Package 'MALDIassist'

August 4, 2026

Title Mathematical Utilities for MALDI-TOF Mass Spectrometry

Version 1.0.2

Description Supports matrix-assisted laser desorption/ionization time-of-flight (MALDI-TOF) mass spectrometry workflows from raw Bruker spectra to cohort-level peak matrices. Provides spectrum loading, Savitzky-Golay smoothing, baseline correction (SNIP and TopHat), Gaussian kernel-regression-based peak detection including shoulder peaks, peak-quality assessment, filtering, and cohort feature analysis. Computationally intensive routines are implemented in C++ using 'Rcpp'. The implemented signal-processing methods include those described by Savitzky and Golay (1964) <doi:10.1021/ac60214a047>, Ryan et al. (1988) <doi:10.1016/0168-583X(88)90063-8>, Stanford, Bagley and Solomon (2016) <doi:10.1186/s12953-016-0107-8>, and Nadaraya-Watson kernel regression (Nadaraya (1964) <doi:10.1137/1109020>; Watson (1964) <https://www.jstor.org/stable/25049340>).

License MIT + file LICENSE

URL https://github.com/hiows/MALDIassist

BugReports https://github.com/hiows/MALDIassist/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports Rcpp, gtools

LinkingTo Rcpp

Suggests colorspace, pheatmap, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Wonseok Oh [aut, cre] (ORCID: <https://orcid.org/0009-0002-0687-8466>)

Maintainer Wonseok Oh <hiows97@gmail.com>

Repository CRAN

Date/Publication 2026-08-04 10:10:13 UTC

Contents

`MALDIassist-package` *MALDIassist: Mathematical utilities for MALDI-TOF mass spectrometry*

Description

Supports loading Bruker MALDI-TOF spectra, preprocessing, Gaussian kernel-regression-based peak detection, peak-quality metrics, and cohort feature analysis workflows.

Main functions

- `load_maldi_spectra()`: load Bruker raw spectra
- `preprocess_maldi_spectra()`: smooth and baseline-correct spectra
- `find_peaks()` / `find_peaks_spectra()`: detect ordinary and shoulder peaks in one spectrum or a list of spectra
- `find_peaks_fast()` / `find_peaks_spectra_fast()`: fast local peak detection in one spectrum or a list of spectra

- filter_peaks() / filter_peaks_spectra(): filter detected peaks by intensity, prominence, and strength
- build_kde_spectrum() / build_kde_spectra(): build Gaussian KDE spectra
- find_frequent_mz(): find frequent m/z values across a cohort
- filter_overlap_mz(): remove overlapping nearby frequent m/z features
- align_spectra(): align spectra to internal standards (linear / lowess)
- build_matched_matrix(): assemble a cohort peak intensity matrix

Suggested packages

Some functions require optional packages that are only loaded when used:

- visualize_spectrum, visualize_spectra: colorspace
- heatmap_matched_matrix: pheatmap

Author(s)

Maintainer: Wonseok Oh <hiows97@gmail.com> (ORCID)

Authors:

- Wonseok Oh <hiows97@gmail.com> (ORCID)

See Also

Useful links:

- https://github.com/hiows/MALDIassist
- Report bugs at https://github.com/hiows/MALDIassist/issues

---

align_spectra *Alignment of MALDI-TOF Spectra to Internal Standards*

Description

Aligns a cohort of MALDI-TOF spectra along the m/z axis using internal standard m/z values selected from frequently observed, high-intensity peaks via find_frequent_mz(). Two alignment strategies are provided:

- "linear": a two-point linear alignment. A low-mass and a high-mass anchor are selected, matched in each spectrum with match_peaks(), and a linear map (aligned_mz = slope * mz + intercept) that sends the detected anchors onto the reference standard m/z is applied.
- "lowess": a multi-point non-linear alignment. All frequent standard m/z values are matched in each spectrum, a locally weighted regression (lowess) of the m/z shift against the detected m/z is fitted per spectrum, and the interpolated shift is added to the m/z axis.

In both cases the alignment is applied to the first column (m/z) of the spectrum and its peak table.

Usage

```
align_spectra(
  spectra,
  peaks_list,
  bin_width = 20,
  alignment_mode = c("linear", "lowess"),
  lowess_span = 2/3,
  freq_ratio_cutoff = 0.9,
  hws_alignment = 50
)
```

Arguments

spectra	A list of spectra. Each element must be a matrix or data frame with m/z in the first column and intensity in the second column. When named, elements are matched to peaks_list by name; otherwise they are matched by position.
peaks_list	A list of peak tables aligned with spectra. Each element must be a matrix or data frame with m/z in the first column and intensity in the second column.
bin_width	A positive finite numeric scalar passed to find_frequent_mz() when locating candidate standard m/z values. The default is 20.
alignment_mode	Character string specifying the alignment strategy, either "linear" (default) or "lowess". See Details.
lowess_span	A finite numeric scalar in (0, 1] giving the smoother span (f) passed to stats::lowess() when alignment_mode = "lowess". Larger values give smoother fits. The default is 2/3.
freq_ratio_cutoff	A finite numeric scalar in [0, 1]. Only frequent m/z values with freq_ratio > freq_ratio_cutoff are considered as standard candidates. The default is 0.9.
hws_alignment	A positive finite numeric scalar specifying the half-window size, in m/z units, passed to match_peaks() when matching each spectrum's peaks to the standard m/z values. The default is 50.

Details

Standard candidates are obtained once for the whole cohort from find_frequent_mz() and filtered by freq_ratio_cutoff. At least two candidates must remain for either mode.

Linear mode

1. Two standard m/z values are chosen. The retained m/z range is split into a low-mass and a high-mass half, and within each half the m/z with the largest median_intensity is used as an anchor. These are named std_1 (low mass) and std_2 (high mass).
2. For each spectrum, the two anchors are matched to detected peaks with match_peaks() using peak_selection_mode = "maximum_intensity", so that when several peaks fall within the matching window the most intense one is used.
3. A two-point linear map is fitted from the two detected anchors to the two reference standard m/z values and applied to the first column (m/z) of the spectrum and its peak table.

A spectrum is aligned only when both anchors are matched and the two detected anchor m/z values differ (so that the slope is finite). Otherwise the spectrum and its peaks are returned unchanged and flagged with `is_aligned = FALSE`.

Lowess mode

1. All retained frequent m/z values are used as standards (`std_1`, ..., `std_N`).
2. For each spectrum, standards are matched to detected peaks with `match_peaks()` using `peak_selection_mode = "nearest_mz"`. For every matched standard the shift `standard_mz - detected_mz` is computed.
3. A `lowess` smoother of the shift against the detected m/z is fitted per spectrum, and the shift interpolated with `stats::approx()` (`rule = 2`) is added, independently, to the m/z axis of the spectrum and of the peak table.

A spectrum is aligned only when at least two distinct matched standards are available for the fit. Otherwise the spectrum and its peaks are returned unchanged and flagged with `is_aligned = FALSE`.

Value

A list with elements:

alignment_results A named list with one element per sample, each a list with components `spectrum` and `peaks` containing the (possibly) m/z-aligned spectrum and peak table.

standard_mz A named numeric vector with the reference standard m/z values (`std_1`, ...). Length two in `"linear"` mode and one per retained frequent m/z in `"lowess"` mode.

matched_mz A data frame with one row per sample and columns `spectrum_name`, one `std_*` column per standard, and `is_aligned`. The `std_*` columns hold the detected m/z for that sample, or `NA` when the standard was not matched. `is_aligned` indicates whether the alignment was applied.

alignment_mode The alignment strategy used.

See Also

`find_frequent_mz()`, `match_peaks()`, `build_matched_matrix()`

Examples

```
make_spectrum <- function(seed_offset = 0L) {
  set.seed(1L + seed_offset)
  x <- seq(3000, 8000, length.out = 400)
  y <- dnorm(x, mean = 4000, sd = 30) * 100 +
       dnorm(x, mean = 5500, sd = 35) * 80 +
       runif(length(x), 0, 5)
  data.frame(mz = x, intensity = y)
}

spectra <- list(
  sample_1 = make_spectrum(0L),
  sample_2 = make_spectrum(1L),
  sample_3 = make_spectrum(2L)
```

```
)
pp_spectra <- preprocess_maldi_spectra(spectra, iter_snip = 20, n_cores = 1)
peaks_list <- find_peaks_spectra_fast(pp_spectra, hws_peaks = 10, n_cores = 1)

aligned <- align_spectra(
  spectra = pp_spectra,
  peaks_list = peaks_list,
  bin_width = 20,
  alignment_mode = "linear",
  freq_ratio_cutoff = 0.5,
  hws_alignment = 50
)

aligned$standard_mz
aligned$matched_mz
```

`build_kde_spectrum` *Build a Gaussian KDE-Smoothed Spectrum*

Description

Builds a Gaussian kernel density estimate (KDE) representation of a mass spectrum. The returned object contains the KDE-smoothed spectrum evaluated on the observed m/z grid, together with the reusable KDE closure and the bandwidth used. This prepares data that matches the Gaussian KDE-based peak detection performed by `find_peaks()` and `find_peaks_spectra()`.

