Package 'metabom8'

May 9, 2026

Type Package

Title A High-Performance R Package for Metabolomics Modeling and Analysis

Version 1.1.0

Description Tools for 1D NMR metabolomics workflows, including import and preprocessing of Bruker experiments, multivariate modeling (PCA, PLS, OPLS) and model analytics and validation (y-permutations, cv-anova). Performance-critical routines are implemented in C++ and use the Armadillo and Eigen linear algebra libraries to improve runtime.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.6.0)

Imports abind, colorRamps, ellipse, fs, ggplot2, ggrepel, graphics, methods, parallel, pcaMethods, plotly, pROC, progress, ptw, Rcpp (>= 1.0.4.6), reshape2, rlang, scales, signal, stats, utils

LinkingTo Rcpp, RcppArmadillo, RcppEigen

biocViews Metabolomics, Cheminformatics, Preprocessing, DataImport, Alignment, WorkflowStep

BugReports https://github.com/tkimhofer/metabom8/issues

URL https://tkimhofer.github.io/metabom8/

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests BiocStyle, dplyr, DT, ExperimentHub, htmlTable, knitr, rmarkdown, testthat

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/metabom8

git_branch devel

git_last_commit 977c8fa

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-08

Author Torben Kimhofer [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7158-9930>)

Maintainer Torben Kimhofer <tkimhofer@gmail.com>

Contents

`.alignSegment` *Align NMR Spectra via Cross-Correlation*

Description

Aligns rows of an NMR spectral matrix to a reference spectrum by maximizing cross-correlation. Optionally uses the row-wise median spectrum as reference. Useful for minor spectral misalignments.

Usage

```
.alignSegment(
  seg,
  idx_ref = 1,
  clim = 0.7,
  med = TRUE,
  norm = FALSE,
  lag.max = 20
)
```

Arguments

`seg`	Numeric matrix. Each row is a 1D NMR spectrum segment.
`idx_ref`	Integer. Row index to use as reference spectrum. Ignored if `med = TRUE`.
`clim`	Numeric. Minimum cross-correlation threshold. Segments with lower similarity are not shifted.
`med`	Logical. If `TRUE`, the row-wise median spectrum is used as the reference.
`norm`	Logical. If `TRUE`, rows are z-scaled prior to alignment.

Details

Each spectrum is aligned to the reference by computing the cross-correlation and applying a lag-based linear interpolated shift. The shift is applied only if the cross-correlation exceeds `clim`. This function is intended for fine-tuning alignment across short ppm windows (i.e., segments).

Value

A numeric matrix of aligned spectra with the same dimensions as `seg`. If `med = TRUE`, the output excludes the median row.

.balanced_bootstrap_mc *Class-balanced resampling (with replacement)*

Description

Generates k training index sets. Total train size per set is floor(N * split), where N is the total number of samples. Each stratum contributes (approximately) equally, sampling WITH replacement within each stratum.

Usage

```
.balanced_bootstrap_mc(
  k,
  split,
  stratified,
  remainder = c("distribute", "drop")
)
```

Arguments

k	Integer. Number of resampling sets.
split	Numeric in (0,1). Fraction of total N used as training size.
stratified	List of length 3: 1. type: "DA" for classification or "R" for regression 2. Y: matrix with one column (n x 1) 3. probs: numeric vector of quantile probs for regression binning (e.g. c(0, .33, .66, 1))
remainder	Character. How to handle n_train %% G remainder: • "drop": keep perfect balance, total size becomes G * floor(n_train/G) • "distribute": distribute leftover 1-by-1 to random strata (still near-balanced)

Value

List of length k. Each element is an integer vector of training indices.

.calibTsp *Calibrate*

Description

Calibrate

Usage

```
.calibTsp(spec, ppm)
```

Value

Shift-adjusted ppm vector

.check1d_files_fid *Check for Intact Bruker NMR File Structures*

Description

Verifies that each NMR experiment has a matching `acqus` and `fid` file.

Usage

```
.check1d_files_fid(datapath, n_max = 10, filter = TRUE, recursive, verbose)
```

Arguments

`datapath`	Character. Directory containing NMR spectra.
`n_max`	Integer. Maximum number of experiments to process.
`filter`	Logical. Whether to filter out incomplete experiments.
`recursive`	Logical. Whether to search directories recursively.
`verbose`	Integer. Controls verbosity level (0 = silent, 1 = basic, 2 = detailed).

Value

A list with paths to intact experiments and matching metadata.

See Also

`.extract_acq_pars1d`, `.filterExp_files`

.checkDimXY *Check Dimensions of X and Y Matrices (PLS Context)*

Description

Checks that the input matrices `X` and `Y` have compatible dimensions for Partial Least Squares (PLS) analysis. Specifically, the number of observations (rows) must match, and the number of variables (columns) in `X` must be greater than in `Y`.

Usage

```
.checkDimXY(X, Y)
```

Arguments

`X`	Numeric matrix. Predictor data matrix (observations x variables).
`Y`	Numeric matrix. Response data matrix (observations x variables).

Value

Invisibly returns `NULL` if checks pass. Throws an error if dimensions are incompatible.

.checkXclassNas *Check X matrix (PLS context)*

Description

Validates that input `X` is a numeric matrix and does not contain missing, NaN, or infinite values.

Usage

```
.checkXclassNas(X)
```

Arguments

`X`	Input data (univariate or multivariate), formatted as a matrix.

Value

`NULL` if all checks pass; otherwise throws an informative error.

.checkYclassNas *Check Y for missing values (PLS context)*

Description

Checks input Y for NA/NaN/Inf values, determines analysis type (Regression or Discriminant Analysis), and returns cleaned Y matrix, associated levels, and type.

Usage

```
.checkYclassNas(Y)
```

Arguments

`Y`	A vector, matrix, or data.frame. Target variable.

Value

A list: cleaned Y matrix, levels (if applicable), and type ('R' or 'DA')

.check_pc_model *Check Validity of Selected Principal or Orthogonal Component*

Description

Internal consistency check for principal (or orthogonal) component selection used in plotting or extraction functions.

Usage

```
.check_pc_model(pc, mod, le = 1, type = "p")
```

Arguments

pc	Character or numeric. Selected component, e.g., 1 or "o1".
mod	An object of class PCA_metabom8 or OPLS_metabom8.
le	Integer. Expected maximum length of pc; usually 1.
type	Character. Type of component to check: "p" for projection or "t" for scores. Currently only used for logic branching.

Value

Nothing. Function is used for its side effect (i.e., error checking).

.check_X_ppm *Check Consistency of Spectral Data and ppm Vector*

Description

Validates that the NMR data matrix and the chemical shift reference vector (ppm) are consistent. Specifically, it checks that

- ppm contains no NA or infinite values.
- The number of columns in X matches the length of ppm.

This function is typically used as a preliminary data validation step before downstream spectral analysis.

Usage

```
.check_X_ppm(X, ppm)
```

Arguments

X	Numeric matrix. NMR spectral matrix with samples in rows and variables in columns (e.g., intensities).
ppm	Numeric vector. Chemical shift reference values corresponding to columns of X.

Value

Logical. Returns TRUE if inputs are valid, FALSE otherwise.

.chemShift *Calculate Chemical Shift Axis*

Description

Computes a 1D NMR chemical shift axis based on sweep width, offset, and size.

Usage

```
.chemShift(swidth, offset, si)
```

Arguments

swidth	Numeric. Sweep width (Hz).
offset	Numeric. Frequency offset (ppm).
si	Integer. Number of data points.

Value

Numeric vector of chemical shift values (ppm).

.cvSetsMethod *Generate Cross-Validation (CV) Training Set Indices*

Description

Generates a list of row indices representing training sets for various CV strategies including stratified and Monte Carlo methods. Handles both regression (R) and discriminant analysis (DA), and multi-column Y.

Usage

```
.cvSetsMethod(
  Y,
  type,
  method = "k-fold_stratified",
  k = 7,
  split = 2/3,
  probs = NULL
)
```

Arguments

Y	matrix or data.frame. Outcome matrix. If multi-column, only the first column will be used.
type	character. Type of analysis: "R" for regression or "DA" for discriminant analysis. Use suffix "-mY" to indicate multi-column Y.
method	character. One of: "k-fold", "k-fold_stratified", "MC", "MC_balanced".

k	integer. Number of CV iterations/folds.
split	numeric. For Monte Carlo methods, the fraction of samples to use in the training set (0 < split < 1).
probs	Numeric vector. Probabilities used for stratification of numeric Y (regression only).

Details

- In "k-fold_stratified" and "MC_balanced" modes, stratification is applied to preserve class proportions or outcome distribution.
- If type is regression ("R"), quantile-based binning is applied to stratify continuous Y values.
- If type is discriminant analysis ("DA"), class labels are used directly.
- If Y has multiple columns, only the first column is used to create resampling sets.

Value

A list of integer vectors. Each vector contains row indices representing a training set.

.detect1d_procs *Check for Intact File Systems - Helper Function for read1d*

Description

Scans the specified directory recursively to find intact sets of Bruker NMR files: 'procs', 'acqus', and '1r' files. Optionally filters incomplete experiments.

Usage

```
.detect1d_procs(
  datapath,
  n_max = 10,
  filter = TRUE,
  recursive = TRUE,
  verbose = 1
)
```

Arguments

datapath	character. Path to directory containing spectra.
n_max	integer. Maximum number of spectra to read (default 10).
filter	logical. Whether to filter out incomplete file sets (default TRUE).
recursive	logical. Whether to search directories recursively.
verbose	integer. Verbosity level for messaging.

Value

A list with elements:

path Character vector of experiment folder paths (intact).

exp_no Character vector of experiment folder IDs.

f_procs Character vector of paths to 'procs' files.

f_acqus Character vector of paths to 'acqus' files.

f_1r Character vector of paths to '1r' files.

.dimX *Ensure Input is a Row Matrix*

Description

Ensures that the input object `X` is in row-matrix form. If `X` is a numeric vector (i.e., has no dimensions), it is converted to a matrix with 1 row and `length(X)` columns. This is helpful for standardizing inputs before applying matrix operations.

Usage

```
.dimX(X)
```

Arguments

`X`	Numeric vector, matrix, or array. Typically NMR data where rows represent spectra.

Value

A numeric matrix with one row if input was a vector; otherwise, returns `X` unchanged.

.draw_base *Draw NMR Spectra Using Base Graphics*

Description

Internal backend renderer for base R plotting.

Usage

```
.draw_base(dat, add, ...)
```

Arguments

`dat`	List returned by `.prep_spec()`.
`add`	Logical. Add to existing plot.
`...`	Additional graphical parameters.

.draw_ggplot2 *Draw NMR Spectra Using ggplot2*

Description

Internal backend renderer using ggplot2. Converts spectra to long format prior to plotting.

Usage

```
.draw_ggplot2(dat, ...)
```

Arguments

dat	List returned by .prep_spec().
...	Additional arguments for ggplot2.

.draw_plotly *Draw NMR Spectra Using Plotly*

Description

Internal backend renderer for interactive plotting.

Usage

```
.draw_plotly(dat, col = NULL, ...)
```

Arguments

dat	List returned by .prep_spec().
...	Additional arguments passed to plotly.

.dynamicIntervalsMedian *Dynamic Interval Segmentation (RSPA-style)*

Description

Defines peak-system intervals from the median spectrum for recursive segment-wise alignment (RSPA).

Usage

```
.dynamicIntervalsMedian(X, ppm, half_win_ppm = 0.02, min_snr = 8)
```

Arguments

X	Numeric matrix (spectra in rows).
ppm	Numeric ppm vector.
half_win_ppm	Numeric. Half window size around detected peaks.

Value

List of index vectors defining alignment intervals.

See Also

Other NMR: get_idx()

.evalFit *Evaluate model fit progression based on cross-validated performance*

Description

Determines whether an additional component should be fitted based on cross-validated generalization performance. For regression models (type = "R"), the decision is based on the Q^2 statistic. For discriminant analysis models (type = "DA"), the decision is based on the cross-validated AUROC (area under the ROC curve).

Usage

```
.evalFit(
  type,
  q2s,
  cv_auc,
  pc_max = 5,
  min_delta = 0.05,
  sat_q2 = 0.98,
  sat_auc = 0.97,
  min_q2 = 0.05,
  min_auc = 0.6
)
```

Arguments

type	Character. Model type: "R" for regression or "DA" for discriminant analysis. The value may optionally include a suffix such as "-mY" to indicate multi-response models.
q2s	Numeric vector of Q^2 values for the fitted components (used for regression models).
cv_auc	Numeric vector of cross-validated AUROC values for the fitted components (used for classification models).
pc_max	Integer. Maximum number of components allowed.
min_delta	Numeric. Minimum improvement in the cross-validated performance metric required to justify fitting an additional component.

`sat_q2`	Numeric. Saturation threshold for Q^2 in regression models.
`sat_auc`	Numeric. Saturation threshold for cross-validated AUROC in discriminant analysis models.
`min_q2`	Numeric. Minimum acceptable Q^2 value for the first component.
`min_auc`	Numeric. Minimum acceptable cross-validated AUROC for the first component.

Details

AUROC is computed on cross-validation test folds to ensure that it reflects out-of-sample classification performance, analogous to the use of Q^2 in regression.

Value

A list containing:

stop Logical indicating whether model fitting should stop.

reason Character string describing the stopping condition (e.g., `"pc_max"`, `"cv_decreased"`, `"cv_improvement_negli`

metric The current cross-validated performance value.

delta Improvement relative to the previous component.

`.extMeanCvFeat` *Extract Cross-Validated OPLS Features*

Description

Extracts a specific data structure (e.g., scores or predictions) from cross-validation output returned by `.oplsComponentCv()`. Depending on the cross-validation type and model structure, the output is aggregated across folds by computing the mean, standard deviation (SD), and coverage.

Here, **Coverage** refers to the proportion of cross-validation folds in which a given sample's value was observed (i.e., not missing), serving as an estimate of how frequently that value was evaluated across folds.

Usage

```
.extMeanCvFeat(cv_obj, feat = "t_xp")
```

Arguments

`cv_obj`	Named list. Output of `.oplsComponentCv()` for a specific OPLS component.
`feat`	Character. Feature name to extract from each CV fold object (e.g., `"t_xp"` or `"y_pred_test"`).

Value

A numeric matrix or array of the requested feature:

- For Monte Carlo CV and single-column Y: matrix with rows `"mean"`, `"sd"`, and `"coverage"` x samples.
- For Monte Carlo CV and multi-column Y: 3D array with shape `[3, samples, outcomes]`.
- For k-fold CV: matrix or array containing values for each sample (no aggregation, just fold values).

