Package 'ImperialNMRTool'

April 19, 2023

Type Package

Title Tool for Nuclear Magnetic Resonance based spectrography.

Description Workflow for automatic processing and annotation of 1D 1H-NMR spectra from MetaboLights repository.
Entrypoint to the workflow is hurricane.r, and it takes an argument which is the location of a parameters file, which itself contains the location of the files the workflow needs, and the workflow parameters. To read more visit the github repository at github.com/EBI-Metabolights/icl_nmr_R

Version 0.2.0.0

Date 2022-09-12

Maintainer Callum Martin <cmartin@ebi.ac.uk>

Depends R (>= 4.0.0)

Remotes tkimhofer/metabom8

Imports optparse, stringr, yaml, pbapply, devtools, batch, flexclust, rlang, pacman, ggplot2, scales, purrr, matrixStats,
pracma, RcppHungarian, grid, lattice, modeltools, ggnewscale, colorRamps, Metrics, transport, fftw, gridExtra, umap,
tictoc, apcluster, plotly, shiny, ggridges, foreach, coop, doParallel, fpc, plyr, reshape2

License GPL-2

RoxygenNote 7.2.1

R topics documented:

`align.max` *Aligns the features in the input matrix based on their maximum peaks*

Description

Aligns the features in the input matrix based on their maximum peaks

Usage

```
align.max(feature, scaling = TRUE)
```

Arguments

`feature`	a list object containing a matrix of feature values and a matrix of their corresponding positions
`scaling`	a logical value indicating whether to scale the feature values to between 0 and 1 based on the highest peak's intensity

Value

a list object containing

`backfit_ref.feats.2.subset.specs` *Back-fit reference features to a subset of spectra*

Description

Back-fit reference features to a subset of spectra

Usage

```
backfit_ref.feats.2.subset.specs(
  m.inds,
  fits.feature,
  match.info,
  feature,
  xmat,
  ppm,
  plots = T
)
```

Arguments

m.inds	A vector of indices corresponding to the matches between reference features and spectra. Necessary to allow for subsetting of matches (i.e. because of filtering)
fits.feature	A list of fitted features (to reference spectra)
match.info	A data frame containing information about the matches between features and reference spectra
feature	List containing feature intensities and positions
xmat	Full spectral matrix from which the features were derived
ppm	A vector containing the ppm axis of the spectra
plots	A logical value indicating whether to generate plots of the fits and back-fit feasibility scores - only use for a couple of backfits at a time as plots will double the weight of the result

Value

A list containing the back-fits and back-fit feasibility scores for each spectrum in the subset, as well as a grid plot of the fits and back-fit feasibility scores.

---

bad.peaks *Calculate how often each feature point is fit across the reference database*

Description

Get weights for feature profile points according to how often they fit against reference DB peaks. mean of the useful to scale this by some exponential (e.g. 2) to select extremely high percentages lapply(allmatches.fits, function(x) x$overshoot.pct.feat)

Usage

```
bad.peaks(pos.res.pct.feat, scale.exp = 2)
```

Arguments

pos.res.pct.feat	matrix of positive residuals for a feature to all references fit as a percent of the feature resonance height
scale.exp	a scaling exponent for the weights to select for values very close to 1 (default is 2)

Value

A vector of weights, one for each point in the feature profile, indicating how rarely the feature point was fit in the ref DB matches the which can be (e.g.) multiplied by and then subtracted from feature to squash not-never-fit peaks in the feature, or (1-result) * rmse values to downweight the effect of not-never-fit peaks (those not represented in the ref DB, and more likely to be false positives in feature extraction).

`clusts2labs` *Convert clusters to feature-sorted cluster labels*

Description

Given a list of clustered feature inds in list elements, this function returns feature-sorted cluster labels.

Usage

```
clusts2labs(clusters)
```

Arguments

`clusters`	A list of cluster-feature pairs.

Value

A numeric vector of feature-sorted cluster labels.

Author(s)

MTJ

`corrPocketPairs_al` *corrPocketPairs*

Description

corrPocketPairs

Usage

```
corrPocketPairs_al(
  x,
  ppm,
  ws,
  reg = NULL,
  plotHeatmap = FALSE,
  wdlimit = 0.99,
  rcutoff = 0.75
)
```

Arguments

x	a matrix with numerical values.
ppm	a numeric vector of the same length as ncol(x) specifying the ppm for each column.
ws	an integer specifying half the size of the sliding window used to calculate correlations.
reg	an optional vector specifying the column indices to consider in x.
plotHeatmap	a logical indicating whether to plot a heatmap of the correlation matrix.
wdlimit	a numeric (0,1) specifying the minimum fraction of sliding windows that need to contain a pocket to consider a point noise.
rcutoff	a numeric (0,1) specifying the correlation coefficient cutoff to use when extracting significant peaks.

Value

a list with the following elements:

regions	a matrix specifying the indices of the peaks in each column of the original matrix, filtered to exclude peaks that are too small or only have unidirectional interactions.
corr	a matrix of correlation coefficients between columns of x.
cov	a matrix of covariance values between columns of x.
peakMap	a matrix specifying the indices of the peaks in each column of the correlation matrix.
noiseDist	the proportion of pockets containing each window index.

---

corr_expand *Expand a peak region to the closest correlation minimums*

Description

This function takes a correlation profile vector and an index of a peak, and returns the bounds of the region containing the peak, extended to the closest local minima on either side of the peak.

Usage

```
corr_expand(peak, localMins, vRange)
```

Arguments

peak	An integer representing the index of the peak in the correlation vector
localMins	A numeric vector containing the indices of local minima in the correlation vector
vRange	A numeric vector containing the range of valid indices for the correlation vector

Value

A list with two elements: lower, the index of the closest local minimum to the left of the peak, and upper, the index of the closest local minimum to the right of the peak

Examples

```
v <- c(1, 2, 3, 2, 1)
localMins <- c(1, 3, 5)
corr_expand(peak = 3, localMins = localMins, vRange = c(1, 5))
# Returns: list(lower = 1, upper = 5)
```

`corr_expand_window` *Expand a correlation window around a given peak index to the nearest correlation minima outside of the window bounds.*

Description

Expand a correlation window around a given peak index to the nearest correlation minima outside of the window bounds.

Usage

```
corr_expand_window(xmat, ppm, peakInd, wind)
```

Arguments

`xmat`	A matrix of NMR spectra.
`ppm`	A vector of ppm values corresponding to the columns of xmat.
`peakInd`	The index of the peak around which to expand the window.
`wind`	A vector defining the window (column inds) around the peak.

Value

A list with the following elements:

`peak`	The index of the peak used for window expansion.
`intcorr`	The integral of the correlation within the expanded window.
`data`	A data frame containing the correlation values, covariance values, and corresponding ppm values within the expanded window.
`sr`	The STOCSY object for the expanded window.
`wind`	The original window vector.
`newWind`	The expanded window vector.
`corrRbound`	The index of the right boundary of the expanded window in the `wind` vector.
`corrLbound`	The index of the left boundary of the expanded window in the `wind` vector.

`detect.baseline.effect` *Detect baseline effect in feature profile (also indicates monotonic features, which are easy to get with STORM) Goes through a couple of checks: - is the peak prominence (max vs. lower bound) of at least one peak > prom.ratio x the entire feature intensity range - do the valleys and peaks in the feature profile correlate strongly (r > cutoff.corr)? If so, likely a baseline effect.*

Description

Detect baseline effect in feature profile (also indicates monotonic features, which are easy to get with STORM) Goes through a couple of checks: - is the peak prominence (max vs. lower bound) of at least one peak > prom.ratio x the entire feature intensity range - do the valleys and peaks in the feature profile correlate strongly (r > cutoff.corr)? If so, likely a baseline effect.

Usage

```
detect.baseline.effect(feat, cutoff.corr = 0.99, prom.ratio = 0.3)
```

Arguments

`feat`	A numeric vector containing feature data.
`cutoff.corr`	The maximum allowed correlation between predicted and observed peak intensities.
`prom.ratio`	The minimum required ratio of peak prominence to feature intensity range.

Value

A list containing two logical values. The first indicates whether the feature passed the peak prominence filter, and the second indicates whether the feature passed the peak fit filter.

`earlyOut` *Return early output for 'xcorr' if no comparisons should be computed*

Description

This function returns a list of NA values for 'rvals', 'pvals', 'lags', and 'bestFit', as well as the 'overlaps' values and 'dontCalc' boolean vector that caused the function to exit early.