Usage

```
build_kde_spectrum(spectrum, bw = stats::median(diff(spectrum[, 1])))

build_kde_spectra(spectra, bw = NULL, n_cores = 1L)
```

Arguments

`spectrum`	A numeric matrix or data frame with at least two columns. The first column must contain strictly increasing x-axis values, typically m/z values, and the second column must contain the corresponding intensities.
`bw`	A positive numeric scalar specifying the bandwidth of the Gaussian KDE. The default is the median interval between adjacent x-axis values.
`spectra`	A two-column `data.frame` or numeric `matrix` representing a single spectrum, or a `list` of such objects. The first column must contain strictly increasing x-axis values and the second column must contain intensity values.
`n_cores`	A positive integer specifying the number of worker processes used when `spectra` is a list. The default is `1L`, which preserves sequential behaviour.

Details

The KDE closure is created with `get_gauss_kde()` using derivative order `0`. The smoothed spectrum is obtained by evaluating this closure on the observed x-axis grid, so the output spectrum has the same x-axis values as the input. The column names of the input spectrum are preserved in the output spectrum. When the input has no usable column names, `x` and `y` are used as a fallback.

`build_kde_spectra()` applies `build_kde_spectrum()` to a single spectrum or to each spectrum in a list. When `bw` is `NULL`, the per-spectrum default bandwidth of `build_kde_spectrum()` (the median interval between adjacent x-axis values) is used for every spectrum. Supplying a non-`NULL` `bw` applies the same bandwidth to all spectra.

Value

A list with three elements:

- `spectrum`: A two-column `data.frame` containing the observed x-axis values and the KDE-smoothed intensities. The column names are inherited from `spectrum`.
- `gauss_kde`: The Gaussian KDE closure returned by `get_gauss_kde()`, which can be re-evaluated at arbitrary x-axis values.
- `bw`: The bandwidth used to build the KDE.

`build_kde_spectrum()` returns a single KDE-spectrum list. `build_kde_spectra()` returns a single KDE-spectrum list when `spectra` is a single spectrum, or a named list of KDE-spectrum lists with the same list structure and names as `spectra`.

See Also

`get_gauss_kde()`, `find_peaks()`, `find_peaks_spectra()`

Examples

```
x <- seq(1000, 2000, length.out = 2000)
y <- dnorm(x, mean = 1400, sd = 20) * 100 +
     dnorm(x, mean = 1700, sd = 25) * 80
spectrum <- data.frame(mz = x, intensity = y)

kde_spectrum <- build_kde_spectrum(spectrum, bw = 1)

head(kde_spectrum$spectrum)
kde_spectrum$gauss_kde(c(1400, 1700))

make_spectrum <- function() {
  x <- seq(1000, 2000, length.out = 500)
  y <- dnorm(x, mean = 1400, sd = 20) * 100 +
       dnorm(x, mean = 1700, sd = 25) * 80
  data.frame(mz = x, intensity = y)
}

spectra <- list(sample_1 = make_spectrum(), sample_2 = make_spectrum())

kde_spectra <- build_kde_spectra(
```

```
  spectra = spectra,
  bw = 1,
  n_cores = 1
)

lapply(kde_spectra, function(k) head(k$spectrum))
```

`build_matched_matrix` *Build a Sample-by-Marker Peak Intensity Matrix*

Description

Matches detected peaks from multiple samples to a shared set of reference m/z values and assembles a sample-by-marker feature matrix.

Usage

```
build_matched_matrix(
  peaks_list,
  reference_mz,
  reference_names = NULL,
  hws_match = 10,
  peak_selection_mode = c("nearest_mz", "maximum_intensity")
)
```

Arguments

`peaks_list`	A named list of peak tables, typically from `find_peaks()`.
`reference_mz`	A numeric vector of reference m/z values, for example from `find_frequent_mz()`.
`reference_names`	Optional character vector of marker names.
`hws_match`	Half-window size passed to `match_peaks()`. The default is `10`.
`peak_selection_mode`	Peak selection rule passed to `match_peaks()`. The default is `"nearest_mz"`.

Value

A list with elements:

detected_matrix Integer matrix with samples as rows and markers as columns, holding `1` where a peak was matched and `0` otherwise. Column names are `reference_names` when supplied, otherwise `paste0("mz_", round(reference_mz, 3))`. Columns with no matched peak in any sample are removed.

delta_mz_matrix Numeric matrix with the same dimensions as `detected_matrix`, holding the signed m/z difference (`detected_mz - reference_mz`). Unmatched cells remain `NA`.

reference_mz Reference m/z values retained in the matrices.

reference_names Marker names retained in the matrices, if supplied.

sample_names Sample names from `peaks_list`.

matches Named list of per-sample `match_peaks()` results.

See Also

`match_peaks()`, `find_frequent_mz()`, `heatmap_matched_matrix()`

Examples

```
peaks_list <- list(
  sample_1 = data.frame(mz = c(1000, 1500, 2000), intensity = c(10, 30, 20)),
  sample_2 = data.frame(mz = c(1002, 2001), intensity = c(12, 25))
)

result <- build_matched_matrix(
  peaks_list = peaks_list,
  reference_mz = c(1000, 1500, 2000),
  reference_names = c("marker_a", "marker_b", "marker_c"),
  hws_match = 10
)

result$detected_matrix
```

`calculate_peak_prominence`

Calculate Peak Prominence

Description

Calculate peak prominence for detected peaks.

This function validates spectrum data and detected peaks, then calls the Rcpp backend `cpp_calculate_peak_prominence()`. Peak prominence is calculated using the left and right valleys around each peak. The reference valley can be selected by `valley_type`.

- `"higher"`: use the higher of the left and right valleys. This is the more conservative and standard-like prominence definition.
- `"lower"`: use the lower of the left and right valleys. This gives a larger prominence value.

Usage

```
calculate_peak_prominence(
  data,
  peaks,
  valley_type = c("higher", "lower"),
  zero_tol = sqrt(.Machine$double.eps)
)
```

Arguments

data	A data frame or matrix with at least two columns. The first column must be x values, such as m/z, and the second column must be intensity values.
peaks	A data frame or matrix with at least two columns. The first column must be peak x positions and the second column must be peak intensities. Only the first two columns are used.
valley_type	Character string specifying which reference valley should be used. One of "higher" or "lower". Default is "higher".
zero_tol	A single non-negative finite numeric value. Prominence values with absolute value smaller than or equal to zero_tol are treated as zero. Default is sqrt(.Machine$double.eps).

Value

A numeric vector of peak prominence values. The returned vector has the same length as the number of input peaks.

Examples

```
x <- seq(1000, 2000, length.out = 2000)
y <- dnorm(x, mean = 1400, sd = 20) * 100 +
     dnorm(x, mean = 1700, sd = 25) * 80
spectrum_data <- data.frame(mz = x, intensity = y)

detected_peaks <- find_peaks_fast(spectrum_data, hws_peaks = 10)

peak_prominence <- calculate_peak_prominence(
  data = spectrum_data,
  peaks = detected_peaks,
  valley_type = "higher"
)

peak_prominence
```

---

estimate_peak_strength *Estimate Peak Strength*

Description

Estimate a bounded peak strength score for each detected peak.

This function validates input spectrum data and detected peaks, then calls the Rcpp backend cpp_estimate_peak_strength(The peak strength score is calculated from the local intensity scale around each peak relative to the global intensity scale of the whole spectrum.

