Package: specProc (via r-universe)

October 15, 2026

Type Package

Title Preprocessing Tools for Laser-Induced Breakdown Spectroscopy

Version 0.8.3

Description Tools for exploring, preprocessing and modeling spectroscopic data, developed for laser-induced breakdown spectroscopy (LIBS) and also applicable to other techniques such as Raman, infrared, and inductively coupled plasma optical emission spectroscopy. Covers the workflow from raw spectra to calibrated results: peak finding and fitting, wavelength calibration, baseline correction, normalization and scaling, orthogonal signal correction including orthogonal partial least squares (OPLS; Trygg and Wold (2002) <doi:10.1002/cem.695>), calibration transfer, calibration curves and figures of merit, and plasma diagnostics (Boltzmann and Saha-Boltzmann plots, Stark broadening). Includes robust tools for outlier detection and exploration: robust principal component analysis (Hubert, Rousseeuw and Vanden Branden (2005) <doi:10.1198/004017004000000563>), detection of deviating cells (Rousseeuw and Van den Bossche (2018) <doi:10.1080/00401706.2017.1340909>), PCA with cellwise outliers and missing values (MacroPCA; Hubert, Rousseeuw and Van den Bossche (2019) <doi:10.1080/00401706.2018.1562989>), robust transformations to central normality (Raymaekers and Rousseeuw (2021) <doi:10.1007/s10994-021-05960-5>), and self-organizing maps (Kohonen (1982) <doi:10.1007/BF00337288>). Preprocessing steps are also available as 'recipes' steps for 'tidymodels' workflows. Computationally intensive steps are implemented in 'C++' via 'Rcpp' and 'RcppEigen'.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

LazyDataCompression xz

Depends R (>= 4.1.0)

Imports dplyr, ggplot2, lifecycle, magrittr, MASS, minpack.lm, moments, pls, Rcpp, rlang, robustbase, tibble, tidyr

Suggests bslib, cellWise, cli, ConfidenceEllipse, dials, embed, generics, ggsci, HotellingEllipse (>= 1.3.0), knitr, mixOmics, parsnip, patchwork, plotly, plsmod, prospectr, purrr, recipes (>= 1.1.0), rmarkdown, ropls, rospca, rsample, scales (>= 1.3.0), shiny, spelling, testthat (>= 3.0.0), tune, workflows, XICOR, xml2, yardstick

VignetteBuilder knitr

LinkingTo Rcpp, RcppEigen

Config/testthat/edition 3

URL https://github.com/ChristianGoueguel/specProc, https://christiangoueguel.com/specProc/

BugReports https://github.com/ChristianGoueguel/specProc/issues

Language en-US

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Christian L. Goueguel [aut, cre] (ORCID: <https://orcid.org/0000-0003-0521-3446>)

Maintainer Christian L. Goueguel <christian.goueguel@gmail.com>

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

Date/Publication 2026-10-15 15:20:02 UTC

RemoteUrl https://github.com/cran/specProc

RemoteRef HEAD

RemoteSha 4b6fcf045e304d415d7e39528d19c3587b69dceb

Contents

adjusted_boxplot 5
apply_calibration 7
average 8
baseline_als 9
baseline_arpls 10
baseline_lambda 11
baseline_lsp 12
biweight_location 14
biweight_midcorrelation 15
biweight_midcovariance 16
biweight_midvariance 17
biweight_scale 18
boltzmann 19
calibration_curve 21

cellpca . . . 23
center . . . 26
cf_libs . . . 27
contributions . . . 30
correct_self_absorption . . . 32
correlation . . . 34
direct_orthogonal . . . 37
direct_osc . . . 39
directional_outlyingness . . . 40
dmodx . . . 42
electron_density . . . 44
emsc . . . 46
epo . . . 48
flagged_regions . . . 49
forageLIBS . . . 51
gaussian_profile . . . 52
generalized_boxplot . . . 53
glsw . . . 55
glsw_alpha . . . 57
hotelling_t2 . . . 57
iqr_outliers . . . 59
libs_lines . . . 61
line_finder . . . 62
line_intensities . . . 64
loading_peaks . . . 66
lorentzian_profile . . . 67
macropca . . . 68
mcwhirter_criterion . . . 71
medcouple_weight . . . 72
minmax . . . 74
msc . . . 75
multipeak_fit . . . 76
nas . . . 78
nist_ionization_energy . . . 80
nist_levels . . . 81
nist_lines . . . 82
normalize . . . 84
o2pls . . . 85
opls . . . 87
osc . . . 90
pareto_scale . . . 92
pds . . . 93
peak_fit . . . 95
plot_boltzmann . . . 97
plot_calibration . . . 98
plot_cell_map . . . 99
plot_contributions . . . 101
plot_embedding . . . 103