Usage

```
earlyOut(dontCalc, overlaps)
```

Arguments

`dontCalc`	A boolean vector indicating which comparisons are not calculated
`overlaps`	A numeric vector of the overlaps between the reference and each shift of the spectrum

Value

A list containing NA values for ‘rvals‘, ‘pvals‘, ‘lags‘, and ‘bestFit‘, as well as the ‘overlaps‘ values and ‘dontCalc‘ boolean vector.

`errorOut` *Create an error output object with default values.*

Description

Create an error output object with default values.

Usage

```
errorOut()
```

Value

A list with the following elements:

best.shifts A numeric vector of length 1, set to 0.

best.rvals A numeric vector of length 1, set to 0.

best.pvals A numeric vector of length 1, set to 1.

shiftedInds A NULL object.

shiftedSpecs A NULL object.

ref.aligned A NULL object.

overlap.sizes A numeric vector of length 1, set to 0.

g A NULL object.

Examples

```
errorOut()
```

`expand_window` *Expand a window by a certain distance while keeping indices within a given range.*

Description

Expand a window by a certain distance while keeping indices within a given range.

Usage

```
expand_window(window, within, by = NULL)
```

Arguments

window	A numeric vector representing the original window to be expanded.
within	A numeric vector representing the range within which the expanded window should be contained.
by	A numeric scalar representing the amount by which to expand the window. If not specified, the default is to expand the window by the span of its original values.

Value

A numeric vector representing the expanded window.

Examples

```
# Expand a window with default arguments
expand_window(c(1, 3), 1:5)

# Expand a window with custom expansion distance
expand_window(c(1, 3), 1:5, by = 2)
```

---

extractPeaks_corr *Extract local maxima and minima from a correlation vector*

Description

This function takes a correlation vector and returns the indices of the local maxima and minima, and the bounds for the corresponding windows around each maximum.

Usage

```
extractPeaks_corr(corr, mask = NULL, plots = FALSE)
```

Arguments

corr	A correlation vector.
mask	A boolean mask for excluding points.
plots	Whether to plot the resulting peaks and bounds.

Value

A list containing the peak indices, the bounds of the windows around each peak, and the indices of any masked regions, and whether or not each peak is a true peak (i.e., require a lower point on each side)

Examples

```
corr <- rnorm(100)
peaks <- extractPeaks_corr(corr, plots = TRUE)
```

`feature_match2ref_slim` *Match a feature against a reference using FFT-based convolution and correlation picks the top max.hits peaks in the convolution, then calculates correlation and pvalue of the PCC at those points in the ref. Returns a dataframe with the xcorr information.*

Description

Match a feature against a reference using FFT-based convolution and correlation picks the top max.hits peaks in the convolution, then calculates correlation and pvalue of the PCC at those points in the ref. Returns a dataframe with the xcorr information.

Usage

```
feature_match2ref_slim(
  f.num,
  r.num,
  feat,
  ref,
  feat.ft.c,
  ref.ft,
  pad.size,
  max.hits = 5,
  r.thresh = 0.8,
  p.thresh = 0.01
)
```

Arguments

`f.num`	numeric: Feature number
`r.num`	numeric: Reference number
`feat`	numeric: Vector of feature intensities
`ref`	numeric: Vector of reference intensities
`feat.ft.c`	numeric: FFT of feature intensities, conjugated
`ref.ft`	numeric: FFT of reference intensities
`pad.size`	numeric: Size of the padding applied during FFT-based convolution
`max.hits`	numeric: Maximum number of candidate lags to consider
`r.thresh`	numeric: Correlation threshold for considering a match
`p.thresh`	numeric: P-value threshold for considering a match

Value

A data frame with the following columns: - feat: Feature number or identifier - ref: Reference number or identifier - lag: The lag that produces the highest correlation - rval: The correlation coefficient - pval: The p-value of the correlation - pts.matched: Number of data points used in the correlation - pts.feat: Number of data points in the feature vector - feat.start: The starting index of the feature vector used in the correlation - feat.end: The ending index of the feature vector used in the correlation - ref.start: The starting index of the reference vector used in the correlation - ref.end: The ending index of the reference vector used in the correlation

fillbetween *fillbetween*

Description

Calculate a sequence of values between two numeric indices (e.g., like ':'), with optional reversal.

Usage

```
fillbetween(inds)
```

Arguments

inds	a numeric vector of length 2 indicating the starting and ending indices

Value

a numeric vector representing the sequence of indices between the starting and ending indices

Examples

```
fillbetween(c(1, 5)) # returns 1 2 3 4 5
fillbetween(c(5, 1)) # returns 5 4 3 2 1
fillbetween(c(1, NA)) # throws an error
```

filter.matches *Match Filtering*

Description

Filter and process matched peaks based on user-specified criteria, such as singlet removal and ppm distance filtering. Also calculate backfits of ref-feats (reference subsignatures fished out by feature matches to reference spectra) to each individual dataset spectrum. Note: ref-feat fit is obtained using the original feature, then backfit feasibility scores are calculated. BFF scores indicate the extent to which ref-feat resonances do not exceed actual spectral signal. Specifically, each single ref-feat resonance positive residual is evaluated as a fraction of the total ref-feat height. Because the absence of a feasible fit for any single reference resonance invalidates a match, the worst-violating resonance gives the score for the whole ref-feat in a particular spectrum. Note that a ref-feat receives a BFF score for each spectrum it is fit to.

Usage

```
## S3 method for class 'matches'
filter(pars)
```

Arguments

pars	A list of input parameters.

Value

A list of filtered and processed matched peak information, including back-fits to the original spectra.

filter.matches_shiftDelta

Filter matches based on ppm shift difference

Description

This function filters the matches in match.info based on the difference in ppm shift between the feature and the match. Only matches with a ppm shift difference less than or equal to ppm.tol are retained.

Usage

```
## S3 method for class 'matches_shiftDelta'
filter(match.info, feature, ppm, fits.feature, ppm.tol = 0.5)
```

Arguments

match.info	A data frame containing the match information; produced in filter.matches()
feature	Object containing features' intensity matrix, position (column inds of spectral matrix) matrix, and other information about the features
ppm	A numeric vector of ppm values; feature$position values index this
fits.feature	List of feature fits to reference spectral regions
ppm.tol	The maximum allowed ppm shift difference between the feature and the match

Value

A list with two elements: the filtered ‘match.info‘ data frame, and the filtered ‘fits.feature‘ list which have been filtered by ppm shift difference.

filter.matches_singlets

Filter matches with singlets

Description

This function removes matches that have only one peak in their respective feature, reference or feature-not-never-fit regions.

Usage

```
## S3 method for class 'matches_singlets'
filter(
  match.info,
  fits.feature,
  peak.qualities,
  pq.featureNumbers,
  res.area.thresh
)
```

Arguments

match.info	The match information data.frame obtained within filter.matches.
fits.feature	A list of fitted features.
peak.qualities	A list of peak quality vectors (~ feature points' relevance in the reference database)
pq.featureNumbers	A vector of feature numbers to index the peak.qualities list. This is necessary when only a subset of features are matched and feature fits are filtered.
res.area.thresh	The minimum reference spectrum resonance area that must be accounted for by the fit feature in order to consider it matched.

Value

A list containing the filtered match information and the filtered fitted features.

---

filterFeatures *Filter features based on specified criteria - null features - features outside of ppm range - features with no runs >= min.runlength - features derived from < min.subset spectra - features with strong baseline effect*

Description

Filter features based on specified criteria - null features - features outside of ppm range - features with no runs >= min.runlength - features derived from < min.subset spectra - features with strong baseline effect

Usage

```
filterFeatures(
  feature,
  ppm,
  ppm.range,
  min.runlength = 3,
  min.subset = 5,
  prom.ratio = 0.3,
  give = "filter"
)
```

Arguments

feature	A feature object to be filtered
ppm	A vector of ppm values for each point in the spectra
ppm.range	A vector specifying the range of ppm values to include
min.runlength	The minimum length of a run of consecutive data points to include
min.subset	The minimum number of spectra a feature must be present in to be included
prom.ratio	The maximum ratio of peak prominence to intensity range for a feature to be considered monotonic
give	A character string indicating whether to return the filtered feature object or the filter as a logical vector

Value

If `give = "features"`, a feature object with the filtered features. If `give = "filter"`, a logical vector indicating which features passed the filter.

