raw_metadata_mzml . . . 34
recalculate_offsets . . . 34
run_mzml_list . . . 35
sample_run_time . . . 35
ScanCentricPeakCharacterization . . . 36
SCCharacterizePeaks . . . 36
SCMzml . . . 41
SCPeakRegionFinder . . . 45
SCPeakRegions . . . 49
SCZip . . . 51
sc_zip . . . 54
set_internal_map . . . 55
show_progress . . . 55
single_pass_normalization . . . 56
split_region_by_peaks . . . 56
ssr . . . 57
transform_residuals . . . 58
zip_list_contents . . . 58

Index **59**

`.get_scan_polarity` *get_scan_mode*

Description

takes a list from xmlToList for "run" and looks at whether all scans are positive, negative, or mixed

Usage

```
.get_scan_polarity(spectrum_list)
```

Arguments

`spectrum_list`	the list of spectra

.remove_attrs *remove attributes*

Description

removes a list entry called ".attrs" from a list, and makes them first level partners

Usage

```
.remove_attrs(in_list)
```

Arguments

in_list	the list to work on

.to_data_frame *transform to data frame*

Description

transform to data frame

Usage

```
.to_data_frame(in_list)
```

Arguments

in_list	the list of xml nodes to work on

add_scan_info *add scan level info*

Description

add scan level info

Usage

```
add_scan_info(mzml_data)
```

Arguments

mzml_data	the MSnbase mzml data object

Details

returns a data.frame with:

- `scanIndex`: the indices of the scans
- `scan`: the number of the scan by number. This will be used to name scans.
- `polarity`: +1 or -1 depending on if the scan is positive or negative
- `rtime`: the retention time or injection time of the scan for for direct-injection data
- `tic`: the total intensity of the scan
- `rtime_lag`: how long between this scan and previous scan
- `rtime_lead`: how long between this scan and next scan

After running `predict_frequency()`, the following fields are added from the information returned from frequency conversion:

- `mad`: mean absolute deviation of residuals
- `frequency model coefficients`: the coefficients from the fit frequency, named whatever you named them
- `mz model coefficients`: similar, but for the m/z model

Value

data.frame, see **Details**

`add_to_zip` *add to zip file*

Description

given an output object, filename, and zip file, write the output object to the file, and then add to the zip file

Usage

```
add_to_zip(object, filename, zip_file)
```

Arguments

`object`	the object to write
`filename`	the file that it should be
`zip_file`	the zip file to add to

Details

a directory created by `tempdir` is used to hold the file, which is then added to the zip file.

area_sum_points *area sum*

Description

calculate the area based on summing the points

Usage

```
area_sum_points(peak_mz, peak_intensity, zero_value = 0)
```

Arguments

peak_mz	the mz in the peak
peak_intensity	the peak intensities
zero_value	what value actually represents zero

Value

numeric

characterize_peaks *characterize peaks from points and picked peaks*

Description

characterize peaks from points and picked peaks

Usage

```
characterize_peaks(peak_region, calculate_peak_area = FALSE)
```

Arguments

peak_region	the PeakRegion object to work on

Value

list

check_frequency_r2 *check r2*

Description

check r2

Usage

```
check_frequency_r2(mz_frequency_list)
```

Arguments

mz_frequency_list	the list of predicted frequency data.frames

check_ranges_convert_to_regions *frequency points to frequency regions*

Description

Given M/Z point data in a data.frame, create IRanges based point "regions" of width 1, using the `frequency_multiplier` argument to convert from the floating point double to an `integer`.

Usage

```
check_ranges_convert_to_regions(frequency_list, frequency_multiplier = 400)
```

Arguments

frequency_list	a list of with a `data.frame` containing `frequency`
frequency_multiplier	a value used to convert to integers.

check_zip_file *check zip file*

Description

checks that the zip file has the basic contents it should have, and that files listed in the metadata actually exist.

Usage

```
check_zip_file(zip_dir)
```

Arguments

zip_dir	the directory of the unzipped data

choose_frequency_model_builtin *default single model*

Description

default single model

Usage

```
choose_frequency_model_builtin(sc_mzml)
```

Arguments

sc_mzml	the sc_mzml object

Details

This is the default function to choose a single frequency and mz model. It takes the scan_info after filtering scans, and calculates the median of the square root terms, and chooses the one closest to the median value.

Please examine this function and write your own if needed. You can view the function definition using choose_frequency_model_builtin

Value

SCmzml

`convert_mz_frequency` *Convert M/Z to Frequency*

Description

Given a data.frame of m/z, generate frequency values for the data.

Usage

```
convert_mz_frequency(mz_data, keep_all = TRUE)
```

Arguments

`mz_data`	a data.frame with `mz`
`keep_all`	keep **all** the variables generated, or just the original + **frequency**?

Details

The **M/Z** values from FTMS data do not have constant spacing between them. This produces challenges in working with ranged intervals and windows. The solution for FTMS data then is to convert them to **frequency** space. This is done by:

- taking subsequent M/Z points
- averaging their M/Z
- taking the difference to get an `offset` value
- dividing averaged M/Z by offset to generate **frequency**
- taking subsequent differences of frequency points
- keep points with a difference in the supplied range as valid for modeling

After deciding on the valid points for modeling, each point gets an interpolated frequency value using the two averaged points to the left and right in M/Z.

Value

list

See Also

mz_scans_to_frequency

correct_mean *correct peak mean*

Description

Given a corrected SD, corrects the mean assuming that it is the result of a truncated normal distribution.

Usage

```
correct_mean(observed_mean, corrected_sd, fraction)
```

Arguments

observed_mean	the observed mean
corrected_sd	a corrected sd, generated by correct_variance
fraction	the fraction of total observations

Value

corrected mean

References

https://en.wikipedia.org/wiki/Truncated_normal_distribution

See Also

correct_peak() correct_variance()

correct_peak *correct peak sd and mean*

Description

Assuming that an observed mean (intensity) and sd are from a truncated normal distribution that is truncated on one side only.

Usage

```
correct_peak(observed_mean, observed_sd, n_observed, n_should_observe)
```

Arguments

observed_mean	the observed mean
observed_sd	the observed sd
n_observed	how many observations went into this mean
n_should_observe	how many observations *should* there have been?

Value

data.frame, with corrected mean and sd

References

https://en.wikipedia.org/wiki/Truncated_normal_distribution

See Also

correct_mean() correct_variance()

---

correct_peak_sd_height

correct peak height and sd

Description

correct peak height and sd

Usage

```
correct_peak_sd_height(
  original_height,
  list_of_heights,
  n_observed,
  n_should_observe
)
```

Arguments

original_height	the original height estimate to correct
list_of_heights	the set of peak heights
n_observed	how many were observed
n_should_observe	how many should have been observed

Value

data.frame

`correct_variance` *correct peak variance*

Description

Given a variance observed from a truncated normal distribution, correct it assuming that it should have had 100% observaionts

Usage

```
correct_variance(observed_variance, fraction)
```

Arguments

`observed_variance`	the observed **variance**
`fraction`	what fraction was it observed in

Value

corrected variance

References

https://en.wikipedia.org/wiki/Truncated_normal_distribution

See Also

`correct_mean()` `correct_peak()`

`create_frequency_regions` *create frequency regions*

Description

Given a point-point spacing and a frequency range, create IRanges based regions of specified width. Overlapping sliding regions can be creating by specifying a `region_size` bigger than `delta`, adjacent tiled regions can be created by specifying a `region_size == delta`.