plot_fit . . . 106
plot_influence . . . 108
plot_lines . . . 110
plot_loadings . . . 111
plot_outlier_map . . . 113
plot_outliers . . . 115
plot_som . . . 117
plot_spectra . . . 118
plot_wavelength_calibration . . . 119
poisson_scale . . . 120
predict.specproc_calibration . . . 121
predict.specproc_emsc . . . 123
predict.specproc_filter . . . 124
predict.specproc_macropca . . . 125
predict.specproc_nas . . . 126
predict.specproc_opls . . . 127
predict.specproc_som . . . 128
predict.specproc_wavelength_calibration . . . 129
projected_osc . . . 130
pseudo_voigt_profile . . . 132
q_residuals . . . 134
quantile_weight . . . 136
read_starkb . . . 138
reject_shots . . . 139
robpca . . . 141
robust_bcyj . . . 143
rospca . . . 145
rousseeuw_croux . . . 147
saha_boltzmann . . . 148
saturation_summary . . . 149
savgol_derivative . . . 150
savitzky_golay . . . 151
self_absorption . . . 152
snv . . . 153
som . . . 154
som_stability . . . 157
spectral_norm_method . . . 158
stark_table . . . 158
stark_width . . . 160
starkb_lines . . . 162
step_baseline . . . 163
step_cellpca . . . 165
step_direct_orthogonal . . . 167
step_direct_osc . . . 169
step_emsc . . . 170
step_epo . . . 172
step_glsw . . . 174
step_line_intensities . . . 175

step_line_ratio . . . 177
step_macropca . . . 179
step_msc . . . 181
step_o2pls . . . 182
step_opls . . . 186
step_osc . . . 187
step_pareto_scale . . . 190
step_poisson_scale . . . 191
step_projected_osc . . . 192
step_reject_shots . . . 194
step_robpca . . . 196
step_robust_bcyj . . . 197
step_rospca . . . 199
step_savgol . . . 201
step_snv . . . 202
step_spectral_norm . . . 204
step_wavelet . . . 205
step_y_gradient_glsw . . . 207
summary_stats . . . 209
tukey_gh . . . 210
umad . . . 212
voigt_fwhm . . . 213
voigt_profile . . . 214
wavelength_calibration . . . 215
wavelet_features . . . 218
wavelet_level . . . 219
y_gradient_glsw . . . 220
zscore . . . 221

Index **223**

`adjusted_boxplot` *Adjusted Boxplot*

Description

This function generates the adjusted boxplot, which is a robust graphical method for visualizing skewed data distributions. It provides a more accurate representation of the data's spread and skewness compared to standard boxplot, especially in the presence of outliers.

Usage

```
adjusted_boxplot(
  x,
  plot = TRUE,
  xlabels.angle = 90,
  xlabels.vjust = 1,
  xlabels.hjust = 1,
```

```
  box.width = 0.5,
  notch = FALSE,
  notchwidth = 0.5,
  staplewidth = 0.5
)
```

Arguments

`x`	A numeric data frame or tibble.
`plot`	A logical value indicating whether to plot the adjusted boxplot (default is `TRUE`).
`xlabels.angle`	A numeric value specifying the angle (in degrees) for x-axis labels (default is 90).
`xlabels.vjust`	A numeric value specifying the vertical justification of x-axis labels (default is 1).
`xlabels.hjust`	A numeric value specifying the horizontal justification of x-axis labels (default is 1).
`box.width`	A numeric value specifying the width of the boxplot (default is 0.5).
`notch`	A logical value indicating whether to display a notched boxplot (default is `FALSE`).
`notchwidth`	A numeric value specifying the width of the notch relative to the body of the boxplot (default is 0.5).
`staplewidth`	A numeric value specifying the width of staples at the ends of the whiskers.

Details

The function is based on the medcouple (MC) measure computed on the data and which robustly measures skewness. This measure is bounded between -1 and 1. The medcouple is equal to zero when the observed distribution is symmetric, whereas a positive (resp. negative) value of MC corresponds to a right (resp. left) tailed distribution. It worth noting that this method is more appropriate for distributions that are not excessively skewed i.e., for $|\text{MC}| \leq 0.6$.

Value

- If `plot = TRUE`, returns a `ggplot2` object containing the adjusted boxplot.
- If `plot = FALSE`, returns a list of tibbles with the adjusted boxplot statistics and potential outliers.