`fit.batman` *Fit a two-component model to a feature and a reference spectrum*

Description

Fit a two-component model to a feature and a reference spectrum

Usage

```
fit.batman(feat, spec, exclude.lowest = 0.5, ppm = NULL, plots = FALSE)
```

Arguments

`feat`	a numeric vector of the feature spectrum
`spec`	a numeric vector of the reference spectrum
`exclude.lowest`	the fraction of the lowest values to exclude from the ratio calculation (default is 0.5)
`ppm`	a numeric vector of the parts per million (ppm) values for the spectra (default is NULL)
`plots`	logical, whether to generate plots (default is FALSE)

Value

a list with the following elements:

- `feat.fit`: a numeric vector of the fitted feature spectrum
- `spec.fit`: a numeric vector of the fitted reference spectrum
- `keypoint`: the index of the keypoint where the ratio of the two spectra is minimum
- `ratio`: the ratio of the feature and reference spectra at the keypoint
- `intercept`: the intercept of the reference spectrum
- `residuals`: a numeric vector of the residuals between the fitted feature and reference spectra
- `plot`: a plot of the fitted feature and reference spectra (if plots = TRUE)

Examples

```
# Example usage:
feat <- c(1, 2, 3, 4, 5)
spec <- c(1, 2, 3, 4, 5)
fit.batman(feat, spec)
```

fit.leastSquares *Fit least squares model between two signals and calculate statistics*

Description

Fit least squares model between two signals and calculate statistics

Usage

```
fit.leastSquares(v1, v2, ppm = NULL, plots = FALSE)
```

Arguments

v1	a numeric vector representing one of the signals
v2	a numeric vector representing the other signal
ppm	a numeric vector representing the ppm values for the signals
plots	a logical value indicating whether to produce a plot (default is FALSE)

Value

A list with the following elements:

fit.minmax *Fit a feature to a spectrum using minmax scaling*

Description

Fit a feature to a spectrum using minmax scaling

Usage

```
fit.minmax(feature, spectrum)
```

Arguments

feature	a numeric vector representing the feature to be fit
spectrum	a numeric vector representing the spectrum to which the feature will be fit

Value

a list with the following components:

feature.fit a numeric vector representing the feature fit

ratio a ratio of the original feature intensity to the intensity of the feature after minmax scaling

fitFeature *Fit a feature to a spectrum*

Description

This function fits a feature to a spectrum using either least squares fitting or minmax scaling.

Usage

```
fitFeature(feature, spectrum, spectrum.position, method = "minmax scaling")
```

Arguments

feature	A numeric vector containing the feature to fit to the spectrum
spectrum	A numeric vector containing the spectrum to fit the feature to
spectrum.position	A numeric value specifying the position of the spectrum
method	A character string specifying the method to use for fitting. Can be "least squares" or "minmax scaling".

Value

A list containing the feature fit, the feature position, the ratio, the residuals, and the overfit

@export

fse *FSE Function*

Description

Extracts feature shapes from a matrix of NMR spectra using the FSE algorithm. # FSE: the spectral matrix is decomposed into compound features using feature shape extraction. First, a local STOCSY is performed at every point along the spectrum (within a sliding window of ~ 100 points; enough to capture multiple resonances within any multiplet). For each of these STOCSYs, the central peak in the correlation profile (correlation pocket; corrpocket) typically captures a resonance, as the correlation is 1 by definition at the STOCSY'd point, and typically falls off as you approach the boundaries of the resonance. This is taken, with the next highest correlation peak within the window, to form a rough statistical description of two resonances which have an correlation in intensity across samples, albeit separated by chemical shift. We term this a 'protofeature'. Importantly, each point and its associated window will capture the dominant protofeature most associated with that point. This changes for adjacent points, and many protofeatures will be duplicated multiple times. A protofeature should be considered as a rough hypothesis about a statistical association between two resonances, which happen to be sufficiently aligned so that they produce a coherent signal. If the windows are all aligned, we can plot the each window point. From this distribution, it is clear that nearly all protofeatures, including those from noise peaks and real peaks, include the most central window points. As such, these cannot reliably be used to identify noise. However, the correlation peak about noise tends to be much smaller, and characteristically so. As such, a noisewidth can be estimated from this distribution. This is the origin of the noise.percentile cutoff, which is

applied like so: "given a noise.percentile = 0.99, consider only those protofeatures for which both peaks have a width > the smallest 1 number therefore gives a more selective cut. Protofeatures are also filtered so that they must be bidirectional (i.e. both peaks must indicate each other as highest correlated), and must include runs > noise.width.

Usage

```
fse(pars)
```

Arguments

pars	A list of parameters for the function.

Details

STORM: Joram Posma's STORM has been adapted and optimized to accept these protofeatures in the following ways: - first, since many of the protofeatures are noise, we provide failure modes and reporting for the following cases: "empty subset", #' empty subset (no spectrum contains signature) "subset degenerated", #' 1-3 spectra in the subset (not enough spectra to get a reliable correlation) "reference degenerated", #' signature degenerates to include < 3 points (not meaningful to correlate shapes) "did not converge" #' subset continues to change after 24 iterations - additionally, the correlation r and p-value cutoff q are both used during both the subset selection and reference update steps. - we also remove any regions of the reference for which there are fewer than minpeak values after r and p value thresholding. This helps avoid noise.

STORM extracts meaningful features using protofeatures to define the region of interest and a rough sketch of the feature shape highly correlated with each spectral point. In the future, HCA could be used to cluster potential starting feature shapes correlated with each driver, or the nonoptimal subset for each point could be re-STORMed to detect any other feature shapes present. It is also perfectly reasonable to combine feature shapes from different STORM runs for a given dataset, as these comprise a list of somewhat independently tested feature shapes, and duplication is not an issue.

Value

A list of features extracted from the spectra.

---

fwhm *Full Width at Half Maximum (FWHM) calculation*

Description

Given an isolated peak, calculates its Full Width at Half Maximum (FWHM), i.e., the width of the peak at half of its maximum height.

Usage

```
fwhm(isolatedpk)
```

Arguments

isolatedpk	a numeric vector representing an isolated peak

Value

a numeric value representing the FWHM of the peak

`ind2subR` *Convert linear indices to row and column subscripts*

Description

This function takes a numeric vector of linear indices `inds` and a positive integer `m`, representing the number of columns in a matrix, and returns a list with two elements: `rows`, a vector of row subscripts corresponding to the linear indices, and `cols`, a vector of column subscripts corresponding to the linear indices. Meant to replicate MATLAB's ind2sub.

Usage

```
ind2subR(inds, m)
```

Arguments

`inds`	A numeric vector of linear indices from a matrix
`m`	A positive integer representing the number of rows in the corresponding matrix

Value

A list with two elements: `rows`, a vector of row subscripts corresponding to the linear indices, and `cols`, a vector of column subscripts corresponding to the linear indices

Examples

```
inds <- 1:9
m <- 4
ind2subR(inds, m)
# Returns: list(rows = c(1, 2, 3, 4, 1, 2, 3, 4, 1), cols = c(1, 1, 1, 1, 2, 2, 2, 2, 3))
```

`is.true.peak` *Label True Peak points in a vector using extractPeaks_corr (wrapper)*

Description

Label True Peak points in a vector using extractPeaks_corr (wrapper)

Usage

```
is.true.peak(v)
```

Arguments

`v`	a vector of intensities

Value

a logical vector indicating the true peaks and tail regions

Examples

```
v <- rnorm(100)
is.true.peak(v)
```

`keep_inds_in_bounds` *keep_inds_in_bounds.R*

Description

Given a set of indices to check and a set of indices to check against, returns the subset of check that lies within the bounds of against.

Usage

```
keep_inds_in_bounds(check, against)
```

Arguments

`check`	A numeric vector of indices to check.
`against`	A numeric vector of indices to check against.

Value

A numeric vector of indices within the bounds of against.

Examples

```
keep_inds_in_bounds(1:10, 5:15)
```

`localMaxima` *Find the indices of the local maxima of a numeric vector*

Description

Given a numeric vector, this function returns the indices of the local maxima. A local maximum is defined as a data point that is greater than its adjacent points. The function adds a minimum value to each end of the vector to ensure that the endpoints can also be identified as local maxima, if appropriate.

Usage

```
localMaxima(v)
```

Arguments

v	A numeric vector

Value

A numeric vector containing the indices of the local maxima of v

Examples

```
localMaxima(c(1, 2, 3, 2, 1))
```

---

localMinima *Find local minima in a vector*

Description

Given a numeric vector, returns the indices of local minima in the vector. Endpoints can be included. Specifically, a point is considered a local minimum if there are no lesser points adjacent.

Usage

```
localMinima(v)
```

Arguments

v	a numeric vector

Value

a numeric vector containing the indices of local minima in v.

