Package: MetaboCoreUtils (via r-universe)

October 14, 2026

Title Core Utils for Metabolomics Data

Version 1.20.1

Description MetaboCoreUtils defines metabolomics-related core functionality provided as low-level functions to allow a data structure-independent usage across various R packages. This includes functions to calculate between ion (adduct) and compound mass-to-charge ratios and masses or functions to work with chemical formulas. The package provides also a set of adduct definitions and information on some commercially available internal standard mixes commonly used in MS experiments.

Depends R (>= 4.0)

Imports utils, MsCoreUtils, BiocParallel, methods, stats

Suggests BiocStyle, testthat, knitr, rmarkdown, robustbase

License Artistic-2.0

Encoding UTF-8

LazyData no

VignetteBuilder knitr

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

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

biocViews Infrastructure, Metabolomics, MassSpectrometry

Roxygen list(markdown=TRUE)

RoxygenNote 7.3.3

Repository Bioconductor 3.23

Date/Publication 2026-04-30 06:18:41 UTC

RemoteUrl https://github.com/bioc/MetaboCoreUtils

RemoteRef RELEASE_3_23

RemoteSha ae5c09690bd839c005d762c5e868a040cbb75dea

Contents

addElements . . . 2
adductFormula . . . 3
adductNames . . . 4
betaValues . . . 5
calculateKendrickMass . . . 7
calculateMass . . . 8
containsElements . . . 9
convertMtime . . . 10
correctRindex . . . 11
countElements . . . 12
fit_lm . . . 12
formula2mz . . . 16
guessSource . . . 17
indexRtime . . . 18
internalStandardMixNames . . . 19
internalStandards . . . 19
isotopicSubstitutionMatrix . . . 20
isotopologues . . . 22
mass2mz . . . 24
mclosest . . . 25
multiplyElements . . . 26
mz2mass . . . 27
nameMapping . . . 28
pasteElements . . . 29
quality_assessment . . . 29
softwareMapping . . . 31
softwareMappingSchema . . . 32
standardizeFormula . . . 33
subtractElements . . . 34
translate . . . 34

Index **36**

`addElements` *Combine chemical formulae*

Description

`addElements` Add one chemical formula to another.

Usage

```
addElements(x, y)
```

Arguments

x	character strings with chemical formula
y	character strings with chemical formula that should be added from x

Value

character Resulting formula

Author(s)

Michael Witting and Sebastian Gibb

Examples

```
addElements("C6H12O6", "Na")

addElements("C6H12O6", c("Na", "H2O"))
```

---

adductFormula *Calculate a table of adduct (ionic) formulas*

Description

adductFormula calculates the chemical formulas for the specified adducts of provided chemical formulas.

Usage

```
adductFormula(formulas, adduct = "[M+H]+", standardize = TRUE)
```

Arguments

formulas	character with molecular formulas for which adduct formulas should be calculated.
adduct	character or data.frame of valid adduct. to be used. Custom adduct definitions can be provided via a data.frame but its format must follow adducts()
standardize	logical(1) whether to standardize the molecular formulas to the Hill notation system before calculating their mass.

Value

character matrix with *formula* rows and *adducts* columns containing all ion formulas. In case an ion can't be generated (eg. [M-NH3+H]+ in a molecule that doesn't have nitrogen), a NA is returned instead.

Author(s)

Roger Gine

See Also

adductNames() for a list of all available predefined adducts and adducts() for the adduct data.frame definition style.

Examples

```
# Calculate the ion formulas of glucose with adducts [M+H]+, [M+Na]+ and [M+K]+
adductFormula("C6H12O6", c("[M+H]+", "[M+Na]+", "[M+K]+"))

# > "[C6H13O6]+" "[C6H12O6Na]+" "[C6H12O6K]+"

# Use a custom set of adduct definitions (For instance, a iron (Fe2+) adduct)
custom_ads <- data.frame(name = "[M+Fe]2+", mass_multi = 0.5, charge = 2,
                         formula_add = "Fe", formula_sub = "C0",
                         positive = "TRUE")
adductFormula("C6H12O6", custom_ads)
```

---

adductNames *Retrieve names of supported adducts*

Description

adductNames returns all supported adduct definitions that can be used by mass2mz() and mz2mass().

adducts returns a data.frame with the adduct definitions.

Usage

```
adductNames(polarity = c("positive", "negative"))

adducts(polarity = c("positive", "negative"))
```

Arguments

polarity	character(1) defining the ion mode, either "positive" or "negative".

Value

for adductNames: character vector with all valid adduct names for the selected ion mode. For adducts: data.frame with the adduct definitions.

Author(s)

Michael Witting, Johannes Rainer

Examples

```
## retrieve names of adduct names in positive ion mode
adductNames(polarity = "positive")

## retrieve names of adduct names in negative ion mode
adductNames(polarity = "negative")
```

`betaValues` *Peak shape quality: beta parameters*

Description

Calculate *beta* parameters for a chromatographic peak, both its similarity to a bell curve of varying degrees of skew (i.e., assymmetry or *tailing factor*) and the standard deviation of the residuals after the best-fit bell is normalized and subtracted. This function requires at least 5 scans or it will return `NA` for both parameters.

Usage

```
betaValues(
  intensity = numeric(),
  rtime = seq_along(intensity),
  skews = c(3, 3.5, 4, 4.5, 5),
  zero.rm = TRUE
)
```

Arguments

`intensity`	A `numeric` vector corresponding to the peak intensities with a minimum length of 5.
`rtime`	A numeric vector corresponding to the retention times of each intensity. Retention times are expected to be in increasing order without duplicates. If not provided, intensities will be assumed to be equally spaced.
`skews`	A `numeric` vector of the skews to try. These values will be passed to `shape1` parameter of the `dbeta()` function (with its `shape2` parameter set to `5`). Values less than 5 will generated increasingly right-skewed curves, while values greater than 5 will result in left-skewed curves. See Figure 8 in the original manuscript for visual examples.
`zero.rm`	`logical(1)` controlling whether *missing* scans are dropped prior to curve fitting. The default, `zero.rm = TRUE`, will remove intensities of zero or `NA`.

Details

The function compares the actual chromatographic peak to *Beta* distribution curves calculated with the `dbeta()` function.

Value

`numeric` of length 2.