The intensity can be transformed before calculation using one of three normalization types:

- "raw": use raw intensity y

- "sqrt": use sqrt(y)
- "log10": use log10(1 + y)

Usage

```
estimate_peak_strength(
  data,
  peaks,
  k = 1,
  normalization_type = c("raw", "sqrt", "log10")
)
```

Arguments

data	A data frame or matrix with at least two columns. The first column must be x values, such as m/z, and the second column must be non-negative intensity values.
peaks	A data frame or matrix with at least two columns. The first column must be peak x positions and the second column must be peak intensities. Only the first two columns are used.
k	A single non-negative finite numeric value. MAD multiplier used when estimating local and global strength scales. Default is 1.
normalization_type	Character string specifying the intensity transformation method. One of "raw", "sqrt", or "log10". Default is "raw".

Value

A numeric vector of peak strength scores. The returned vector has the same length as the number of input peaks. Peaks for which local valleys cannot be defined may return NA_real_.

Examples

```
x <- seq(1000, 2000, length.out = 2000)
y <- dnorm(x, mean = 1400, sd = 20) * 100 +
     dnorm(x, mean = 1700, sd = 25) * 80
spectrum_data <- data.frame(mz = x, intensity = y)

detected_peaks <- find_peaks_fast(spectrum_data, hws_peaks = 10)

peak_strength <- estimate_peak_strength(
  data = spectrum_data,
  peaks = detected_peaks,
  k = 1,
  normalization_type = "log10"
)

peak_strength
```

`estimate_significance` *Estimate Significant m/z Features Between Two Groups*

Description

Performs a per-feature two-group comparison on a sample-by-feature matrix, such as the intensity or detection matrix produced by `build_matched_matrix()`. For each feature (column), a two-sided test is run between the two sample groups and the resulting p-values are optionally adjusted for multiple comparisons.

Usage

```
estimate_significance(
  matched_matrix,
  group,
  feat_names = NULL,
  stat_method = c("t.test", "wilcox"),
  adj_method = c("none", "BH", "bonferroni")
)
```

Arguments

`matched_matrix`	A numeric matrix or data frame with samples in rows and features (m/z markers) in columns. A data frame is coerced with `as.matrix()` and must contain only numeric columns.
`group`	A vector of group labels with one entry per row of `matched_matrix`. It must contain exactly two distinct (non-`NA`) levels. The first level in factor order is treated as the reference group.
`feat_names`	Optional character vector of feature names, one per column of `matched_matrix`. If `NULL`, the column names of `matched_matrix` are used when available, otherwise `paste0("feat_", seq_len(ncol))`.
`stat_method`	The statistical test to apply per feature, either `"t.test"` (Welch two-sample t-test) or `"wilcox"` (Wilcoxon rank-sum test with `exact = FALSE`). The default is `"t.test"`.
`adj_method`	The multiple-comparison adjustment passed to `stats::p.adjust()`, one of `"none"`, `"BH"`, or `"bonferroni"`. The default is `"none"`.

Details

The two group levels are determined from `as.factor(group)`. Rows are split into the reference group (the first factor level) and the other group, and a two-sided test is applied to each feature column:

- For `"t.test"`, if both groups are constant for a feature (zero standard deviation), the p-value is set to `NaN`, matching the behavior of a degenerate comparison.
- Tests that fail (for example, on constant data) return `NaN` for that feature instead of raising an error, so a single problematic feature does not abort the whole scan.

Value

A data frame with one row per feature and columns:

- `feat_names`: feature name.
- `pvalue`: raw two-sided p-value.
- `adj_pvalue`: p-value after `adj_method` adjustment.

See Also

`build_matched_matrix()`, `stats::p.adjust()`

Examples

```
set.seed(1)
n_per_group <- 10
matched_matrix <- rbind(
  matrix(rnorm(n_per_group * 3, mean = 0), nrow = n_per_group),
  matrix(rnorm(n_per_group * 3, mean = 1), nrow = n_per_group)
)
colnames(matched_matrix) <- paste0("mz_", c(1000, 2000, 3000))
group <- rep(c("control", "case"), each = n_per_group)

estimate_significance(
  matched_matrix = matched_matrix,
  group = group,
  stat_method = "t.test",
  adj_method = "BH"
)
```

`filter_overlap_mz` *Filter Overlapping Frequent m/z Features*

Description

Removes redundant frequent m/z values that fall within a local half-window of each other. For every candidate, the function retains the row only when its `freq_ratio` equals the maximum `freq_ratio` among all candidates in `[mz - hws_selection, mz + hws_selection]`. This is a post-processing step for tables returned by `find_frequent_mz()`, which can otherwise report nearby m/z values (for example, differing by only about 1 Da) as separate features.

Usage

```
filter_overlap_mz(freq_data, hws_selection = 5)
```

Arguments

freq_data	A data frame or matrix containing at least the columns mz and freq_ratio, typically the output of find_frequent_mz(). Additional columns are preserved in the returned object.
hws_selection	A non-negative finite numeric scalar specifying the half-window size, in m/z units, used to define the local neighborhood for each candidate. The default is 5.

Details

Ties are retained: if two or more candidates share the same maximum freq_ratio inside a neighborhood, all of them are kept.

The returned table keeps the original row order of retained candidates. Row names are reset.

Value

An object of the same class as freq_data, containing only the non-overlapping frequent m/z rows. If no rows remain, an empty object with the same columns is returned.

See Also

find_frequent_mz(), build_matched_matrix()

Examples

```
freq_data <- data.frame(
  mz = c(3000, 3001, 5000, 5010),
  median_intensity = c(2.0, 1.8, 2.5, 2.1),
  count = c(18, 10, 20, 12),
  freq_ratio = c(0.9, 0.5, 1.0, 0.6)
)

filter_overlap_mz(freq_data, hws_selection = 5)
```

---

filter_peaks *Filter Detected Peaks Using Intensity, Prominence, and Peak Strength*

Description

Filters candidate peaks detected from a MALDI-TOF mass spectrum using minimum thresholds for peak intensity, peak prominence, and peak strength. Peak prominence is calculated using calculate_peak_prominence(), and peak strength is estimated using estimate_peak_strength().

Usage

```
filter_peaks(
  data,
  peaks,
  cutoff_peak_intensity = NULL,
  cutoff_peak_prominence = NULL,
  cutoff_peak_strength = 0.2,
  k = 1,
  normalization_type = c("raw", "sqrt", "log10")
)

filter_peaks_spectra(
  spectra,
  peaks_list,
  cutoff_peak_intensity = NULL,
  cutoff_peak_prominence = NULL,
  cutoff_peak_strength = 0.2,
  k = 1,
  normalization_type = c("raw", "sqrt", "log10"),
  n_cores = 1L
)
```

Arguments

data	A numeric matrix or data frame containing the mass spectrum. The first column must contain strictly increasing x-axis values, typically m/z values, and the second column must contain the corresponding intensities.
peaks	A numeric matrix or data frame containing candidate peaks. The first column must contain peak positions and the second column must contain the corresponding peak intensities. Additional columns are preserved in the returned object.
cutoff_peak_intensity	A non-negative finite numeric scalar specifying the minimum peak intensity. Peaks with intensities less than or equal to this threshold are removed. If NULL, the median absolute deviation (MAD) of the spectrum intensities is used.
cutoff_peak_prominence	A non-negative finite numeric scalar specifying the minimum peak prominence. Peaks with prominences less than or equal to this threshold are removed. If NULL, the median absolute deviation (MAD) of the spectrum intensities is used.
cutoff_peak_strength	A non-negative finite numeric scalar specifying the minimum peak-strength score. Peaks with strength values less than or equal to this threshold are removed. The default is 0.2.
k	A positive finite numeric scalar passed to estimate_peak_strength(). This parameter controls the contribution of the noise-related term when calculating peak strength. The default is 1.

normalization_type
: Character string specifying the intensity transformation passed to `estimate_peak_strength()`. Available options are `"raw"`, `"sqrt"`, and `"log10"`.

spectra
: A two-column `data.frame` or numeric `matrix` representing a single spectrum, or a `list` of such objects aligned with `peaks_list`.

peaks_list
: A peak table (`data.frame` or `matrix`) when `spectra` is a single spectrum, or a `list` of peak tables aligned with `spectra`.

n_cores
: A positive integer specifying the number of worker processes used when `spectra` is a list. The default is `1L`, which preserves sequential behaviour.

Details

A candidate peak is retained only when all three criteria are satisfied:

```
peak intensity  > cutoff_peak_intensity
peak prominence > cutoff_peak_prominence
peak strength   > cutoff_peak_strength
```

Peaks with non-finite intensity, prominence, or strength values are removed. Such values may occur, for example, when a boundary peak does not have a well-defined neighboring valley.