---

match.features2refs.par *Match STORM features to GISSMO database using parallel computing*

Description

This function matches STORM'd feature shapes to a reference database using cross-correlation, least-squares fitting, and peak quality weighting. The matching process is parallelized for speed, and results are saved as an RDS file. The function reads in parameters from a YAML file and data from RDS files generated by previous function calls (from fse() and tina())

Usage

```
match.features2refs.par(pars)
```

Arguments

pars	a list of parameters for matching, read in from a YAML file

Details

Matching: Matching is accomplished by cross-correlating all pairwise feature - reference spectrum pairs. Since there are feature-level comparisons to make (i.e. a single feature across all matches), iteration over features is serial. Iteration over references for each feature is done in parallel. This comparison is not optimal, but works as a proof of concept and can be scaled up. Increasing numbers of features results in a linear increase in computational time, but also more memory usage for each core. Generally, it is advisable to leave 1 or 2 cores on one's machine free for system operations and the main R instance. For each comparison that is, for each feature - reference pair being tested, the top max.hits (e.g., 5) convolution hits are assessed for pearson correlation coefficient (r.thresh) and pvalue (p.thresh). For hits passing both thresholds, the feature is fit to the reference using least squares. RMSE is reported.

Next, it's best to avoid penalizing a feature fit just because it contains false positive points. Once a feature has been fit to all reference spectra (at 0-5 more places), resonances which were never fit are identified using peak poorness: 1) Line up all fits for that feature across the database. 2) For each fit, take the positive residuals, divide by feature intensity. If values are close to 1, the feature was completely absent in the ref signature at those point. This is a measure of ~ peak poorness for this database 3) Take the mean of those values across all fits for each point in the feature. 4) Square peak-poorness to squash them down unless very high: peak.quality = 1-(peak.poorness^2) 5) For each fit's non-missing values: rmse.weighted = sum(residuals * peak.quality)/n.points

This metric appears to give more intuitive results. Missing reference resonances compared to the feature is preferable to the opposite, which will be measured during backfitting. Matches are written to file. Note: matching in parallel is much lighter (~50 outside of RStudio due to memory leaks/overhead.

Value

nothing; results are saved as an RDS file

`padmat` *Pad matrix with rows and columns of specified value*

Description

This function takes a matrix and pads it with rows and columns of a specified value. The number of rows and columns to add can also be specified. The new rows and columns will have the same value.

Usage

```
padmat(x, use = NA, row.by = 0, col.by = 1)
```

Arguments

`x`	A matrix to pad
`use`	The value to fill new rows and columns with (default is NA)
`row.by`	The number of rows to add to each side of the matrix
`col.by`	The number of columns to add to each side of the matrix

Value

A padded matrix

See Also

sub2indR

Examples

```
m <- matrix(1:4, ncol = 2)
padmat(m, use = 0, row.by = 1, col.by = 1)

m2 <- matrix(1:6, ncol = 2)
padmat(m2, row.by = 2, col.by = 3)

m3 <- matrix(1:9, ncol = 3)
padmat(m3, use = -1, row.by = 1, col.by = 2)
```

`pair.score.summation` *Calculates interaction scores between all reference spectra and spectral sets. The FSE module is one way to produce likely feature shapes supported by statistical association of spectral points across dataset spectra. Singlets contain very little specific information, can match to any feature shape and are therefore excluded.*

Description

TINA attempts to reduce the number of duplicate features, including those identified in different places on the spectral axis (i.e., misaligned).

Usage

```
pair.score.summation(pars)
```

Arguments

`pars`	A list of parameters specifying input/output directories and parallelization settings.

Details

These features are then matched to full-resolution pure compound reference spectra (e.g. GISSMO, or any other full-res source). The matching reference regions are termed ref-feats, and indeed are hypothesized subsignatures. To test whether these hypothetical subsignatures are present in the actual data, these are then back-fit to the original spectral data using the feature position and quantification.

Then, for each backfit (ref-feat to dataset spectrum), a backfit feasibility score is calculated (see filter.matches() for details). This roughly indicates whether or not the fit of the ref-feat is feasible (doesn't grossly exceed the spectral data in any places). Perfectly feasible is 1, not feasible is 0.

A good weighted rmse and rval therefore indicates a good shape match between the feature and the ref-feat. A high bff score for a ref-feat and a given dataset spectrum indicates that reference

spectrum region has a feasible fit to the data. These two pieces of information provide a basis for a believable association between the reference spectrum and the dataset spectrum.

However, there can be numerous versions of each ref-feat, and numerous ref-feats associating a given reference spectrum with a particular dataset spectrum. In fact, the ideal situation is numerous, highly feasible ref-feats associating the two and accounting for 100 point along the reference spectrum, we take the highest bff score associated with that point (from any ref-feat that covers a region including that point), with the default being 0. We could sum all these values for a given reference spectrum (again, focusing specifically on a single dataset spectrum). However, small peaks are downweighted by multiplying this composite "point-wise best bff score" vector by the the best-case feasibility of association of each ref spectral point, scaled by that point's overall relevance to the spectral signature. Summing these values across the score vector gives a total association score between a reference spectrum and a dataset spectrum.

Value

A table of scores for all possible reference spectra and spectral sets.

`pipeline` *Pipeline function to execute the entire ImperialNMRTool workflow*

Description

Feature shape based annotation pipeline / To run from here, you need the following files in ./data: params.yaml # parameter file spectral.matrix.RDS # spectral matrix data.list.RDS # library files lib.info.RDS # index for library files

Usage

```
pipeline(params_loc, params_obj)
```

Arguments

`params_loc`	Path to a YAML file containing user-specified parameters
`params_obj`	List of parameters

Details

Running the setup script will accomplish this, as well as ensuring that the correct library is being used. This pipeline accepts a spectral matrix (preferably with minimal processing; e.g. no strong normalization, no scaling, no alignment) in the form of and RDS file with a matrix (row 1 = ppm vector, each additional row is a 1D NMR sample).

MTJ 2023

pk.bounds — *Get bounds for local maxima in a vector*

Description

This function takes a numeric vector v and returns the bounds of the regions containing the local maxima in v. Updated to include only true peaks (those with a lower value on either side; no endpoints).

Usage

```
pk.bounds(v)
```

Arguments

v	A numeric vector

Value

A list of numeric vectors, where each element corresponds to the bounds of a region containing a local maximum in v

Examples

```
v <- c(1, 2, 3, 2, 1, 2, 2, 1, 2)
pk.bounds(v)
# Returns: list(c(1, 3), c(4, 5))
```

pk.maxs — *Get indices of maxima of true peaks in a vector. Wrapper for extractPeaks_corr()*

Description

This function takes a numeric vector v and a logical mask mask, and returns the indices of the local maxima in v that are not masked by mask.

Usage

```
pk.maxs(v, mask = NULL)
```

Arguments

v	A numeric vector
mask	A logical vector of the same length as v, indicating which values of v should be masked (i.e., excluded from consideration as local maxima)

Value

A numeric vector containing the indices of the local maxima in v that are not masked by mask

Examples

```
v <- c(1, 2, 3, 2, 1)
mask <- c(FALSE, FALSE, TRUE, FALSE, FALSE)
pk.maxs(v, mask)
# Returns: 3
```

`plot.fit` *Plot the spectral and feature fits*

Description

Plot the spectral and feature fits

Usage

```
## S3 method for class 'fit'
plot(fit, type = "simple", ppm = NULL, color = NULL)
```

Arguments

`fit`	An object of class "fit"
`type`	A character string specifying the type of plot. Possible values are "simple", "auc", and "color.line".
`ppm`	A numeric vector containing the ppm values.
`color`	A numeric vector containing the color values.

Value

A ggplot object

Examples

```
plot.fit(fit, type = "simple", ppm = ppm, color = color)
```

`plot_addRef` *Add reference region to an existing ggplot object*

Description

Add reference region to an existing ggplot object

Usage

```
plot_addRef(g, specRegion, ref, ppm, scaling = TRUE, type = "bold line")
```

Arguments

g	existing ggplot object (e.g. from simplePlot)
specRegion	x[,ref.idx]
ref	entire ref object - this function uses information from window and signature
ppm	ppm[,ref$wind$inds]
scaling	logical, whether to use new scaling
type	character vector specifying the type of plot to make ("bold line", "line", "shaded area")

Value

ggplot object with reference region added

---

plot_addSTOCSYLine *Plot STOCSY line over the current plot*

Description

Plot STOCSY line over the current plot

Usage

```
plot_addSTOCSYLine(
  g,
  specRegion,
  ppm,
  corr = NULL,
  covar = NULL,
  driver = NULL,
  bounds = NULL
)
```

Arguments

g	plot to add STOCSY line to
specRegion	region of the NMR spectrum to plot
ppm	vector of chemical shift values
corr	matrix of correlations between reference and query spectra
covar	matrix of covariances between reference and query spectra
driver	chemical shift value of the driver peak, if given
bounds	chemical shift values of the boundaries of the region of interest, if given

Value

plot with STOCSY line added

plot_correlation_pocket

Plot correlation window for a given pocket

Description

Plot correlation window for a given pocket

Usage

```
plot_correlation_pocket(vals)
```

Arguments

vals	a list containing values for the correlation window including sr, wind, corrRbound, and corrLbound

Value

a ggplot object displaying the correlation window

plot_correlation_pockets_grid

Plot correlation pockets in a grid

Description

Wrapper function for plot_correlation_pocket to generate a grid of plots from a list of correlation pocket data. Prints the resulting grid to a PDF file located in the specified directory.