.extract_acq_pars1d *Extract Bruker NMR Acquisition Parameters*

Description

Parses acqus files to extract acquisition parameters for each experiment.

Usage

```
.extract_acq_pars1d(f_list)
```

Arguments

f_list	List. Contains paths to NMR experiment folders (e.g., as returned by .check1d_files_fid()).

Value

A data frame containing extracted acquisition parameters for each experiment.

See Also

.filterExp_files, .check1d_files_fid

.extract_pars1d *Read Bruker NMR Parameter Files*

Description

Helper function used by read1d() to extract acquisition (acqus) and processing (procs) parameters from Bruker-formatted 1D NMR experiments.

Usage

```
.extract_pars1d(f_list)
```

Arguments

f_list	List of file paths. Output from .detect1d_procs().

Value

A data frame of extracted acquisition and processing metadata. Row names correspond to spectrum filenames.

`.fidApodisationFct` *Apodisation function dispatcher*

Description

Generates apodisation (window) functions for FID processing. The function applies one of several predefined window shapes, selected via `pars$fun`.

Usage

```
.fidApodisationFct(n, pars)
```

Arguments

`n`	Integer. Number of data points.
`pars`	List of parameters defining the apodisation. Must contain element `fun` specifying the window type.

Details

Supported apodisation functions and parameters:

`"uniform"` No apodisation (all weights = 1).

`"exponential"` Exponential decay window.

- `lb` Line broadening parameter.

`"cosine"` Cosine window (no additional parameters).

`"sine"` Sine window (no additional parameters).

`"sem"` Sine-modulated exponential window.

- `lb` Exponential decay parameter.

`"expGaus_resyG"` Exponential-Gaussian hybrid window.

- `lb` Exponential decay parameter.
- `gb` Gaussian broadening parameter.
- `aq_t` Acquisition time.

`"gauss"` Gaussian window (Bruker-style).

- `lb` Line broadening parameter.
- `gb` Gaussian broadening factor.
- `para` List containing acquisition parameters (e.g. `a_SW_h`, `a_TD`).

Value

Numeric vector of length `n` with apodisation weights.

Examples

```
# Internal FID apidisation functions (illustrative only)
f_apod <- c("sine","cosine","exponential","sem")

# create toy fid
n <- 200; t <- seq(0,1,len=n)
fid <- exp(-5*t)*cos(20*pi*t)

# generate & apply apodisation windows
A  <- sapply(f_apod, \(f) metabom8:::.fidApodisationFct(n, list(fun=f, lb=-0.2)))
Fp <- sweep(A, 1, fid, `*`)

# generate spectra
S  <- apply(Fp, 2, \(x) Mod(fft(x))[1:(n/2)])

# graphhics
cols <- 1:ncol(A)

par(mfrow=c(2,2), mar=c(3,3,2,1))
plot(t, fid, type="l", lwd=2, main="FID")
matplot(t, A, type="l", lwd=2, col=cols, main="Windows")
matplot(t, Fp, type="l", lwd=2, col=cols, main="Windowed FID")
matplot(S, type="l", lwd=2, col=cols, main="Spectrum")

legend("topright", f_apod, col=cols, lty=1, bty="n", cex=.8)
```

`.filterExp_files` *Filter Bruker NMR Experiments*

Description

Filters Bruker 1D experiments based on acquisition metadata and optionally limits the number of returned entries. Character parameters are matched by value, while numeric parameters support equality, set inclusion, range filtering (list(range = c(min, max))), and generic operators (list(op = ">", value = x)).

Usage

```
.filterExp_files(pars, exp_type, f_list, n_max)
```

Arguments

`pars`	Data frame. Parsed acquisition and processing parameters.
`exp_type`	Named list. Filtering conditions for parameters. Elements may be character vectors, numeric values/vectors, or operator/range lists for numeric fields.
`f_list`	List. File paths of spectra and associated metadata (e.g., f_fid, f_1r).
`n_max`	Integer. Maximum number of experiments to retain.

Value

A list containing filtered and ordered `f_list` and `pars`.

See Also

.extract_acq_pars1d, .check1d_files_fid

.is_numeric_trycatch *Check if input is numeric-like using tryCatch*

Description

Tests whether a vector x can be safely coerced to numeric without producing NAs (excluding original NAs) or warnings.

This function attempts to coerce the input to numeric and returns TRUE if all non-missing elements convert without coercion failure; otherwise FALSE. It catches warnings and errors during coercion and treats those as non-numeric.

Usage

```
.is_numeric_trycatch(x)
```

Arguments

x	A vector to test for numeric coercion.

Value

Logical TRUE if x is numeric-like, FALSE otherwise.

.kFold *k-fold cross-validation index generator*

Description

Creates a list of training set indices for k-fold cross-validation (CV), without considering class balance in Y. Each element of the list represents the row indices used for training in one CV fold.

Usage

```
.kFold(k, Y)
```

Arguments

k	Integer. Number of CV folds. If k is not valid or too high relative to number of rows in Y, it defaults to leave-one-out CV (LOO-CV).
Y	Matrix. Outcome matrix (observations x variables).

Value

A list of length k, where each element contains the training indices (integers) for one CV fold.

.kFoldStratified *Stratified k-fold cross-validation index generator*

Description

Generates a list of training set indices for stratified k-fold cross-validation (CV). Stratification is performed based on the first column of `Y`. For regression tasks, `Y` is binned by quantiles to emulate class balance. Ensures class proportions are preserved across folds when possible.

Usage

```
.kFoldStratified(k, stratified)
```

Arguments

`k`	Integer. Number of folds.
`stratified`	List with three elements: • `type`: Character. Either `"R"` for regression or `"DA"` for discriminant analysis. • `Y`: Matrix. Outcome matrix with a single column. • `probs`: Numeric vector. Probabilities used for stratification of numeric `Y` (regression only).

Value

A list of length `k`, each element containing the training set row indices (integers) for one CV fold. Returns `NULL` and a warning if stratification is not feasible due to class imbalance.

.list_to_string *Convert named list to compact string representation*

Description

Internal utility that converts a named list into a single character string of the form `name=value`. Nested sublists are represented recursively using braces.

Usage

```
.list_to_string(lst)
```

Arguments

`lst`	A named list.

Details

This function is primarily used for compact parameter logging in provenance metadata and should not be called directly by users.

Value

A character string.

.mc *Generate Monte Carlo Cross-Validation (MCCV) Training Indices*

Description

Generates a list of training set indices for Monte Carlo Cross-Validation.

Usage

```
.mc(k, Y, split)
```

Arguments

k	Integer. Number of MCCV iterations (i.e., training sets to generate).
Y	A matrix (n x p), where rows are observations and columns are outcomes. Only the number of rows is used here.
split	Numeric. Fraction of the data to include in each training set. Should be between 0 and 1.

Details

For each fold, a random sample (without replacement) of size `floor(nrow(Y) * split)` is drawn.

Value

A list of length `k`, each containing a vector of training set indices.

.mcBalanced *Class-balanced Monte Carlo Cross-Validation (MCCV) splits*

Description

Generates class/group-balanced training set indices for each MCCV round.

Usage

```
.mcBalanced(k, split, stratified)
```

Arguments

k	Integer. Number of MCCV training sets to generate.
split	Numeric. Fraction of observations per class to include in each training set. Must be between 0 and 1.
stratified	List of three elements: • 1: Character. Outcome type: 'R' (regression) or 'DA' (discriminant analysis), optionally with suffix '-mY' for multi-column Y. • 2: Matrix. Response matrix `Y`, with `ncol(Y) == 1`. • 3: Numeric vector of probabilities used to stratify numeric Y if regression.

Details

Each round samples a class-balanced subset without replacement. Useful for high-variance, imbalanced-class modeling.

Value

A list of length k, each element containing a vector of row indices for the training set.

.oplsComponentCv *OPLS Component Estimation via Cross-Validation*

Description

Performs orthogonal projections to latent structures (OPLS) modeling on training folds of a cross-validation (CV) scheme. The function fits one orthogonal and one predictive component per fold, tracking scores, predictions, and residuals. For the first orthogonal component (nc = 1), the full model structure is initialized. For later components (nc > 1), modeling proceeds on residual matrices carried over from previous iterations.

Usage

```
.oplsComponentCv(X, Ycs_fold, cv.set, nc, mod.cv, acc)
```

Arguments

X	Numeric matrix. Input data matrix (samples x features); only required for nc = 1.
Ycs_fold	List containing the response values (Y) for each cross-validation split.
cv.set	List of integer vectors. Each element contains training indices for one CV round. These indices are adjusted internally by -1 due to 0-based indexing in the underlying C++ (Rcpp) routines.
nc	Integer. Component number currently being estimated (nc = 1 for first orthogonal component).
mod.cv	List. Model state to be updated with results from each CV iteration.

Value

A list of the same length as cv.set. Each list element contains:

- t_xo: Orthogonal component scores (matrix, samples x components).
- t_xp: Predictive component scores (matrix, samples x 1).
- y_pred_train: Predicted responses for training samples.
- y_pred_test: Predicted responses for held-out test samples.
- x_res: Residual matrix after filtering out orthogonal structure.

Note

Training indices in cv.set are internally adjusted by -1 before being passed to Rcpp routines. This is required because C++ uses zero-based indexing.

.permYmod *OPLS Y-permutation Modeling*

Description

Performs OPLS modeling on permuted Y to estimate cross-validated model performance under the null hypothesis. Extracts predictive performance (R2, Q2, AUROC) from cross-validation for either regression or classification.

Usage

```
.permYmod(XcsTot, Y, cv, type, nc_o)
```

Arguments

XcsTot	Numeric matrix. Input data matrix (samples x features).
Y	Numeric matrix. Response variable (numeric or dummy-coded).
cv	List. Cross-validation parameters, including method and cv_sets.
type	Character. Model type: "DA", "R", possibly with "-mY" suffix for multi-column Y.
nc_o	Integer. Number of orthogonal components to be fitted.

Details

When nc_o > 1, the function fits additional orthogonal components sequentially, updating the model object. Classification performance is computed using AUROC (via pROC). Regression performance uses R2 and Q2.

Value

List with elements:

- r2_comp: Numeric, R2 statistic for test data.
- q2_comp: Numeric, Q2 statistic from cross-validation.
- aucs_tr: Numeric, AUROC on training data (for classification only).
- aucs_te: Numeric, AUROC on test data (for classification only).

.perm_test_from_table *Permutation-test summary from an opls_perm out_df table*

Description

Takes the out_df you showed (perm rows + one "non-permuted" row) and returns observed values, permutation distributions, and permutation p-values.

Usage

```
.perm_test_from_table(
  out_df,
  observed_label = "non-permuted",
  alternative = c("greater", "less"),
  add_one = TRUE,
  na_rm = TRUE
)
```

Arguments

out_df	data.frame with columns like q2_comp, r2_comp, aucs_te, aucs_tr, model (with one row "non-permuted"), and optionally r/r_abs.
observed_label	character. Label used in model for the observed row.
alternative	character. "greater" (default) tests obs > perm; "less" tests obs < perm.
add_one	logical. If TRUE, uses (1 + count)/(B + 1) p-value correction.
na_rm	logical. Drop NA values before computing p-values.

Value

A list with:

- observed: named numeric vector of observed metrics
- perm: list of numeric vectors for each metric
- p_value: named numeric vector of permutation p-values
- B: number of permutations used

.prepareY *Prepare Response Vector for OPLS/OPLS-DA*

Description

Converts a response vector Y into a numeric matrix suitable for regression or classification models.

- Numeric input is returned as a column matrix.
- Factor or character input:
  - If binary (2 levels): returns a single numeric column (values 1 and 2).
  - If multiclass (>2 levels): returns a dummy matrix with 1 for presence and -1 for absence. Also returns a mapping between original labels and numeric values (only for categorical input).

Usage

```
.prepareY(Y)
```

Arguments

Y	A numeric, factor, or character vector.

Value

A list with:

`[[1]]` Numeric matrix of processed response.

`[[2]]` Data frame mapping class labels to encoding (empty for numeric input).

.prep_spec *Prepare Spectral Data for Plotting*

Description

Internal helper that validates input, enforces ppm ordering, subsets the selected shift region, and returns matrix-form data.

Usage

```
.prep_spec(x, ppm, shift)
```

Arguments

`x`	Numeric vector or matrix of spectra.
`ppm`	Numeric chemical shift vector.
`shift`	Length-2 numeric vector specifying ppm window.

Value

A list with elements:

ppm Subsetted chemical shift axis

X Matrix of subsetted spectra

.r2 *Rˆ2 and Qˆ2 Calculation for OPLS Models*

Description

Computes the coefficient of determination (R^2 or Q^2) for OPLS regression models using the formula:

$$R^2 = 1 - \frac{PRESS}{TSS}$$

, where PRESS is the prediction error sum of squares, and TSS is the total sum of squares. If `ytss` is not provided, it is computed directly from `Y`.

This function supports matrix inputs (e.g., for multi-class outcomes) and averages across columns to return a single summary R^2/Q^2 value.

Usage

```
.r2(Y, Yhat, ytss = NULL)
```

Arguments

Y	Numeric matrix. True response values. For classification, this should be a dummy matrix.
Yhat	Numeric matrix. Predicted response values from the model.
ytss	Numeric (optional). Total sum of squares of Y. If not provided, it will be computed internally.

Value

A single numeric value representing R^2 or Q^2.

.scaleMatRcpp *Column-wise matrix scaling (C++ backend)*

Description

Column-wise matrix scaling (C++ backend)

Usage

```
.scaleMatRcpp(X, idc, center, scale_type)
```

Arguments

X	num matrix
idc	int row indices of X
center	logical mean centering
scale_type	int 0: no scaling, 1: SD scaling, 2: Pareto scaling

Value

list: 1. scale X matrix, 2. mean (sd vec), 3: sd (num vec)

.sdRcpp *Column-wise standard deviation and mean for a matrix Welford algorithm (single loop for mean/sd)*

Description

Column-wise standard deviation and mean for a matrix Welford algorithm (single loop for mean/sd)

Usage

```
.sdRcpp(X)
```

Arguments

X	num matrix

Value

list: 1. sd (num vec), 2. mean (sd vec)

.shift_interp *Linear Interpolated Shift (Internal)*

Description

Applies a linear shift to a numeric vector using interpolation. Values outside the original domain are extended using the boundary values (no zero padding).