Usage

```
create_frequency_regions(
  point_spacing = 0.5,
  frequency_range = NULL,
  n_point = 10,
  delta_point = 1,
  multiplier = 500
)
```

Arguments

point_spacing	how far away are subsequent points.
frequency_range	the range of frequency to use
n_point	how many points you want to cover
delta_point	the *step* size between the beginning of each subsequent region
multiplier	multiplier to convert from frequency to integer space

Details

For Fourier-transform mass spec, points are equally spaced in frequency space, which will lead to unequal spacing in M/Z space. Therefore, we create regions using the point-point differences in frequency space.

What will be returned is an `IRanges` object, where the widths are constantly increasing over M/Z space.

Value

IRanges

---

create_value *create a value from mantissa and exponent*

Description

given a mantissa and exponent, returns the actual value as a numeric

Usage

```
create_value(mantissa, exponent)
```

Arguments

mantissa	the base part of the number
exponent	the exponent part

Value

numeric

`default_correct_offset_function` *correct offsets*

Description

Given a MasterPeakList object and the MultiScansPeakList that generated it, correct the m/z values using offset predictions

Usage

```
default_correct_offset_function(
  master_peak_list,
  multi_scan_peaklist,
  min_scan = 0.1
)
```

Arguments

`master_peak_list`	the MasterPeakList object of correspondent peaks
`multi_scan_peaklist`	the MultiScansPeakList to be corrected
`min_scan`	what is the minimum number of scans a peak should be in to be used for correction.

Value

list

`default_offset_predict_function` *offset predict function*

Description

The offset predictor using loess

Usage

```
default_offset_predict_function(model, x)
```

Arguments

model	the model to use
x	the new values

Value

numeric

---

disable_logging *turn logging off*

Description

There may be good reasons to turn the logging off after it's been turned on. This basically tells the package that the logger isn't available.

Usage

```
disable_logging()
```

---

enable_logging *turn logging on*

Description

Choose to enable logging, to a specific file if desired.

Usage

```
enable_logging(log_file = NULL, memory = FALSE)
```

Arguments

log_file	the file to log to
memory	provide memory logging too? Only available on Linux and MacOS

Details

Uses the logger package under the hood, which is suggested in the dependencies. Having logging enabled is nice to see when things are starting and stopping, and what exactly has been done, without needing to write messages to the console. It is especially useful if you are getting errors, but can't really see them, then you can add "memory" logging to see if you are running out of memory.

Default log file has the pattern:

YYYY.MM.DD.HH.MM.SS_ScanCentricPeakCharacterization_run.log

extract *extract parts of a value*

Description

Often we want to transform a number into it's exponential representation, having the number itself and the number of decimal places. This function provides that functionality

Usage

```
extract(x)
```

Arguments

x	the number to extract the parts from

Value

list

extract_raw_method *extract raw method*

Description

Given a Thermo ".raw" file, attempts to extract the "method" definition from a translated hexdump of the file.

Usage

```
extract_raw_method(in_file, output_type = "data.frame")
```

Arguments

in_file	The Thermo raw file to extract
output_type	string, data.frame or json

Value

string or data.frame

filter_scans_builtin *built in filter scan function*

Description

built in filter scan function

Usage

```
filter_scans_builtin(sc_mzml)
```

Arguments

sc_mzml	the sc_mzml object

Details

This is the built in filtering and removing outliers function. It is based on the Moseley groups normal samples and experience. However, it does not reflect everyone's experience and needs. We expect that others have different use cases and needs, and therefore they should create their own function and use it appropriately.

Please examine this function and write your own as needed. It **must** take an `SCMzml` object, work on the `scan_info` slot, and then create a column with the name "keep" denoting which scans to keep. To view the current definition, you can do `filter_scans_builtin`

Value

SCmzml

find_signal_regions *find signal*

Description

Given some regions and point_regions, find the regions that actually should contain **real** data. See *details* for an explanation of what is considered **real**.

Usage

```
find_signal_regions(
  regions,
  point_regions_list,
  region_percentile = 0.99,
  multiplier = 1.5,
  n_point_region = 2000
)
```

Arguments

regions	the regions we want to query
point_regions_list	the individual points
region_percentile	the cumulative percentile cutoff to use
multiplier	how much above base quantiles to use (default = 1.5)
n_point_region	how many points make up a large segment to do percentile on?

Value

IRanges

---

frequency_points_to_frequency_regions

frequency points to frequency_regions

Description

Given a set of frequency points in a data.frame, create IRanges based point "regions" of width 1, using the multiplier to convert from a floating point double to an integer

Usage

```
frequency_points_to_frequency_regions(
  frequency_data,
  frequency_variable = "frequency",
  multiplier = 400
)
```

Arguments

frequency_data	a data.frame
frequency_variable	which column is the frequency stored in
multiplier	value used to convert to integers

get_fitted_peak_info *fitted peak information*

Description

given the peak, returns the location and intensity

Usage

```
get_fitted_peak_info(
  possible_peak,
  use_loc = "mz",
  w = NULL,
  addend = 1e-08,
  calculate_peak_area = FALSE
)
```

Arguments

possible_peak	data.frame of mz, intensity and log intensity
use_loc	which field to use for locations, default is "mz"
w	the weights to use for the points
addend	how much was added to the peak intensity
calculate_area	should the area of the peak be calculated too?

Value

list

get_mzml_header *extract mzML header*

Description

extract mzML header

Usage

```
get_mzml_header(mzml_file)
```

Arguments

mzml_file	the mzML file to get the header from

get_mzml_metadata *get mzML metadata*

Description

get mzML metadata

Usage

```
get_mzml_metadata(mzml_file)
```

Arguments

mzml_file	the mzML file to get metadata from

get_raw_ms_metadata *get raw ms metadata*

Description

figures out which metadata function to run, and returns back the metadata generated by it.

Usage

```
get_raw_ms_metadata(in_file)
```

Arguments

in_file	the file to use

Value

list

import_json *import json*

Description

import json from a file correctly given some things where things get written differently

Usage

```
import_json(json_file)
```

Arguments

json_file	the json file to read

Value

list

import_sc_mzml *import mzml mass spec data*

Description

function to import mzml mass spec data in a way that provides what we need to work with it. mzml_data should be the *full path* to the data.