Usage

```
plot_correlation_pockets_grid(allvals, plotLoc)
```

Arguments

allvals	a list of correlation pocket data (output from find_correlation_pockets)
plotLoc	character string specifying the directory to save the resulting PDF file

Value

None

plot_corrPocketPairs *Plot Stackplot of Correlation Pocket Pairs*

Description

This function plots a stackplot of correlation pocket peaks (same data as the heatmap, but plotted like spectral regions)

Usage

```
plot_corrPocketPairs(
  pocketPairs,
  region = NULL,
  vshift = 5,
  xvect = NULL,
  show = TRUE
)
```

Arguments

pocketPairs	Result of corrPocketPairs_al(...) with elements named "peakMap", "corr", "cov", and "regions".
region	A vector of integers specifying the columns of the peakMap to be included in the plot.
vshift	The vertical shift to apply to the distributions for stacking.
xvect	The x values at which to plot the distributions.
show	Logical. Whether or not to display the plot.

Value

The corrPocketPairs stack plot.

Examples

```
# Load data
pocketPairs <- corrPocketPairs_al(...)

# Plot correlation heatmaps
plot_corrPocketPairs(pocketPairs, region = 1000:1500)
```

plot_corrPocketPairs_grid

Plot correlation pairs of pockets in a grid

Description

Plot correlation pairs of pockets in a grid

Usage

```
plot_corrPocketPairs_grid(xmat, ppm, pocketPairs, res = 4000, plotLoc = "./")
```

Arguments

xmat	A matrix of dimensions (n, m), where each row is a pocket
ppm	A vector of ppm values
pocketPairs	A list of 2-tuples, each of which indicates which pockets to compare
res	Resolution of the plot in ppm
plotLoc	A string specifying the directory to save the plot in

Value

A PDF file of correlation plots between the specified pocket pairs arranged in a grid

plot_stocsy_corrWindow

Plot a correlation window with local minima

Description

Plot a correlation window with local minima

Usage

```
plot_stocsy_corrWindow(sr, lbound, rbound, showplot = TRUE, restrictTo = NULL)
```

Arguments

sr	An object of class Spectrum containing correlation information
lbound	An integer specifying the left boundary of the correlation window to be plotted
rbound	An integer specifying the right boundary of the correlation window to be plotted
showplot	A logical indicating whether the plot should be displayed on the screen (default = TRUE)
restrictTo	A numeric vector of length two specifying the limits of the x-axis in the plot

Value

A ggplot2 object of the correlation window with local minima highlighted

plot_stormRefRegions_grid *Plot storm Results for reference regions in grid format*

Description

Wrapper for plot_storm_refRegions to generate plots for multiple reference regions in a grid format and save the result to a pdf file.

Usage

```
plot_stormRefRegions_grid(
  xmat,
  ppm,
  stormResults,
  plotLoc,
  filename = "storm_results_refPlots_grid.pdf",
  calcStocsy = TRUE,
  n_xticks = 4
)
```

Arguments

xmat	a numeric matrix of NMR spectra with ppm values in the first column.
ppm	a numeric vector of ppm values corresponding to the NMR spectra.
stormResults	a list containing the storm analysis results for each reference region.
plotLoc	a character string indicating the directory to save the output file.
filename	a character string of the output filename. Defaults to "storm_results_refPlots_grid.pdf".
calcStocsy	a logical indicating whether to calculate and include STOCSY scores in the plots.
n_xticks	an integer indicating the number of x-axis ticks to use in the plots.

Value

a grid of plots of the storm analysis results for each reference region, saved to a pdf file.

plot_storm_refRegions *Plot the STORM results*

Description

Plot the storm results (s) from storm_play, etc. Plot light gray ref region for optimal subset. use in lapply(). Pass in s: nothing outside

Usage

```
plot_storm_refRegions(
  xmat,
  ppm,
  s,
  bgplot = "overlayed",
  vshift = 1,
  hshift = 0.05,
  calcStocsy = TRUE,
  n_xticks = 4
)
```

Arguments

xmat	numeric matrix of spectral data
ppm	numeric vector of length equal to ncol(xmat)
s	STORM results from storm_play or similar function
bgplot	a character vector indicating how the plot is rendered: "overlayed" or "stack". Default is "overlayed".
vshift	numeric scalar indicating the vertical shift between spectra when bgplot is "stack". Default is 1.
hshift	numeric scalar indicating the horizontal shift between spectra when bgplot is "stack". Default is 0.05.
calcStocsy	a logical indicating whether to calculate STOCSY plot. Default is TRUE.
n_xticks	an integer indicating the number of ticks on the x-axis. Default is 4.

Value

a ggplot object with the plot of the spectral data, the reference region, and the STOCSY plot (if applicable).

See Also

plot_stormRefRegions_grid, plot_stormRefRegions_summarize

Examples

```
plot_storm_refRegions(xmat = xmat, ppm = ppm, s = stormResults[[1]])
```

---

plot_umap.scores *Plot UMAP Scores*

Description

This function plots the UMAP scores (layout) and optionally labels the clusters (if given).

Usage

```
plot_umap.scores(umap.obj, clusters = NULL)
```

Arguments

umap.obj	A UMAP object generated using the umap package.
clusters	A vector of cluster labels for each row of umap.obj$layout.

Value

A plotly plot of the UMAP scores, with optional cluster labels.

---

plot_umap.scores.df *Plot UMAP scores*

Description

Plots UMAP scores (layout) and labels clusters (if given).

Usage

```
plot_umap.scores.df(df, clusters = NULL)
```

Arguments

df	A data.frame with columns "x", "y", and "cluster".
clusters	An optional vector of cluster labels for each point in the layout.

Value

A plotly figure.

---

prepRefs.for.dataset *Prepare reference spectra for matching to a dataset To match features from a dataset to reference spectra in fine detail, they need to have the same spectral axis, more or less. This function interpolates each reference spectrum in the list to the dataset ppm axis, and then filters its signal to avoid inclusion of noise/zeros, which are replaced with NAs so that cross-correlations, etc. are minimized. In future versions, a solvent region could be provided for each (or all) that will be replaced with NAs to avoid matching that peak.*

Description

Takes gissmo (or other) reference spectra as a list with - tag (short name, e.g. "bmse000005" - not necessarily unique) - ref.name (should be unique with field strength etc.) - ppm axis - ref spectrum data

Usage

```
prepRefs.for.dataset(data.list, ppm.dataset, ref.sig.SD.cutoff = 0.01)
```

Arguments

data.list	a list of reference spectra, where each element of the list is a list containing the fields "tag", "ref.name", "compound.name", "ppm", and "data"
ppm.dataset	the ppm axis of the dataset to which the reference spectra will be matched
ref.sig.SD.cutoff	the cutoff for reference spectra signal, expressed as a fraction of the standard deviation

Value

a list of processed reference spectra, where each element of the list is a list containing the fields "tag", "ref.name", "compound.name", "ppm", "data", and "mapped". The "mapped" field contains a list with the fields "ppm", "data", "sig.cutoff", and "sd.cutoff"

Examples

```
data.list <- list(ref1, ref2, ref3)
ppm.dataset <- c(0, 1, 2, 3, 4)
ref.sig.SD.cutoff <- 0.01
prepRefs.for.dataset(data.list, ppm.dataset, ref.sig.SD.cutoff)
```

---

project_features.stackplot

Generate a stacked area plot of the spectra for each feature in xmat*, overlaid with the corresponding best-fit features*

Description

Generate a stacked area plot of the spectra for each feature in xmat, overlaid with the corresponding best-fit features

Usage

```
project_features.stackplot(
  xmat,
  ppm,
  label,
  bestfits,
  exp.by = 0.05,
  vshift = 2,
  hshift = 0.002,
  sort.rows = F
)
```

Arguments

xmat	A matrix of spectral data with rows corresponding to spectra and columns to ppm values
ppm	A vector of ppm values corresponding to the columns of xmat

label	A character string indicating the feature label to be used in the plot title
bestfits	A list containing the best fit data for each feature, as generated by project_features.getBestFits
exp.by	The amount by which to expand the range of each feature's ppm range in the plot
vshift	The amount by which to vertically shift each spectrum in the plot
hshift	The amount by which to horizontally shift each spectrum in the plot
sort.rows	A logical indicating whether or not to sort the rows of the spectra in each plot by their summed intensity

Value

A list containing the plot objects and parameters used to create the plot

---

prominences *Calculate the prominence of each peak*

Description

The prominence of a peak is the height of the peak relative to the lowest contour line that encloses the peak and no higher peak. This function calculates the prominence of each peak in a vector.