Author(s)

William Kumler

References

Kumler W, Hazelton B J and Ingalls A E (2023) "Picky with peakpicking: assessing chromatographic peak quality with simple metrics in metabolomics" *BMC Bioinformatics* 24(1):404. doi: 10.1186/s12859-023-05533-4

See Also

`dbeta()` for the function to calculate the Beta distribution.

Examples

```
## Define intensity values of a typical chromatographic peak
skinny_peak <- c(9107, 3326, 9523, 3245, 3429, 9394, 1123, 935, 5128, 8576,
                 2711, 3427, 7294, 8109, 9288, 6997, 9756, 8034, 1317, 8866,
                 13877, 14854, 28296, 57101, 92209, 151797, 222386, 299402,
                 365045, 394255, 402680, 363996, 293985, 222989, 147007,
                 94947, 52924, 32438, 11511, 10836, 8046, 601, 889, 5917,
                 2690, 5381, 9901, 8494, 3349, 8283, 3410, 5935, 3332,
                 7041, 3284, 7478, 76, 3739, 2158, 5507)
skinny_peak_rt <- seq_along(skinny_peak)+100

plot(skinny_peak_rt, skinny_peak, xlab = "RT",
    ylab = "intensity", type = "l")

## Calculate beta parameters for the full region including noise signal
## around the peak
res <- betaValues(skinny_peak, skinny_peak_rt)
res

## Calculate beta parameters for peak signal
res <- betaValues(skinny_peak[20:40], skinny_peak_rt[20:40])
res

## Define noisy chromatographic signal
noise_peak <- c(0.288, 0.788, 0.409, 0.883, 0.94, 0.046, 0.528, 0.892,
                0.551, 0.457, 0.957, 0.453, 0.678, 0.573, 0.103, 0.9,
                0.246, 0.042, 0.328, 0.955, 0.89, 0.693, 0.641, 0.994,
                0.656, 0.709, 0.544, 0.594, 0.289, 0.147, 0.963, 0.902,
                0.691, 0.795, 0.025, 0.478, 0.758, 0.216, 0.318, 0.232,
                0.143, 0.415, 0.414, 0.369, 0.152, 0.139, 0.233, 0.466,
                0.266, 0.858, 0.046, 0.442, 0.799, 0.122, 0.561, 0.207,
                0.128, 0.753, 0.895, 0.374, 0.665, 0.095, 0.384, 0.274,
                0.815, 0.449, 0.81, 0.812, 0.794, 0.44, 0.754, 0.629, 0.71,
                0.001, 0.475, 0.22, 0.38, 0.613, 0.352, 0.111, 0.244, 0.668,
```

```
                0.418, 0.788, 0.103, 0.435, 0.985, 0.893, 0.886, 0.175,
                0.131, 0.653, 0.344, 0.657, 0.32, 0.188, 0.782, 0.094,
                0.467, 0.512)
noise_peak_rt <- seq_along(noise_peak) + 10
res <- betaValues(noise_peak, noise_peak_rt)
res
```

`calculateKendrickMass` *Kendrick mass defects*

Description

Kendrick mass defect analysis is a way to analyze high-resolution MS data in order to identify homologous series. The Kendrick mass (KM) is calculated by choosing a specific molecular fragment (e.g. CH2) and settings its mass to an integer mass. In case of CH2 the mass of 14.01565 would be set to 14.The Kendrick mass defect (KMD) is defined as the difference between the KM and the nominial (integer) KM. All molecules of homologoues series, e.g. only differing in the number of CH2, will have an identical KMD. In an additional step the KMD can be referenced to the mass defect of specific lipid backbone and by this normalize values to the referenced KMD (RKMD). This leads to values of 0 for saturated species or -1, -2, -3, etc for unsaturated species.

Available functoins are:

- `calculateKm`: calculates the Kendrick mass from an exact mass for a specific molecular fragment, e.g. `"CH2"`.
- `calculateKmd`: calculates the Kendrick mass defect from an exact mass for a specific molecular fragment, e.g. `"CH2"`.
- `calculateRkmd`: calculates the referenced Kendrick mass defect from an exact mass for a specific molecular fragment, e.g. `"CH2"`, and a reference KMD.
- `isRkmd`: Checks if a calculated RKMD falls within a specific error range around an negative integer corresponding the number of double bonds, in case of CH2 as fragment.

Usage

```
calculateKm(x, fragment = 14/14.01565)

calculateKmd(x, fragment = 14/14.01565)

calculateRkmd(x, fragment = 14/14.01565, rkmd = 0.749206)

isRkmd(x, rkmdTolerance = 0.1)
```

Arguments

`x`	`numeric` with exact masses or calculated RKMDs in case of `isRkmd`.
`fragment`	`numeric(1)` or `character(1)` corresponding factor or molecular formula of molecular fragment, e.g. `14 / 14.01565` or `"CH2"` for CH2.

rkmd	numeric(1) KMD used for referencing of KMDs.
rkmdTolerance	numeric(1) Tolerance to check if RKMD fall around a negative integer corresponding to the number of double bonds

Value

numeric or boolean. All functions, except isRkmd return a numeric with same length as the input corresponding to the KM, KMD or RMKD. isRkmd returns a logical with TRUE or FALSE indicating if the RKMD falls within a specific range around a negative integer corresponding to the number of double bonds.

Author(s)

Michael Witting

Examples

```
calculateKm(760.5851)

calculateKmd(760.5851)

calculateRkmd(760.5851, rkmd = 0.749206)

isRkmd(calculateRkmd(760.5851, rkmd = 0.749206))
```

---

calculateMass *Calculate exact mass*

Description

calculateMass calculates the exact mass from a formula. Isotopes are also supported. For isotopes, the isotope type needs to be specified as an element's prefix, e.g. "[13C]" for carbon 13 or "[2H]" for deuterium. A formula with 2 carbon 13 isotopes and 3 carbons would thus contain e.g. "[13C2]C3".

Usage

```
calculateMass(x)
```

Arguments

x	character representing chemical formula(s) or a list of numeric with element counts such as returned by countElements(). Isotopes and deuterated elements are supported (see examples below).

Value

numeric Resulting exact mass.

Author(s)

Michael Witting

Examples

```
calculateMass("C6H12O6")
calculateMass("NH3")
calculateMass(c("C6H12O6", "NH3"))

## Calculate masses for formulas containing isotope information.
calculateMass(c("C6H12O6", "[13C3]C3H12O6"))

## Calculate mass for a chemical with 5 deuterium.
calculateMass("C11[2H5]H7N2O2")
```

---

containsElements *Check if one formula is contained in another*

Description

`containsElements` checks if one sum formula is contained in another.