Usage

```
import_sc_mzml(mzml_data, ms_level = 1)
```

Arguments

mzml_data	the mzml mass spec file to import
ms_level	which MS-level data to import

Value

MSnbase

`indicate_standards_contaminents` *indicate standards*

Description

Given a directory of characterized samples, attempts to determine which peaks may be standards or contaminants that should be removed after assignment.

Usage

```
indicate_standards_contaminents(
  zip_dir,
  file_pattern = ".zip",
  blank_pattern = "^blank",
  save_dir = NULL,
  conversion_factor = 400,
  progress = TRUE
)
```

Arguments

`zip_dir`	which directories to look for files within
`file_pattern`	what files are we actually using
`blank_pattern`	regex indicating that a sample may be a blank
`save_dir`	where to save the files (default is to overwrite originals)
`conversion_factor`	how much to multiply frequencies by
`progress`	should progress messages be displayed?

Details

For each sample, the scan level frequencies are read in and converted to ranges, and then compared with tiled ranges over the whole frequency range. For those ranges that have 90 to 110% of scan level peaks in ALL blanks, and have 10 to 110% of scan level peaks in at least N-sample - 1, we consider a possible standard or contaminant. The peak is marked so that it can be removed by filtering out it's assignments later.

Value

NULL nothing is returned, files are overwritten

`initialize_metadata_from_mzml` *initialize metadata from mzML*

Description

initialize metadata from mzML

Usage

```
initialize_metadata_from_mzml(zip_dir, mzml_file)
```

Arguments

`zip_dir`	the directory containing unzipped data
`mzml_file`	the mzML file to extract metadata from

`initialize_zip_metadata` *initialize metadata*

Description

initialize metadata

Usage

```
initialize_zip_metadata(zip_dir)
```

Arguments

`zip_dir`	the temp directory that represents the final zip

integrate_model *integrate model*

Description

provides the area integration for the peak that fits the parabolic model

Usage

```
integrate_model(model_mz, model_coeff, n_point = 100, log_transform = "log")
```

Arguments

model_mz	the mz values for the model peak
model_coeff	the model of the peak
n_point	how many points to use for integration
log_transform	what kind of transform was applied

Value

numeric

integrate_sides *integrate sides*

Description

provides ability to calculate the area on the sides of a peak that are not caught by the parabolic model assuming a triangle to each side of the parabola

Usage

```
integrate_sides(peak_mz, peak_int, full_peak_loc, model_peak_loc)
```

Arguments

peak_mz	the mz in the peak
peak_int	the intensity in the peak
full_peak_loc	what defines all of the peak
model_peak_loc	what defined the peak fitting the parabolic model

Value

numeric

integration_based_area *area from integration*

Description

gives the area of the peak based on integrating the model bits and the sides

Usage

```
integration_based_area(
  mz_data,
  int_data,
  full_peak_loc,
  model_peak_loc,
  model_coeff,
  n_point = 100,
  log_transform = "log"
)
```

Arguments

mz_data	peak mz values
int_data	peak intensity values
full_peak_loc	indices defining the full peak
model_peak_loc	indices defining the model peak
model_coeff	the model of the peak
n_point	number of points for integration of the model section
log_transform	which log transformation was used

json_2_peak_list *PeakList from json*

Description

takes json representing a PeakList object, and generates the data.frame version

Usage

```
json_2_peak_list(json_string, in_var = "Peaks")
```

Arguments

json_string	the json to convert
in_var	the top level variable containing the "Peaks"

Value

tbl_df

---

json_mzML_2_df *json mzML to data.frame*

Description

Given a json file or list of lists, return a data.frame with the most important bits of the data.

Usage

```
json_mzML_2_df(in_file)
```

Arguments

in_file	the file to read from

Value

data.frame

---

lists_2_json *lists_2_json*

Description

lists_2_json

Usage

```
lists_2_json(
  lists_to_save,
  zip_file = NULL,
  digits = 8,
  temp_dir = tempfile(pattern = "json")
)
```

Arguments

lists_to_save	the set of lists to create the json from
zip_file	should the JSON files be zipped into a zip file? Provide the zip file name
digits	how many digits to use for the JSON representation
temp_dir	temp directory to write the JSON files to

Value

character

load_metadata *load metadata*

Description

given a zip and a metadata file, load it and return it

Usage

```
load_metadata(zip_dir, metadata_file)
```

Arguments

zip_dir	the directory of the unzipped data
metadata_file	the metadata file

Value

list

loess_to_df *create loess data.frame*

Description

Given a loess model, creates a data.frame suitable for plotting via ggplot2

Usage

```
loess_to_df(loess_model)
```

Arguments

loess_model	the model object generated by loess

Value

data.frame

log_memory *log memory usage*

Description

Logs the amount of memory being used to a log file if it is available, and generating warnings if the amount of RAM hits zero.

Usage

```
log_memory()
```

log_message *log messages*

Description

If a log_appender is available, logs the given message at the `info` level.

Usage

```
log_message(message_string)
```

Arguments

`message_string`	the string to put in the message

log_with_min *log-transform data*

Description

performs a log-transform while adding a small value to the data based on finding the smallest non-zero value in the data

Usage

```
log_with_min(data, min_value = NULL, order_mag = 3, log_fun = log)
```

Arguments

`data`	the data to work with
`min_value`	the minimum value
`order_mag`	how many orders of magnitute smaller should min value be?
`log_fun`	what log function to use for the transformation

Value

matrix

`meta_export_json` *export metadata to json*

Description

export the list metadata to a json string

Usage

```
meta_export_json(meta_list)
```

Arguments

`meta_list`	a list of metadata

`model_peak_center_intensity` *model peak center*

Description

use the derivative of the parabolic equation to find the peak center, and then put the center into the equation to find the intensity at that point.

Usage

```
model_peak_center_intensity(x, coefficients)
```

Arguments

`x`	the x-values to use (non-centered)
`coefficients`	the model coefficients generated from centered model

Details

The coefficients are generated using the linear model:

$$y = a + bx + cx^2$$

.

The derivative of this is:

$$y = b + 2cx$$

The peak of a parabola is defined where y is zero for the derivative.

$$x = -b/2c$$

We can use this to derive where the center of the peak is, and then put the center value back into the equation to get the intensity.

Value

numeric

`mzml_to_zip` *create initial zip from mzML*

Description

given an mzML file, create the initial zip file containing the zipped *mzML*, *metadata.json*, and *mzml_metadata.json*. This zip file is what will be operated on by anything that accesses files, so that our interface is consistent.

Usage

```
mzml_to_zip(mzml_file, out_file)
```

Arguments

`mzml_file`	the mzML file to zip up
`out_file`	the directory to save the zip file

mz_frequency_interpolation *convert mz to frequency using linear fit*

Description

Given a query, and either two values of M/Z and two values of frequency or a previously generated model, return a data.frame with the predicted value, and the slope and the intercept so the model can be re-used later for other points when needed.