Usage

```
prominences(peaks, v2)
```

Arguments

peaks	A list of peak information as output from the peakdet function.
v2	A vector of the y-values of the peaks.

Value

A vector of the prominences of each peak.

---

range.groups *Divide Ranges into Groups with Overlapping Sections*

Description

Given a set of ranges, return a list of groups where each group contains ranges with overlapping sections.

Usage

```
## S3 method for class 'groups'
range(ranges, operation = "intersection")
```

Arguments

ranges	A matrix with two rows, representing ranges in the form c(min,max).
operation	The type of intersection operation to use when determining overlap.

Value

A list of groups, where each group is a vector of indices corresponding to overlapping ranges.

---

range.intersect *Calculate the intersection or union of two numeric ranges*

Description

Calculate the intersection or union of two numeric ranges

Usage

```
## S3 method for class 'intersect'
range(a, b, operation = "intersection")
```

Arguments

a	a numeric vector representing a range
b	a numeric vector representing a range
operation	a character string indicating whether to calculate the "intersection" or "union" of the two ranges

Value

a numeric vector representing the intersection or union of the two ranges, or a vector of NA if the ranges do not overlap

Examples

```
range.intersect(c(1, 5), c(3, 6), operation = "intersection")
range.intersect(c(1, 5), c(3, 6), operation = "union")
```

range.intersect.all *Calculate pairwise intersection or union between ranges*

Description

Given a matrix of ranges (rows: features, columns: ranges), calculates the pairwise intersection or union between each range. Returns a matrix with dimensions (ncol(ranges), ncol(ranges)).

Usage

```
## S3 method for class 'intersect.all'
range(ranges, operation = "intersection")
```

Arguments

ranges	a matrix with dimensions (n x m), where n is the number of features and m is the number of ranges
operation	a string indicating the type of operation to perform. Can be either "intersection" or "union". Defaults to "intersection".

Value

a matrix with dimensions (ncol(ranges), ncol(ranges)) containing the pairwise intersection or union between each range.

run.labels *Assigns a unique label to each run of 1s in a binary vector.*

Description

This function takes a binary vector and assigns a unique integer label to each run of 1s in the vector. The labels start at 1 and increment by 1 for each new run. Runs of 0s are assigned a label of 0.

Usage

```
run.labels(v)
```

Arguments

v	A binary vector.

Value

A numeric vector with the same length as ‘v‘ containing the labels assigned to each run of 1s.

Examples

```
v <- c(0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1)
run.labels(v)
```

runFeatureSpace.shiny *Run Feature Space Shiny app*

Description

Runs a Shiny app that displays a UMAP layout with associated clusters, and allows the user to select points and view their associated extracted features.

Usage

```
runFeatureSpace.shiny(study, umap.obj, cluster.labs, featureStack)
```

Arguments

study	A character string specifying the accession number of the study.
umap.obj	A UMAP object, output of umap() or a related function.
cluster.labs	A vector of cluster assignments for each point in the UMAP layout.
featureStack	A object of extracted features for each point in the UMAP layout.

Author(s)

MTJ2022

runs.labelBy.lengths *Computes the lengths of runs of 1s in a binary vector.*

Description

This function takes a binary vector and computes the lengths of all runs of consecutive 1s in the vector. The output is a numeric vector, with length equal to the number of 1s in the input vector, where each point in each run is labeled by its run length.

Usage

```
runs.labelBy.lengths(v)
```

Arguments

v	A binary vector.

Value

A numeric vector containing the lengths of all runs of consecutive 1s in the input vector.

Examples

```
v <- c(0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1)
runs.labelBy.lengths(v)
```

scale.between *Scale values in a vector to be between a specified range.*

Description

This function takes a numeric vector and scales the values to be between a specified lower and upper bound.

Usage

```
## S3 method for class 'between'
scale(data, lower = 0, upper = 1)
```

Arguments

data	A numeric vector to be scaled.
lower	A numeric value representing the lower bound of the output range. Default is 0.
upper	A numeric value representing the upper bound of the output range. Default is 1.

Value

A numeric vector with values scaled between the lower and upper bounds.

Examples

```
x <- c(1, 3, 5, 7, 9)
scale.between(x, 0, 10)
```

scale.to.minmax *Scale a vector to the range of a matrix, element-wise*

Description

Scale a vector to the range of a matrix, element-wise

Usage

```
## S3 method for class 'to.minmax'
scale(v, mat, useInds = TRUE)
```

Arguments

v	A numeric vector to be scaled
mat	A numeric matrix used to define the scaling range
useInds	A logical vector or integer vector indicating the columns of ‘mat‘ or the elements of ‘v‘ to use in the calculation. Default is to use all columns or elements.

Value

A numeric vector with the same length as ‘v‘, scaled to the range of ‘mat‘ element-wise.

score.matches *score.matches function*

Description

This function computes match scores between subset spectra and library reference spectra. Uses pair.score.summation() to actually compute the scores It builds a ss-ref matrix and looks for any compounds that match known annotations.

Usage

```
score.matches(pars)
```

Arguments

pars	a list containing necessary parameters for the function

Value

RDS file containing match scores between library reference spectra and the best subset spectrum score for each.

score.wasserstein *Calculate the Wasserstein distance between two vectors using transport::wasserstein1d.*

Description

Calculate the Wasserstein distance between two vectors using transport::wasserstein1d.

Usage

```
score.wasserstein(v1, v2)
```

Arguments

v1	numeric vector of values.
v2	numeric vector of values.

Value

The Wasserstein distance between v1 and v2.

Examples

```
v1 <- c(1, 2, 3, 4, 5)
v2 <- c(3, 5, 7, 9, 11)
score.wasserstein(v1, v2)
```

setup — *Setup function for initializing the analysis environment*

Description

This function performs various setup tasks, such as copying library data and metadata, and finding the best fit for the spectrometer frequency.

Usage

```
setup(params_obj)
```

Arguments

params_obj	An object containing various input parameters, such as the study and library directories.

Value

None

Examples

```
setup(params_obj)
```

shiftedInds_toVals — *Extract values from a matrix given shifted indices*

Description

Given a matrix and a matrix of shifted indices (with the same number of rows), extract the values of the original matrix at those shifted indices. The shifted indices can be created by aligning a subregion of the original matrix with a reference vector.

Usage

```
shiftedInds_toVals(xmat, shiftedInds)
```

Arguments

xmat	A matrix containing the values to extract
shiftedInds	A matrix of shifted indices (with the same number of rows as xmat) indicating which values to extract from xmat

Value

A matrix containing the values of xmat at the shifted indices given in shiftedInds

Examples

```
xmat <- matrix(1:9, nrow = 3)
shiftedInds <- matrix(c(1, 2, 3, 2, 3, 4, 3, 4, 5), nrow = 3)
shiftedInds_toVals(xmat, shiftedInds)

# Output:
#      [,1] [,2] [,3]
# [1,]    1    2    3
# [2,]    2    3    4
# [3,]    3    4    5
```

---

simplePlot *Create a simple plot using ggplot2*

Description

This function creates a simple plot using ggplot2, based on a matrix or dataframe input. It can handle both single vector and multiple vectors input, and can generate x-axis ticks with a specified number of ticks or using a default breaks_pretty() function.

Usage

```
simplePlot(
  ymat = NULL,
  xvect = NULL,
  n_xticks = NULL,
  xdir = "reverse",
  linecolor = "gray",
  opacity = 0.6,
  linewidth = 0.5
)
```

Arguments

ymat	A matrix or dataframe, with each column being a spectrum to plot.
xvect	A numeric vector for the x-axis values.
n_xticks	An integer specifying the number of x-axis ticks to generate.
xdir	A character string indicating the direction of the x-axis, either "reverse" or "forward".
linecolor	A character string indicating the color of the plotted lines.
opacity	A numeric value specifying the opacity of the plotted lines.
linewidth	A numeric value specifying the width of the plotted lines.

Value

A ggplot object.