The default intensity and prominence thresholds are calculated from the spectrum intensity vector rather than from the detected peak intensities.

`filter_peaks_spectra()` applies `filter_peaks()` to a single spectrum and its peak table, or to each spectrum-peak pair in a list. When `spectra` and `peaks_list` are lists, they are matched by name: the intersection of `names(spectra)` and `names(peaks_list)` determines which samples are processed. Unnamed lists are matched positionally by assigning sequential names.

Value

An object of the same class as `peaks`, containing only peaks that satisfy all filtering criteria. All original columns in `peaks` are preserved. If no peaks remain, an empty object with the same columns is returned.

`filter_peaks()` returns a single filtered peak table. `filter_peaks_spectra()` returns a single filtered peak table when `spectra` is a single spectrum, or a named list of filtered peak tables for the shared sample names when `spectra` and `peaks_list` are lists.

See Also

`filter_peaks_spectra()`

`find_peaks_spectra()`, `preprocess_maldi_spectra()`

Examples

```
x <- seq(1000, 2000, length.out = 2000)
y <- dnorm(x, mean = 1400, sd = 20) * 100 +
     dnorm(x, mean = 1700, sd = 25) * 80
processed_spectrum <- data.frame(mz = x, intensity = y)
```

```
detected_peaks <- find_peaks_fast(processed_spectrum, hws_peaks = 10)

filtered_peaks <- filter_peaks(
  data = processed_spectrum,
  peaks = detected_peaks,
  cutoff_peak_intensity = NULL,
  cutoff_peak_prominence = NULL,
  cutoff_peak_strength = 0.2,
  k = 1,
  normalization_type = "raw"
)

filtered_peaks

make_spectrum <- function() {
  x <- seq(1000, 2000, length.out = 500)
  y <- dnorm(x, mean = 1400, sd = 20) * 100 +
       dnorm(x, mean = 1700, sd = 25) * 80
  data.frame(mz = x, intensity = y)
}

spectra <- list(sample_1 = make_spectrum(), sample_2 = make_spectrum())
pp_spectra <- preprocess_maldi_spectra(spectra, iter_snip = 20, n_cores = 1)
peaks_list <- find_peaks_spectra_fast(pp_spectra, hws_peaks = 10, n_cores = 1)

filtered <- filter_peaks_spectra(
  spectra = pp_spectra,
  peaks_list = peaks_list,
  cutoff_peak_strength = 0.2,
  n_cores = 1
)

lapply(filtered, head)
```

`find_extrema` *Find Local Extrema from First and Second Derivative Functions*

Description

Finds local extrema by locating the roots of a first derivative function and classifying the stationary points using the corresponding second derivative values.

This function first identifies points where the first derivative is zero using the internal C++ root finder. The second derivative is then evaluated at each root position to classify the stationary points as local minima, local maxima, or plateau-like points.

Usage

```
find_extrema(first_deriv, second_deriv, x, tol = 1e-05, max_iter = 100L)
```

Arguments

first_deriv	A function representing the first derivative of the original curve. It must accept a numeric vector and return a numeric vector of the same length.
second_deriv	A function representing the second derivative of the original curve. It must accept a numeric vector and return a numeric vector of the same length.
x	A strictly increasing numeric vector defining the search range for root detection.
tol	A positive numeric scalar used as the numerical tolerance for root finding and second-derivative-based classification. Default is 1e-5.
max_iter	A positive numeric scalar specifying the maximum number of iterations used by the root-finding procedure. Default is 100L.

Details

Stationary points are classified according to the second derivative value at each root of the first derivative:

- second_deriv(x_root) > tol: local minimum
- second_deriv(x_root) < -tol: local maximum
- abs(second_deriv(x_root)) <= tol: plateau-like stationary point

Root finding is performed by the package's internal C++ backend, which returns numeric root positions within the range of x.

Value

A list with three elements:

local_min Numeric vector of x positions classified as local minima.

local_max Numeric vector of x positions classified as local maxima.

plateau Numeric vector of x positions classified as plateau-like stationary points.

If no root is found, or if no point belongs to a given category, NA_real_ is returned for that element.

Examples

```
f_d1 <- function(x) 2 * x
f_d2 <- function(x) rep(2, length(x))

x <- seq(-5, 5, length.out = 101)

find_extrema(
  first_deriv = f_d1,
  second_deriv = f_d2,
  x = x
)
```

`find_extrema_fast` *Fast Local Extrema Detection by Sign Changes*

Description

Detect local extrema from discrete signal data using sign changes in the first difference.

This function identifies local maxima and minima directly from discrete `x` and `y` values. A local maximum is detected when the sign of the first difference changes from positive to negative, and a local minimum is detected when the sign changes from negative to positive.

Flat-top and flat-bottom plateau regions are also handled. A flat-top plateau, represented by a positive slope followed by one or more zero differences and then a negative slope, is classified as a local maximum. A flat-bottom plateau, represented by a negative slope followed by one or more zero differences and then a positive slope, is classified as a local minimum.

Compared with derivative-based extrema detection, this function does not require continuous derivative functions. It is intended as a fast candidate detection method for discrete signals such as mass spectrometry spectra.

Usage

```
find_extrema_fast(x, y, plateau = c("middle", "first", "last"), na_rm = TRUE)
```

Arguments

`x`	A numeric vector representing the x-axis values, such as m/z values.
`y`	A numeric vector representing the signal intensity values.
`plateau`	A character string specifying which point should represent a plateau-type extremum. One of `"middle"`, `"first"`, or `"last"`. `"first"` uses the first point of the plateau, `"last"` uses the last point of the plateau, and `"middle"` uses the middle point of the plateau. Default is `"middle"`.
`na_rm`	Logical. If `TRUE`, pairs with `NA` in either `x` or `y` are removed before extrema detection. Default is `TRUE`.

Details

The detection rule is based on the sign of `diff(y)`:

- positive to negative sign change: local maximum
- negative to positive sign change: local minimum
- positive to zero to negative sign pattern: flat-top local maximum
- negative to zero to positive sign pattern: flat-bottom local minimum

Boundary plateaus are not classified because their left or right slope cannot be determined safely.

This function is fast because it only uses first differences and sign changes. However, it can be sensitive to local noise. For noisy signals, smoothing or additional filtering using intensity, signal-to-noise ratio, prominence, peak width, or statistical significance is recommended after candidate detection.

The output format is intentionally matched to `find_extrema()` as a named list. Local minima and local maxima are returned separately as `local_min` and `local_max`. Elements with no detected values are returned as `NA_real_`.

Value

A named list with the following elements:

`local_min` A numeric vector containing the x-coordinates of detected local minima. Flat-bottom plateau regions are included as local minima. If no local minimum is detected, `NA_real_` is returned.

`local_max` A numeric vector containing the x-coordinates of detected local maxima. Flat-top plateau regions are included as local maxima. If no local maximum is detected, `NA_real_` is returned.

Examples

```
x <- seq(0, 2 * pi, length.out = 101)
y <- sin(x)

find_extrema_fast(
  x = x,
  y = y
)

# Flat-top plateau example
x2 <- 1:6
y2 <- c(1, 3, 5, 5, 5, 2)

find_extrema_fast(
  x = x2,
  y = y2,
  plateau = "middle"
)

# With noisy signal
set.seed(1)
y_noise <- sin(x) + rnorm(length(x), sd = 0.05)

find_extrema_fast(
  x = x,
  y = y_noise,
  plateau = "middle"
)
```

`find_frequent_mz` *Find Frequent m/z Values from a List of Peak Tables*

Description

Identifies frequently observed m/z values across a cohort of samples. Each sample is represented by a peak table with m/z in the first column and intensity in the second column. The pooled m/z values are scanned with overlapping bins, and each bin location is refined to the strongest local maximum of a Gaussian kernel density estimate (KDE) using `get_gauss_kde()` and `find_extrema()`.

Usage

```
find_frequent_mz(
  peaks_list,
  bin_width = 20,
  exclude_mz = NULL,
  hws_exclude = bin_width/2
)
```

Arguments

`peaks_list`	A list of peak tables. Each element must be a matrix or data frame with at least two columns: m/z in the first column and intensity in the second column.
`bin_width`	A positive finite numeric scalar specifying the bin width used for the initial scan. The default is `20`.
`exclude_mz`	A numeric vector of m/z values to remove from the result, for example known contaminant or calibrant peaks. For each value, refined m/z values within `[exclude_mz - hws_exclude,` are removed. If `NULL`, no values are excluded. The default is `NULL`.
`hws_exclude`	A non-negative finite numeric scalar specifying the half-window size, in m/z units, used to exclude values around each `exclude_mz`. A value of `0` removes only exact matches. The default is `bin_width / 2`.