Author(s)

Christian L. Goueguel

References

The adjusted boxplot is based on the methodology described in:

- Brys, G., Hubert, M., Struyf, A., (2004). A Robust Measure of Skewness. Journal of Computational and Graphical Statistics, 13(4):996-1017
- Hubert, M., Vandervieren, E., (2008). An adjusted boxplot for skewed distributions. Computational Statistics and Data Analysis, 52(12):5186-5201

Examples

```
data(forageLIBS)
# mineral contents (%) of the forage samples
adjusted_boxplot(forageLIBS[c("Ca", "Mg", "P", "K", "S")])
adjusted_boxplot(forageLIBS[c("Ca", "Mg", "P", "K", "S")], plot = FALSE)
```

---

apply_calibration *Apply a Wavelength Calibration*

Description

Relabels the wavelengths of spectra with the correction fitted by `wavelength_calibration()`. The intensities are not changed.

Usage

```
apply_calibration(spectra, calibration)
```

Arguments

`spectra`	Spectra with the wavelength axis of the calibration: a numeric vector named by wavelength, or a matrix or data frame with the wavelengths as column names (other columns are kept unchanged).
`calibration`	The result of `wavelength_calibration()`.

Details

The spectra must have the wavelength axis of the calibration (the same channels, in the same order), so that each channel gets the correction of its detector segment. The corrected wavelengths are rounded to 4 decimals (0.1 pm) for the column names.

Value

`spectra`, with the corrected wavelengths as names.

See Also

`wavelength_calibration()`

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
first_detector <- spectra[seq_len(which(diff(wl) < 0)[1])]
reference <- c(`Mg I` = 285.213, `Ca II` = 317.933, `Ca II` = 393.366, `K I` = 404.414,
               `Ca I` = 422.673, `Mg I` = 518.360, `Na I` = 588.995, `Ca I` = 643.907)
cal <- wavelength_calibration(first_detector, reference)
corrected <- apply_calibration(first_detector, cal)
head(names(corrected))
```

`average` *Fast Average for Large Spectral Dataset*

Description

This function efficiently averages the samples spectra in a large dataset. It provides flexibility by computing either the overall mean across all spectra or group-wise means based on the values of a specified grouping column.

Usage

```
average(x, .group_by = NULL)
```

Arguments

`x`	A data frame or tibble.
`.group_by`	The column to group the data by (optional), given either unquoted or as a string. If not provided, the average of the overall data will be computed.

Details

The function leverages the power of `Rcpp` to perform the mean calculations in C++. The underlying C++ implementation has a time complexity of *O(n x m)*, where *n* is the number of rows and *m* is the number of columns in the data. Missing values are ignored in the computation of each mean.

Value

- If `.group_by = NULL`, a one-row tibble containing the mean of each column.
- If `.group_by` is provided, a tibble with one row per group: the first column holds the group labels and the remaining columns hold the group means.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
# one mean spectrum per sample: three samples were measured twice
means <- average(forageLIBS[-c(1, 3:14)], Sample)
dim(means)
```

baseline_als — *Asymmetric Least Squares*

Description

Baseline correction based on asymmetric least squares (ALS) algorithm as proposed by Eilers *et al.* (2005).

Usage

```
baseline_als(x, lambda = 1000, p = 0.001, max.iter = 10)
```

Arguments

`x`	A numeric matrix or data frame, with one spectrum per row.
`lambda`	A numeric value specifying the smoothing parameter, which controls the amount of curvature allowed for the baseline. The smaller the lambda, the more curvature in the baseline fitting. Default is 1000.
`p`	A numeric value specifying the extent of asymmetry required of the fit. Larger values allow more negative-going regions. Smaller values disallow negative-going regions. `p` must be between 0 and 1. Default is 0.001.
`max.iter`	Maximum number of iterations for the algorithm. Default is 10.

Details

The function applies Eilers' method based on a Whittaker filter. The algorithm estimates a baseline curve by minimizing the asymmetric least squares criterion, which allows for different weights for positive and negative residuals. The resulting baseline curve is subtracted from the input data, providing a baseline-corrected version.

The penalized system is pentadiagonal and is solved in C++ with a banded Cholesky decomposition, so the cost grows linearly with the number of spectral channels. Negative values in the corrected spectra are kept as they are (they typically reflect noise around the baseline).

Value

A list containing two tibbles:

- `correction`: The baseline-corrected spectral matrix.
- `background`: The fitted background emission.