Usage

```
containsElements(x, y)
```

Arguments

`x`	`character` strings with a chemical formula
`y`	`character` strings with a chemical formula that shall be contained in `x`

Value

`logical` TRUE if `y` is contained in `x`

Author(s)

Michael Witting and Sebastian Gibb

Examples

```
containsElements("C6H12O6", "H2O")
containsElements("C6H12O6", "NH3")
```

convertMtime *Convert migration times to effective mobility*

Description

convertMtime performs effective mobility scale transformation of CE(-MS) data, which is used to overcome variations of the migration times, caused by differences in the Electroosmotic Flow (EOF) between different runs. In order to monitor the EOF and perform the transformation, neutral or charged EOF markers are spiked into the sample before analysis. The information of the EOF markers (migration time and effective mobility) will be then used to perform the effective mobility transformation of the migration time scale.

Usage

```
convertMtime(
  x = numeric(),
  rtime = numeric(),
  mobility = numeric(),
  tR = 0,
  U = numeric(),
  L = numeric()
)
```

Arguments

x	numeric vector with migration times in minutes.
rtime	numeric vector that holds the migration times (in minutes) of either one or two EOF markers in the same run of which the migration time is going to be transformed.
mobility	numeric vector containing the respective effective mobility (in in mm^2 / (kV * min)) of the EOF markers. If two markers are used, one is expected to be the neutral marker, i.e. having a mobility of 0.
tR	numeric a single value that defines the time (in minutes) of the electrical field ramp. The default is 0.
U	numeric a single value that defines the voltage (in kV) applied. Note that for reversed polarity CE mode a negative value is needed. Is only used if the transformation is performed based on a single marker.
L	numeric a single value that defines the total length (in mm) of the capillary that was used for CE(-MS) analysis. Is only used if the transformation is performed based on a single marker.

Value

numeric vector of same length as x with effective mobility values.

Author(s)

Liesa Salzer

Examples

```
rtime <- c(10,20,30,40,50,60,70,80,90,100)
marker_rt <-  c(20,80)
mobility <- c(0, 2000)
convertMtime(rtime, marker_rt, mobility)
```

---

correctRindex *2-point correction of RIs*

Description

`correctRindex` performs correction of retention indices (RIs) based on reference substances. Even after conversion of RTs to RIs slight deviations might exist. These deviations can be further normalized, if they are linear, by using two metabolites for which the RIs are known (e.g. internal standards).

Usage

```
correctRindex(x, y)
```

Arguments

`x`	`numeric` vector with retention indices, calculated by `indexRtime`
`y`	`data.frame` containing two columns. The first is expected to contain the measured RIs of the reference substances and the second the reference RIs.

Value

`numeric` vector of same length than `x` with corrected retention indices. Values are floating point decimals. If integer values shall be used conversion has to be performed manually.

Author(s)

Michael Witting

Examples

```
ref <- data.frame(rindex = c(110, 210),
refindex = c(100, 200))
rindex <- c(110, 210)
correctRindex(rindex, ref)
```

countElements *Count elements in a chemical formula*

Description

countElements parses strings representing a chemical formula into a named vector of element counts.

Usage

```
countElements(x)
```

Arguments

x	character() representing a chemical formula.

Value

list of integer with the element counts (names being elements).

Author(s)

Michael Witting and Sebastian Gibb

See Also

pasteElements()

Examples

```
countElements(c("C6H12O6", "C11H12N2O2"))
```

fit_lm *Linear model-based normalization of abundance matrices*

Description

The fit_lm and adjust_lm functions facilitate linear model-based normalization of abundance matrices. The expected noise in a numeric data matrix can be modeled with a linear regression model using the fit_lm function and the data can subsequently be adjusted using the adjust_lm function (i.e., the modeled noise will be removed from the abundance values). A typical use case would be to remove injection index dependent signal drifts in a LC-MS derived metabolomics data: a linear model of the form y ~ injection_index could be used to model the measured abundances of each feature (each row in a data matrix) as a function of the injection index in which a specific sample was measured during the LC-MS measurement run. The fitted linear regression models can

subsequently be used to adjust the abundance matrix by removing these dependencies from the data. This allows to perform signal adjustments as described in (Wehrens et al. 2016).

The two functions are described in more details below:

`fit_lm` allows to fit a linear regression model (defined with parameter `formula`) to each row of the numeric data matrix submitted with parameter `y`. Additional covariates of the linear model defined in `formula` are expected to be provided as columns in a `data.frame` supplied *via* the `data` parameter.

The linear model is expected to be defined by a formula starting with `y ~ .`. To model for example an injection index dependency of values in `y` a formula `y ~ injection_index` could be used, with values for the injection index being provided as a column `"injection_index"` in the `data` data frame. `fit_lm` would thus fit this model to each row of `y`.

Linear models can be fitted either with the standard least squares of `lm()` by setting `method = "lm"` (the default), or with the more robust methods from the *robustbase* package with `method = "lmrob"`.

`adjust_lm` can be used to adjust abundances in a data matrix `y` based on linear regression models provided with parameter `lm`. Parameter `lm` is expected to be a `list` of length equal to the number of rows of `y`, each element being a linear model (i.e., a results from `lm` or `lmrob`). Covariates for the model need to be provided as columns in a `data.frame` provided with parameter `data`. The number of rows of that `data.frame` need to match the number of columns of `y`. The function returns the input matrix `y` with values in rows adjusted with the linear models provided by `lm`. No adjustment is performed for rows for which the respective element in `lm` is `NA`. See examples below for details or the vignette for more examples, descriptions and information.