Usage

```
.shift_interp(x, lag, ppm = NULL)
```

Arguments

x	Numeric vector.
lag	Numeric scalar. Shift in index units. Positive values shift the signal to the right, negative values shift to the left.

Value

Numeric vector of the same length as x.

.vip *Compute Variable Importance in Projection (VIP)*

Description

Calculates VIP scores for a PLS/OPLS model using predictive components only.

Usage

```
.vip(Tx, W, C)
```

Arguments

Tx	Numeric matrix (n x A). Score matrix.
W	Numeric matrix (A x p). Weight matrix. Rows correspond to components, columns to variables.
C	Numeric vector or matrix of length A. Y-loadings.

Details

VIP is computed as:

$$VIP_j = \sqrt{p \frac{\sum_a SSY_a w_{aj}^2}{\sum_a SSY_a}}$$

where SSY_a = (t_a^T t_a) * c_a^2

By construction, mean(VIP^2) = 1.

Value

Numeric vector of length p containing VIP scores.

`add_note` *Add user note to metabom8 provenance Appends a user annotation to the* `"m8_prep"` *attribute. The step title is formatted as* `"note {username}"`*. The timestamp is stored in* `params`*, and the user message is stored in* `notes`*.*

Description

Add user note to metabom8 provenance Appends a user annotation to the `"m8_prep"` attribute. The step title is formatted as `"note {username}"`. The timestamp is stored in `params`, and the user message is stored in `notes`.

Usage

```
add_note(x, note, params = NULL)
```

Arguments

`x`	A metabom8 object or matrix with `"m8_prep"` metadata.
`note`	Character string describing the annotation.
`params`	Named list providing paramter key-value pairs.

Value

The input object with updated provenance metadata.

See Also

Other provenance: `get_provenance()`, `print_provenance()`

Examples

```
params <- list(
  runtime  = "docker",
  image    = Sys.getenv("IMAGE", "docker-image-dummy"),
  workflow = Sys.getenv("M8_WORKFLOW", "std_prof-urine"),
  agent    = paste0("snakemake/", Sys.getenv("SNAKEMAKE_VERSION", "v?")),
  run_id   = Sys.getenv("M8_RUN_ID", "m8-2605-001")
)

data(hiit_raw)
print_provenance(hiit_raw)

hiit_proc <- hiit_raw |>
  calibrate(type = "tsp") |>
  excise() |>
  add_note('dilution-adaptive acquisition mode -> verify snr after normalisation',
    params)

print_provenance(hiit_proc)
```

`align_segment` *Align NMR Spectra in a Selected Shift Region*

Description

Aligns spectra within a specified ppm window using cross-correlation. Only the selected region is aligned; the remainder of the spectra is unchanged.

Usage

```
align_segment(
  dat,
  shift,
  idx_ref = 1,
  med = TRUE,
  clim = 0.7,
  norm = FALSE,
  lag.max = 20
)
```

Arguments

`dat`	Named list with elements: **X** Numeric matrix (spectra in rows) **ppm** Numeric vector of chemical shift values **meta** Optional metadata
`shift`	Numeric vector of length 2 specifying ppm region to align.
`idx_ref`	Integer. Row index to use as reference spectrum.
`med`	Logical. Use row-wise median spectrum as reference?
`clim`	Numeric. Minimum correlation threshold.
`norm`	Logical. Z-scale before alignment.
`lag.max`	Integer. Maximum lag allowed.

Value

Updated `dat` list with aligned spectra in selected region.

See Also

Other preprocessing: `align_spectra()`, `binning()`, `calibrate()`, `correct_baseline()`, `correct_lw()`, `pqn()`, `print_preprocessing()`

Examples

```
data(hiit_raw)
plot_spec(hiit_raw, shift=c(1.3,1.4))
hiit_aligned <- align_segment(hiit_raw, c(1.3, 1.35))
plot_spec(hiit_aligned, shift=c(1.3,1.4)) # aligned segment
plot_spec(hiit_aligned, shift=c(3, 3.1)) # segment not aligned
```

`align_spectra` *Cohort-Guided Interval Alignment for 1D NMR Spectra*

Description

Aligns 1D NMR spectra using signal-adaptive ppm intervals derived from the cohort median spectrum. Intervals are constructed around median peak systems and aligned locally via cross-correlation. The function accepts either a metabom8-style `dat` list or plain matrix/vector with `ppm`.

Usage

```
align_spectra(x, ppm = NULL, half_win_ppm = 0.007, lag.max = 200)
```

Arguments

`x`	A numeric matrix/vector of spectra or a named list with elements `X`, `ppm`, and optionally `meta`.
`ppm`	Numeric vector of chemical shift values (ppm). Ignored if `x` is a `dat` list.
`half_win_ppm`	Numeric scalar controlling alignment interval width.
`lag.max`	Integer. Maximum allowed lag for cross-correlation alignment.

Details

Alignment intervals are derived from peaks in the cohort median spectrum.

The `half_win_ppm` parameter controls interval granularity:

- Smaller values (e.g. 0.005–0.007 ppm) generate more, narrower intervals (RSPA-like behaviour).
- Larger values (e.g. 0.008–0.015 ppm) merge nearby multiplets into broader regions (icoshift-like behaviour).

Typical 600–800 MHz ^{1}H NMR data:

- 0.006–0.008 ppm: stable multiplet-level alignment
- <0.005 ppm: may split J-coupled systems
- >0.015 ppm: may merge unrelated resonances

Alignment parameters and interval definitions are recorded in `attr(X, "m8_prep")`.

Value

If input is `dat`, returns updated `dat`. Otherwise returns aligned matrix.

See Also

`align_segment`

Other preprocessing: `align_segment()`, `binning()`, `calibrate()`, `correct_baseline()`, `correct_lw()`, `pqn()`, `print_preprocessing()`

Examples

```
data(hiit_raw)
plot_spec(hiit_raw, shift=c(1.3,1.4))
hiit_aligns <- align_spectra(hiit_raw)
plot_spec(hiit_aligns, shift=c(1.3,1.4)) # aligned segment
plot_spec(hiit_aligns, shift=c(3, 3.1)) # aligned segment
```

---

balanced_boot *Balanced bootstrap resampling strategy*

Description

Balanced bootstrap resampling strategy

Usage

```
balanced_boot(k, split, type = c("DA", "R"), probs = NULL)
```

Arguments

k	Integer. Number of bootstrap resamples.
split	Numeric. Fraction of samples drawn for the training set (e.g. 2/3). Sampling is performed with replacement.
type	Character. Either "DA" (classification) or "R" (regression).
probs	Numeric vector of quantile probabilities used to stratify continuous Y when type = "R".

Details

Generates k bootstrap samples (training sets sampled with replacement). The remaining samples (the out-of-bag set) can be used as a test set.

Balancing ensures equal representation of strata in the training data:

type = "DA" Class labels define the strata, and sampling is balanced across classes.

type = "R" The response is discretised into bins using quantiles defined by probs, and each bin contributes equally to the training set.

Value

A named list with elements:

train List of integer vectors containing training set indices for each resampling iteration.

strategy Character string indicating the resampling strategy.

n Integer. Number of samples in the dataset.

seed Integer. Random seed used to generate the resampling splits, ensuring reproducibility.

See Also

Other resampling strategies: balanced_mc(), kfold(), mc(), stratified_kfold()

Examples

```
n <- 100
# bivariate outcome
thr <- 1.5
Y <- c(rnorm(80, thr-3, 0.3), rnorm(20, thr+3, 0.3))  # unbalanced low/high outcome
mean(Y>thr)

cv_k <- kfold(k = 10)
cv_boot <- balanced_boot(k = 10, split = 2/3, type = "R", probs = c(0, 0.8, 1))

k_inst <- metabom8:::.arg_check_cv(cv_pars=cv_k, model_type='R', n=n, Y_prepped=cbind(Y))
b_inst <- metabom8:::.arg_check_cv(cv_pars=cv_boot, model_type='R', n=n, Y_prepped=cbind(Y))

# balanced splits: proportion above global median stays ~0.5
q80 <- quantile(Y, 0.8)
round(sapply(k_inst$train, function(i) mean(Y[i] > q80)), 2) # resembles original Y distr.
round(sapply(b_inst$train, function(i) mean(Y[i] > q80)), 2) # balanced strata (low/high)
```

---

balanced_mc *Balanced Monte-Carlo resampling strategy*

Description

Balanced Monte-Carlo resampling strategy

Usage

```
balanced_mc(k, split, type = c("DA", "R"), probs = NULL)
```

Arguments

k	Integer. Number of repeated random splits.
split	Numeric. Fraction of samples assigned to the training set (e.g. 2/3).
type	Character. Either "DA" (classification) or "R" (regression).
probs	Numeric vector of quantile probabilities used to stratify continuous Y when type = "R".

Details

Generates k Monte-Carlo resampling splits by randomly partitioning the data into training and test sets without replacement.

Balancing ensures equal representation of strata in the training data:

type = "DA" Class labels define the strata, and sampling is balanced across classes.

type = "R" The response is discretised into bins using quantiles defined by probs, and each bin contributes equally to the training set.

This strategy can improve robustness of model evaluation in settings with limited samples size and imbalanced or unevenly distributed outcome variables.

Value

A named `list` with elements:

train List of integer vectors containing training set indices for each resampling iteration.

strategy Character string indicating the resampling strategy.

n Integer. Number of samples in the dataset.

seed Integer. Random seed used to generate the resampling splits, ensuring reproducibility.

See Also

Other resampling strategies: `balanced_boot()`, `kfold()`, `mc()`, `stratified_kfold()`

Examples

```
n <- 100
# bivariate outcome
thr <- 1.5
Y <- c(rnorm(80, thr-3, 0.3), rnorm(20, thr+3, 0.3))  # unbalanced low/high outcome
mean(Y>thr)

cv_k <- kfold(k = 10)
cv_mc <- balanced_mc(k = 10, split = 2/3, type = "R", probs = c(0, 0.8, 1))

k_inst <- metabom8:::.arg_check_cv(cv_pars=cv_k, model_type='R', n=n, Y_prepped=cbind(Y))
mc_inst <- metabom8:::.arg_check_cv(cv_pars=cv_mc, model_type='R', n=n, Y_prepped=cbind(Y))

# balanced splits: proportion above global median stays ~0.5
q80 <- quantile(Y, 0.8)
round(sapply(k_inst$train, function(i) mean(Y[i] > q80)), 2) # resembles original Y distr.
round(sapply(mc_inst$train, function(i) mean(Y[i] > q80)), 2) # balanced strata (low/high)
```

`binning` *Spectral data binning*

Description

Equidistant binning of spectra by summarising intensities within ppm bins.

Usage

```
binning(X, ppm = NULL, width = NULL, npoints = NULL, fun = sum)
```

Arguments

`X`	Numeric matrix or data frame with spectra in rows, or a named list as returned by `read1d`/`read1d_proc` containing `X` and `ppm`.
`ppm`	Numeric vector of chemical shift positions (length must match `ncol(X)`). If `NULL`, `ppm` is inferred in the following order: 1. `X$ppm` if `X` is a list input, 2. `attr(X, "m8_axis")$ppm` (if present), 3. numeric `colnames(X)` (if present).

width	Numeric. Bin size in ppm, or NULL if npoints is specified.
npoints	Integer. Desired number of bins per spectrum, or NULL if width is specified. If both are provided, npoints is used.
fun	Function. Summary function applied to intensities within each bin. Must return a single numeric value (e.g. sum, mean, max).

Details

If present, preprocessing provenance is appended to attr(X, "m8_prep") using .m8_stamp(). The ppm axis is updated in attr(X, "m8_axis")$ppm and column names are set to the bin centres.

When width is specified, spectra are interpolated onto a regular ppm grid and then aggregated within bins (interp = TRUE in provenance). When npoints is specified, aggregation is performed by index bins on the original grid (interp = FALSE in provenance).

Value

Numeric matrix with spectra in rows and binned ppm variables in columns.

See Also

Other preprocessing: align_segment(), align_spectra(), calibrate(), correct_baseline(), correct_lw(), pqn(), print_preprocessing()

Examples

```
set.seed(1)
X <- matrix(rnorm(2 * 100), nrow = 2)
ppm <- round(seq(10, 0.5, length.out = 100), 3)
colnames(X) <- ppm

Xb <- binning(X, ppm, width = 0.5)
Xb_mean <- binning(X, ppm, width = 0.5, fun = mean)
dim(Xb)
```

---

calibrate *Chemical Shift Calibration*

Description

Aligns 1D ^{1}H NMR spectra to a reference signal on the existing ppm grid. Supports singlet (e.g. TSP or custom range) and predefined doublet references (glucose, alanine).

Usage

```
calibrate(X, ppm = NULL, type = "tsp")
```

Arguments

X	Numeric matrix/vector of spectra or a metabom8 data list.
ppm	Numeric chemical shift vector. If X is a metabom8 list, this is taken from X$ppm.
type	Character ("tsp", "glucose", "alanine"), or list.

Details

In addition to predefined references, custom calibration targets can be supplied as a list with elements:

- `mode`: `"singlet"` or `"doublet"`
- `window`: numeric vector of length 2 defining the ppm search region
- `centre`: target ppm position (optional; defaults to mean(window))
- `j`: numeric vector of length 2 specifying expected J-coupling (required for doublet calibration)

Custom doublet calibration requires the expected J-coupling range (`j`) in ppm to distinguish the two peaks of the multiplet.

For example:

```
calibrate(
  X, ppm,
  list(
    mode   = "doublet",
    window = c(1.2, 1.35),
    j      = c(0.007, 0.009)
  )
)
```

Value

Calibrated spectra in the same structure as input.

See Also

Other preprocessing: `align_segment()`, `align_spectra()`, `binning()`, `correct_baseline()`, `correct_lw()`, `pqn()`, `print_preprocessing()`

Examples

```
data("covid_raw")
X=covid_raw$X
ppm=covid_raw$ppm
X_tsp <- calibrate(X, ppm, type = "tsp")
X_glu <- calibrate(X, ppm, type = "glucose")
X_custom <- calibrate(X, ppm, type = c(1.9, 2.1))
```

---

cliffs_d *Cliff's Delta Effect Size*

Description

Calculates Cliff's delta (`Cd`) as a non-parametric effect size for comparing two numeric vectors. `Cd` quantifies the directional difference between a reference (`ref`) and comparison (`comp`) distribution based on pairwise comparisons of their values.