Examples

```
data(mtcars)
plotmat <- scale(mtcars[, -1], center = TRUE, scale = TRUE)
simplePlot(plotmat, linecolor = "black", opacity = 0.6, linewidth = 0.5)

simplePlot(plotmat[, 1], linecolor = "red", opacity = 0.6, linewidth = 0.5)

simplePlot(plotmat, xvect = mtcars$wt, n_xticks = 5, linecolor = "black", opacity = 0.6, linewidth = 0.5)
```

---

slidingCorr *Calculate sliding window correlation matrix, and label correlation peaks (pockets)*

Description

Calculate sliding window correlation matrix, and label correlation peaks (pockets)

Usage

```
slidingCorr(
  x,
  ws,
  extractPockets = FALSE,
  plotting = TRUE,
  vshift = 20,
  ppm = NULL
)
```

Arguments

x	Input matrix.
ws	Size of sliding window.
extractPockets	If TRUE, calculate correlation pockets for each column of x within each sliding window.
plotting	If TRUE, produce a stackplot of the correlation matrix with pockets highlighted.
vshift	Vertical shift in plot.
ppm	X-axis values for plot. If not specified, uses column numbers of x.

Value

A list containing the following items:

corr_compact	The compact form of the sliding correlation matrix.
indsmat	The offset matrix used to calculate the sliding correlation matrix.
cov_compact	The compact form of the sliding covariance matrix.
isPocket	A logical matrix indicating whether each point in the sliding window is within a correlation pocket.
plot	If plotting is TRUE, a stackplot of the correlation matrix. If extractPockets = T, show only the pockets.
window	The relative inds in the sliding window.
center	The center position of the sliding window.

Examples

```
# Generate a test matrix
x <- matrix(rnorm(2000), nrow = 100)

# Calculate sliding correlation matrix with pockets and plot
slidingCorr(x, ws = 20, extractPockets = TRUE, plotting = TRUE)
```

`sortedStack` *sortedStack*

Description

Sorts a matrix by row sum and stacks the rows.

Usage

```
sortedStack(mat, ppm = NULL, vshift = 10, hshift = 0)
```

Arguments

`mat`	The matrix to sort and stack.
`ppm`	The x-axis values for the columns of the matrix. Defaults to NULL.
`vshift`	The vertical shift for the stacked plots. Defaults to 10.
`hshift`	The horizontal shift for the stacked plots. Defaults to 0.

Value

A stacked plot of the sorted matrix.

`sortPairs` *sortPairs.R*

Description

Given a two-row matrix where each column contains a pair of numbers, this function sorts the pairs such that each column's first element is less than its second element. The function assumes that pairs are already sorted, and quickly checks if any pairs need to be switched.

Usage

```
sortPairs(pairs)
```

Arguments

`pairs`	A two-row matrix where each column contains a pair of numbers

Value

A two-row matrix with pairs sorted in ascending order

Examples

```
pairs <- matrix(c(1, 3, 2, 5, 4, 3), nrow = 2)
sortPairs(pairs)
# Output:
#      [,1] [,2] [,3]
# [1,]    1    2    3
# [2,]    3    5    4
```

`span` *Calculates the span of a numeric vector*

Description

Calculates the span of a numeric vector, which is defined as the range of the values plus one.

Usage

```
span(v)
```

Arguments

v	A numeric vector

Value

The span of the vector '

`stackplot` *Plot stacked spectra with ggplot2 and ggridges*

Usage

```
stackplot(
  ymat = NULL,
  xvect = NULL,
  vshift = 1,
  hshift = 0,
  shift_by = NULL,
  xdir = "reverse",
  show = TRUE
)
```

Arguments

ymat	A numeric matrix of spectral intensities.
xvect	A numeric vector of ppm values.
vshift	A numeric value for shifting spectra vertically (spectra are scaled down to this value).
hshift	A numeric value for shifting spectra horizontally (as
	\itemshift_byA numeric vector for shifting spectra by row number.
	\itemxdirA character string for x-axis direction, either "reverse" or "forward".
	\itemshowA logical value for whether or not to display the plot.
	A ggplot2 object.
	This function plots stacked spectra using ggplot2 and ggridges, which creates a density plot of the spectra that can help to visualize overlaps and peaks.
	data(metab) stackplot(metab[, 1:10], xvect = metab[, 11])

---

stocsy_slim *Calculate Pearson correlations and covariance using STOCSY for a subset of variables.*

Description

Calculate Pearson correlations and covariance using STOCSY for a subset of variables.

Usage

```
stocsy_slim(X, driverInd)
```

Arguments

X	A matrix of spectral data with variables in the columns.
driverInd	A vector of indices for the variables that will be used as drivers.

Value

A list with two elements: cc, a vector of Pearson correlations between each variable in X and the variables in driverInd; and cv, a vector of covariance values between each variable in X and the variables in driverInd.

`storm_pairplay` *storm_pairplay: Run modified STORM on the provided spectral region and ref shape. Built for accepting corrPocketPairs results. Notes:*

Description

STORM: Joram Posma's STORM has been adapted and optimized to accept these

Usage

```
storm_pairplay(
  xmat = NULL,
  ppm = NULL,
  b = 30,
  corrthresh = 0.8,
  q = 0.05,
  minpeak = 10,
  refSpec,
  ref.idx
)
```

Arguments

`xmat`	A matrix of spectral data (rows are spectra, columns are spectral points)
`ppm`	A vector of the spectral points in ppm (optional, default is all columns of xmat)
`b`	An integer giving the expansion parameter for the reference peak
`corrthresh`	A numeric giving the minimum correlation value to be considered for inclusion (for both subset AND reference optimization)
`q`	A numeric giving the p-value threshold for correlation significance (both subset AND reference optimization)
`minpeak`	An integer giving the minimum number of points allowed in a run of significant points in the reference
`refSpec`	A vector of spectral data to use as the initial reference
`ref.idx`	A vector of the spectral points (columns of xmat) to use as the initial reference

Value

A list with components "reconstructed" and "status". "reconstructed" is a matrix containing the reconstructed metabolite concentrations (rows are samples, columns are metabolites). "status" is a character string indicating whether the method converged successfully or failed.

sub2indR *Convert Subscripts to Linear Indices in Column-Major Order*

Description

This function converts subscripts of a matrix to their corresponding linear indices in a column-major order. Intended to replicate MATLAB's sub2ind.

Usage

```
sub2indR(rows, cols, m)
```

Arguments

rows	The row subscripts.
cols	The column subscripts.
m	The number of rows in the matrix.

Value

The linear indices corresponding to the input subscripts.

Examples

```
sub2indR(1:3, 1:3, 3)
# [1] 1 4 7 2 5 8 3 6 9
```

tina *Tina Function. Tina Is Not Alignment.*

Description

This is a bit of a semantic argument, but spectral alignment typically refers to modifying spectra or reposition peaks such that they align with each other from sample to sample. Because we've done FSE, features have much more information which allows them to be detected in multiple regions combined by shape similarity. Instead of modifying spectra, TINA reformulates alignment as a clustering problem in feature (shape) space.

Usage

```
tina(pars)
```

Arguments

pars	A list of parameters for the TINA pipeline.

Details

There are also several filtering steps employed before this. We cannot eliminate all poor shapes, but there are a couple of useful heuristics which generally reduce unnecessary computation downstream. First, there are many feature shapes which are either quite poor in quality, or do not contain sufficient information to be useful for annotation. We keep features with: - in defined ppm range (generally [-1, 11]) - long enough runs? (contains runs > noisewidth*3 adjacent points) - large enough subset? (>= 5 spectra with feature, good correlation reliability) - has at least 1 true peak > 0.3 of the range of intensities? (not just the side of a broad peak; not monotonic) Since the same features will often be extracted multiple times (either in the same spectral region, or other regions; i.e. same peak, but misaligned), it is advantageous to reduce this redundancy by clustering feature shapes. We accomplish this using a combination of UMAP projection and Affinity Propagation clustering. Before comparing feature shapes, we align them to their maximum intensity resonance. This is quick, and usually performs well enough. UMAP uses euclidean distance as a default. In practice, UMAP is able to sort feature shapes into tight clusters when low n_neighbors (e.g. 5) and min_dist (e.g. 0.05) are used. This does not capture global relationships as well, but we only use it to identify tight clusters. All pairwise correlations (PCCs) are calculated for the feature shapes. A mask for correlation thresholding is applied to the distance matrix (generated using apcluster::negDistMat(pts, r=2), squared negative euclidean distance) to ensure that clustered features have a high correlation as well. apcluster uses a q parameter to optimize the initial preferences. Higher q -> stricter clusters. Raising the lambda (dampening) parameter helps avoid oscillations which prevent convergence, although raising this too high can make updates too slow to converge within the number of iterations. See https://cran.r-project.org/web/packages/apcluster/vignettes/apcluster.pdf for a full description of affinity propagation parameters Ultimately, it doesn't matter much what clustering method is used, as this is primarily a means of combining highly similar features to reduce the computational burden of pairwise comparisons to reference spectra.