Usage

```
fit_lm(
  formula,
  data,
  y,
  method = c("lm", "lmrob"),
  control = NULL,
  minVals = ceiling(nrow(data) * 0.75),
  model = TRUE,
  ...,
  BPPARAM = SerialParam()
)

adjust_lm(y = matrix(), data = data.frame(), lm = list(), ...)
```

Arguments

`formula`	`formula` defining the model that should be fitted to the data. See also `lm()` for more information. Formulas should begin with `y ~` as values in rows of `y` will be defined as *y*. See description of the `fit_lm` function for more information.
`data`	`data.frame` containing the covariates for the linear model defined by `formula` (for `fit_lm`) or used in `lm` (for `adjust_lm`). The number of rows has to match the number of columns of `y`.

y	for fit_lm: matrix of abundances on which the linear model pefined with formula should be fitted. For adjust_lm: matrix of abundances that should be adjusted using the models provided with parameter lm.
method	character(1) defining the linear regression function that should be used for model fitting. Can be either method = "lm" (the default) for standard least squares model fitting or method = "lmrob" for a robust alternative defined in the *robustbase* package.
control	a list speficying control parameters for lmrob. Only used if method = "lmrob". See help of lmrob.control in the robustbase package for details. By default control = NULL the *KS2014* settings are used and scale-finding iterations are increased to 10000.
minVals	numeric(1) defining the minimum number of non-missing values (per feature/row) required to perform the model fitting. For rows in y for which fewer non-NA values are available no model will be fitted and a NA will be reported instead.
model	logical(1) whether the model frame are included in the returned linear models. Passed to the lm or lmrob functions.
...	for fit_lm: additional parameters to be passed to the downstream calls to lm or lmrob. For adjust_lm: ignored.
BPPARAM	parallel processing setup. See BiocParallel::bpparam() for more information. Parallel processing can improve performance especially for method = "lmrob".
lm	list of linear models (as returned by lm or lmrob) such as returned by the fit_lm function. The length of the list is expected to match the number of rows of y, i.e., each element should be a linear model to adjust the specific row, or NA to skip adjustment for that particular row in y.

Value

```
For `fit_lm`: a `list` with linear models (either of type *lm* or
*lmrob*) or length equal to the number of rows of `y`. `NA` is
reported for rows with too few non-missing data points (depending
on parameter `minValues`).
For `adjust_lm`: a numeric matrix (same dimensions as input matrix
`y`) with the values adjusted with the provided linear models.
```

Author(s)

Johannes Rainer

References

Wehrens R, Hageman JA, van Eeuwijk F, Kooke R, Flood PJ, Wijnker E, Keurentjes JJ, Lommen A, van Eekelen HD, Hall RD Mumm R and de Vos RC. Improved batch correction in untargeted MS-based metabolomics. *Metabolomics* 2016; 12:88.

Examples

```
## See also the vignette for more details and examples.

## Load a test matrix with abundances of features from a LC-MS experiment.
vals <- read.table(system.file("txt", "feature_values.txt",
                               package = "MetaboCoreUtils"), sep = "\t")
vals <- as.matrix(vals)

## Define a data.frame with the covariates to be used to model the noise
sdata <- data.frame(injection_index = seq_len(ncol(vals)))

## Fit a linear model describing the feature abundances as a
## function of the index in which samples were injected during the LC-MS
## run. We're fitting the model to log2 transformed data.
## Note that such a model should **only** be fitted if the samples
## were randomized, i.e. the injection index is independent of any
## experimental covariate. Alternatively, the injection order dependent
## signal drift could be estimated using QC samples (if they were
## repeatedly injected) - see vignette for more details.
ii_lm <- fit_lm(y ~ injection_index, data = sdata, y = log2(vals))

## The result is a list of linear models
ii_lm[[1]]

## Plotting the data for one feature:
plot(x = sdata$injection_index, y = log2(vals[2, ]),
    ylab = expression(log[2]~abundance), xlab = "injection index")
grid()
## plot also the fitted model
abline(ii_lm[[2]], lty = 2)

## For this feature (row) a decreasing signal intensity with injection
## index was observed (and modeled).

## For another feature an increasing intensity can be observed.
plot(x = sdata$injection_index, y = log2(vals[3, ]),
    ylab = expression(log[2]~abundance), xlab = "injection index")
grid()
## plot also the fitted model
abline(ii_lm[[3]], lty = 2)

## This trend can be removed from the data using the `adjust_lm` function
## by providing the linear models descring the drift. Note that, because
## we're adjusting log2 transformed data, the resulting abundances are
## also in log2 scale.
vals_adj <- adjust_lm(log2(vals), data = sdata, lm = ii_lm)

## Plotting the data before (open circles) and after adjustment (filled
## points)
plot(x = sdata$injection_index, y = log2(vals[2, ]),
    ylab = expression(log[2]~abundance), xlab = "injection index")
points(x = sdata$injection_index, y = vals_adj[2, ], pch = 16)
```

```
grid()
## Adding the line fitted through the raw data
abline(ii_lm[[2]], lty = 2)
## Adding a line fitted through the adjusted data
abline(lm(vals_adj[2, ] ~ sdata$injection_index), lty = 1)
## After adjustment there is no more dependency on injection index.
```

`formula2mz` *Calculate mass-to-charge ratio from a formula*

Description

`formula2mz` calculates the m/z values from a list of molecular formulas and adduct definitions.

Custom adduct definitions can be passed to the `adduct` parameter in form of a `data.frame`. This `data.frame` is expected to have columns `"mass_add"` and `"mass_multi"` defining the *additive* and *multiplicative* part of the calculation. See `adducts()` for examples.

Usage

```
formula2mz(formula, adduct = "[M+H]+", standardize = TRUE)
```

Arguments

`formula`	`character` with one or more valid molecular formulas for which their adduct m/z shall be calculated.
`adduct`	either a `character` specifying the name(s) of the adduct(s) for which the m/z should be calculated or a `data.frame` with the adduct definition. See `adductNames()` for supported adduct names and the description for more information on the expected format if a `data.frame` is provided.
`standardize`	`logical` whether to standardize the molecular formulas to the Hill notation system before calculating their mass.

Value

Numeric `matrix` with same number of rows than elements in `formula` and number of columns being equal to the length of `adduct` (adduct names are used as column names). Each column thus represents the m/z of `formula` for each defined `adduct`.

Author(s)

Roger Gine

Examples

```
## Calculate m/z values of adducts of a list of formulas
formulas <- c("C6H12O6", "C9H11NO3", "C16H13ClN2O")
ads <- c("[M+H]+", "[M+Na]+", "[2M+H]+", "[M]+")
formula2mz(formulas, ads)
formula2mz(formulas, adductNames()) #All available adducts

## Use custom-defined adducts as input
custom_ads <- data.frame(mass_add = c(1, 2, 3), mass_multi = c(1, 2, 0.5))
formula2mz(formulas, custom_ads)

## Use standardize = FALSE to keep formula unaltered
formula2mz("H12C6O6")
formula2mz("H12C6O6", standardize = FALSE)
```

---

guessSource *Guess the source of names*

Description

Guess the source of names based on predefined mappings. The function counts how many elements in the input vector `x` match the names defined in the mapping schema for each software and returns the software with the highest match count.