`Cd` ranges from -1 to 1, where:

- -1: values in `comp` are consistently larger than those in `ref`
- 0: both distributions are similar, with no systematic difference
- 1: values in `ref` are consistently larger than those in `comp`

Usage

```
cliffs_d(ref, comp)

es_cdelta(ref, comp)
```

Arguments

ref	Numeric vector representing the reference group.
comp	Numeric vector representing the comparator group.

Details

The effect size is calculated as the scaled difference in dominance between groups. Missing or infinite values are removed with a message. Synonyms are

Value

A single numeric value: Cliff's delta effect size.

References

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3), 494–509. doi:10.1037/00332909.114.3.494

Examples

```
ref <- rnorm(100, mean = 0)
comp <- rnorm(100, mean = 1)

hist(ref,  col=rgb(0,0,1,0.3), breaks=50, xlim=c(-4,4))
hist(comp, col=rgb(1,0,0,0.3), breaks=50, add=TRUE)
legend("topright", legend=c("ref","comp"), fill=c("blue","red"))

cliffs_d(ref, comp)
cliffs_d(comp, ref)

comp1 <- rnorm(100, mean = 10)
cliffs_d(ref, comp1)

cliffs_d(ref, ref)
```

---

correct_baseline *Baseline Correction for Spectral Data*

Description

Applies baseline correction to each spectrum (row) of a spectral matrix. Multiple correction algorithms are available and selected via the method argument.

Usage

```
correct_baseline(X, method = c("asls", "linear"), ...)

bline(X, ...)
```

Arguments

X	Numeric matrix or metabom8 dat object containing spectra in rows and spectral variables in columns.
method	Character specifying the baseline correction algorithm. One of: "asls" Asymmetric least squares baseline estimation. "linear" Linear baseline estimated from edge regions.
...	Additional parameters passed to the selected method. **Arguments for** method = "asls" **lambda** Numeric smoothing parameter controlling baseline stiffness. Larger values produce smoother baselines. Default 1e7. **iter_max** Maximum number of iterations used in the baseline estimation procedure. Default 30. **Arguments for** method = "linear" **ppm** Numeric vector describing the spectral axis (e.g. chemical shift). Must have length equal to ncol(X). **edge_frac** Fraction of points at each spectrum edge used to estimate the baseline. Default 0.1.

Details

Baseline correction is performed independently for each spectrum.

The "asls" method estimates a smooth baseline using asymmetric least squares smoothing, which is well suited for spectra with positive peaks such as NMR metabolomics data.

The "linear" method estimates a straight baseline from the edges of each spectrum and subtracts the fitted trend.

The returned object includes a .stamp attribute recording the baseline correction method and parameters used.

Value

Numeric matrix containing baseline-corrected spectra with the same dimensions as X. If X is a metabom8 dat object, the corrected matrix replaces the X component while preserving associated metadata.

References

Eilers PHC, Boelens HFM (2005). Baseline correction with asymmetric least squares smoothing.

See Also

Other preprocessing: align_segment(), align_spectra(), binning(), calibrate(), correct_lw(), pqn(), print_preprocessing()

Examples

```
data(hiit_raw)

plot_spec(hiit_raw$X[1,], hiit_raw$ppm, shift=c(3.1,4), backend='base')
hiit_proc <-
  hiit_raw |>
```

```
  excise() |>
  correct_baseline()

plot_spec(hiit_proc$X[1,], hiit_proc$ppm, shift=c(3.1,4), backend='base', add=TRUE, col='red')
```

`correct_lw` *Linewidth correction by scaling spectra to a reference linewidth*

Description

Applies a multiplicative correction to each spectrum to compensate for linewidth-induced peak height variation, using a reference peak region (e.g. TSP). The correction assumes an empirical power-law relationship between peak height and linewidth:

Usage

```
correct_lw(
  x,
  lw_ref = 0.8,
  beta = 0.6,
  shift = c(-0.1, 0.1),
  ppm = NULL,
  sf = NULL,
  estimate_beta = TRUE,
  beta_min_n = 6,
  only_if_improves = TRUE,
  notes = NULL
)
```

Arguments

`x`	A numeric matrix (samples x variables) or a named list containing `X`, `ppm`, and optionally `meta`.
`lw_ref`	Numeric reference linewidth (FWHM) to which spectra are normalized. If `NULL`, the median linewidth across samples is used.
`beta`	Numeric exponent governing the linewidth-to-height relationship. Ignored if `estimate_beta = TRUE` and sufficient samples are available.
`shift`	Numeric vector of length 2 specifying the ppm region used for linewidth estimation and peak height measurement (e.g. `c(-0.1, 0.1)` for TSP).
`ppm`	Optional numeric ppm axis. If `NULL`, inferred from `attr(X, "m8_axis")$ppm` or numeric `colnames(X)`.
`sf`	Optional spectrometer frequency (MHz). If `NULL`, obtained from `meta$a_SFO1` in list input or `attr(X, "m8_meta")$a_SFO1`.
`estimate_beta`	Logical; if `TRUE`, estimate `beta` from a log-log regression of reference peak height versus FWHM.
`beta_min_n`	Minimum number of valid samples required to estimate `beta`.
`only_if_improves`	Logical; if `TRUE`, apply correction only if the CV of the reference peak height decreases after correction.
`notes`	Optional character string appended to the preprocessing log.

Details

$$I \propto \mathrm{FWHM}^{-\beta}$$

Spectra are scaled such that all samples are adjusted to a common reference linewidth `lw_ref`:

$$X_{\mathrm{adj}} = X \cdot \left(\frac{\mathrm{FWHM}}{lw_{\mathrm{ref}}} \right)^{\beta}$$

where FWHM is estimated per spectrum within the specified `shift` region (typically containing an internal reference signal).

If `estimate_beta = TRUE`, the exponent `beta` is estimated from a log-log regression between peak height (max within `shift`) and FWHM across samples. Otherwise, the user-supplied `beta` is used.

The correction is only applied if it reduces the coefficient of variation (CV) of the reference peak height across samples when `only_if_improves = TRUE`.

Input may be either:

- a numeric matrix `X` (samples x variables), with ppm axis supplied via `attr(X, "m8_axis")$ppm` or numeric `colnames(X)`, or
- a named list with elements `X`, `ppm`, and optionally `meta`.

Attributes `"m8_prep"`, `"m8_axis"`, and `"m8_meta"` are preserved and the applied correction is appended to the preprocessing log via `.m8_stamp()`.

This correction removes systematic peak height variation due to spectral broadening (e.g. shimming differences) under the assumption that the internal reference signal has constant concentration across samples. The exponent `beta` reflects the empirical sensitivity of peak height to linewidth in the current acquisition and processing regime.

This adjustment improves comparability of signal intensities across spectra by reducing linewidth-induced amplitude bias, thereby enhancing the detectability of small effects in downstream multivariate analyses (e.g. PLS-type models).

Value

An object of the same type as input:

- If input is a matrix, returns a corrected matrix with attributes preserved and updated.
- If input is a list, returns the same list with corrected `$X`.

See Also

`lw`

Other preprocessing: `align_segment()`, `align_spectra()`, `binning()`, `calibrate()`, `correct_baseline()`, `pqn()`, `print_preprocessing()`

Examples

```
data(covid_raw)
# matrix input with ppm in m8_axis
Xcor <- correct_lw(covid_raw, shift = c(-0.1, 0.1))
```

covid — *COVID-19 blood plasma proton NMR spectra (processed)*

Description

1D proton NMR spectra from SARS-CoV-2–positive patients (n = 10) and healthy controls (n = 13), collected in Perth, Western Australia. Spectra were pre-processed (residual water and signal-free regions excised, baseline corrected, and normalized to account for line-width differences).

Format

A numeric matrix/data frame with 23 rows (samples) and 27,819 columns (chemical shift variables in parts per million, ppm).

Details

FIDs were acquired with a standard 90° RF pulse sequence on a 600 MHz Bruker Avance II spectrometer using IVDr methods for blood plasma (300 K, 32 scans). The spectrometer was equipped with a double-resonance broadband (BBI) probe and a refrigerated autosampler.

Source

Australian National Phenome Centre (ANPC), Murdoch University.

References

Kimhofer et al. (2020) doi:10.1021/acs.jproteome.0c00280

Examples

```
data(covid)
```

covid_raw — *COVID-19 blood plasma proton NMR spectra (raw)*

Description

1D proton NMR spectra from SARS-CoV-2–positive patients (n = 10) and healthy controls (n = 13), collected in a research study in Perth, Western Australia. Spectra are raw and require processing before statistical analysis (see `?covid` for processed spectra).

Format

A numeric matrix/data frame with 23 rows and 29,782 columns:

rows Spectra (samples)

columns Chemical shift variables in parts per million (ppm)

Details

FIDs were acquired using a Carr–Purcell–Meiboom–Gill (CPMG) pulse sequence on a 600 MHz Bruker Avance II spectrometer using IVDr methods for blood plasma (300 K, 32 scans). The spectrometer was equipped with a double-resonance broadband (BBI) probe and a refrigerated autosampler at 4°C.

Source

Australian National Phenome Centre (ANPC), Murdoch University.

References

Kimhofer et al. (2020) doi:10.1021/acs.jproteome.0c00280

Examples

```
data(covid_raw)
```

---

cv_anova *Cross-validated ANOVA for O-PLS models*

Description

Performs a cross-validated ANOVA (CV-ANOVA) test for OPLS models. The function compares residuals from a null model and a model using cross-validated predictive scores to assess the significance of the OPLS model.

Usage

```
cv_anova(smod)
```

Arguments

`smod`	An object of class `m8_model`, generated by function **opls** in the **metabom8** package.

Details

Interpretation of the CV-ANOVA table

The CV-ANOVA compares two nested linear models:

- *Null model*: $Y = \beta_0 + \epsilon$
- *Full model*: $Y = \beta_0 + \beta_1 t_{\text{pred,cv}} + \epsilon$

where $t_{\text{pred,cv}}$ represents the cross-validated predictive component score(s) obtained from the OPLS model.

The ANOVA table contains:

- **SS** – Sum of Squares
- **DF** – Degrees of Freedom
- **MS** – Mean Squares (SS / DF)

- **F_value** – F statistic comparing model vs. null
- **p_value** – P-value from the F-test

The *Regression* row quantifies the reduction in residual variance achieved by including the cross-validated predictive score(s). The *Residual* row represents the unexplained variance of the full model.

The F-statistic is computed as:

$$F = \frac{(RSS_0 - RSS_1)/df_{reg}}{RSS_1/df_{res}}$$

where RSS_0 and RSS_1 are the residual sums of squares of the null and full models, respectively.

Predictive interpretation

A small p-value (typically < 0.05) indicates that the cross-validated predictive score(s) significantly reduce residual variance compared to an intercept-only model. This suggests that the OPLS model captures statistically meaningful predictive structure in Y.

A large p-value indicates that the predictive component does not explain Y significantly better than chance, implying weak or unstable predictive performance.

Importantly, CV-ANOVA evaluates the *linear explanatory power of the cross-validated predictive scores*, not the descriptive separation of the latent space. A model may show visual class separation or moderate R^2 yet fail CV-ANOVA if predictive performance is weak.

Sample size strongly affects statistical power. With small n, even models with moderate predictive strength may not reach statistical significance.

CV-ANOVA is intended for continuous response (regression) models.

Value

A `data.frame` containing:

- `SS` - Sum of Squares
- `DF` - Degrees of Freedom
- `MS` - Mean Squares
- `F_value` - F statistic
- `p_value` - P-value from the F-test

References

Eriksson, L., et al. (2008). CV-ANOVA for significance testing of PLS and OPLS models. *Journal of Chemometrics*, 22(11-12), 594–600.

See Also

Other model_validation: `dmodx()`, `opls_perm()`

Examples

```
data("covid")

X <- covid$X
Y <- as.numeric(factor(covid$an$type)) - 1
```

```
scaling <- uv_scaling(center=TRUE)
cv <- balanced_mc(10, split=2/3, type='R', probs = c(0, 0.5, 1))
mod <- opls(X, Y, scaling, cv)

cv_anova(mod)
```

---

dmodx *Distance to the Model in X-Space (DModX)*

Description

Calculates the orthogonal distance of each observation to an OPLS model in X-space. DModX can be used for identifying outliers.

Usage

```
dmodx(mod, plot = TRUE)
```

Arguments

`mod`	An OPLS model object of class `m8_model` (engine = "opls").
`plot`	Logical. If TRUE, a plot of DModX values with an approximate cutoff is shown.

Details

DModX is computed from the X-residual matrix as a scaled RMSE of residuals. The empirical cutoff (dashed line in plot) uses a t-based confidence interval and assumes approximate normality of DModX values. This assumption may not be satisfied in all datasets, so the resulting threshold should be regarded as a pragmatic heuristic for outlier detection.

Value

A data frame with columns `ID`, `DmodX`, and `passed`.

See Also

Other model_validation: `cv_anova()`, `opls_perm()`

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
dX <- dmodx(model)
print(dX[[1]])
df <- dX[[2]]; head(df)
```

`ellipse2d` *Calculate 2D Hotelling T^2 Ellipse*

Description

Generates coordinates of a two-dimensional ellipse corresponding to a Hotelling T^2 region projected onto selected dimensions.

Usage

```
ellipse2d(obj, dims = c(1, 2), npoints = 100)
```

Arguments

`obj`	A `list` as returned by `hotellingsT2()`.
`dims`	Integer vector of length 2 specifying which dimensions to project onto. Default is `c(1, 2)`.
`npoints`	Integer. Number of points used to approximate the ellipse. Default is 100.

Details

The ellipse is obtained by projecting the multivariate T^2 ellipsoid onto the specified dimensions. The scaling is derived from the Hotelling T^2 statistic and accounts for sample size and dimensionality.

Value

A `data.frame` with columns `x` and `y` containing ellipse coordinates.

Examples

```
set.seed(1)
X <- cbind(rnorm(100), rnorm(100) + 0.5)

t2 <- hotellingsT2(X)
ell <- ellipse2d(t2)

plot(X[,1], X[,2], asp = 1, pch = 16,
     xlab = "Score 1", ylab = "Score 2")
lines(ell$x, ell$y, col = "red", lwd = 2)
```

`excise` *Excise Chemical Shift Regions from 1D NMR Spectra*

Description

Removes specified chemical shift regions from 1D ^{1}H NMR spectra. By default, commonly excluded metabolomics regions are removed (upfield/downfield noise, water, urea).

Usage

```
excise(x, ppm = NULL, regions = NULL)
```

Arguments

`x`	Numeric matrix or vector. Spectra in rows and variables in columns.
`ppm`	Numeric vector of chemical shift positions (ppm). If omitted, `ppm` is inferred from `colnames(X)`.
`regions`	Named list of numeric vectors (length 2), specifying ppm regions to remove. Each element must define lower and upper bounds.

Details

Default regions removed (ppm):

- Upfield noise: `[min(ppm), 0.25]`
- Residual water: `[4.5, 5.2]`
- Urea region: `[5.5, 6.0]`
- Downfield noise: `[9.7, max(ppm)]`

Removed regions are recorded in `attr(X, "m8_prep")` if present. Updated ppm values are stored in column names and in `attr(X, "m8_axis")$ppm`.

Value

Numeric matrix with specified chemical shift regions removed.

Examples

```
set.seed(1)
ppm <- seq(0, 10, length.out = 1000)
X <- matrix(rnorm(100 * length(ppm)), nrow = 100)

Xe <- excise(X, ppm)

dim(Xe)
names(attributes(Xe))
```

fitted *Extract fitted Y values*

Description

Extract fitted Y values

Usage

```
fitted(object, ...)