Usage

```
mz_frequency_interpolation(
  mz_query,
  mz_values = NULL,
  frequency_values = NULL,
  model = NULL
)
```

Arguments

mz_query	the M/Z value to fit
mz_values	two M/Z values
frequency_values	two frequency values
model	a model to use instead of actual values

Value

data.frame with predicted_value, intercept, and slope

mz_scans_to_frequency *convert mz to frequency across scans*

Description

Given a multi-scan data.frame of m/z, generate frequency values for the data.

Usage

```
mz_scans_to_frequency(
  mz_df_list,
  frequency_fit_description,
  mz_fit_description,
  ...
)
```

Arguments

mz_df_list	a list of data.frame with at least mz and scan columns
frequency_fit_description	the exponentials to use in fitting the frequency ~ mz model
mz_fit_description	the exponentials to use in fitting the mz ~ frequency model
...	other parameters for convert_mz_frequency

Value

list

See Also

convert_mz_frequency

---

parabolic_fit *parabolic fit*

Description

calculates the coefficients of a parabolic fit (y = x + x^2) of x to y

Usage

```
parabolic_fit(x, y, w = NULL)
```

Arguments

x	the x-values, independent
y	the y-values, dependent
w	weights

Value

list

peak_list_2_json *PeakList to json*

Description

takes a PeakList object, and generates a json version

Usage

```
peak_list_2_json(peak_list)
```

Arguments

peak_list	a data.frame or tbl_df to convert

Value

json_string

predicted_frequency_r2 *calculate r2*

Description

calculate r2

Usage

```
predicted_frequency_r2(mz_frequency_df)
```

Arguments

mz_frequency_df	the data.frame with predicted frequencies

raw_metadata_mzml *get mzml metadata*

Description

When raw files are copied, we also generated metadata about their original locations and new locations, and some other useful info. We would like to capture it, and keep it along with the metadata from the mzml file. So, given a list of mzml files, and a location for the raw files, this function creates metadata json files for the mzml files.

Usage

```
raw_metadata_mzml(mzml_files, raw_file_loc, recursive = TRUE)
```

Arguments

mzml_files	the paths to the mzml files
raw_file_loc	the directory holding raw files and json metadata files
recursive	should we go recursively down the directories or not (default = TRUE)

recalculate_offsets *recalculate offsets*

Description

Given a previously generated zip file of characterized peaks, now we've realized that the offsets on each peak should be somehow different. This function takes a zip file, adjusts the offsets, and writes the file back out.

Usage

```
recalculate_offsets(in_zip, offset = 2, out_file = in_zip)
```

Arguments

in_zip	the zip file to work with
offset	the offset to use
out_file	the file to write too (optional)

run_mzml_list *run list of mzML files*

Description

Given a data.frame or character vector of files to run characterization on, processes them in sequence, to a particular saved location.

Usage

```
run_mzml_list(
  mzml_files,
  json_files = NULL,
  progress = TRUE,
  save_loc = ".",
  ...
)
```

Arguments

mzml_files	the list of mzML files to use
json_files	the list of corresponding json meta-data files
progress	whether to give messages about the progress of things
save_loc	where should the file files be saved
...	other parameters for SCCharacterizePeaks

Value

list

sample_run_time *determine sample run time*

Description

determine sample run time

Usage

```
sample_run_time(zip, units = "m")
```

Arguments

zip	the zip object you want to use
units	what units should the run time be in? (s, m, h)

Value

data.frame with sample, start and end time

`ScanCentricPeakCharacterization` *ScanCentricPeakCharacterization: A package for direct injection FT-MS data processing.*

Description

The ScanCentricPeakCharacterization package provides several classes and functions for working with direct injection, high-resolution mass spectrometry data.

`SCCharacterizePeaks` *R6 Class Controlling Peak Characterization*

Description

Peak characterization control

Details

Peak characterization associates data with the `SCZip`, `SCPeakRegionFinder`, and controls their execution.

Public fields

`found_peaks` peaks found by a function

`id` a holder for the ID of the sample

`frequency_fit_description` the model for conversion to frequency

`mz_fit_description` the model for converting back to m/z

`calculate_peak_area` whether to calculate peak area or not

`sc_peak_region_finder` the peak finder object

`sc_zip` the `SCZip` that represents the final file

`in_file` the input file

`metadata_file` the metadata file

`out_file` where everything gets saved

`temp_loc` where intermediates get saved

Methods

Public methods:

- SCCharacterizePeaks$load_file()
- SCCharacterizePeaks$filter_scans()
- SCCharacterizePeaks$choose_frequency_model()
- SCCharacterizePeaks$prepare_mzml_data()
- SCCharacterizePeaks$set_frequency_fit_description()
- SCCharacterizePeaks$set_mz_fit_description()
- SCCharacterizePeaks$generate_filter_scan_function()
- SCCharacterizePeaks$generate_choose_frequency_model_function()
- SCCharacterizePeaks$predict_frequency()
- SCCharacterizePeaks$check_frequency_model()
- SCCharacterizePeaks$get_frequency_data()
- SCCharacterizePeaks$scan_info()
- SCCharacterizePeaks$find_peaks()
- SCCharacterizePeaks$summarize()
- SCCharacterizePeaks$save_peaks()
- SCCharacterizePeaks$write_zip()
- SCCharacterizePeaks$run_all()
- SCCharacterizePeaks$prep_data()
- SCCharacterizePeaks$add_regions()
- SCCharacterizePeaks$run_splitting()
- SCCharacterizePeaks$new()
- SCCharacterizePeaks$clone()

Method `load_file()`**:** Loads the mzml data into the `SCZip`

Usage:

```
SCCharacterizePeaks$load_file()
```

Method `filter_scans()`**:** Filter the scans in data.

Usage:

```
SCCharacterizePeaks$filter_scans()
```

Method `choose_frequency_model()`**:** Choose the single frequency model.

Usage:

```
SCCharacterizePeaks$choose_frequency_model()
```

Method `prepare_mzml_data()`**:** Prepare the mzml data.

Usage:

```
SCCharacterizePeaks$prepare_mzml_data()
```

Method `set_frequency_fit_description()`**:** Set the frequency fit description

Usage:

```
SCCharacterizePeaks$set_frequency_fit_description(frequency_fit_description)
```

Arguments:

`frequency_fit_description` the frequency model description

Method `set_mz_fit_description()`: Set the mz fit description

Usage:

```
SCCharacterizePeaks$set_mz_fit_description(mz_fit_description)
```

Arguments:

`mz_fit_description` the m/z model description

Method `generate_filter_scan_function()`: Sets the scan filtering and check for outlier function.