Usage

```
guessSource(x = character(), map = softwareMappingSchema())
```

Arguments

`x`	A `character` vector of names for which the source should be guessed.
`map`	Optional `data.frame` with a custom mapping schema. If not provided, the default mapping schema defined in the package will be used. The `data.frame` must contain the software names as column names and the corresponding names as values.

Value

A `character(1)` with the name of the guessed source software.

Author(s)

Gabriele Tomè

See Also

Other name translation functions: `nameMapping()`, `softwareMapping()`, `softwareMappingSchema()`, `translate()`

Examples

```
## MS-Dial names
x <- c("Average Rt(min)", "Alignment ID", "Average Mz")
guessSource(x)
```

---

indexRtime *Convert retention times to retention indices*

Description

indexRtime uses a list of known substances to convert retention times (RTs) to retention indices (RIs). By this retention information is normalized for differences in experimental settings, such as gradient delay volume, dead volume or flow rate. By default linear interpolation is performed, other ways of calculation can supplied as function.

Usage

```
indexRtime(x, y, FUN = rtiLinear, ...)
```

Arguments

x	numeric vector with retention times
y	data.framedata.frame containing two columns, where the first holds the retention times of the indexing substances and the second the actual index value
FUN	function function defining how the conversion is performed, default is linear interpolation
...	additional parameter used by FUN

Value

numeric vector of same length as x with retention indices. Values floating point decimals. If integer values shall be used conversion has to be performed manually

Author(s)

Michael Witting

Examples

```
rti <- data.frame(rtime = c(1,2,3),
rindex = c(100,200,300))
rtime <- c(1.5, 2.5)
indexRtime(rtime, rti)
```

internalStandardMixNames *Get names of internal standard mixes provided by the package*

Description

`internalStandardMixNames` returns available names of internal standard mixes provided by the `MetaboCoreUtils` package.

Usage

```
internalStandardMixNames()
```

Value

`character` names of available IS mixes

Author(s)

Michael Witting

Examples

```
internalStandardMixNames()
```

internalStandards *Get definitions for internal standards*

Description

`internalStandards` returns a table with metabolite standards available in commercial internal standard mixes. The returned data frame contains the following columns:

- `"name"`: the name of the standard
- `"formula_salt"`: chemical formula of the salt that was used to produce the standard mix
- `"formula_metabolite"`: chemical formula of the metabolite in free form
- `"smiles_salt"`: SMILES of the salt that was used to produced the standard mix
- `"smiles_metabolite"`: SMILES of the metabolite in free form
- `"mol_weight_salt"`: molecular (average) weight of the salt (can be used for calculation of molar concentration, etc.)
- `"exact_mass_metabolite"`: exact mass of free metabolites
- `"conc"`: concentration of the metabolite in ug/mL (of salt form)
- `"mix"`: name of internal standard mix

Usage

```
internalStandards(mix = "QReSS")
```

Arguments

mix	character(1) Name of the internal standard mix that shall be returned. One of internalStandardMixNames().

Value

data.frame data on internal standards

Author(s)

Michael Witting

See Also

internalStandardMixNames() for provided internal standard mixes.

Examples

```
internalStandards(mix = "QReSS")
internalStandards(mix = "UltimateSplashOne")
```

---

isotopicSubstitutionMatrix

Definitions of isotopic substitutions

Description

In order to identify potential isotopologues based on only m/z and intensity values with the isotopologues() function, sets of pre-calculated parameters are required. This function returns such parameter sets estimated on different sources/databases. The nomenclature used to describe isotopes follows the following convention: the number of neutrons is provided in [as a prefix to the element and the number of atoms of the element as suffix. [13]C2[37]Cl3 describes thus an isotopic substitution containing 2 [13]C isotopes and 3 [37]Cl isotopes.

Each row in the returned data.frame is associated with an isotopic substitution (which can involve isotopes of several elements or different isotopes of the same element). In general for each isotopic substitution multiple rows are present in the data.frame. Each row provides parameters to compute bounds (for the ratio between the isotopologue peak and the monoisotopic one) on a certain mass range. The provided isotopic substitutions are in general the most frequently observed substitutions in the database (e.g. HMDB) on which they were defined. Parameters (columns) defined for each isotopic substitution are:

- "minmass": the minimal mass of a compound for which the isotopic substitution was found. Peaks with a mass lower than this will most likely not have the respective isotopic substitution.

- "maxmass": the maximal mass of a compound for which the isotopic substitution was found. Peaks with a mass higher than this will most likely not have the respective isotopic substitution.
- "md": the mass difference between the monoisotopic peak and a peak of an isotopologue characterized by the respective isotopic substitution.
- "leftend": left endpoint of the mass interval.
- "rightend": right endpoint of the mass interval.
- "LBint": intercept of the lower bound line on the mass interval whose endpoints are "leftend" and "rightend".
- "LBslope": slope of the lower bound line on the mass interval.
- "UBint": intercept of the upper bound line on the mass interval.
- "UBslope": slope of the upper bound line on the mass interval.

Usage

```
isotopicSubstitutionMatrix(source = c("HMDB_NEUTRAL"))
```

Arguments

source	character(1) defining the set of predefined parameters and isotopologue definitions to return.

Value

data.frame with parameters to detect the defined isotopic substitutions

Available pre-calculated substitution matrices

- source = "HMDB": most common isotopic substitutions and parameters for these have been calculated for all compounds from the Human Metabolome Database (HMDB, July 2021). Note that the substitutions were calculated on the **neutral masses** (i.e. the chemical formulas of the compounds, not considering any adducts).

Author(s)

Andrea Vicini

Examples

```
## Get the substitution matrix calculated on HMDB
isotopicSubstitutionMatrix("HMDB_NEUTRAL")
```

`isotopologues` *Identfying isotopologue peaks in MS data*

Description

Given a spectrum (i.e. a peak matrix with m/z and intensity values) the function identifies groups of potential isotopologue peaks based on pre-defined mass differences and intensity (probability) ratios that need to be passed to the function with the `substDefinition` parameter. Each isotopic substitution in a compound determines a certain isotopologue and it is associated with a certain mass difference of that with respect to the monoisotopic isotopologue. Also each substitution in a compound is linked to a certain ratio between the intensities of the peaks of the corresponding isotopologue and the monoisotopic one. This ratio isn't the same for isotopologues corresponding to the same isotopic substitution but to different compounds. Through the `substDefinition` parameter we provide upper and lower values to compute bounds for each isotopic substitution dependent on the peak's mass.