## S4 method for signature 'm8_model'
fitted(object)
```

Arguments

object	An object of class m8_model.
...	Additional arguments (currently ignored).

Value

Numeric vector or matrix containing the fitted response values.

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
show(model)
Y_hat_dummy <- fitted(model)
```

get_idx *Select Indices for a Chemical Shift Region*

Description

Returns the indices of the chemical shift vector (ppm) that fall within the specified range.

Usage

```
get_idx(range = c(1, 5), ppm)

get.idx(range = c(1, 5), ppm)
```

Arguments

range	Numeric vector of length 2 specifying lower and upper bounds (order does not matter).
ppm	Numeric vector. The full chemical shift axis (in ppm).

Value

Integer vector of indices corresponding to ppm values within the given range.

See Also

Other NMR: .dynamicIntervalsMedian()

Examples

```
data(covid_raw)
X <- covid_raw$X
ppm <- covid_raw$ppm
idx_tsp <- get_idx(c(-0.1, 0.1), ppm)
ppm[range(idx_tsp)]
plot(ppm[idx_tsp], X[1, idx_tsp], type = 'l')
```

---

get_provenance *Retrieve metabom8 provenance metadata*

Description

Extracts preprocessing provenance stored in the "m8_prep" attribute. Allows access to the full processing log, a specific step, or a specific parameter within a step.

Usage

```
get_provenance(x, step = NULL, param = NULL)
```

Arguments

x	A metabom8 object (named list with element X) or a numeric matrix containing metabom8 provenance metadata.
step	Optional. Either: • Numeric index of the preprocessing step • Character string matching the recorded step name If NULL, the full provenance log is returned.
param	Optional character string specifying a parameter name within the selected step. If provided, only this parameter value is returned.

Details

Provenance metadata are recorded automatically by metabom8 preprocessing functions and stored as structured attributes on the spectral matrix. This function provides programmatic access to these records.

Value

Depending on the arguments:

- Full provenance list (if step = NULL)
- A single preprocessing step (if step specified)
- A single parameter value (if param specified)

See Also

Other provenance: `add_note()`, `print_provenance()`

Examples

```
data(hiit_raw)

hiit_proc <- hiit_raw |>
  calibrate(type = "tsp") |>
  excise()

# Retrieve full log
log <- get_provenance(hiit_proc)

# Retrieve specific step
get_provenance(hiit_proc, step = 2)

# Retrieve parameter from a named step
get_provenance(hiit_proc, step = "calibrate", param = "target")
```

---

hiit_raw *High-intensity interval training (HIIT) 1H NMR urine dataset*

Description

Urine samples collected from a single individual performing a VO_2max-type exercise protocol over a time period of 3h.

Format

A list with four elements:

X Numeric matrix of spectral intensities (samples x variables).

ppm Numeric vector of chemical shift values corresponding to columns of `X`.

meta Data frame containing acquisition metadata.

df Data frame containing sample annotations.

Details

The spectra were acquired on a 600 MHz Bruker NMR spectrometer. The dataset is included for demonstration of preprocessing and modelling workflows in **metabom8**.

Source

Example dataset bundled with the package.

Examples

```
data(hiit_raw)
```

`hotellingsT2` *Hotelling T^2 Statistic*

Description

Computes the Hotelling T^2 statistic defining a multivariate confidence region for a set of observations. The region corresponds to an ellipsoid in p-dimensional space and is commonly visualised as an ellipse in two-dimensional (OPLS) score plots.

Usage

```
hotellingsT2(X, alpha = 0.95)
```

Arguments

`X`	Numeric matrix with observations in rows and variables (dimensions) in columns.
`alpha`	Numeric scalar. Confidence level for the T^2 region. Default is 0.95.

Details

The Hotelling T^2 region is defined as

$$(x - \mu)^T S^{-1} (x - \mu) \leq c^2$$

where $c^2 = (p(n-1)/(n-p))F_{p,n-p}(\alpha)$.

Value

A named `list` with elements:

center Numeric vector of column means.

cov Sample covariance matrix.

c2 Squared radius of the Hotelling T^2 region.

Examples

```
set.seed(1)
X <- matrix(rnorm(200), ncol = 2)
t2 <- hotellingsT2(X)
ell <- ellipse2d(t2)
plot(X, asp = 1)
lines(ell$x, ell$y, col = "red", lwd = 2)
```

`kfold` *K-fold cross-validation strategy*

Description

K-fold cross-validation strategy

Usage

```
kfold(k)
```

Arguments

`k`	Integer number of folds.

Details

Partitions the data into `k` folds. Each fold is used once as a test set, with the remaining folds used for training. No stratification is applied; folds are created by random partitioning.

Value

A named `list` with elements:

train List of integer vectors containing training set indices for each resampling iteration.

strategy Character string indicating the resampling strategy.

n Integer. Number of samples in the dataset.

seed Integer. Random seed used to generate the resampling splits, ensuring reproducibility.

See Also

Other resampling strategies: `balanced_boot()`, `balanced_mc()`, `mc()`, `stratified_kfold()`

Examples

```
n <- 100
thr <- 1.5
Y <- c(rnorm(80, thr - 3, 0.3), rnorm(20, thr + 3, 0.3))  # unbalanced outcome
mean(Y > thr)
cv_k  <- kfold(k = 10)
k_inst <- metabom8:::.arg_check_cv(cv_pars=cv_k, model_type='R', n=n, Y_prepped=cbind(Y))
sapply(k_inst$train, function(i) length(i))
```

list_preprocessing *List available preprocessing steps*

Description

Returns a named character vector describing the preprocessing operations implemented in **metabom8**. For more information on individual functionalities please refer to the function help pages.

Usage

```
list_preprocessing()
```

Value

A named character vector with preprocessing function identifiers as names and short descriptions as values.

Examples

```
list_preprocessing()
```

loadings,m8_model-method *Model loadings*

Description

Model loadings

Usage

```
## S4 method for signature 'm8_model'
loadings(x, orth = FALSE, ...)
```

Arguments

x	An object of class m8_model.
orth	Logical indicating whether orthogonal scores should be returned (only applicable for OPLS models).
...	Additional arguments (currently ignored).

Value

Numeric vector or matrix containing loadings.

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
show(model)
P <- loadings(model)
Po <- loadings(model, orth = TRUE)
dim(P)
dim(Po) == dim(P)
```

---

lw *Full Width at Half Maximum (FWHM) Estimation*

Description

Estimates the full width at half maximum (FWHM; line width) of a singlet-like peak within a specified chemical-shift range for each spectrum.

Usage

```
lw(X, ppm = NULL, shift = c(-0.1, 0.1), sf)
```

Arguments

`X`	Numeric matrix (spectra in rows) *or* a named list as returned by `read1d`/`read1d_proc` containing `X`, `ppm`, and `meta`.
`ppm`	Numeric vector of chemical shift values (ppm) corresponding to columns of `X`. If `NULL`, `ppm` is inferred in the following order: 1. `attr(X, "m8_axis")$ppm` (if present), 2. numeric `colnames(X)` (if present).
`shift`	Numeric vector of length 2. Chemical shift range containing the peak (e.g., `c(-0.1, 0.1)` for TSP).
`sf`	Spectrometer frequency in MHz. Either a single numeric (recycled across spectra) or a numeric vector of length `nrow(X)` (one value per spectrum). If `X` is a list input and `sf` is missing, `sf` is taken from `X$meta$a_SFO1` when available. If `sf` is missing for a matrix input, the function attempts to use `attr(X, "m8_meta")$a_SFO1` when present.

Details

For each spectrum, the function:

1. extracts the region defined by `shift`,
2. finds the peak apex within that region,
3. computes the half-height level relative to the local baseline (minimum in the window),
4. estimates the left and right half-height crossing points by linear interpolation,
5. converts the width from ppm to Hz using `sf` (MHz), i.e. `Hz = ppm * sf`.

If no valid half-height crossings can be found (e.g., very low SNR or truncated peak), `NA` is returned for that spectrum.

Value

Numeric vector of FWHM values in Hz (length `nrow(X)`).

Note

The ppm axis may be increasing or decreasing; FWHM is computed as an absolute width and is therefore independent of axis direction.

Examples

```
# Simulated NMR peaks with different linewidths
ppm <- seq(-0.2, 0.2, length.out = 1000)

# generate peaks with increasing width
sds <- seq(0.01, 0.03, length.out = 10)

X <- t(sapply(sds, function(s)
  dnorm(ppm, mean = 0, sd = s)
))

sf <- 600  # spectrometer frequency in MHz

fwhm_vals <- lw(X, ppm = ppm, shift = c(-0.1, 0.1), sf = sf)

plot(sds, fwhm_vals,
  xlab = "Gaussian sd",
  ylab = "Estimated FWHM (Hz)",
  pch = 16)
```

`m8_model-class` *m8_model class Model object returned by* `pca()`, `pls()`, *and* `opls()`.

Description

m8_model class Model object returned by `pca()`, `pls()`, and `opls()`.

Usage

```
## S4 method for signature 'm8_model'
summary(object)

## S4 method for signature 'm8_model'
show(object)
```

Arguments

`object`	An object of class `m8_model`.

Value

An object of class `m8_model`.

Functions

- `summary(m8_model)`: Summarise model performance and component selection.
- `show(m8_model)`: Show a compact model header.

Slots

`engine` Character. Model engine ("pca", "pls", "opls").

`ctrl` List. Engine-specific control and performance information.

`fit` List. Fitted data (engine specific).

`cv` Resampling instance (may be NULL if not used).

`prep` Scaling and centering information

`provenance` Preprocessing attributes of spectral matrix X

`session` R-session information

`call` Function call

`dims` List with `n` and `p`.

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
class(model)
show(model)
```

`mc` *Monte-Carlo cross-validation strategy*

Description

Monte-Carlo cross-validation strategy

Usage

```
mc(k, split)
```

Arguments

`k`	Integer. Number of repeated random splits.
`split`	Numeric. Fraction of samples assigned to the training set (e.g. `2/3`).

Details

Monte-Carlo cross-validation generates `k` random train/test splits without replacement. No stratification is applied; samples are drawn uniformly at random.

Value

A named `list` with elements:

train List of integer vectors containing training set indices for each resampling iteration.

strategy Character string indicating the resampling strategy.

n Integer. Number of samples in the dataset.

seed Integer. Random seed used to generate the resampling splits, ensuring reproducibility.

See Also

Other resampling strategies: `balanced_boot()`, `balanced_mc()`, `kfold()`, `stratified_kfold()`

Examples

```
n <- 100
# bivariate outcome
thr <- 1.5
Y <- c(rnorm(80, thr-3, 0.3), rnorm(20, thr+3, 0.3))  # unbalanced low/high outcome
mean(Y>thr)

cv_mc <- mc(k = 10, split = 2/3)
mc_inst <- metabom8:::.arg_check_cv(cv_pars=cv_mc, model_type='R', n=n, Y_prepped=cbind(Y))
sapply(mc_inst$train, function(i) length(i))
```

`metabom8` *metabom8: A High-Performance R Package for Metabolomics Modeling and Analysis*

Description

metabom8 (pronounced *metabo-mate*) provides pipelines for 1D NMR data import, preprocessing, multivariate modeling (PCA, OPLS), metabolite identification, and visualisation. Core functions are accelerated in C++ via **Rcpp**, **RcppArmadillo**, and **RcppEigen** for improved computational performance.

Features

- Import, preprocessing, and analysis of 1D NMR spectra.
- **Principal Components Analysis (PCA)** with *back-scaled* loadings (projected back to the spectral domain and visualized as spectra).
- **Orthogonal Partial Least Squares (OPLS)** fitted iteratively via the NIPALS algorithm, with an *automatic stopping criterion* to determine the optimal number of components.
- Automatic model selection using cross-validated performance (R^2, Q^2, and cross-validated AUC for classification), with safeguards against overfitting.
- Robust statistical validation for small to large sample sizes: stratified Monte Carlo cross-validation and k-fold CV.
- Model diagnostics and validation (DModX, permutation testing).
- Metabolite identification via STOCSY and STORM.
- Native C++ acceleration via **RcppArmadillo** and **RcppEigen**.

Vignettes

- Getting Started: `vignette("Getting Started")`

Author(s)

Maintainer: Torben Kimhofer <tkimhofer@gmail.com> (ORCID)

See Also

Useful links:

- https://tkimhofer.github.io/metabom8/
- Report bugs at https://github.com/tkimhofer/metabom8/issues

---

minmax *Min-Max Scaling to* $[0, 1]$

Description

Scales a numeric vector to the range $[0, 1]$ using min-max normalization. This is a special case of scRange.

Usage

```
minmax(x, na.rm = FALSE)
```

Arguments

x	Numeric vector. Input values to be scaled.
na.rm	Logical; if TRUE, ignore NAs when computing the range.

Details

The scaled values are computed as:

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Equivalent to `scRange(x, ra = c(0, 1))`.

Value

A numeric vector of the same length as x, scaled to the range $[0, 1]$.

See Also

`scRange()` for flexible output ranges.

Examples

```
x <- rnorm(20)
plot(x, type = 'l'); abline(h = range(x), lty = 2)
points(minmax(x), type = 'l', col = 'blue')
abline(h = c(0, 1), col = 'blue', lty = 2)
```

noise_sd *Estimate Noise Standard Deviation in 1D NMR Spectra*

Description

Estimates the noise standard deviation (σ) for each spectrum from a signal-free ppm region. The default estimator is robust (MAD-based) and suitable for signal-to-noise calculations.