Details

The function performs the following steps:

1. Pool the m/z and intensity values across all peak tables and drop any non-finite observations.
2. Scan the pooled m/z range with three overlapping sets of bins of width `bin_width` to reduce edge effects.
3. For each bin, build a histogram, refine the m/z location as the strongest local maximum of a Gaussian KDE, and re-count pooled observations within `[refined_mz - bin_width, refined_mz + bin_width]`.
4. Drop duplicate refined m/z values produced by the overlapping bins.
5. Remove refined m/z values within `[exclude_mz - hws_exclude, exclude_mz + hws_exclude]` of any value listed in `exclude_mz`.

The returned table is sorted in ascending order of `mz`.

Value

A data frame with columns:

- `mz`: density-refined frequent m/z value.
- `median_intensity`: geometric median intensity (via `log10`) of pooled observations within `[mz - bin_width, mz + bin_width]`.
- `count`: number of pooled observations within `[mz - bin_width, mz + bin_width]`.
- `freq_ratio`: `count / length(peaks_list)`.

If no frequent m/z values remain, an empty data frame with the same columns is returned.

See Also

`filter_overlap_mz()`, `find_peaks()`, `get_gauss_kde()`, `find_extrema()`

Examples

```
set.seed(1)
peaks_list <- lapply(seq_len(20), function(i) {
  mz <- c(3000 + rnorm(1, sd = 1), 5000 + rnorm(1, sd = 1))
  data.frame(mz = mz, intensity = runif(2, 100, 1000))
})

find_frequent_mz(
  peaks_list = peaks_list,
  bin_width = 20,
  exclude_mz = NULL
)
```

---

find_peaks *Detect Ordinary Peaks and Shoulder Peaks Using Weighted Curvature*

Description

Detects ordinary peaks and shoulder peaks from a smoothed spectrum. Ordinary peaks are identified from the local maxima of a Gaussian kernel density estimate (KDE). Shoulder peaks are inferred from strong local maxima of the weighted reverse-signed curvature that are not located near ordinary KDE peaks.

Usage

```
find_peaks(
  data,
  bw = stats::median(diff(data[, 1])),
  hws_peaks = 10,
  merge_tol = NULL,
  tol = 1e-05,
```

```
  max_iter = 100L,
  weight_type = c("raw", "sqrt", "log10", "none"),
  hws_grid_kappa_smooth = 3:20,
  cutoff_kappa_peak_strength = 0.5,
  peak_retention_fraction = 0.25
)

find_peaks_spectra(
  spectra,
  bw = NULL,
  hws_peaks = 10,
  merge_tol = NULL,
  tol = 1e-05,
  max_iter = 100L,
  weight_type = c("raw", "sqrt", "log10", "none"),
  hws_grid_kappa_smooth = 3:20,
  cutoff_kappa_peak_strength = 0.5,
  peak_retention_fraction = 0.25,
  n_cores = 1L
)
```

Arguments

`data`	A numeric data frame or matrix with at least two columns. The first column must contain strictly increasing x-axis values, such as m/z values. The second column must contain intensity values.
`bw`	A positive numeric scalar specifying the bandwidth of the Gaussian KDE. The default is the median interval between adjacent x-axis values.
`hws_peaks`	A positive numeric scalar specifying the half-window size used by `find_peaks_fast()` to filter neighboring local maxima from the curvature profile. The value is interpreted as an x-axis distance. For MALDI-TOF MS spectra, the unit is typically m/z. For each local maximum, only the highest local maximum within `[x_peak - hws_peaks, x_peak + hws_peaks]` is retained.
`merge_tol`	A positive numeric scalar specifying the maximum x-axis distance within which two detected candidates are treated as nearby peaks. For MALDI-TOF MS spectra, the unit is typically m/z. If `NULL`, `hws_peaks` is used.
`tol`	A positive numeric scalar specifying the numerical tolerance used by `find_extrema()`.
`max_iter`	A positive integer specifying the maximum number of iterations used by `find_extrema()`.
`weight_type`	A character string specifying the intensity-weighting method applied to the reverse-signed curvature. One of: • `"none"`: do not apply intensity weighting. • `"raw"`: multiply the curvature by the non-negative KDE intensity. • `"sqrt"`: multiply the curvature by the square root of the non-negative KDE intensity. • `"log10"`: multiply the curvature by `log10(non-negative KDE intensity + 1)`.

`hws_grid_kappa_smooth`
: A numeric vector of positive integers specifying the half-window sizes of the Savitzky-Golay filters applied to the weighted curvature profile. Each value must be at least 2, because a third-order polynomial is used.

`cutoff_kappa_peak_strength`
: A non-negative numeric scalar specifying the upper bound of the curvature-peak-strength threshold. A curvature peak is retained as a shoulder candidate only when its strength is greater than the final cutoff.

`peak_retention_fraction`
: A numeric scalar in the interval `(0, 1]`. The curvature-peak-strength cutoff is calculated as the smaller value between `cutoff_kappa_peak_strength` and the `(1 - peak_retention_fraction)` quantile of the curvature peak strengths.

`spectra`
: A two-column `data.frame` or numeric `matrix` representing a single spectrum, or a `list` of such objects. The first column must contain strictly increasing x-axis values and the second column must contain intensity values.

`n_cores`
: A positive integer specifying the number of worker processes used when `spectra` is a list. The default is `1L`, which preserves sequential behaviour.

Details

The function performs the following steps:

1. Construct a Gaussian KDE and its first and second derivatives.
2. Calculate the signed curvature of the KDE.
3. Retain only the negative signed-curvature component and optionally apply intensity weighting.
4. Smooth the weighted curvature using multiple Savitzky-Golay filters and average the resulting profiles.
5. Detect strong local maxima from the averaged curvature profile as shoulder-peak candidates.
6. Detect ordinary KDE peaks from derivative-based extrema.
7. Remove curvature candidates that are located near ordinary KDE peaks.
8. Remove weaker candidates located near stronger candidates.

The reverse-signed curvature is defined as:

```
abs(pmin(0, curvature))
```

Therefore, only the concave-down component of the signed curvature contributes to shoulder-peak detection.

`hws_peaks` and `merge_tol` are both expressed as x-axis distances but serve different purposes. `hws_peaks` controls the local-maximum filtering window used by `find_peaks_fast()`, whereas `merge_tol` controls the distance used to merge nearby ordinary and shoulder-peak candidates.

`find_peaks_spectra()` applies `find_peaks()` to a single spectrum or to each spectrum in a list. When `bw` is `NULL`, the per-spectrum default bandwidth of `find_peaks()` (the median interval between adjacent x-axis values) is used for every spectrum. Supplying a non-`NULL` `bw` applies the same bandwidth to all spectra.

Value

A data frame with three columns:

- The first column contains the x-axis positions of the detected peaks. Its name is inherited from the first column of `data`.
- The second column contains KDE-estimated intensities. Its name is inherited from the second column of `data`.
- `type` indicates whether each detected feature is an ordinary KDE peak (`"peak"`) or a curvature-derived shoulder peak (`"shoulder"`).

The rows are sorted in ascending order of the x-axis values.

`find_peaks()` returns a single peak table. `find_peaks_spectra()` returns a single peak table when `spectra` is a single spectrum, or a named list of peak tables with the same list structure and names as `spectra`.

See Also

`find_peaks_fast()`, `find_extrema()`, `estimate_peak_strength()`

`find_peaks_spectra_fast()`, `preprocess_maldi_spectra()`, `filter_peaks_spectra()`

Examples

```
x <- seq(1000, 2000, length.out = 2000)
y <- dnorm(x, mean = 1400, sd = 20) * 100 +
     dnorm(x, mean = 1450, sd = 10) * 30 +
     dnorm(x, mean = 1700, sd = 25) * 80
spectrum <- data.frame(mz = x, intensity = y)

peaks <- find_peaks(
  data = spectrum,
  hws_peaks = 10,
  weight_type = "raw",
  hws_grid_kappa_smooth = 3:20,
  cutoff_kappa_peak_strength = 0.5,
  peak_retention_fraction = 0.25
)

head(peaks)

make_spectrum <- function() {
  x <- seq(1000, 2000, length.out = 500)
  y <- dnorm(x, mean = 1400, sd = 20) * 100 +
       dnorm(x, mean = 1700, sd = 25) * 80
  data.frame(mz = x, intensity = y)
}

spectra <- list(sample_1 = make_spectrum(), sample_2 = make_spectrum())

peaks_list <- find_peaks_spectra(
  spectra = spectra,
```

```
  hws_peaks = 10,
  weight_type = "raw",
  n_cores = 1
)

lapply(peaks_list, head)
```

`find_peaks_fast` *Fast Local Peak Detection from a Mass Spectrum*

Description

Detects local peaks from a two-column mass spectrum using a C++ backend. The function first identifies local maxima, including optional plateau handling, and then keeps only the highest local maximum within each `hws_peaks`-based window.