Usage

```
isotopologues(
  x,
  substDefinition = isotopicSubstitutionMatrix(),
  tolerance = 0,
  ppm = 20,
  seedMz = numeric(),
  charge = 1,
  .check = TRUE
)
```

Arguments

`x`	`matrix` or `data.frame` with spectrum data. The first column is expected to contain *m/z* and the second column intensity values. The *m/z* values in that matrix are expected to be increasingly ordered and no `NA` values should be present.
`substDefinition`	`matrix` or `data.frame` with definition of isotopic substitutions (columns `"name"` and `"md"` are among the mandatory columns). The rows in this matrix have to be ordered by column `md` in increasing order. See `isotopicSubstitutionMatrix()` for more information on the format and content.
`tolerance`	`numeric(1)` representing the absolute tolerance for the relaxed matching of m/z values of peaks. See `MsCoreUtils::closest()` for details.
`ppm`	`numeric(1)` representing a relative, value-specific parts-per-million (PPM) tolerance for the relaxed matching of m/z values of peaks. See `MsCoreUtils::closest()` for details.
`seedMz`	`numeric` vector of **ordered** m/z values. If provided, the function checks if there are peaks in `x` which m/z match them. If any, it looks for groups where the first peak is one of the matched ones.

charge — numeric(1) representing the expected charge of the ionized compounds.

.check — logical(1) to disable input argument check. Should only be set to FALSE if provided *m/z* values are guaranteed to be increasingly ordered and don't contain NA values.

Details

The function iterates over the peaks (rows) in x. For each peak (which is assumed to be the monoisotopic peak) it searches other peaks in x with a difference in mass matching (given ppm and tolerance) any of the pre-defined mass differences in substDefinitions (column "md"). The mass is obtained by multiplying the m/z of the peaks for the charge expected for the ionized compounds.

For matching peaks, the function next evaluates whether their intensity is within the expected (pre-defined) intensity range. Using "LBint", "LBslope", "UBint", "UBslope" of the previously matched isotopic substitution in substDefinition, the function estimates a (mass dependent) lower and upper intensity ratio limit based on the peak's mass.

When some peaks are grouped together their indexes are excluded from the set of indexes that are searched for further groups (i.e. peaks already assigned to an isotopologue group are not considered/tested again thus each peak can only be part of one isotopologue group).

Value

list of integer vectors. Each integer vector contains the indixes of the rows in x with potential isotopologues of the same compound.

Author(s)

Andrea Vicini

Examples

```
## Read theoretical isotope pattern (high resolution) from example file
x <- read.table(system.file("exampleSpectra",
    "serine-alpha-lactose-caffeine.txt", package = "MetaboCoreUtils"),
    header = TRUE)
x <- x[order(x$mz), ]
plot(x$mz, x$intensity, type = "h")

isos <- isotopologues(x, ppm = 5)
isos

## highlight them in the plot
for (i in seq_along(isos)) {
    z <- isos[[i]]
    points(x$mz[z], x$intensity[z], col = i + 1)
}
```

mass2mz *Calculate mass-to-charge ratio*

Description

mass2mz calculates the m/z value from a neutral mass and an adduct definition.

Custom adduct definitions can be passed to the adduct parameter in form of a data.frame. This data.frame is expected to have columns "mass_add" and "mass_multi" defining the *additive* and *multiplicative* part of the calculation. See adducts() for examples.

Usage

```
mass2mz(x, adduct = "[M+H]+")
```

Arguments

x	numeric neutral mass for which the adduct m/z shall be calculated.
adduct	either a character specifying the name(s) of the adduct(s) for which the m/z should be calculated or a data.frame with the adduct definition. See adductNames() for supported adduct names and the description for more information on the expected format if a data.frame is provided.

Value

numeric matrix with same number of rows than elements in x and number of columns being equal to the length of adduct (adduct names are used as column names). Each column thus represents the m/z of x for each defined adduct.

Author(s)

Michael Witting, Johannes Rainer

See Also

mz2mass() for the reverse calculation, adductNames() for supported adduct definitions.

Examples

```
exact_mass <- c(100, 200, 250)
adduct <- "[M+H]+"

## Calculate m/z of [M+H]+ adduct from neutral mass
mass2mz(exact_mass, adduct)

exact_mass <- 100
adduct <- "[M+Na]+"

## Calculate m/z of [M+Na]+ adduct from neutral mass
```

```
mass2mz(exact_mass, adduct)

## Calculate m/z of multiple adducts from neutral mass
mass2mz(exact_mass, adduct = adductNames())

## Provide a custom adduct definition.
adds <- data.frame(mass_add = c(1, 2, 3), mass_multi = c(1, 2, 0.5))
rownames(adds) <- c("a", "b", "c")
mass2mz(c(100, 200), adds)
```

`mclosest` *Extract closest values in a pairwise manner between two matrices*

Description

The `mclosest` function calculates the closest rows between two matrices (or data frames) considering pairwise differences between values in columns of `x` and `table`. It returns the index of the closest row in `table` for each row in `x`.

Usage

```
mclosest(x, table, ppm = 0, tolerance = Inf)
```

Arguments

`x`	`numeric` matrix or data frame representing the query data. Each row in `x` will be compared to every row in `table`. Both `x` and `table` are expected to have the same number of columns, and the columns are expected to be in the same order.
`table`	`numeric` matrix or data frame containing the reference data to be matched with each row of `x`. Each row in `table` will be compared to every row in `x`. Both `table` and `x` are expected to have the same number of columns, and the columns are expected to be in the same order.
`ppm`	`numeric` representing a relative, value-specific parts-per-million (PPM) tolerance that is added to tolerance (default is 0).
`tolerance`	`numeric` accepted tolerance. Defaults to `tolerance = Inf`, thus for each row in x the closest row in table is reported, regardless of the magnitude of the (absolute) difference.

Details

If, for a row of `x`, two rows of `table` are closest only the index of first row will be returned.

For both the `tolerance` and `ppm` arguments, if their length is different to the number of columns of `x` and `table`, the input argument will be replicated to match it.

Value

`integer` vector of indices indicating the closest row of `table` for each row of `x`. If no suitable match is found for a row in `x` based on the specified `tolerance` and `ppm`, the corresponding index is set to `NA`.