Usage

```
noise_sd(
  X,
  ppm = NULL,
  where = c(14.6, 14.7),
  method = c("mad", "sd", "p95"),
  baseline_correct = FALSE,
  lambda = 10000,
  min_points = 50L,
  ns = NULL,
  normalise_scans = FALSE
)
```

Arguments

X	Numeric matrix (spectra in rows), numeric vector (single spectrum), or a named list as returned by read1d/read1d_proc containing X, ppm, and meta.
ppm	Numeric vector of chemical shift values (ppm) corresponding to columns of X. If NULL, ppm is inferred in the following order: 1. X$ppm if X is a list input, 2. attr(X, "m8_axis")$ppm (if present), 3. numeric colnames(X) (if present).
where	Numeric vector of length 2. ppm range used for noise estimation. Should be free of metabolite signals (default c(14.6, 14.7)).
method	Character. Noise estimator: "mad" (default), "sd", or "p95" (legacy amplitude).
baseline_correct	Logical. If TRUE, subtract a smooth baseline in the noise window using asymmetric least squares (asysm). Default FALSE.
lambda	Numeric. Smoothing parameter for asysm when baseline_correct = TRUE.
min_points	Integer. Minimum number of points required in the noise window. May be a single value (recycled across spectra) or a numeric vector of length nrow(X). If X is a list input as returned by read1d/read1d_proc and ns is NULL, the function attempts to extract the number of scans from X$meta$a_NS when available.
ns	Number of scans / transients (required if normalise_scans=TRUE)
normalise_scans	Logical. If TRUE, noise estimates are multiplied by $\sqrt{NS}$ to account for the theoretical scaling of noise with the number of scans ($\sigma \propto 1/\sqrt{NS}$). This is useful when comparing noise levels across spectra acquired with different numbers of scans. Default is FALSE.

Details

In NMR spectroscopy, noise scales predictably with the number of scans (NS). For otherwise identical acquisition settings:

- Signal increases approximately proportional to NS.
- Noise increases approximately proportional to $\sqrt{NS}$.
- Consequently, signal-to-noise ratio (SNR) increases proportional to $\sqrt{NS}$.

Equivalently, the noise standard deviation scales as:

$$\sigma(NS) \propto \frac{1}{\sqrt{NS}},$$

assuming a fixed underlying signal scale and comparable acquisition conditions.

To compare noise levels across datasets acquired with different numbers of scans, a scan-normalised noise estimate may be used:

$$\sigma_{\text{norm}} = \sigma \cdot \sqrt{NS}.$$

Under stable receiver gain and processing conditions, this normalised noise should be approximately constant across runs.

Value

Numeric vector of noise estimates (length `nrow(X)`).

Examples

```
data(hiit_raw)
X <- hiit_raw$X
ppm <- hiit_raw$ppm
sigma <- noise_sd(X, ppm, where = c(10,11))
plot(hiit_raw$meta$a_NS, sigma,
  xlab = "Number of scans (NS)",
  ylab = expression(sigma~"(noise estimate)"),
   pch = 16)
 lines(lowess(hiit_raw$meta$a_NS, sigma), col = "red", lwd = 2)
```

---

norm_eretic *Normalise Spectra Using ERETIC Signal*

Description

Normalises 1D NMR spectra using an ERETIC reference signal. By default the ERETIC position is discovered in a search window and the spectra are normalised by the integral in a narrow window around the detected position. Power users can supply `pos` to enforce a fixed ERETIC position.

Usage

```
norm_eretic(
  X,
  integr = FALSE,
  ppm = NULL,
  pos = NULL,
  search = c(10, 16),
  width = 0.2,
  warn_absent = TRUE,
  noise_win = c(9.8, 10.2),
  snr_factor = 10
)
```

Arguments

X	Numeric matrix or vector. NMR spectra with spectra in rows. If ppm is not provided, it is inferred from colnames(X).
integr	Logical. If TRUE, returns the ERETIC integral per spectrum. If FALSE, returns the normalised spectra.
ppm	Numeric vector of chemical shift positions. If NULL, inferred from colnames(X).
pos	Numeric scalar or NULL. If NULL (default), ERETIC position is discovered within search. If provided, the ERETIC window is centered on pos.
search	Numeric vector of length 2. Ppm window used to discover ERETIC when pos = NULL.
width	Numeric scalar. Full integration window width in ppm (default 0.2 gives pos ± 0.1).
warn_absent	Logical. If TRUE, warn when ERETIC integral is zero/invalid for individual spectra.
noise_win	Numeric vector of length 2. Ppm window for estimating noise intensity.
snr_factor	Numeric scaler. Minimum Signal-to-Noise Ratio required for detecting the ERETIC peak.

Details

Binning/normalisation history is recorded in attr(X, "m8_prep") if present. Ppm axis values are stored in attr(X, "m8_axis")$ppm.

Value

If integr = TRUE, numeric vector of ERETIC integrals. If integr = FALSE, numeric matrix of normalised spectra.

Examples

```
set.seed(123)

n <- 1000
ppm <- seq(0, 14, length.out = n)

gauss <- function(x, c, h, w = 0.05) {
  h * exp(-((x - c)^2) / (2 * w^2))
```

```
}

heights <- c(100, 80, 60, 40, 20)

spectra <- sapply(heights, function(h)
  rnorm(n, 0, 0.01) + gauss(ppm, 12, h)
)

spectra <- t(spectra)   # 5 spectra × 1000 variables

plot_spec(spectra, ppm, shift = c(11, 13))

X_norm <- norm_eretic(spectra, ppm=ppm)
plot_spec(X_norm, ppm, shift=c(11, 13))
```

`opls` *Fit an Orthogonal Partial Least Squares (O-PLS) model*

Description

Fits a supervised Orthogonal Partial Least Squares (O-PLS) model using a NIPALS-based algorithm with optional cross-validation and automatic component selection.

Usage

```
opls(X, Y, scaling, validation_strategy)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, columns = variables).
`Y`	Numeric matrix or factor vector of responses.
`scaling`	A scaling strategy object (e.g., `uv_scaling(center = TRUE)`), specifying model-internal centering and/or scaling applied during fitting. This does not modify the original spectral matrix.
`validation_strategy`	A cross-validation strategy object defining how resampling is performed (e.g., k-fold, Monte Carlo).

Details

O-PLS decomposes the predictor matrix into:

- One predictive component capturing variation correlated with `Y`
- Orthogonal components capturing structured variation in `X` unrelated to `Y`

Predictive and orthogonal components are estimated sequentially. Cross-validated performance metrics (e.g., Q^2, R^2, classification AUC) are computed for each model configuration according to the supplied `validation_strategy`.

The model extracts a single predictive component and iteratively adds orthogonal components until the `stopRule` indicates overfitting or `maxPCo` is reached.

Scaling specified via `scaling` is applied internally during model fitting and does not alter the input matrix `X`. Spectral preprocessing (e.g., alignment or baseline correction) should be performed prior to model fitting.

The returned model object stores:

- Predictive and orthogonal component models
- Cross-validation results
- Performance metrics (R^2, Q^2, AUC)
- Model control parameters
- Input data provenance metadata
- Session information for reproducibility

Value

An object of class `m8_model` containing the fitted O-PLS model, cross-validation results, and performance statistics.

See Also

`pls`, `uv_scaling`

Other modelling: `pca()`, `pls()`

Examples

```
data(covid)

cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)

show(model)
summary(model)

# scores
Tp <- scores(model)
To <- scores(model, orth=TRUE)

t2 <- hotellingsT2(cbind(Tp, To))
ell <-ellipse2d(t2)

plot(Tp, To, asp = 1,
  col = as.factor(covid$an$type),
  xlim = range(c(Tp, ell$x)),
  ylim = range(c(To, ell$y))
 )
lines(ell$x, ell$y, col = "grey", lty=2)

# loadings & vip's
Pp <- loadings(model)
Po <- loadings(model, orth=TRUE)
vips <- vip(model)

x=covid$ppm
y = Pp * apply(covid$X, 2, sd)
```

```
palette <-  colorRampPalette(c("blue", "cyan", "yellow", "red"))(100)
idx <- cut(vips, breaks = 100, labels = FALSE)
plot(x, y, type = "n", xlim = rev(range(x)), xlab='ppm', ylab='t_pred_sc')

for (i in seq_len(length(x) - 1)) {
  segments(x[i], y[i], x[i+1], y[i+1], col = palette[idx[i]], lwd = 2)
}
```

`opls_perm` *OPLS Model Validation via Y-Permutation*

Description

Performs Y-permutation tests to assess the robustness of OPLS models by comparing model performance statistics on real vs. permuted response labels.

Usage

```
opls_perm(smod, n = 10, plot = TRUE, mc = FALSE)
```

Arguments

`smod`	An OPLS model object of class `OPLS_metabom8`.
`n`	Integer. Number of permutations to perform.
`plot`	Logical. If `TRUE`, generates a visual summary of permutation statistics.
`mc`	Logical. If `TRUE`, enables multicore processing (currently not implemented).

Details

Each permutation shuffles the response labels and fits a new OPLS model. The function captures model statistics (R2, Q2, AUC) to compare against the non-permuted model. This helps determine whether the original model performance is better than expected by chance.

Value

A `data.frame` with model metrics (e.g., R2, Q2, AUROC) from both permuted and original models.

References

Wiklund, S. et al. (2008). A Tool for Improved Validation of OPLS Models. *Journal of Chemometrics*, 22(11–12), 594–600.

See Also

Other model_validation: `cv_anova()`, `dmodx()`

pareto_scaling *Pareto Scaling Leaves variables unscaled. Optional centering.*

Description

Pareto Scaling Leaves variables unscaled. Optional centering.

Usage

```
pareto_scaling(center = FALSE)
```

Arguments

center	Logical. If TRUE, variables are mean-centered before scaling.

Details

Scales variables by the square root of their standard deviation.

Value

A list with elements:

X Numeric matrix containing the scaled data.

prep List describing the preprocessing, including centering and scaling parameters (center, scale, X_mean, X_sd).

See Also

Other scaling_strategies: unscaled(), uv_scaling()

Examples

```
paritalUV <- pareto_scaling(center=TRUE)
X <- matrix(c(10,10, 0,0, 0, 10, 0, 1000), ncol=4)
X_scaled <- prep_X(paritalUV, X)
str(X_scaled)
X_scaled$X
```

pca *Principal Component Analysis (PCA)*

Description

Fits an unsupervised Principal Component Analysis (PCA) model to a spectral data matrix. Model-internal centering and scaling are controlled via a ScalingStrategy object and do not modify the input matrix.

The default backend uses an internal NIPALS implementation. Alternatively, PCA algorithms from the **pcaMethods** package (e.g. "ppca", "svd", "robustPca") can be used.

Usage

```
pca(X, scaling, ncomp, method = "nipals")
```

Arguments

X	Numeric matrix (rows = samples, columns = variables).
scaling	A ScalingStrategy object defining model-internal centering and scaling (e.g. uv_scaling(center = TRUE)).
ncomp	Integer. Number of principal components to compute.
method	Character. PCA backend. Use "nipals" for the internal implementation or any method supported by pcaMethods::pca().

Details

PCA decomposes X into orthogonal score and loading matrices:

$$X \approx TP$$

where:

- T contains the principal component scores
- P contains the loadings

The number of components is fixed by ncomp. Unlike supervised models, PCA does not use cross-validation or stopping rules.

Scaling and centering are applied internally during model fitting. The original input matrix is not modified.

Value

An object of class m8_model with engine = "pca". The object contains:

- fit$t: Score matrix (samples × components)
- fit$p: Loading matrix (components × variables)
- ctrl: Model control information (variance explained, scaling settings)
- provenance: Attributes inherited from the input matrix

See Also

uv_scaling

Other modelling: opls(), pls()

Examples

```
data(covid)

uv <- uv_scaling(center=TRUE)
model <- pca(X=covid$X, scaling=uv, ncomp=2)

show(model)
summary(model)
```

```
Tx <- scores(model)
Px <- loadings(model)

t2 <- hotellingsT2(Tx)
ell <-ellipse2d(t2)

# scores plot
plot(Tx, asp = 1,
  col = as.factor(covid$an$type),
  xlim = range(c(Tx[1,], ell$x)),
  ylim = range(c(Tx[2,], ell$y))
 )
lines(ell$x, ell$y, col = "grey", lty=2)
```

`plotStocsy` *Plot STOCSY result*

Description

Generates a STOCSY plot (covariance trace coloured by absolute correlation).

Usage

```
plotStocsy(stoc_mod, shift = c(0, 10), title = NULL)
```

Arguments

`stoc_mod`	An object of class `m8_stocsy1d` returned by `stocsy()`.
`shift`	Numeric vector of length 2 specifying the chemical shift range (ppm).
`title`	Optional character plot title.

Value

A `ggplot2` object.

Examples

```
# st <- stocsy(X, ppm, driver = 5.233, plotting = FALSE)
# plotStocsy(st, shift = c(5.15, 5.30), title = "Glucose")

data(covid)
cs = 5.233 # anomeric H of gluc
s1 <- stocsy(covid$X, driver=cs, plotting = FALSE)
plotStocsy(s1)
```

plot_spec *Plot 1D NMR Spectra*

Description

Plot one or multiple 1D ¹H NMR spectra using different rendering backends.

Usage

```
plot_spec(
  x,
  ppm = NULL,
  shift = c(0, 10),
  backend = c("plotly", "base", "ggplot2"),
  add = FALSE,
  ...
)

spec(...)

matspec(...)
```

Arguments

x	Numeric vector (single spectrum) or numeric matrix (spectra in rows).
ppm	Numeric vector of chemical shift values. Must match ncol(x).
shift	Numeric vector of length 2. Chemical shift window to display (e.g., c(0, 10)).
backend	Character. Rendering backend. One of "plotly", "base", or "ggplot". Defaults to "plotly".
add	Logical. If TRUE and backend = "base", add spectra to an existing plot.
...	Additional arguments passed to the selected backend.

Details

The function accepts both single spectra and matrices of spectra. Input is internally normalized to matrix form, subset to the selected ppm region, and then rendered using the chosen backend.

For large NMR datasets (e.g., >500 spectra × >10k ppm), the "base" and "plotly" backends are substantially more memory-efficient than "ggplot", which requires reshaping to long format.

Chemical shift axes are automatically displayed in decreasing order (NMR convention).

Value

- "plotly": a plotly object.
- "ggplot": a ggplot2 object.
- "base": NULL (invisibly).

Examples