Value

A list containing the results and cluster labels from TINA

---

tina_combineFeatures *Combines and clusters features in a matrix*

Description

Combines and clusters features in a matrix

Usage

```
tina_combineFeatures(
  featureStack,
  doUMAP = TRUE,
  umap.n_neighbors = 20,
  umap.min_dist = 0.1,
  umap.n_epochs = 500,
  ap.rcutoff = 0.99,
  ap.q = 0.95,
  ap.max.iter = 1000,
  ap.lam = 0.9,
  max.plots = 250,
```

```
  plot.loc = "./",
  plot.name = "study_feature_clusters.pdf"
)
```

Arguments

featureStack	A matrix of features to be clustered.
doUMAP	Logical indicating whether to perform UMAP projection on the features.
umap.n_neighbors	the number of approximate nearest neighbors used to construct the initial high-dimensional graph. It effectively controls how UMAP balances local versus global structure - low values will push UMAP to focus more on local structure by constraining the number of neighboring points considered when analyzing the data in high dimensions, while high values will push UMAP towards representing the big-picture structure while losing fine detail.
umap.min_dist	the minimum distance between points in low-dimensional space. This parameter controls how tightly UMAP clumps points together, with low values leading to more tightly packed embeddings. Larger values of min_dist will make UMAP pack points together more loosely, focusing instead on the preservation of the broad topological structure.
umap.n_epochs	The number of iterations to run.
ap.rcutoff	rcutoff for filtering out correlations before clustering
ap.q	q parameter is the shared value for the initial preferences s(i,k) for a priori clustering. See apcluster doc.
ap.max.iter	The maximum number of iterations for the ap cluster.
ap.lam	The "self-responsibility" parameter for the affinity propagation clustering algorithm.
max.plots	The maximum number of scatter plots to generate for the clusters.
plot.loc	The directory in which to save the scatter plot PDF.
plot.name	The name of the scatter plot PDF.

Value

A list containing UMAP and clustering results, cluster assignments, pairwise distances and correlations.

---

tina_setup *TINA Setup*

Description

Given a list of peak regions and a matrix of spectra, creates a matrix to house all the features and returns it in a list format along with additional information about the subsets and regions of interest.

Usage

```
tina_setup(a, xmat)
```

Arguments

a	A list of peak regions.
xmat	A matrix of spectra.

Value

A list containing the feature stack, position stack, subsets, and region information.

Examples

```
# Create a mock input list of peak regions and a matrix of spectra
peaks <- list(list(finalRegion = c(1, 2, 3), ref.vals = c(1, 2, 3),
                   ref.idx = c(1, 2, 3), subset = c(1, 2)),
              list(finalRegion = c(4, 5, 6), ref.vals = c(4, 5, 6),
                   ref.idx = c(4, 5, 6), subset = c(3, 4)))
spectra <- matrix(1:12, ncol = 4)

# Run the function
tina_setup(a = peaks, xmat = spectra)
```

---

trim.sides *Trim the sides of a matrix to remove columns with all NA values*

Description

Trim the sides of a matrix to remove columns with all NA values

Usage

```
trim.sides(mat, out = "mat")
```

Arguments

mat	a matrix or data.frame to be trimmed
out	character string indicating the output type. Possible values are "mat" (default), which returns the trimmed matrix, or "inds", which returns the indices of the columns that were kept.

Value

a matrix with the same number of rows as `mat` but fewer columns, with columns containing NA values taken out.

Examples

```
mat <- matrix(c(1,2,NA,4,NA,NA,7,8,NA), nrow = 3)
trim.sides(mat)
```

vectInds *Return the index/indices in a vector closest to given value(s).*

Description

Given a vector of values, find the index/indices in another vector which is closest to each value.

Usage

```
vectInds(vals, vect)
```

Arguments

vals	a numeric vector of values to find the closest index/indices to.
vect	a numeric vector of values to search for closest index/indices.

Value

a numeric vector of indices in vect which are closest to each value in vals.

Examples

```
vectInds(1:5, c(3.5, 4.6, 2.1))
```

xcorr *Calculate cross-correlation between reference and shifted spectra*

Description

Wrote my own cross-correlation function for more consistent behavior in the context of comparing ref signatures to spec data. This takes a (wider) spec vector and slides it across the ref signature lags. The lags are used to extract a window of spec for each shift (lag). These windows are collected as rows of a matrix, which is then used to compute all correlations to the ref. MTJ 2023

Usage

```
xcorr(spec, ref.vals, ref.idx, lags, min.overlap = 3)
```

Arguments

spec	Numeric matrix. A matrix of shifted spectra.
ref.vals	Numeric vector. The values of the reference spectrum.
ref.idx	Integer vector. The indices of the reference spectrum.
lags	Integer vector. The range of lags to consider.
min.overlap	Integer. The minimum number of overlapping data points between reference and shifted spectra to calculate correlation.

Value

A list containing the following elements:

rvals	Numeric vector. The correlation scores for each shifted spectrum.
pvals	Numeric vector. The p-values for each correlation score.
overlaps	Numeric vector. The number of overlapping data points between reference and each shifted spectrum.
not.used	Logical vector. Indicates which shifted spectra did not meet the minimum overlap requirement and were not used in calculating correlation scores or p-values.
lags	Integer vector. The lags used for each shifted spectrum.
bestFit	List. Contains information on the best-matching shifted spectrum. The list has the following elements: \itemwhich.lagLogical vector. Indicates which lags were the best match. \itemrNumeric. The correlation score for the best match. \itemlagInteger. The lag corresponding to the best match. \itempvalNumeric. The p-value for the correlation score of the best match. \itemoverlapsInteger. The number of overlapping data points between the reference and the best-matching shifted spectrum. \itemspecInds.matchedInteger vector. The indices of the data points in the shifted spectrum that correspond to the overlapping region with the reference spectrum. This can be used to plot the overlapping regions.

Examples

```
# Generate sample data
set.seed(123)
ref <- rnorm(10)
ref.idx <- 1:10
spec <- matrix(rnorm(1000), nrow = 10)
lags <- -5:5

# Calculate cross-correlation
xcorr(spec, ref, ref.idx, lags)
```

---

xcorr_localign *Cross-correlation with local alignment*

Description

This function calculates the cross-correlation between a reference vector and a matrix of spectra, where each spectrum is a row of the matrix. The function aligns the reference vector with each spectrum by shifting the spectrum and then calculating the cross-correlation between the aligned spectrum and the reference vector. The function returns the best shift for each spectrum, along with the corresponding correlation score, p-value, and overlap size. The function can also generate a plot showing the aligned spectra and reference vector.

Usage

```
xcorr_localign(
  x,
  ppm,
  signature.vals,
  signature.idx.wind,
  currentInds,
  min.overlap = 3,
  slide = "internal",
  lag.limit = NA,
  plots = F
)
```

Arguments

x	A matrix where each row is a spectrum.
ppm	A vector of column names for the matrix x.
signature.vals	A vector of values matching the signature index window.
signature.idx.wind	The reference indices (across all rows of currentInds).
currentInds	The window(s) in which the reference sits in x (often larger than the reference).
min.overlap	The minimum number of points on which to base the correlation.
slide	If "internal", the function just slides the reference vector until it hits the bounds of currentInds; if "external", the function pads currentInds to ensure that all reference vector points cross the bounds of currentInds; if "limited", the user provides the number of lags (duplicated for both directions).
lag.limit	The limit on the number of lags to consider (for "limited" slide option).
plots	A logical value indicating whether to generate a plot showing the aligned spectra and reference vector.

Value

A list with the following elements:

best.shifts	A vector of the best shift for each spectrum.
best.rvals	A vector of the correlation scores for each spectrum.
best.pvals	A vector of the p-values for each spectrum.
shiftedInds	A matrix of the indices in x for the aligned spectra.
shiftedSpecs	A matrix of the aligned spectra.
ref.aligned	A vector of the reference vector values (aligned with the spectra).
overlap.sizes	A vector of the overlap sizes for each spectrum.
g	A plot showing the aligned spectra and reference vector (if `plots = TRUE`).

Examples

```
x <- matrix(rnorm(100), ncol = 10)
ppm <- seq(100)
signature.vals <- rnorm(10)
signature.idx.wind <- seq(5, 14)
```

```
currentInds <- matrix(seq(1, 100, by = 10), ncol = 2)
xcorr_localign(x, ppm, signature.vals, signature.idx.wind, currentInds, min.overlap = 3, slide = "internal", l
```

Index