Author(s)

Philippine Louail

Examples

```
x <- data.frame(a = 1:5, b = 3:7)
table <- data.frame(c = c(11, 23, 3, 5, 1), d = c(32:35, 45))

## Get for each row of `x` the index of the row in `table` with the smallest
## difference of values (per column)
mclosest(x, table)

## If the absolute difference is larger than `tolerance`, return `NA`. Note
## that the tolerance value of `25` is used for difference for each pairwise
## column in `x` and `table`.
mclosest(x, table, tolerance = 25)
```

---

multiplyElements *Multiply chemical formulas by a scalar*

Description

`multiplyElements` Multiply the number of atoms of each element by a constant, positive, integer

Usage

```
multiplyElements(x, k)
```

Arguments

x	`character` strings with chemical formula
k	`numeric(1)` positive integer by which each formula will be multiplied.

Value

`character` strings with the standardized chemical formula.

Author(s)

Roger Gine

Examples

```
multiplyElements("H2O", 3)

multiplyElements(c("C6H12O6", "Na", "CH4O"), 2)
```

mz2mass *Calculate neutral mass*

Description

mz2mass calculates the neutral mass from a given m/z value and adduct definition.

Custom adduct definitions can be passed to the adduct parameter in form of a data.frame. This data.frame is expected to have columns "mass_add" and "mass_multi" defining the *additive* and *multiplicative* part of the calculation. See adducts() for examples.

Usage

```
mz2mass(x, adduct = "[M+H]+")
```

Arguments

x	numeric m/z value for which the neutral mass shall be calculated.
adduct	either a character specifying the name(s) of the adduct(s) for which the m/z should be calculated or a data.frame with the adduct definition. See adductNames() for supported adduct names and the description for more information on the expected format if a data.frame is provided.

Value

numeric matrix with same number of rows than elements in x and number of columns being equal to the length of adduct (adduct names are used as column names. Each column thus represents the neutral mass of x for each defined adduct.

Author(s)

Michael Witting, Johannes Rainer

See Also

mass2mz() for the reverse calculation, adductNames() for supported adduct definitions.

Examples

```
ion_mass <- c(100, 200, 300)
adduct <- "[M+H]+"

## Calculate m/z of [M+H]+ adduct from neutral mass
mz2mass(ion_mass, adduct)

ion_mass <- 100
adduct <- "[M+Na]+"

## Calculate m/z of [M+Na]+ adduct from neutral mass
```

```
mz2mass(ion_mass, adduct)

## Provide a custom adduct definition.
adds <- data.frame(mass_add = c(1, 2, 3), mass_multi = c(1, 2, 0.5))
rownames(adds) <- c("a", "b", "c")
mz2mass(c(100, 200), adds)
```

---

nameMapping *Get mapping vector between two software*

Description

Get a named `character` vector that defines the mapping between names of two software based on the mapping schema. The names of the returned vector are the names of the `from` software and the values are the corresponding names of the `to` software.

Usage

```
nameMapping(
  from = character(),
  to = character(),
  map = softwareMappingSchema()
)
```

Arguments

`from`	`character(1)` with the name of the source software.
`to`	`character(1)` with the name of the target software.
`map`	Optional `data.frame` with a custom mapping schema. If not provided, the default mapping schema defined in the package will be used. The `data.frame` must contain the `from` and `to` parameter as columns names.

Value

A named `character` vector with the mapping between the two software.

Author(s)

Gabriele Tomè

See Also

Other name translation functions: `guessSource()`, `softwareMapping()`, `softwareMappingSchema()`, `translate()`

Examples

```
nameMapping(from = "MS-Dial", to = "mzTab-M")
```

pasteElements *Create chemical formula from a named vector*

Description

pasteElements creates a chemical formula from element counts (such as returned by countElements()).

Usage

```
pasteElements(x)
```

Arguments

x	list/integer with element counts, names being individual elements.

Value

character() with the chemical formulas.

Author(s)

Michael Witting and Sebastian Gibb

See Also

countElements()

Examples

```
elements <- c("C" = 6, "H" = 12, "O" = 6)
pasteElements(elements)
```

quality_assessment *Basic quality assessment functions for metabolomics*

Description

The following functions allow to calculate basic quality assessment estimates typically employed in the analysis of metabolomics data. These functions are designed to be applied to entire rows of data, where each row corresponds to a feature. Subsequently, these estimates can serve as a foundation for feature filtering.

- rsd and rowRsd are convenience functions to calculate the relative standard deviation (i.e. coefficient of variation) of a numerical vector or for rows of a numerical matrix, respectively.
- rowDratio computes the D-ratio or *dispersion ratio*, defined as the standard deviation for QC (Quality Control) samples divided by the standard deviation for biological test samples, for each feature (row) in the matrix.

- `percentMissing` and `rowPercentMissing` determine the percentage of missing values in a vector or for each row of a matrix, respectively.
- `rowBlank` identifies rows (i.e., features) where the mean of test samples is lower than a specified multiple (defined by the `threshold` parameter) of the mean of blank samples. This can be used to flag features that result from contamination in the solvent of the samples.

These functions are based on standard filtering methods described in the literature, and they are implemented to assist in preprocessing metabolomics data.

Usage

```
rsd(x, na.rm = TRUE, mad = FALSE)

rowRsd(x, na.rm = TRUE, mad = FALSE)

rowDratio(x, y, na.rm = TRUE, mad = FALSE)

percentMissing(x)

rowPercentMissing(x)

rowBlank(x, y, threshold = 2, na.rm = TRUE)
```

Arguments

`x`	`numeric` For `rsd`, a numeric vector; for `rowRsd`, `rowDratio`, `percentMissing` and `rowBlank`, a numeric matrix representing the biological samples.
`na.rm`	`logical(1)` indicates whether missing values (`NA`) should be removed prior to the calculations.
`mad`	`logical(1)` indicates whether the *Median Absolute Deviation* (MAD) should be used instead of the standard deviation. This is suggested for non-gaussian distributed data.
`y`	`numeric` For `rowDratio` and `rowBlank`, a numeric matrix representing feature abundances in QC samples or blank samples, respectively.
`threshold`	`numeric` For `rowBlank`, indicates the minimum difference required between the mean of a feature in samples compared to the mean of the same feature in blanks for it to not be considered a possible contaminant. For example, the default threshold of 2 signifies that the mean of the features in samples has to be at least twice the mean in blanks for it not to be flagged as a possible contaminant.

Value

See individual function description above for details.