Usage

```
find_peaks_fast(
  data,
  hws_peaks = 10,
  plateau = c("middle", "first", "last"),
  na_rm = TRUE
)

find_peaks_spectra_fast(
  spectra,
  hws_peaks = 10,
  plateau = c("middle", "first", "last"),
  na_rm = TRUE,
  n_cores = 1L
)
```

Arguments

`data`	A data frame or matrix containing the mass spectrum. The first column is assumed to contain x values, usually m/z values, and the second column is assumed to contain y values, usually intensity values.
`hws_peaks`	Numeric. Half-window size used to filter neighboring local maxima. For each local maximum, only the highest local maximum within `x_peak - hws_peaks` and `x_peak + hws_peaks` is retained. Default is `10`.
`plateau`	Character. Method used to represent flat local maxima. One of `"middle"`, `"first"`, or `"last"`.

- `"middle"`: use the middle point of a plateau.
- `"first"`: use the first point of a plateau.

- "last": use the last point of a plateau.

na_rm — Logical. If TRUE, rows with missing values in the first two columns of data are removed before peak detection. Default is TRUE.

spectra — A two-column data.frame or numeric matrix representing a single spectrum, or a list of such objects. The first column is assumed to contain x values, usually m/z values, and the second column is assumed to contain y values, usually intensity values.

n_cores — A positive integer specifying the number of worker processes used when spectra is a list. The default is 1L, which preserves sequential behaviour.

Details

This function is an R wrapper around the C++ backend cpp_find_peaks_fast().

The input data is converted to a two-column numeric matrix before calling the C++ backend. The first column must be strictly increasing and must not contain duplicated values.

Local maxima are first detected from the intensity profile. Then, if multiple local maxima exist within the same hws_peaks-based x-window, only the local maximum with the highest intensity is retained.

This function is intended for fast peak candidate detection from mass spectrum-like one-dimensional signals.

find_peaks_spectra_fast() applies find_peaks_fast() to a single spectrum or to each spectrum in a list.

Value

A data frame with two columns. The column names are inherited from the first two columns of data. Each row represents one detected peak.

find_peaks_fast() returns a single peak table. find_peaks_spectra_fast() returns a single peak table when spectra is a single spectrum, or a named list of peak tables with the same list structure and names as spectra.

See Also

find_peaks_spectra(), preprocess_maldi_spectra(), filter_peaks_spectra()

Examples

```
x <- seq(1000, 2000, length.out = 5000)
y <- dnorm(x, mean = 1400, sd = 20) * 100 +
     dnorm(x, mean = 1450, sd = 10) * 30 +
     dnorm(x, mean = 1700, sd = 25) * 80

data <- data.frame(mz = x, intensity = y)

peaks <- find_peaks_fast(
  data = data,
  hws_peaks = 10,
  plateau = "middle"
```

```
)

head(peaks)

make_spectrum <- function() {
  x <- seq(1000, 2000, length.out = 2000)
  y <- dnorm(x, mean = 1400, sd = 20) * 100 +
       dnorm(x, mean = 1700, sd = 25) * 80
  data.frame(mz = x, intensity = y)
}

spectra <- list(sample_1 = make_spectrum(), sample_2 = make_spectrum())

peaks_list <- find_peaks_spectra_fast(spectra, hws_peaks = 10)

lapply(peaks_list, head)
```

`get_curvature_fun` *Create a Curvature Function from First and Second Derivative Functions*

Description

Creates a closure function that calculates the curvature of a one-dimensional curve $y = f(x)$ using the first and second derivative functions.

The returned function evaluates curvature at arbitrary x values. By default, it calculates the absolute curvature:

$$\kappa(x) = \frac{|f''(x)|}{(1 + [f'(x)]^2)^{3/2}}$$

If `absolute = FALSE`, the sign of the second derivative is retained:

$$\kappa_s(x) = \frac{f''(x)}{(1 + [f'(x)]^2)^{3/2}}$$

Usage

```
get_curvature_fun(first_deriv, second_deriv, absolute = TRUE)
```

Arguments

`first_deriv`	A function representing the first derivative $f'(x)$. It must take a numeric vector as input and return a numeric vector of the same length.
`second_deriv`	A function representing the second derivative $f''(x)$. It must take a numeric vector as input and return a numeric vector of the same length.
`absolute`	Logical. If `TRUE`, the absolute value of the second derivative is used in the numerator, returning non-negative curvature values. If `FALSE`, the sign of the second derivative is retained, returning a signed curvature-like value. Default is `TRUE`.

Details

This function returns a closure. The derivative functions `first_deriv` and `second_deriv`, as well as the `absolute` option, are stored inside the returned function environment. Therefore, the returned curvature function can be evaluated repeatedly without passing the derivative functions again.

The actual curvature calculation is performed by the internal C++ function `cpp_curvature()`.

In MALDI-TOF spectrum analysis, curvature can be used to quantify local peak sharpness or shape changes in a smoothed continuous representation of the spectrum. Absolute curvature is useful when only the magnitude of local bending is required, whereas signed curvature-like values can be useful when distinguishing concave-up and concave-down regions.

Value

A function that takes a numeric vector `x` and returns curvature values at `x`. If `absolute = TRUE`, the returned values are non-negative. If `absolute = FALSE`, the returned values retain the sign of the second derivative.

Examples

```
first_deriv <- function(x) 2 * x
second_deriv <- function(x) rep(2, length(x))

curvature_fun <- get_curvature_fun(
  first_deriv = first_deriv,
  second_deriv = second_deriv
)

curvature_fun(seq(-2, 2, length.out = 5))

signed_curvature_fun <- get_curvature_fun(
  first_deriv = first_deriv,
  second_deriv = second_deriv,
  absolute = FALSE
)

signed_curvature_fun(seq(-2, 2, length.out = 5))
```

---

get_gauss_kde *Create a Gaussian KDE Function for a Mass Spectrum*

Description

Creates a closure function for Gaussian kernel regression-based smoothing of a two-column mass spectrum. The returned function can be evaluated at arbitrary m/z values and can also return the first, second, or third derivative of the smoothed function.

The input spectrum is assumed to contain m/z values in the first column and intensity values in the second column. The observed m/z values, observed intensities, bandwidth, and derivative order are stored inside the returned function environment. Therefore, the returned function can be saved with `save()` and reused after `load()`, as long as the MALDIassist package is loaded.

Usage

```
get_gauss_kde(data, bw = 1, d = 0)
```

Arguments

data	A matrix or data frame with at least two columns. The first column should contain m/z values and the second column should contain intensity values.
bw	A positive numeric scalar specifying the bandwidth of the Gaussian kernel. Default is 1.
d	An integer-like scalar specifying the derivative order to return. 0 returns the smoothed Gaussian KDE function, 1 returns the first derivative, 2 returns the second derivative, and 3 returns the third derivative. Default is 0.

Details

This function returns a closure. The returned function stores x_obs, y_obs, bw, and d internally, so the user does not need to provide the original spectrum again when evaluating the function.

The actual numerical evaluation is performed by internal Rcpp functions: cpp_gaussKDE(), cpp_gaussKDE_1st_deriv(), cpp_gaussKDE_2nd_deriv(), or cpp_gaussKDE_3rd_deriv().

Value

A function that takes a numeric vector x and returns the estimated Gaussian KDE-smoothed intensity or its derivative at x.

Examples

```
mz <- seq(100, 200, by = 1)
intensity <- dnorm(mz, mean = 150, sd = 10) * 100
spectrum <- data.frame(mz = mz, intensity = intensity)

kde_fun <- get_gauss_kde(data = spectrum, bw = 1, d = 0)
kde_fun(c(140, 150, 160))

d1_fun <- get_gauss_kde(data = spectrum, bw = 1, d = 1)
d1_fun(c(140, 150, 160))
```

---

heatmap_matched_matrix

Heatmap of a Sample-by-Marker Matched-Peak Matrix

Description

Draws a clustered heatmap of a sample-by-marker matched-peak matrix, such as the detected_matrix or delta_mz_matrix returned by build_matched_matrix(), with optional group annotation and a zero-centered diverging palette. Requires the suggested package pheatmap.