```
data(hiit_raw)
plot_spec(hiit_raw)
plot_spec(hiit_raw, shift=c(-0.05,0.05), backend='base')
plot_spec(hiit_raw, shift=c(-0.05,0.05), backend='ggplot2')
```

`pls` *Fit a Partial Least Squares (PLS) model*

Description

Fits a supervised Partial Least Squares (PLS) model using a NIPALS-based algorithm with optional cross-validation and automatic component selection.

Usage

```
pls(X, Y, scaling, validation_strategy, maxPCo = 5)
```

Arguments

`X`	Numeric matrix of predictors (rows = samples, columns = variables).
`Y`	Numeric matrix or factor vector of responses.
`scaling`	A scaling strategy object (e.g., `uv_scaling(center = TRUE)`), specifying model-internal centering and/or scaling applied during fitting. This does not modify the original spectral matrix.
`validation_strategy`	A cross-validation strategy object defining how resampling is performed (e.g., k-fold, Monte Carlo).
`maxPCo`	Integer. Maximum number of predictive components to evaluate.

Details

Model components are estimated sequentially using a NIPALS-based algorithm. For each component, cross-validated performance metrics (e.g., Q^2, R^2, classification AUC) are computed according to the supplied `validation_strategy`. Component extraction stops when the `stopRule` indicates overfitting or when `maxPCo` is reached.

Scaling specified via `scaling` is applied internally during model fitting and does not alter the input matrix `X`. Spectral preprocessing steps (e.g., alignment, baseline correction) should be performed prior to model fitting.

The returned model object stores:

- Fitted component models
- Cross-validation results
- Performance metrics (R^2, Q^2, AUC)
- Model control parameters
- Input data provenance metadata
- Session information for reproducibility

Value

An object of class `m8_model` containing the fitted PLS model, cross-validation results, and performance statistics.

See Also

`opls`, `uv_scaling`

Other modelling: `opls()`, `pca()`

Examples

```
data(covid)

cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-pls(X=covid$X, Y=covid$an$type, scaling, cv)

show(model)
summary(model)

Tp <- scores(model)
Pp <- loadings(model)
```

---

ppick *Find Local Extrema in NMR Spectra (Peak Picking)*

Description

Identifies local maxima, minima, or both from smoothed NMR spectra using Savitzky–Golay filtering.

Usage

```
ppick(X, ppm, fil_p = 3, fil_n = 5, type = "max")
```

Arguments

`X`	Numeric matrix. NMR data with spectra in rows and chemical shifts in columns.
`ppm`	Numeric vector. Chemical shift values corresponding to columns in `X`.
`fil_p`	Integer. Polynomial order of the Savitzky–Golay filter.
`fil_n`	Integer. Filter length (must be odd) of the Savitzky–Golay filter.
`type`	Character. Type of extrema to return: `"max"`, `"min"`, or `"both"`.

Details

The spectra are smoothed using a Savitzky–Golay filter to reduce noise. Extrema are then detected by identifying sign changes in the first derivative of the smoothed signal.

Value

A list of data frames, one per spectrum. Each data frame contains:

- `idc`: Index of the detected peak.
- `ppm`: Chemical shift at the peak.
- `Int`: Intensity at the peak.
- `Etype`: Extrema type: `1` for minima, `-1` for maxima.

See Also

`ppick2`

Examples

```
data(covid)
X <- covid$X
ppm <- covid$ppm

peaklist <- ppick(X, ppm)
plot_spec(X[1, ], ppm, shift = c(1.2, 1.4), backend='base')
points(peaklist[[1]]$ppm, peaklist[[1]]$Int, col = 'cyan')
```

---

ppick2 *Peak picking using Savitzky–Golay derivatives*

Description

Finds local extrema in 1D spectra using Savitzky–Golay first and second derivatives. Candidate peaks are identified at zero-crossings of the first derivative and classified by the sign of the second derivative. Optional filters control peak height, prominence, SNR, curvature and minimum separation.

Usage

```
ppick2(
  X,
  ppm = NULL,
  type = c("max", "min", "both"),
  fil_p = 3L,
  fil_n = 11L,
  noise_win = NULL,
  min_snr = 10,
  min_height = NULL,
  min_prominence = NULL,
  prom_half_window_ppm = 0.02,
  min_distance_ppm = 0.005,
  min_curvature = NULL,
  keep_cols = c("height", "snr", "curvature", "prominence")
)
```

Arguments

X	Numeric matrix or vector. Spectra in rows.
ppm	Numeric vector or NULL. If NULL, inferred from colnames(X).
type	Character. "max", "min", or "both".
fil_p	Integer. Polynomial order for SG filter.
fil_n	Integer. Window length for SG filter (odd).
noise_win	Numeric length-2 vector or NULL. Ppm window used to estimate noise per spectrum. If NULL, a robust noise estimate is computed from the full spectrum (MAD of first differences).
min_snr	Numeric. Minimum SNR (peak height / noise). Set NULL to disable.
min_height	Numeric. Minimum absolute peak height (in original intensity units). NULL disables.
min_prominence	Numeric. Minimum local prominence. NULL disables.
prom_half_window_ppm	Numeric. Half-window (ppm) for prominence estimation around each peak.
min_distance_ppm	Numeric. Minimum separation between peaks (ppm). NULL disables.
min_curvature	Numeric. Minimum absolute curvature at peak (|d2|). NULL disables.
keep_cols	Character. Extra columns to keep (default keeps all computed).

Value

List of data.frames (one per spectrum). Each contains: idc, ppm, Int, Etype (+1 max, -1 min), height, snr, curvature, prominence.

Examples

```
data(covid)
X <- covid$X
ppm <- covid$ppm

peaklist <- ppick2(X[1,], ppm, min_snr=50)

plot_spec(X[1, ], ppm, shift = c(3, 4.5), backend='base')
points(peaklist[[1]]$ppm, peaklist[[1]]$Int, col = "cyan")
head(peaklist[[1]])
```

---

pqn *Probabilistic Quotient Normalisation (PQN)*

Description

Applies probabilistic quotient normalisation (PQN) to spectra. PQN estimates a sample-specific dilution factor from the median of quotients relative to a reference spectrum and scales each spectrum accordingly.

Usage

```
pqn(
  X,
  ref_index = NULL,
  total_area = FALSE,
  bin = NULL,
  iref = NULL,
  TArea = NULL
)
```

Arguments

X	Numeric matrix or data.frame. Each row is a sample spectrum and each column is a variable (e.g. chemical shift point or bin).
ref_index	Integer vector of row indices used to compute the reference spectrum. If NULL, all rows are used.
total_area	Logical. If TRUE, total area normalisation is applied to the working copy before estimating the PQN dilution factors. See Notes.
bin	Optional named list controlling binning for reference estimation, e.g. list(ppm = ppm, width = 0.05) or list(ppm = ppm, npoints = 400). If NULL, no binning is applied.
iref	Deprecated. Use ref_index.
TArea	Deprecated. Use total_area.

Details

Mechanics. Let x_i be spectrum i and r a reference spectrum. PQN computes quotients $q_{ij} = x_{ij}/r_j$ and defines the dilution factor as $d_i = 1/\mathrm{median}_j(q_{ij})$. The PQN-normalised spectrum is $x_i^{(PQN)} = d_i\, x_i$.

The reference spectrum r is typically the median spectrum across all samples or across QC samples (ref_index).

If bin is provided, r and dilution factors are computed on binned spectra, but applied to the original spectra.

Dilution factors are stored in attr(X, "m8_pqn")$dilution_factor.

Value

Numeric matrix of PQN-normalised spectra.

Notes on total area normalisation

Total area normalisation prior to PQN is *usually not recommended*. Total area scaling removes global intensity differences by enforcing equal total signal per sample. PQN is itself a global scaling method intended to estimate dilution. Applying both can substantially change results because PQN no longer estimates dilution alone, but also compensates compositional distortions introduced by total area scaling.

Situations where total_area = TRUE can be defensible include:

- when spectra have large, non-dilution-related amplitude differences caused by acquisition artefacts (receiver gain / baseline offset) and you explicitly want to stabilise the reference estimation step;

- when the measured total signal is expected to be constant by design (e.g. strictly controlled sample mass/volume and stable overall metabolite pool), and the main goal is to reduce technical scaling variation before PQN.

In most metabolomics settings, prefer PQN without total area scaling.

#' @section On spectral alignment and binning: PQN assumes that corresponding variables represent the same chemical signal across spectra. If spectra are not well aligned, small peak shifts can inflate the variability of pointwise quotients x_{ij}/r_j, leading to unstable dilution factor estimates.

In such cases, slight binning (e.g. narrow fixed-width bins) prior to reference estimation is recommended. Binning reduces sensitivity to minor misalignments by aggregating neighbouring variables. However, excessive binning may obscure narrow signals and should be avoided.

Alternatively, prior spectral alignment is preferable when available.

References

Dieterle F, Ross A, Schlotterbeck G, Senn H (2006). Probabilistic Quotient Normalization as Robust Method to Account for Dilution of Complex Biological Mixtures. *Analytical Chemistry*, 78(13), 4281–4290.

See Also

Other preprocessing: `align_segment()`, `align_spectra()`, `binning()`, `calibrate()`, `correct_baseline()`, `correct_lw()`, `print_preprocessing()`

Examples

```
set.seed(1)
ppm <- seq(0, 10, length.out = 1000)
ref <- dnorm(ppm, 3, 0.15) + dnorm(ppm, 6, 0.20) + dnorm(ppm, 7.5, 0.18)
dil <- c(1, 0.8, 0.6, 0.4, 0.2)            # true dilution factors
X <- t(sapply(dil, function(d) d * ref + rnorm(length(ref), 0, 0.005)))
plot_spec(X, ppm)

Xn <- pqn(X, ref_index=1)
dil_est <- attr(Xn, "m8_pqn")$dilution_factor

cbind(true = dil, estimated = dil_est)
```

`prep_X` *Applies a preprocessing strategy to a numeric matrix.*

Description

Applies a preprocessing strategy to a numeric matrix.

Usage

```
prep_X(preproc_strategy, X)
```

Arguments

preproc_strategy	list with elements 'center' (logical) and 'scale' (character).
X	numeric matrix - processing is done column-wise.

Value

A list containing the processed matrix and parameters.

Examples

```
X <- matrix(c(10,10, 0,0, 0, 10), ncol=3)
autoscale <- uv_scaling(center=TRUE)
X_scaled <- prep_X(autoscale, X)
str(X_scaled)
```

---

print_preprocessing *List available preprocessing functions Returns the preprocessing utilities provided by* **metabom8**.

Description

List available preprocessing functions Returns the preprocessing utilities provided by **metabom8**.

Usage

```
print_preprocessing()
```

Value

A named character vector describing preprocessing functions.

See Also

Other preprocessing: `align_segment()`, `align_spectra()`, `binning()`, `calibrate()`, `correct_baseline()`, `correct_lw()`, `pqn()`

Examples

```
list_preprocessing()
```

`print_provenance` *Print metabom8 preprocessing pipeline*

Description

Displays the recorded preprocessing history attached to a metabom8 spectral matrix. The function reads the `"m8_prep"` attribute and prints each processing step in chronological order, including parameters and notes.

Usage

```
print_provenance(x, detail = FALSE, max_items = 8)
```

Arguments

`x`	A metabom8 object or spectral matrix with attached `"m8_prep"` provenance metadata.
`detail`	Prints full information trail, incl. timestamp / versioning
`max_items`	Limits individual list entries (e.g., parameter) to specified number

Details

The input can be either:

- A metabom8-style list containing `$X`, or
- A matrix with metabom8 provenance attributes attached.

If no preprocessing metadata is found, a message is printed.

metabom8 records preprocessing steps as an ordered list of transformation descriptors stored in the `"m8_prep"` attribute. Each step typically contains:

- `step`: Name of the preprocessing operation
- `params`: Parameters used
- `notes`: Optional description
- `time`: Timestamp
- `pkg`: Package name and version

This function provides a compact audit trail of the processing workflow, facilitating reproducibility and provenance inspection.

Value

Invisibly returns `NULL`. This function is called for its side effect of printing pipeline information.

See Also

Other provenance: `add_note()`, `get_provenance()`

Examples

```
params <- list(
  runtime  = "docker",
  image    = Sys.getenv("IMAGE", "docker-image-dummy"),
  workflow = Sys.getenv("M8_WORKFLOW", "std_prof-urine"),
  agent    = paste0("snakemake/", Sys.getenv("SNAKEMAKE_VERSION", "v?")),
  run_id   = Sys.getenv("M8_RUN_ID", "m8-2605-001")
)

data(hiit_raw)
print_provenance(hiit_raw)

hiit_proc <- hiit_raw |>
  calibrate(type = "tsp") |>
  excise() |>
  add_note('dilution-adaptive acquisition mode -> verify snr after normalisation',
    params)

print_provenance(hiit_proc)
```

`read1d` *Import 1D NMR spectra (TopSpin processed)*

Description

Imports TopSpin-processed 1D NMR spectra together with spectrometer acquisition and TopSpin processing parameters (`acqus` and `procs`, respectively).

Usage

```
read1d(
  path,
  exp_type = list(pulprog = "noesygppr1d"),
  n_max = 1000,
  filter = TRUE,
  recursive = TRUE,
  verbose = 1,
  to_global = FALSE
)

read1d_proc(
  path,
  exp_type = list(pulprog = "noesygppr1d"),
  n_max = 1000,
  filter = TRUE,
  recursive = TRUE,
  verbose = 1,
  to_global = FALSE
)
```

Arguments

path	Character. Directory path containing Bruker NMR experiments.
exp_type	Named list. Optional filtering specification based on acquisition or processing metadata. Each list element must correspond to a metadata field (e.g. pulprog, ns, rg). Filtering supports: • Exact match: list(pulprog = "noesygppr1d") • Membership: list(pulprog = c("zg30", "noesygppr1d")) • Numeric range: list(ns = list(range = c(16, 128))) • Generic comparison: list(ns = list(op = ">=", value = 32)) Multiple fields are combined using logical AND.
n_max	Integer. Maximum number of spectra to import. Default: 1000.
filter	Logical. Filter out experiments with incomplete file systems.
recursive	Logical. Search path recursively. Default: TRUE.
verbose	Logical or numeric. Verbosity level.
to_global	Logical. If TRUE, the returned objects are additionally assigned to the global environment.

Value

A named list with three elements:

X A numeric matrix of spectra (rows = samples, columns = ppm values).

ppm A numeric vector of chemical shift values (ppm).

meta A data frame of acquisition and processing metadata, row-aligned with X.

If to_global = TRUE, objects with the same names in the global environment will be overwritten.

Examples