Note

For `rsd` and `rowRsd` the feature abundances are expected to be provided in natural scale and not e.g. log2 scale as it may lead to incorrect interpretations.

Author(s)

Philippine Louail, Johannes Rainer

References

Broadhurst D, Goodacre R, Reinke SN, Kuligowski J, Wilson ID, Lewis MR, Dunn WB. Guidelines and considerations for the use of system suitability and quality control samples in mass spectrometry assays applied in untargeted clinical metabolomic studies. Metabolomics. 2018;14(6):72. doi: 10.1007/s11306-018-1367-3. Epub 2018 May 18. PMID: 29805336; PMCID: PMC5960010.

Examples

```
## coefficient of variation
a <- c(4.3, 4.5, 3.6, 5.3)
rsd(a)

A <- rbind(a, a, a)
rowRsd(A)

## Dratio
x <- c(4.3, 4.5, 3.6, 5.3)
X <- rbind(a, a, a)
rowDratio(X, X)

#' ## Percent Missing
b <- c(1, NA, 3, 4, NA)
percentMissing(b)

B <- matrix(c(1, 2, 3, NA, 5, 6, 7, 8, 9), nrow = 3)
rowPercentMissing(B)

## Blank Rows
test_samples <- matrix(c(13, 21, 3, 4, 5, 6), nrow = 2)
blank_samples <- matrix(c(0, 1, 2, 3, 4, 5), nrow = 2)
rowBlank(test_samples, blank_samples)
```

---

softwareMapping *Get supported software for name translation*

Description

Get the names of the supported software defined in the default mapping schema

Usage

```
softwareMapping()
```

Value

A `character` vector with the names of the supported software.

Author(s)

Gabriele Tomè

See Also

Other name translation functions: `guessSource()`, `nameMapping()`, `softwareMappingSchema()`, `translate()`

Examples

```
softwareMapping()
```

`softwareMappingSchema` *Get mapping schema for name translation*

Description

Get the mapping schema as a `data.frame` that defines the mapping between names of different software.

Usage

```
softwareMappingSchema(path = NULL)
```

Arguments

`path`	Optional `character(1)` with the path to a custom mapping schema file in TSV format. If not provided, the default mapping schema defined in the package will be used. The file must have a header row with software names as names and the corresponding names as values.

Value

A `data.frame` with the mapping schema.

Author(s)

Gabriele Tomè

See Also

Other name translation functions: `guessSource()`, `nameMapping()`, `softwareMapping()`, `translate()`

Examples

```
softwareMappingSchema()
```

`standardizeFormula` *Standardize a chemical formula*

Description

`standardizeFormula` standardizes a supplied chemical formula according to the Hill notation system.

Usage

```
standardizeFormula(x)
```

Arguments

`x`	`character`, strings with the chemical formula to standardize.

Value

`character` strings with the standardized chemical formula.

Author(s)

Michael Witting and Sebastian Gibb

See Also

`pasteElements()` `countElements()`

Examples

```
standardizeFormula("C6O6H12")
```

subtractElements *subtract two chemical formula*

Description

subtractElements subtracts one chemical formula from another.

Usage

```
subtractElements(x, y)
```

Arguments

x	character strings with chemical formula
y	character strings with chemical formula that should be subtracted from x

Value

character Resulting formula

Author(s)

Michael Witting and Sebastian Gibb

Examples

```
subtractElements("C6H12O6", "H2O")

subtractElements("C6H12O6", "NH3")
```

translate *Translate names based on a provided mapping*

Description

Map names from one software to another. The function replaces elements in the input vector x with their corresponding values in the mapping if a match is found. If an element in x does not have a corresponding mapping, it is returned unchanged with a warning.

Usage

```
translate(x = character(), mapping = NULL)
```

Arguments

x	character vector of names to be translated.
mapping	A named character vector that defines the mapping for translation. The names of the vector should be the original names and the values should be the translated names.

Value

A character vector with the translated names.

Author(s)

Gabriele Tomè

See Also

Other name translation functions: guessSource(), nameMapping(), softwareMapping(), softwareMappingSchema()

Examples

```
## MS-Dial names
x <- c("Average Rt(min)", "Alignment ID", "Average Mz")
map_vec <- nameMapping(from = "MS-Dial", to = "mzTab-M")
translate(x, mapping = map_vec)
```

Index

∗ **name translation functions**
  guessSource, 17
  nameMapping, 28
  softwareMapping, 31
  softwareMappingSchema, 32
  translate, 34

addElements, 2
adductFormula, 3
adductNames, 4
adductNames(), *4*, *16*, *24*, *27*
adducts *(*adductNames*)*, 4
adducts(), *3*, *4*, *16*, *24*, *27*
adjust_lm *(*fit_lm*)*, 12

betaValues, 5
BiocParallel::bpparam(), *14*

calculateKendrickMass, 7
calculateKm *(*calculateKendrickMass*)*, 7
calculateKmd *(*calculateKendrickMass*)*, 7
calculateMass, 8
calculateRkmd *(*calculateKendrickMass*)*, 7
containsElements, 9
convertMtime, 10
correctRindex, 11
countElements, 12
countElements(), *8*, *29*, *33*

dbeta(), *5*, *6*

fit_lm, 12
formula2mz, 16

guessSource, 17, *28*, *32*, *35*

indexRtime, 18
internalStandardMixNames, 19
internalStandardMixNames(), *20*
internalStandards, 19
isotopicSubstitutionMatrix, 20
isotopicSubstitutionMatrix(), *22*
isotopologues, 22
isotopologues(), *20*
isRkmd *(*calculateKendrickMass*)*, 7

lm(), *13*

mass2mz, 24
mass2mz(), *4*, *27*
mclosest, 25
MsCoreUtils::closest(), *22*
multiplyElements, 26
mz2mass, 27
mz2mass(), *4*, *24*

nameMapping, *17*, 28, *32*, *35*

pasteElements, 29
pasteElements(), *12*, *33*
percentMissing *(*quality_assessment*)*, 29

quality_assessment, 29

rowBlank *(*quality_assessment*)*, 29
rowDratio *(*quality_assessment*)*, 29
rowPercentMissing *(*quality_assessment*)*, 29
rowRsd *(*quality_assessment*)*, 29
rsd *(*quality_assessment*)*, 29

softwareMapping, *17*, *28*, 31, *32*, *35*
softwareMappingSchema, *17*, *28*, 32, *32*, *35*
standardizeFormula, 33
subtractElements, 34

translate, *17*, *28*, *32*, 34