Usage:

```
SCCharacterizePeaks$generate_filter_scan_function(
  rtime = NA,
  y.freq = NA,
  f_function = NULL
)
```

Arguments:

`rtime` retention time limits of scans to keep

`y.freq` y-frequency coefficient limits of scans to keep (NA)

`f_function` a full function to set as the filtering function

Method `generate_choose_frequency_model_function()`: Sets the function for choosing a single frequency model

Usage:

```
SCCharacterizePeaks$generate_choose_frequency_model_function(f_function = NULL)
```

Arguments:

`f_function` the function for choosing a single model

Method `predict_frequency()`: Run frequency prediction

Usage:

```
SCCharacterizePeaks$predict_frequency()
```

Method `check_frequency_model()`: Check the frequency model

Usage:

```
SCCharacterizePeaks$check_frequency_model()
```

Method `get_frequency_data()`: Get the frequency data from the `SCMzml` bits

Usage:

```
SCCharacterizePeaks$get_frequency_data()
```

Method `scan_info()`: Get the `SCMzml$scan_info` out

Usage:

```
SCCharacterizePeaks$scan_info()
```

Method `find_peaks()`**:** Do the peak characterization without saving

Usage:

```
SCCharacterizePeaks$find_peaks(stop_after_initial_detection = FALSE)
```

Arguments:

`stop_after_initial_detection` should it stop after the initial peak finding

Method `summarize()`**:** Generates the JSON output summary.

Usage:

```
SCCharacterizePeaks$summarize()
```

Method `save_peaks()`**:** Saves the peaks and JSON to the temp file

Usage:

```
SCCharacterizePeaks$save_peaks()
```

Method `write_zip()`**:** Write the zip file

Usage:

```
SCCharacterizePeaks$write_zip()
```

Method `run_all()`**:** Runs all of the pieces for peak characterization in order

Usage:

```
SCCharacterizePeaks$run_all(
  filter_scan_function = NULL,
  choose_frequency_model_function = NULL
)
```

Arguments:

`filter_scan_function` the scan filtering function

`choose_frequency_model_function` the function for choosing a frequency model

Method `prep_data()`**:** Loads and preps the data for characterization

Usage:

```
SCCharacterizePeaks$prep_data()
```

Method `add_regions()`**:** Adds initial regions for finding real peak containing regions

Usage:

```
SCCharacterizePeaks$add_regions()
```

Method `run_splitting()`**:** Does initial region splitting and peak finding in scans

Usage:

```
SCCharacterizePeaks$run_splitting()
```

Method `new()`**:** Creates a new `SCCharacterizePeaks` class

Usage:

```
SCCharacterizePeaks$new(
  in_file,
  metadata_file = NULL,
  out_file = NULL,
  temp_loc = tempfile("scpcms"),
  frequency_fit_description = NULL,
  mz_fit_description = NULL,
  filter_remove_outlier_scans = NULL,
  choose_single_frequency_model = NULL,
  sc_peak_region_finder = NULL,
  calculate_peak_area = FALSE
)
```

Arguments:

`in_file` the mass spec data file to use (required)

`metadata_file` a json metadata file (optional)

`out_file` where to save the final zip container

`temp_loc` a specified temporary location

`frequency_fit_description` mz -> frequency model

`mz_fit_description` frequency -> mz model

`filter_remove_outlier_scans` function for scan filtering

`choose_single_frequency_model` function to choose a single frequency model

`sc_peak_region_finder` a blank `SCPeakRegionFinder` to use instead of the default

`calculate_peak_area` should peak areas be returned as well as height?

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SCCharacterizePeaks$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
  lipid_sample = system.file("extdata",
     "lipid_example.mzML",
     package = "ScanCentricPeakCharacterization")
  sc_char = SCCharacterizePeaks$new(lipid_sample)

  # prep data and check model
  library(ggplot2)
  library(patchwork)
  sc_char$load_file()
  sc_char$generate_filter_scan_function()
  sc_char$generate_choose_frequency_model_function()
  sc_char$prepare_mzml_data()
  sc_char$check_frequency_model()
```

```
  # run characterization
  save_loc = "test.zip"
  sc_char = SCCharacterizePeaks$new(lipid_sample,
                                    out_file = save_loc)
  sc_char$run_all()

## End(Not run)
```

`SCMzml` *R6 Class For mzML Data*

Description

mzML mass spectrometry data container with some useful methods.

Details

Provides our own container for mzML data, and does conversion to frequency, filtering scans, choosing a single frequency regression model, and generating the frequency data for use in the peak characterization.

Public fields

`mzml_file` the mzml file location

`mzml_metadata` metadata from an external json file

`mzml_data` the actual mzml data from MSnbase

`mzml_df_data` a list of data.frames of the data

`scan_range` the range of scans to be used

`rtime_range` the range of retention times to keep

`mz_range` the mz range to use

`scan_info` data.frame of scan information

`remove_zero` should zero intensity data points be removed?

`frequency_fit_description` the model for m/z -> frequency

`mz_fit_description` the model for going from frequency -> m/z

`frequency_coefficients` the coefficients for the frequency model

`mz_coefficients` the coefficients for the m/z model

`ms_level` which MS level will we be using from the mzml file?

`memory_mode` how will the mzml data be worked with to start, inMemory or onDisk?

`difference_range` how wide to consider adjacent frequency points as *good*

`choose_frequency_model_function` where the added model selection function will live

`filter_scan_function` where the added filter scan function will live.

`choose_single_frequency_model` function to choose a single frequency model

Methods

Public methods:

- SCMzml$import_mzml()
- SCMzml$extract_mzml_data()
- SCMzml$predict_frequency()
- SCMzml$convert_to_frequency()
- SCMzml$choose_frequency_model()
- SCMzml$generate_choose_frequency_model_function()
- SCMzml$filter_scans()
- SCMzml$generate_filter_scan_function()
- SCMzml$check_frequency_model()
- SCMzml$get_instrument()
- SCMzml$get_frequency_data()
- SCMzml$new()
- SCMzml$clone()

Method `import_mzml()`**:** import the mzml file defined

Usage:

```
SCMzml$import_mzml(
  mzml_file = self$mzml_file,
  ms_level = self$ms_level,
  memory_mode = self$memory_mode
)
```

Arguments:

`mzml_file` what file are we reading in?

`ms_level` which ms level to import (default is 1)

`memory_mode` use inMemory or onDisk mode

Method `extract_mzml_data()`**:** get the mzml data into data.frame form so we can use it

Usage:

```
SCMzml$extract_mzml_data(remove_zero = self$remove_zero)
```

Arguments:

`remove_zero` whether to remove zero intensity points or not

Method `predict_frequency()`**:** predict frequency and generate some summary information. This does regression of frequency ~ m/z for each scan separately.

Usage:

```
SCMzml$predict_frequency(
  frequency_fit_description = self$frequency_fit_description,
  mz_fit_description = self$mz_fit_description
)
```

Arguments:

`frequency_fit_description` the regression model definition

`mz_fit_description` the regression model definition

Method `convert_to_frequency()`: actually do the conversion of m/z to frequency

Usage:

```
SCMzml$convert_to_frequency()
```

Method `choose_frequency_model()`: choose a frequency model using the previously added function

Usage:

```
SCMzml$choose_frequency_model()
```

Method `generate_choose_frequency_model_function()`: generate a frequency model choosing function and attach it

Usage:

```
SCMzml$generate_choose_frequency_model_function(f_function = NULL)
```

Arguments:

`f_function` the function you want to pass in

Details: Creates a new function that access the `scan_info` slot of an `SCMzml` object after conversion to frequency space, and chooses a single model based on the information there.

Method `filter_scans()`: filter the scans using the previously added function

Usage:

```
SCMzml$filter_scans()
```

Method `generate_filter_scan_function()`: generate a filter function and attach it

Usage:

```
SCMzml$generate_filter_scan_function(
  rtime = NA,
  y.freq = NA,
  f_function = NULL
)
```

Arguments:

`rtime` retention time limits of scans to keep (NA)

`y.freq` y-frequency coefficient limits of scans to keep (NA)

`f_function` a full function to set as the filtering function

Details: Creates a new function that accesses the `scan_info` slot of an `SCMzml` object, filters the scans by their retention-time and y-frequency coefficients, tests for outliers in the y-frequency coefficients, and denotes which scans will be kept for further processing.
`NA` means no filtering will be done, one-sided limits, eg. `(NA, 10)` or `(10, NA)` implies to filter <= or >=, respectively.

Method `check_frequency_model()`: check how well a given frequency model works for this data

Usage:

```
SCMzml$check_frequency_model(scan = NULL, as_list = FALSE)
```

Arguments:

`scan` which scan to show predictions for

`as_list` whether plots should be returned as a single plot or a list of plots

Method `get_instrument()`**:** get instrument data from associated mzml file metadata

Usage:

```
SCMzml$get_instrument()
```

Method `get_frequency_data()`**:** get the frequency data to go into the next steps of analysis.

Usage:

```
SCMzml$get_frequency_data()
```

Method `new()`**:**

Usage:

```
SCMzml$new(
  mzml_file,
 frequency_fit_description = c(a.freq = 0, x.freq = -1, y.freq = -1/2, z.freq = -1/3),
  mz_fit_description = c(a.mz = 0, x.mz = -1, y.mz = -2, z.mz = -3),
  metadata_file = NULL,
  scan_range = NULL,
  rtime_range = NULL,
  mz_range = NULL,
  remove_zero = FALSE,
  ms_level = 1,
  memory_mode = "inMemory"
)
```

Arguments:

`mzml_file` the file to load and use

`frequency_fit_description` a description of the regression model for frequency ~ m/z

`mz_fit_description` a description of the regression model for m/z ~ frequency

`metadata_file` a metadata file generated by ...

`scan_range` which scans can be used for analysis

`rtime_range` the retention time to use for scans

`mz_range` what m/z range to use

`remove_zero` should zero intensity data be removed?

`ms_level` what MS level should be extracted (default is 1)

`memory_mode` what memory mode should MSnbase be using (inMemory or onDisk)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SCMzml$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

choose_single_frequency_model_default()

Examples

```
## Not run:
  lipid_sample = system.file("extdata", "lipid_example.mzML",
  package = "ScanCentricPeakCharacterization")

## End(Not run)
```

SCPeakRegionFinder *R6 Peak Region Finder*

Description

R6 Peak Region Finder

R6 Peak Region Finder

Details

Think of it like managing all the stuff that needs to happen to find the peaks in the regions.

Public fields

`run_time` how long did the process take

`start_time` when did we start

`stop_time` when did we start

`peak_regions` SCPeakRegions object

`sliding_region_size` how big are the sliding regions in data points

`sliding_region_delta` how much space between sliding region starts

`quantile_multiplier` how much to multiply quantile based cutoff by

`n_point_region` how many points are there in the big tiled regions for quantile based cutoff

`tiled_region_size` how wide are the tiled regions in data points

`tiled_region_delta` how far in between each tiled region

`region_percentile` ??

`peak_method` what method to extract peak center, height, area, etc

`min_points` how many points wide does a peak have to be to get characterized

`sample_id` what sample are we processing

`n_zero_tiles` how many zero count tiled regions split up a region into multiple peaks?

`zero_normalization` do we want to pretend to do normalization

`calculate_peak_area` should peak area be calculated as well?

Methods

Public methods:

- SCPeakRegionFinder$add_regions()
- SCPeakRegionFinder$reduce_sliding_regions()
- SCPeakRegionFinder$split_peak_regions()
- SCPeakRegionFinder$remove_double_peaks_in_scans()
- SCPeakRegionFinder$normalize_data()
- SCPeakRegionFinder$find_peaks_in_regions()
- SCPeakRegionFinder$model_mzsd()
- SCPeakRegionFinder$model_heightsd()
- SCPeakRegionFinder$indicate_high_frequency_sd()
- SCPeakRegionFinder$add_data()
- SCPeakRegionFinder$summarize_peaks()
- SCPeakRegionFinder$add_offset()
- SCPeakRegionFinder$sort_ascending_mz()
- SCPeakRegionFinder$characterize_peaks()
- SCPeakRegionFinder$summarize()
- SCPeakRegionFinder$peak_meta()
- SCPeakRegionFinder$new()
- SCPeakRegionFinder$clone()

Method `add_regions()`**:** Add the sliding and tiled regions

Usage:

```
SCPeakRegionFinder$add_regions()
```

Method `reduce_sliding_regions()`**:** Find the regions most likely to contain real signal

Usage:

```
SCPeakRegionFinder$reduce_sliding_regions()
```

Method `split_peak_regions()`**:** Split up signal regions by peaks found

Usage:

```
SCPeakRegionFinder$split_peak_regions(
  use_regions = NULL,
  stop_after_initial_detection = FALSE
)
```

Arguments:

`use_regions` an index of the regions we want to split up

`stop_after_initial_detection` should it do full characterization or stop

Method `remove_double_peaks_in_scans()`**:** Check for the presence of two peaks with the same scan number in each region and remove them. Any regions with zero peaks left, remove the region.

Usage:

```
SCPeakRegionFinder$remove_double_peaks_in_scans()
```

Method `normalize_data()`**:** Normalize the intensity data

Usage:

```
SCPeakRegionFinder$normalize_data(which_data = "both")
```

Arguments:

`which_data` raw, characterized, or both (default)

Method `find_peaks_in_regions()`**:** Find the peaks in the regions.

Usage:

```
SCPeakRegionFinder$find_peaks_in_regions()
```

Method `model_mzsd()`**:** Model the m/z standard deviation.

Usage:

```
SCPeakRegionFinder$model_mzsd()
```

Method `model_heightsd()`**:** Model the intensity height standard deviation.

Usage:

```
SCPeakRegionFinder$model_heightsd()
```

Method `indicate_high_frequency_sd()`**:** Look for peaks with higher than expected frequency standard deviation.

Usage:

```
SCPeakRegionFinder$indicate_high_frequency_sd()
```

Method `add_data()`**:** Add the data from an SCMzml object to the underlying SCPeakRegions object.

Usage:

```
SCPeakRegionFinder$add_data(sc_mzml)
```

Arguments:

`sc_mzml` the SCMzml object being passed in

Method `summarize_peaks()`**:** Summarize the peaks to go into JSON form.

Usage:

```
SCPeakRegionFinder$summarize_peaks()
```

Method `add_offset()`**:** Add an offset based on width in frequency space to m/z to describe how wide the peak is.

Usage:

```
SCPeakRegionFinder$add_offset()
```

Method `sort_ascending_mz()`**:** Sort the data in m/z order, as the default is frequency order

Usage:

```
SCPeakRegionFinder$sort_ascending_mz()
```

Method `characterize_peaks()`**:** Run the overall peak characterization from start to finish.

Usage:

```
SCPeakRegionFinder$characterize_peaks(stop_after_initial_detection = FALSE)
```

Arguments:

`stop_after_initial_detection` do we stop the whole process after finding initial peaks in each scan?

Method `summarize()`**:** Summarize everything for output to the zip file after completion.

Usage:

```
SCPeakRegionFinder$summarize(
  package_used = "package:ScanCentricPeakCharacterization"
)
```

Arguments:

`package_used` which package is being used for this work.

Method `peak_meta()`**:** Generate the meta data that goes into the accompanying JSON file.

Usage:

```
SCPeakRegionFinder$peak_meta()
```

Method `new()`**:** Make a new SCPeakRegionFinder object.

Usage:

```
SCPeakRegionFinder$new(
  sc_mzml = NULL,
  sliding_region_size = 10,
  sliding_region_delta = 1,
  tiled_region_size = 1,
  tiled_region_delta = 1,
  region_percentile = 0.99,
  offset_multiplier = 1,
  frequency_multiplier = 400,
  quantile_multiplier = 1.5,
  n_point_region = 2000,
  peak_method = "lm_weighted",
  min_points = 4,
  n_zero_tiles = 1,
  zero_normalization = FALSE,
  calculate_peak_area = FALSE
)
```

Arguments:

`sc_mzml` the SCMzml object to use (can be missing)

`sliding_region_size` how wide to make the sliding regions in data points

`sliding_region_delta` how far apart are the starting locations of the sliding regions

`tiled_region_size` how wide are the tiled regions

`tiled_region_delta` how far apart are the tiled reigons

`region_percentile` cumulative percentile cutoff to use

`offset_multiplier` what offset multiplier should be used

`frequency_multiplier` how much to multiply frequency points to interval ranges

`quantile_multiplier` how much to adjust the quantile cutoff by

`n_point_region` how many points in the large tiled regions

`peak_method` the peak characterization method to use (lm_weighted)

`min_points` how many points to say there is a peak (4)

`n_zero_tiles` how many tiles in a row do there need to be to split things up? (1)

`zero_normalization` don't actually do normalization (FALSE)

`calculate_peak_area` should peak area as well as peak height be returned? (FALSE)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SCPeakRegionFinder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`SCPeakRegions` *Holds all the peak region data*

Description

Holds all the peak region data

Holds all the peak region data

Details

This reference class represents the peak region data.

Public fields

`frequency_point_regions` the frequency data

`frequency_fit_description` the model of frequency ~ m/z

`mz_fit_description` the model of m/z ~ frequency

`peak_regions` the peak regions

`sliding_regions` the sliding regions used for density calculations

`tiled_regions` the tiled regions used for grouping and splitting peak regions

`peak_region_list` list of regions

`frequency_multiplier` how much to multiplier frequency by to make interval points

`scan_peaks` the peaks by scans

`peak_data` the data.frame of final peak data

`scan_level_arrays` scan level peak data as matrices

`is_normalized` are the peak intensities normalized

`normalization_factors` the normalization factors calculated

`n_scan` how many scans are we working with

`scans_per_peak` ??

`scan_perc` what percentage of scans is a minimum

`min_scan` based on `scan_perc`, how many scans minimum does a peak need to be in

`max_subsets` ??

`scan_subsets` ??

`frequency_range` what is the range in frequency space

`scan_correlation` ??

`keep_peaks` which peaks are we keeping out of all the peaks we had

`peak_index` the indices for the peaks

`scan_indices` the names of the scans

`instrument` the instrument serial number if available

Methods

Public methods:

- `SCPeakRegions$set_min_scan()`
- `SCPeakRegions$add_data()`
- `SCPeakRegions$new()`
- `SCPeakRegions$clone()`

Method `set_min_scan()`**:** sets the minimum number of scans to use

Usage:

```
SCPeakRegions$set_min_scan()
```

Method `add_data()`**:** Adds the data from an SCMzml object to the SCPeakRegion.

Usage:

```
SCPeakRegions$add_data(sc_mzml)
```

Arguments:

`sc_mzml` the SCMzml object being passed in

Method `new()`**:** Creates a new SCPeakRegions object

Usage:

```
SCPeakRegions$new(
  sc_mzml = NULL,
  frequency_multiplier = 400,
  scan_perc = 0.1,
  max_subsets = 100
)
```

Arguments:

`sc_mzml` the SCMzml object to get data from

`frequency_multiplier` how much to multiply frequency by

`scan_perc` how many scans are required to be in to be a "peak"

`max_subsets` ??

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SCPeakRegions$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`SCZip` *Represents the zip mass spec file*

Description

Represents the zip mass spec file

Represents the zip mass spec file

Details

This reference class represents the zip mass spec file. It does this by providing objects for the zip file, the metadata, as well as various bits underneath such as the mzml data and peak lists, and their associated metadata. Although it is possible to work with the SCZip object directly, it is heavily recommended to use the SCCharacterizePeaks object for carrying out the various steps of an analysis, including peak finding.

Public fields

`zip_file` the actual zip file

`zip_metadata` the metadata about the zip file

`metadata` the metadata itself

`metadata_file` the metadata file

`sc_mzml` the mzML data object.

`peaks` ??

`sc_peak_region_finder` the peak finder object

`json_summary` jsonized summary of the peak characterization

`id` the identifier of the sample

`out_file` where to put the final file

`temp_directory` where we keep everything until peak characterization is done

Methods

Public methods:

- SCZip$load_mzml()
- SCZip$load_sc_peak_region_finder()
- SCZip$save_json()
- SCZip$save_sc_peak_region_finder()
- SCZip$load_peak_list()
- SCZip$compare_mzml_corresponded_densities()
- SCZip$new()
- SCZip$show_temp_dir()
- SCZip$write_zip()
- SCZip$cleanup()
- SCZip$finalize()
- SCZip$add_peak_list()
- SCZip$clone()

Method `load_mzml()`**:** Loads the mzML file

Usage:

```
SCZip$load_mzml()
```

Method `load_sc_peak_region_finder()`**:** Loads the SCPeakRegionFinder object

Usage:

```
SCZip$load_sc_peak_region_finder()
```

Method `save_json()`**:** Save the jsonized summary out to actual json files

Usage:

```
SCZip$save_json()
```

Method `save_sc_peak_region_finder()`**:** Saves the SCPeakRegionFinder binary object

Usage:

```
SCZip$save_sc_peak_region_finder()
```

Method `load_peak_list()`**:** loads just the peak list data-frame instead of peak region finder

Usage:

```
SCZip$load_peak_list()
```

Method `compare_mzml_corresponded_densities()`**:** compare peak densities

Usage:

```
SCZip$compare_mzml_corresponded_densities(
  mz_range = c(150, 1600),
  window = 1,
  delta = 0.1
)
```

Arguments:

`mz_range` the mz range to work over

`window` the window size in m/z

`delta` how much to move the window

Method `new()`**:** Create a new SCZip object.

Usage:

```
SCZip$new(
  in_file,
  mzml_meta_file = NULL,
  out_file = NULL,
  load_mzml = TRUE,
  load_peak_list = TRUE,
  temp_loc = tempfile("scpcms")
)
```

Arguments:

`in_file` the mzML file to load

`mzml_meta_file` an optional metadata file

`out_file` where to save the final file

`load_mzml` should the mzML file actually be loaded into an SCMzml object?

`load_peak_list` should the peak list be loaded if this is previously characterized?

`temp_loc` where to make the temp file while working with the data

Method `show_temp_dir()`**:** Show the temp directory where everything is being worked with

Usage:

```
SCZip$show_temp_dir()
```

Method `write_zip()`**:** Write the zip file

Usage:

```
SCZip$write_zip(out_file = NULL)
```

Arguments:

`out_file` where to save the zip file

Method `cleanup()`**:** delete the temp directory

Usage:

```
SCZip$cleanup()
```

Method `finalize()`**:** delete when things are done

Usage:

```
SCZip$finalize()
```

Method `add_peak_list()`**:** Add peak list data to the temp directory

Usage:

```
SCZip$add_peak_list(peak_list_data)
```

Arguments:

`peak_list_data` the peak list data

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SCZip$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

SCMzml

SCPeakRegionFinder

`sc_zip` *make a new SCZip*

Description

make a new SCZip

Usage

```
sc_zip(
  in_file,
  mzml_meta_file = NULL,
  out_file = NULL,
  load_raw = TRUE,
  load_peak_list = TRUE
)
```

Arguments

`in_file`	the file to use (either .zip or .mzML)
`mzml_meta_file`	metadata file (.json)
`out_file`	the file to save to at the end
`load_raw`	logical to load the raw data
`load_peak_list`	to load the peak list if it exists

Value

SCZip

set_internal_map *pick mapping function*

Description

Allows the user to set which mapping function is being used internally in the functions.

Usage

```
set_internal_map(map_function = NULL)
```

Arguments

map_function	which function to use, assigns it to an internal object

Details

by default, the package uses purrr::map to iterate over things. However, if you have the furrr package installed, you could switch it to use furrr::future_map instead.

Examples

```
## Not run:
 library(furrr)
 future::plan(multicore)
 set_internal_map(furrr::future_map)

## End(Not run)
```

show_progress *turn progress on off*

Description

Allow the user to turn progress messages to the console and off. Default is to provide messages to the console.

Usage

```
show_progress(progress = TRUE)
```

Arguments

progress	logical to have it on or off

single_pass_normalization *Single Pass Normalization*

Description

Does a single pass of normalizing scans to each other.

Usage

```
single_pass_normalization(
  scan_peaks,
  intensity_measure = c("RawHeight", "Height"),
  summary_function = median,
  use_peaks = NULL,
  min_ratio = 0.7
)
```

Arguments

scan_peaks	the scan peaks to normalize
intensity_measure	which intensities to normalize
summary_function	which function to use to calculate summaries (median)
use_peaks	which peaks to use for normalization
min_ratio	what ratio of maximum intensity of peaks should we use for normalization

Value

scan_peaks list

split_region_by_peaks *split signal region*

Description

Given a region that should contain signal, and the point data within it, find the peaks, and return the region, and the set of points that make up each peak from each scan.

Usage

```
split_region_by_peaks(
  region_list,
  min_points = 4,
  metadata = NULL,
  calculate_peak_area = FALSE
)
```

Arguments

region_list	a list with points and tiles IRanges objects
min_points	how many points are needed for a peak
metadata	metadata that tells how things should be processed

Value

list

---

ssr *sum of squares residuals*

Description

returns the sum of squares residuals from an lm object

Usage

```
ssr(object)
```

Arguments

object	the lm object

Value

numeric

transform_residuals *transformed residuals*

Description

given a set of original and fitted values and a transform, return a set of transformed residuals.

Usage

```
transform_residuals(original, fitted, transform = exp)
```

Arguments

original	the original points
fitted	the fitted points
transform	the function that should be used to transform the values

Value

numeric

zip_list_contents *list zip file contents*

Description

given a zip file, list the contents

Usage

```
zip_list_contents(zip_file)
```

Arguments

zip_file	the zip file

Index

.get_scan_polarity, 3
.remove_attrs, 4
.to_data_frame, 4

add_scan_info, 4
add_to_zip, 5
area_sum_points, 6

characterize_peaks, 6
check_frequency_r2, 7
check_ranges_convert_to_regions, 7
check_zip_file, 8
choose_frequency_model_builtin, 8
choose_single_frequency_model_default(), *45*
convert_mz_frequency, 9
correct_mean, 10
correct_mean(), *11, 12*
correct_peak, 10
correct_peak(), *10*, *12*
correct_peak_sd_height, 11
correct_variance, 12
correct_variance(), *10, 11*
create_frequency_regions, 12
create_value, 13

default_correct_offset_function, 14
default_offset_predict_function, 14
disable_logging, 15

enable_logging, 15
extract, 16
extract_raw_method, 16

filter_scans_builtin, 17
find_signal_regions, 17
frequency_points_to_frequency_regions, 18

get_fitted_peak_info, 19
get_mzml_header, 19
get_mzml_metadata, 20
get_raw_ms_metadata, 20

import_json, 21
import_sc_mzml, 21
indicate_standards_contaminents, 22
initialize_metadata_from_mzml, 23
initialize_zip_metadata, 23
integrate_model, 24
integrate_sides, 24
integration_based_area, 25

json_2_peak_list, 25
json_mzML_2_df, 26

lists_2_json, 26
load_metadata, 27
loess_to_df, 27
log_memory, 28
log_message, 28
log_with_min, 28

meta_export_json, 29
model_peak_center_intensity, 29
mz_frequency_interpolation, 31
mz_scans_to_frequency, 31
mzml_to_zip, 30

parabolic_fit, 32
peak_list_2_json, 33
predicted_frequency_r2, 33

raw_metadata_mzml, 34
recalculate_offsets, 34
run_mzml_list, 35

sample_run_time, 35
sc_zip, 54
ScanCentricPeakCharacterization, 36
SCCharacterizePeaks, 36
SCMzml, 41, *54*

SCPeakRegionFinder, 45, *54*
SCPeakRegions, 49
SCZip, 51
set_internal_map, 55
show_progress, 55
single_pass_normalization, 56
split_region_by_peaks, 56
ssr, 57

transform_residuals, 58

zip_list_contents, 58