```
path <- system.file("extdata", package = "metabom8")

read1d_proc(path, exp_type = list(pulprog = "noesygppr1d"), n_max = 2)
```

---

read1d_raw *Read raw FIDs and process to spectra*

Description

Reads Bruker 1D NMR FIDs, corrects the digital filter (group delay), applies apodisation (windowing), optional zero-filling, FFT, phasing and ppm calibration.

Returns either absorption-, dispersion-, or magnitude-mode spectra.

If to_global = TRUE, objects are assigned to the global environment.

Usage

```
read1d_raw(
  path,
  exp_type = list(exp = "PROF_PLASMA_CPMG128_3mm", pulprog = "noesygppr1d"),
  apodisation = list(fun = "exponential", lb = 0.2),
  zerofill = 1L,
  mode = c("absorption", "dispersion", "magnitude"),
  verbose = 1,
  recursive = TRUE,
  n_max = 1000,
  filter = TRUE,
  to_global = FALSE
)
```

Arguments

path	Character. Path to the root directory containing Bruker experiment folders.
exp_type	Named list. Acquisition-parameter filter to select experiments (e.g., list(PULPROG = "noesygppr1d")).
apodisation	Named list. Apodisation function and parameters. fun must be one of "exponential", "cosine", "sine", "sem".
zerofill	Integer. Zero-filling exponent (1 doubles points, 2 quadruples, ...).
mode	Character. Spectrum type to return: "absorption", "dispersion", or "magnitude".
verbose	Integer. Verbosity level: 0 = silent, 1 = summary (default), 2 = detailed, 3 = debug.
recursive	Logical. Recursively search subdirectories for FIDs.
n_max	Integer. Maximum number of experiments to process.
filter	Logical. Remove experiments with incomplete file structures.
to_global	Logical. If TRUE, objects are also assigned to the global environment; otherwise, only an invisible list is returned.

Details

FIDs are read from Bruker acquisition folders and processed by the following pipeline:

1. Digital-filter (group-delay) correction: the initial n complex points are invalid due to the causal DSP decimation filter and are discarded; n equals GRPDLY when present, or is looked up from Bruker tables indexed by DECIM and DSPFVS on older systems.
2. Apodisation (windowing).
3. Zero-filling (optional).
4. FFT to the frequency domain.
5. Phase correction.
6. PPM calibration (e.g., to TSP).

A common ppm scale is then interpolated across spectra.

Digital filter note: On newer systems GRPDLY is written in acqus/acqu2s and should be used directly. For older data sets (GRPDLY < 0 or missing), the group delay is derived from DECIM and DSPFVS via an internal look-up table.

Apodisation functions:

- "uniform"
- "cosine",
- "sine",
- "exponential" (parameter: lb)
- "sem" (sine * exponential, parameter: lb)
- "gauss" (parameter: lb, 'gb', and 'para')
- "expGaus_resyG" (parameter: lb, 'gb', and 'aq_t')

Value

A named list with three elements:

X A numeric matrix of spectra (rows = samples, columns = ppm values).

ppm A numeric vector of chemical-shift axis (ppm).

meta A data frame of acquisition metadata, row-aligned with X.

If to_global = TRUE, these objects are also assigned to the global environment. In that case, any existing objects with the same names will be overwritten.

See Also

read1d_proc for importing TopSpin-processed spectra

Examples

```
path <- system.file("extdata", package = "metabom8")
read1d_raw(
  path,
  exp_type    = list(PULPROG = "noesygppr1d"),
  apodisation = list(fun = "exponential", lb = 0.2),
  zerofill    = 1,
  n_max       = 3
)
```

---

scores *PLS/OPLS model scores*

Description

PLS/OPLS model scores

Usage

```
scores(object, ...)

## S4 method for signature 'm8_model'
scores(object, orth = FALSE, cv = FALSE)
```

Arguments

object	An object of class m8_model.
...	Additional arguments (currently ignored).
orth	Logical indicating whether orthogonal scores should be returned (only applicable for OPLS models).
cv	Logical indicating whether cross-validated scores should be returned

Value

Numeric vector or matrix containing scores.

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
show(model)
scores(model, orth=FALSE)
scores(model, orth=TRUE)
scores(model, cv=TRUE)
```

---

scRange *Min-Max Scaling to Arbitrary Range*

Description

Rescales a numeric vector to a specified range using min-max scaling. This is a generalized form of min-max normalization allowing any output range.

Usage

```
scRange(x, ra)
```

Arguments

x	Numeric vector. Input values to be scaled.
ra	Numeric vector of length 2. Desired output range (e.g., c(5, 10)).

Details

The scaled values are computed as:

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)} \cdot (r_{max} - r_{min}) + r_{min}$$

Value

A numeric vector of the same length as x, scaled to the range ra.

See Also

minmax()

Examples

```
x <- rnorm(20)
plot(x, type = 'l'); abline(h = range(x), lty = 2)
points(scRange(x, ra = c(5, 10)), type = 'l', col = 'red');
abline(h = c(5, 10), col = 'red', lty = 2)
```

---

stocsy *Statistical Total Correlation Spectroscopy (STOCSY)*

Description

Performs STOCSY analysis on 1D NMR spectra. Computes correlation and covariance of all variables in X against a driver signal (internal ppm position or an external numeric vector).

Usage

```
stocsy(X, ppm = NULL, driver, plotting = TRUE, title = NULL)
```

Arguments

X	Numeric matrix or data frame. Rows are spectra, columns are variables.
ppm	Optional numeric vector of chemical shift values (length = ncol(X)). If missing or NULL, will try to read from colnames(X).
driver	Numeric scalar (ppm value, internal driver) or numeric vector (external driver, length = nrow(X)).
plotting	Logical. If TRUE, plot the STOCSY spectrum.
title	Optional character plot title.

Value

An object of class m8_stocsy1d (S3), a named list with entries: version, X, ppm, driver, r, cov, extD.

See Also

Other structural_annotation: storm()

Examples

```
# st <- stocsy(X, ppm, driver = 5.233, plotting = FALSE)
# plotStocsy(st, shift = c(5.15, 5.30))

data(covid)
cs = 5.233
stocsy(covid$X, driver=cs, plotting = TRUE)
```

storm *Subset Optimisation by Reference Matching (STORM)*

Description

Selects an optimal subset of spectra that best match a specified target signal region, improving downstream correlation-based structural analysis such as STOCSY.

Usage

```
storm(X, ppm, b = 30, q = 0.05, idx.refSpec, shift)
```

Arguments

X	Numeric matrix (or data.frame) of NMR spectra with samples in rows and spectral variables in columns.
ppm	Numeric vector of chemical shift values corresponding to the columns of X.
b	Integer. Half-window size expressed as number of spectral variables (data points). The effective window width therefore depends on the ppm spacing.
q	Numeric. P-value threshold for including spectral variables when updating the reference region.
idx.refSpec	Integer. Row index of X defining the initial reference spectrum.
shift	Numeric vector of length 2 giving the ppm range (minimum, maximum) defining the initial target signal region.

Details

STORM iteratively refines a subset of spectra exhibiting consistent signal position and multiplicity within a specified ppm region.

Starting from an initial reference spectrum and ppm window:

1. Spectra positively correlated with the current reference signal are retained.
2. A driver peak (maximum intensity within the reference window) is identified.
3. Correlation and covariance are evaluated within a local window of size b around the driver peak.
4. The reference region is updated using variables that satisfy the p-value threshold (q) and show positive correlation.

The procedure continues until the selected subset stabilises. The resulting row indices define spectra that most consistently represent the structural pattern of the target signal.

STORM does not perform metabolite identification directly. Instead, it refines the dataset to enhance structural coherence prior to correlation-based interpretation methods such as STOCSY.

Value

Integer vector of row indices of X defining the selected spectral subset.

References

Posma, J. M., et al. (2012). Subset optimisation by reference matching (STORM): an optimised statistical approach for recovery of metabolic biomarker structural information from ^1H NMR spectra of biofluids. *Analytical Chemistry*, 84(24), 10694–10701.

See Also

Other structural_annotation: `stocsy()`

Examples

```
## Simulated example with three Gaussian signals
set.seed(123)

n <- 100          # number of spectra
S <- 1000         # number of spectral variables
ppm <- seq(10, 0, length.out = S)

gauss <- function(x, centre, width, height) {
  height * exp(-((x - centre)^2) / (2 * width^2))
}

## Generate base signals
sig1 <- gauss(ppm, 7,   0.05, 10)
sig2 <- gauss(ppm, 3.5, 0.10,  8)
sig3 <- gauss(ppm, 1,   0.07,  5)

spectra <- matrix(0, n, S)

for (i in seq_len(n)) {
  spectra[i, ] <- sig1 + sig2 + rnorm(S, 0, 0.1)
  if (i <= 25) {
    spectra[i, ] <- spectra[i, ] + sig3
  }
}

## Apply STORM to refine spectra containing the 1 ppm signal
idx <- storm(
  X = spectra,
  ppm = ppm,
  b = 30,
  q = 0.05,
  idx.refSpec = 1,
  shift = c(0.75, 1.25)
)

length(idx)   # number of spectra retained
```

---

stratified_kfold *Y-stratified k-fold cross-validation strategy*

Description

Y-stratified k-fold cross-validation strategy

Usage

```
stratified_kfold(k, type = c("DA", "R"), probs = NULL)
```

Arguments

k	Integer. Number of folds.
type	Character. Either "DA" (classification) or "R" (regression).
probs	Numeric vector of quantile probabilities used to stratify continuous Y when type = "R".

Details

For classification (type = "DA"), folds are generated such that class proportions are approximately preserved in each fold.

For regression (type = "R"), Y is discretised into bins defined by probs, and folds are generated to approximately preserve the bin distribution.

Value

A named list with elements:

train List of integer vectors containing training set indices for each resampling iteration.

strategy Character string indicating the resampling strategy.

n Integer. Number of samples in the dataset.

seed Integer. Random seed used to generate the resampling splits, ensuring reproducibility.

See Also

Other resampling strategies: balanced_boot(), balanced_mc(), kfold(), mc()

Examples

```
set.seed(1)
n <- 100
thr <- 1.5
Y <- c(rnorm(80, thr - 3, 0.3), rnorm(20, thr + 3, 0.3))  # unbalanced outcome
mean(Y > thr)

q80 <- quantile(Y, 0.8)  # defines the rare "high" stratum (top 20%)

cv_k  <- kfold(k = 10)
cv_sk  <- stratified_kfold(k = 10, type = "R", probs = c(0, 0.8, 1))

k_inst <- metabom8:::.arg_check_cv(cv_pars=cv_k, model_type='R', n=n, Y_prepped=cbind(Y))
sk_inst <- metabom8:::.arg_check_cv(cv_pars=cv_sk, model_type='R', n=n, Y_prepped=cbind(Y))

round(sapply(k_inst$train,  function(i) mean(Y[i] > q80)), 2)  # reflects imbalance
round(sapply(sk_inst$train, function(i) mean(Y[i] > q80)), 2)  # balanced across strata
```

unscaled *No Scaling This function defines a preprocessing strategy that is applied via* prep_X.

Description

No Scaling This function defines a preprocessing strategy that is applied via prep_X.

Usage

```
unscaled(center = FALSE)
```

Arguments

center	Logical. If TRUE, variables are mean-centered before scaling.

Value

A list with elements:

X Numeric matrix containing the scaled data.

prep List describing the preprocessing, including centering and scaling parameters (center, scale, X_mean, X_sd).

#' @details Leaves variables unscaled. Optional centering.

See Also

Other scaling_strategies: pareto_scaling(), uv_scaling()

Examples

```
just_centering <- unscaled(center=TRUE)
X <- matrix(c(10,10, 0,0, 0, 10), ncol=3)
X_centered <- prep_X(just_centering, X)
str(X_centered)
X_centered$X
```

uv_scaling *Unit Variance Scaling This function defines a preprocessing strategy that is applied via* prep_X.

Description

Unit Variance Scaling This function defines a preprocessing strategy that is applied via prep_X.

Usage

```
uv_scaling(center = TRUE)
```

Arguments

center	Logical. If TRUE, variables are mean-centered before scaling.

Details

Centers variables and scales each feature to unit variance. UV scaling divides each variable by its standard deviation. This is the default scaling in many multivariate methods such as PCA and PLS.

Value

A list with elements:

X Numeric matrix containing the scaled data.

prep List describing the preprocessing, including centering and scaling parameters (center, scale, X_mean, X_sd).

See Also

Other scaling_strategies: pareto_scaling(), unscaled()

Examples

```
autoscale <- uv_scaling(center=TRUE)
X <- matrix(c(10,10, 0,0, 0, 10), ncol=3)
X_scaled <- prep_X(autoscale, X)
str(X_scaled)
X_scaled$X
```

---

vip *Variable Importance in Projection (VIP)*

Description

Variable Importance in Projection (VIP)

Usage

```
vip(object)

## S4 method for signature 'm8_model'
vip(object)
```

Arguments

object	An object of class m8_model.

Value

Numeric vector or matrix containing the variable importance in projection (VIP) scores.

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
show(model)
vip_scores <- vip(model)
dim(vip_scores)
```

---

weights *Extract model weights*

Description

Extract model weights

Usage

```
weights(object, ...)

## S4 method for signature 'm8_model'
weights(object, orth = FALSE)
```

Arguments

object	An object of class m8_model.
...	Additional arguments (currently ignored).
orth	Logical indicating whether orthogonal scores should be returned (only applicable for OPLS models).

Value

Numeric vector or matrix containing model weights.

Examples

```
data(covid)
cv <- balanced_mc(k=5, split=2/3)
scaling <- uv_scaling(center=TRUE)
model <-opls(X=covid$X, Y=covid$an$type, scaling, cv)
show(model)
W <- weights(model)
Wo <- weights(model, orth = TRUE)
dim(W)
dim(Wo) == dim(W)
```

`xres` *Compute X residual matrix Returns the residual matrix (E) of an OPLS model.*

Description

Compute X residual matrix Returns the residual matrix (E) of an OPLS model.

Usage

```
xres(object)

## S4 method for signature 'm8_model'
xres(object)
```

Arguments

`object`	An object.

Value

Numeric matrix containing the X residuals.

Examples

```
data(covid)
cv <- balanced_mc(k = 5, split = 2/3)
scaling <- uv_scaling(center = TRUE)
model <- opls(X = covid$X, Y = covid$an$type, scaling, cv)
X_res <- xres(model)
dim(X_res) == dim(covid$X)
```

Index