Usage

```
heatmap_matched_matrix(
  matched_matrix,
  row_cluster = TRUE,
  col_cluster = TRUE,
  clustering_method = "ward.D2",
  group = NULL,
  title = "Matched peaks heatmap",
  center_at_zero = TRUE,
  hide_rownames = FALSE,
  hide_colnames = FALSE
)
```

Arguments

matched_matrix	A numeric matrix with samples as rows and markers as columns, such as the detected_matrix or delta_mz_matrix element of build_matched_matrix().
row_cluster	Logical indicating whether to cluster rows. The default is TRUE.
col_cluster	Logical indicating whether to cluster columns. The default is TRUE.
clustering_method	Hierarchical clustering linkage method passed to pheatmap / stats::hclust(). One of "ward.D2" (default), "ward.D", "complete", "average", "single", "mcquitty", "median", or "centroid". Applied to both rows and columns when clustering is enabled.
group	Optional group labels for sample annotation. When supplied, group colors use a fixed colorspace Viridis palette.
title	Optional plot title.
center_at_zero	Logical. When TRUE (default), uses a diverging blue-white-red palette with limits symmetric around 0.
hide_rownames	Logical. When TRUE, sample (row) labels are hidden. The default is FALSE.
hide_colnames	Logical. When TRUE, marker (column) labels are hidden. The default is FALSE.

Details

Missing values (NA) are shown as grey cells via pheatmap's na_col. When center_at_zero = TRUE, finite values are mapped with equal positive and negative limits so that 0 sits at the neutral center color. This is especially useful for delta_mz_matrix matrices. When the matrix contains many NA values, consider setting row_cluster = FALSE or col_cluster = FALSE because clustering may be unstable with sparse data. The default linkage is "ward.D2" (previously pheatmap's "complete").

Value

A pheatmap object.

See Also

build_matched_matrix()

Examples

```
if (requireNamespace("pheatmap", quietly = TRUE)) {
  set.seed(1)
  m <- matrix(rnorm(40), nrow = 5)
  rownames(m) <- paste0("sample_", 1:5)
  colnames(m) <- paste0("mz_", 1:8)

  heatmap_matched_matrix(m, title = "Example heatmap")
}
```

---

load_maldi_spectra *Load Bruker MALDI-TOF MS Spectra*

Description

Loads Bruker MALDI-TOF MS spectra from a directory containing Bruker flex data files.

This function searches for spectrum directories containing Bruker `fid` files, loads each spectrum using the internal `.load_spectrum()` function, assigns sample names based on the directory structure, and returns a named list of raw spectra.

The loading procedure is designed for Bruker MALDI-TOF MS directory structures and follows the general concept of `readBrukerFlexData`.

Usage

```
load_maldi_spectra(spectra_dir, return_dir = NULL)
```

Arguments

`spectra_dir`	Character. Path to the directory containing Bruker MALDI-TOF MS spectra.
`return_dir`	Deprecated. Ignored; retained for backward compatibility.

Details

The function normalizes `spectra_dir`, then searches recursively for Bruker `1SLin` directories using the internal `.search_fid_dir()` function and loads each spectrum from its directory path.

Sample names are inferred from directories named `"1SLin"` in the Bruker directory hierarchy. If duplicate sample names are detected, parent directory names are appended to make the names more distinguishable.

The bundled example directory `system.file("extdata", "bruker_sample", package = "MALDIassist")` contains one Bruker spectrum from PRIDE project PXD058284 (sample 9); see `inst/extdata/README` in the source package.

This function assumes that the internal helper functions `.search_fid_dir()` and `.load_spectrum()` are available.

Value

A named list of raw spectra. Each element corresponds to one loaded MALDI-TOF MS spectrum. The list names are generated from the sample directory structure and sorted using `gtools::mixedsort()`.

See Also

`gtools::mixedsort`

Examples

```
spectra_dir <- system.file("extdata", "bruker_sample", package = "MALDIassist")
raw_spectra <- load_maldi_spectra(spectra_dir = spectra_dir)
names(raw_spectra)
```

---

match_peaks *Match Detected Peaks to Reference m/z Values*

Description

For each reference m/z value, searches detected peaks within a symmetric half-window and selects one peak per reference. Returns one row per reference m/z with match status and detected peak properties.

Usage

```
match_peaks(
  peaks,
  reference_mz,
  reference_names = NULL,
  hws_match = 10,
  peak_selection_mode = c("nearest_mz", "maximum_intensity")
)
```

Arguments

`peaks`	A numeric matrix or data frame containing detected peaks. The first column must contain peak positions, typically m/z values, and the second column must contain the corresponding peak intensities.
`reference_mz`	A numeric vector of reference m/z values to match against.
`reference_names`	A character vector of names corresponding to `reference_mz`. If `NULL`, `NA` is used for all reference names. The default is `NULL`.
`hws_match`	A positive finite numeric scalar specifying the half-window size used for matching. For each reference m/z, only peaks within `[reference_mz - hws_match, reference_mz + hws_mat` are considered. The default is `10`.
`peak_selection_mode`	Character string specifying how a peak is chosen when multiple peaks fall within the matching window. Either `"nearest_mz"` (default) or `"maximum_intensity"`.

Details

When multiple peaks fall within the matching window, selection depends on `peak_selection_mode`:

- `"nearest_mz"`: the peak whose m/z is closest to `reference_mz`.
- `"maximum_intensity"`: the peak with the highest intensity in the second column of `peaks`.

If `peaks` is not sorted by m/z, it is sorted internally before matching.

Value

A data frame with one row per reference m/z, sorted in ascending order of `reference_mz`. Columns are:

- `reference_name`: reference label from `reference_names`, or `NA`.
- `reference_mz`: reference m/z value.
- `is_matched`: logical flag indicating whether a peak was found in the matching window.
- `detected_mz`: m/z of the matched peak, or `NA` when unmatched.
- `detected_intensity`: intensity of the matched peak, or `NA` when unmatched.
- `delta_mz`: signed m/z difference, `detected_mz - reference_mz`, or `NA` when unmatched.

Examples

```
peaks <- data.frame(
  mz = c(1000, 2000.5, 3000),
  intensity = c(10, 50, 20)
)

match_peaks(
  peaks = peaks,
  reference_mz = c(999, 2005, 4000),
  reference_names = c("A", "B", "C"),
  hws_match = 10
)
```

---

preprocess_maldi_spectra

Preprocess MALDI-TOF Mass Spectra

Description

Applies Savitzky-Golay smoothing and baseline subtraction to one or more MALDI-TOF mass spectra.

Usage

```
preprocess_maldi_spectra(
  spectra,
  hws_sg = 10L,
  pno_sg = 3L,
  baseline_type = c("snip", "tophat"),
  iter_snip = 50L,
  hws_tophat = 50L,
  n_cores = 1L
)
```

Arguments

spectra	A two-column data.frame or numeric matrix representing a single spectrum, or a list of such objects. The first column must contain strictly increasing m/z values and the second column must contain intensity values.
hws_sg	A positive integer specifying the half-window size used for Savitzky-Golay smoothing. The full smoothing window size is 2 * hws_sg + 1.
pno_sg	A non-negative integer specifying the polynomial order used for Savitzky-Golay smoothing. It must be smaller than the full smoothing window size.
baseline_type	A character string specifying the baseline subtraction method. One of "snip" or "tophat".
iter_snip	A positive integer specifying the number of iterations used for SNIP baseline estimation.
hws_tophat	A positive integer specifying the half-window size used for TopHat baseline estimation.
n_cores	Number of worker processes used when spectra is a list. The default is 1L, which preserves sequential behaviour.

Details

Each spectrum is smoothed using smooth_savitzky_golay() and subsequently baseline-corrected using subtract_baseline(). Negative values produced during baseline subtraction are truncated according to nonnegative_baseline = TRUE.

Value

A preprocessed spectrum or a list of preprocessed spectra. The returned object has the same list structure and names as spectra.

Examples

```
set.seed(1)
spectrum <- data.frame(
  mz = seq(2000, 8000, by = 1),
  intensity = runif(6001)
)
```

```
pp_spectrum <- preprocess_maldi_spectra(spectrum)

spectra <- list(sample_1 = spectrum, sample_2 = spectrum)

pp_spectra <- preprocess_maldi_spectra(
  spectra = spectra,
  hws_sg = 10L,
  pno_sg = 3L,
  baseline_type = "snip",
  iter_snip = 50L
)
```

`smooth_savitzky_golay` *Smooth a Spectrum Using the Savitzky-Golay Filter*

Description

Applies Savitzky-Golay smoothing to a two-column spectrum using a C++ backend.

The first column of `data` is treated as the x-axis, usually m/z values, and the second column is treated as the signal intensity. Only the intensity values are smoothed; the x-axis values are returned unchanged.

Usage

```
smooth_savitzky_golay(data, hws = 10L, pno = 3L)
```

Arguments

`data`	A matrix or data frame containing a two-column spectrum. The first column is assumed to contain x values, usually m/z values, and the second column is assumed to contain y values, usually intensity values.
`hws`	Integer-like numeric value. Half-window size for the Savitzky-Golay filter. The full window size is `2 * hws + 1`. Default is `10L`.
`pno`	Integer-like numeric value. Polynomial order used for local polynomial fitting. Default is `3L`.

Details

This function is an R wrapper around the C++ backend `cpp_savitzkyGolay_filter()`.

Boundary regions are handled using boundary-specific Savitzky-Golay coefficients. The center region is smoothed using the central filter coefficients.

The full window size, `2 * hws + 1`, must be larger than the polynomial order `pno`. The number of rows in `data` must be greater than or equal to the full window size.

Value

A data frame with two columns. The first column contains the original x values, and the second column contains the smoothed y values. Column names are inherited from the first two columns of `data`.

Examples

```
x <- seq(1000, 2000, length.out = 5000)
y <- dnorm(x, mean = 1500, sd = 30) * 100 +
     rnorm(length(x), sd = 2)

data <- data.frame(mz = x, intensity = y)

smoothed <- smooth_savitzky_golay(
  data = data,
  hws = 10,
  pno = 3
)

head(smoothed)
```

---

subtract_baseline *Subtract Baseline from a Mass Spectrum*

Description

Estimates and subtracts the baseline from a two-column mass spectrum using either the SNIP or TopHat baseline correction method.

The first column of `data` is treated as the x-axis, usually m/z values, and the second column is treated as the signal intensity. The original column names of the first two columns in `data` are preserved in the returned raw and baseline-subtracted spectra.

Usage

```
subtract_baseline(
  data,
  baseline_type = c("snip", "tophat"),
  iter_snip = 50L,
  hws_tophat = 50L,
  nonnegative_baseline = TRUE
)
```

Arguments

`data`	A matrix or data frame containing a two-column spectrum. The first column is assumed to contain x values, usually m/z values, and the second column is assumed to contain y values, usually intensity values.

baseline_type Character. Baseline correction method. One of "snip" or "tophat". Default is "snip".

iter_snip Integer-like numeric value. Number of iterations used for SNIP baseline estimation. Used only when baseline_type = "snip". Default is 50L.

hws_tophat Integer-like numeric value. Half-window size used for TopHat baseline estimation. Used only when baseline_type = "tophat". Default is 50L.

nonnegative_baseline Logical. If TRUE, the estimated baseline is constrained to be non-negative inside the C++ baseline estimation backend. Default is TRUE.

Details

This function is an R wrapper around the C++ baseline estimation functions cpp_SNIP() and cpp_TopHat().

If baseline_type = "snip", the baseline is estimated using cpp_SNIP(). If baseline_type = "tophat", the baseline is estimated using cpp_TopHat().

The argument nonnegative_baseline controls whether the estimated baseline itself is constrained to be non-negative. This is different from the final baseline-subtracted signal, which is clipped to be non-negative by applying:

$$y_{corrected} = max(y - baseline, 0)$$

Therefore, the returned baseline-subtracted intensity is always non-negative.

This function assumes that cpp_SNIP() and cpp_TopHat() are available.

Value

A list with four elements:

- raw_data: A data frame containing the original spectrum. Column names are inherited from the first two columns of data.
- subtracted_data: A data frame containing the baseline-subtracted spectrum. Column names are inherited from the first two columns of data.
- baseline: A numeric vector containing the estimated baseline.
- param: A list containing the parameters used for baseline subtraction.

Examples

```
x <- seq(1000, 2000, length.out = 5000)
y <- dnorm(x, mean = 1500, sd = 30) * 100 +
     seq(20, 50, length.out = length(x))

data <- data.frame(mz = x, intensity = y)

result <- subtract_baseline(
  data = data,
  baseline_type = "snip",
```

```
  iter_snip = 50,
  nonnegative_baseline = TRUE
)

head(result$subtracted_data)
```

`visualize_spectra` *Visualize One or More Overlaid Mass Spectra*

Description

Draws one or more mass spectra as overlaid base-R line plots, colored by a sequential palette. No legend is drawn.

Usage

```
visualize_spectra(
  spectra,
  interest_range = NULL,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  cex.axis = 1,
  cex.lab = 1,
  cex.main = 1.3,
  lwd = 1.5,
  col_platte = c("Viridis", "YlOrRd")
)
```

Arguments

`spectra`	A list of spectrum matrices/data frames, each with m/z in the first column and intensity in the second.
`interest_range`	Optional numeric vector giving an m/z range to restrict the spectra to before plotting.
`xlim`	Optional numeric vector of length two limiting the x-axis range.
`ylim`	Optional numeric vector of length two limiting the y-axis range.
`main`	Optional plot title.
`cex.axis`	Axis annotation scaling factor.
`cex.lab`	Axis label scaling factor.
`cex.main`	Title scaling factor.
`lwd`	Line width of the spectra traces.
`col_platte`	Sequential `colorspace` palette used to color the traces.

Value

Invisibly NULL; called for the side effect of drawing a plot.

See Also

visualize_spectrum()

Examples

```
if (requireNamespace("colorspace", quietly = TRUE)) {
  x <- seq(1000, 2000, length.out = 1000)
  spectra <- list(
    sample_1 = data.frame(mz = x, intensity = dnorm(x, 1400, 25) * 100),
    sample_2 = data.frame(mz = x, intensity = dnorm(x, 1600, 25) * 80)
  )

  visualize_spectra(spectra, main = "Example spectra")
}
```

---

visualize_spectrum *Visualize a Single Mass Spectrum with Optional Peak Segments*

Description

Draws a single mass spectrum as a base-R line plot. When a peak table is supplied, vertical segments are drawn from zero to each peak intensity, with an option to annotate the most intense peaks.

Usage

```
visualize_spectrum(
  spectrum,
  peaks = NULL,
  interest_range = NULL,
  annotate_topN = FALSE,
  topN = 10,
  xlim = NULL,
  ylim = NULL,
  main = NULL,
  cex.axis = 1,
  cex.lab = 1,
  cex.main = 1.3,
  lwd = 1,
  col = "black",
  peaks_lwd = 2,
  peaks_col = "red",
  peaks_lty = 3
)
```

Arguments

spectrum	A numeric matrix or data frame with m/z in the first column and intensity in the second column.
peaks	Optional peak matrix or data frame. When supplied, vertical peak segments are drawn from zero to each peak intensity.
interest_range	Optional numeric vector giving an m/z range to restrict the spectrum (and peaks) to before plotting.
annotate_topN	Logical. When TRUE, the most intense peaks are labelled with their m/z values. The default is FALSE.
topN	Number of peaks to annotate when annotate_topN = TRUE.
xlim	Optional numeric vector of length two limiting the x-axis range.
ylim	Optional numeric vector of length two limiting the y-axis range.
main	Optional plot title.
cex.axis	Axis annotation scaling factor.
cex.lab	Axis label scaling factor.
cex.main	Title scaling factor.
lwd	Line width of the spectrum trace.
col	Line color of the spectrum trace.
peaks_lwd	Line width of the peak segments.
peaks_col	Color of the peak segments.
peaks_lty	Line type of the peak segments.

Value

Invisibly NULL; called for the side effect of drawing a plot.

See Also

visualize_spectra(), find_peaks()

Examples

```
x <- seq(1000, 2000, length.out = 1000)
y <- dnorm(x, mean = 1500, sd = 25) * 100
spectrum <- data.frame(mz = x, intensity = y)

visualize_spectrum(spectrum, main = "Example spectrum")
```

Index