Author(s)

Christian L. Goueguel

References

- Eilers, P.H.C., Boelens, H.F.M., (2005). Baseline correction with asymmetric least squares smoothing. Leiden University Medical Centre report.

Examples

```
data(forageLIBS)
spectrum <- forageLIBS[1, -(1:14)]
wl <- as.numeric(names(spectrum))
in_region <- wl > 240 & wl < 300
region <- spectrum[in_region]
res <- baseline_als(region, lambda = 1e6)
oldpar <- par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))
# the spectrum and its fitted baseline
plot(wl[in_region], unlist(region), type = "l", col = "grey40", ylim = c(800, 3000),
     xlab = "Wavelength (nm)", ylab = "Counts", main = "Spectrum and baseline")
lines(wl[in_region], unlist(res$background), col = "red")
# the corrected spectrum: the background is now around zero
plot(wl[in_region], unlist(res$correction), type = "l", col = "grey40",
     ylim = c(-200, 2000), xlab = "Wavelength (nm)", ylab = "Counts",
     main = "Corrected spectrum")
abline(h = 0, col = "red", lty = 2)
par(oldpar)
```

`baseline_arpls` *Asymmetrically Reweighted Penalized Least Squares*

Description

Baseline correction based on asymmetrically reweighted penalized least squares smoothing algorithm as proposed by Baek *et al.* (2015).

Usage

```
baseline_arpls(x, lambda = 1000, ratio = 0.05, max.iter = 10)
```

Arguments

`x`	A numeric matrix or data frame, with one spectrum per row.
`lambda`	A numeric value specifying the smoothing parameter, which controls the amount of curvature allowed for the baseline. The smaller the lambda, the more curvature in the baseline fitting. Default is 1000.
`ratio`	A numeric value specifying the convergence ratio for the iterative algorithm. The algorithm stops when the relative change in the weights is less than this ratio. Typical values are between 0.01 and 0.1. Default is 0.05.
`max.iter`	Maximum number of iterations for the algorithm. Default is 10.

Details

The algorithm estimates a baseline curve by iteratively updating weights based on the residuals and minimizing a penalized least squares criterion. The resulting baseline curve is subtracted from the input data, providing a baseline-corrected version.

The penalized system is pentadiagonal and is solved in C++ with a banded Cholesky decomposition, so the cost grows linearly with the number of spectral channels.

Value

A list containing two tibbles:

- `correction`: The baseline-corrected spectral matrix.
- `background`: The fitted background emission.

Author(s)

Christian L. Goueguel

References

- Baek, S.-J., Park, A., Ahn, Y.-J., Choo, J., (2015). Baseline correction using asymmetrically reweighted penalized least squares smoothing. Analyst, 140(1):250–257.

Examples

```
data(forageLIBS)
spectrum <- forageLIBS[1, -(1:14)]
wl <- as.numeric(names(spectrum))
in_region <- wl > 240 & wl < 300
region <- spectrum[in_region]
res <- baseline_arpls(region, lambda = 1e5)
oldpar <- par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))
# the spectrum and its fitted baseline
plot(wl[in_region], unlist(region), type = "l", col = "grey40", ylim = c(800, 3000),
     xlab = "Wavelength (nm)", ylab = "Counts", main = "Spectrum and baseline")
lines(wl[in_region], unlist(res$background), col = "red")
# the corrected spectrum: the background is now around zero
plot(wl[in_region], unlist(res$correction), type = "l", col = "grey40",
     ylim = c(-200, 2000), xlab = "Wavelength (nm)", ylab = "Counts",
     main = "Corrected spectrum")
abline(h = 0, col = "red", lty = 2)
par(oldpar)
```

`baseline_lambda` *Baseline Smoothness Parameter*

Description

A dials parameter for the `lambda` argument of `step_baseline()`, on a log10 scale.

Usage

```
baseline_lambda(range = c(2, 8), trans = scales::transform_log10())
```

Arguments

`range`	A two-element vector with the range of `lambda`, in log10 units. Default is `c(2, 8)`, i.e. 100 to 10^8.
`trans`	A transformation object from the scales package. Default is log10.

Value

A `quant_param` object.

Examples

```
baseline_lambda()
```

---

baseline_lsp *Least-Squares Polynomial*

Description

This function performs baseline correction on the spectral matrix by estimating and removing the continuous background emission using least-squares polynomial curve fitting approach.

Usage

```
baseline_lsp(
  x,
  degree = 4,
  tol = 0.001,
  max.iter = 100,
  method = c("imodpoly", "modpoly")
)
```

Arguments

`x`	A matrix or data frame, with one spectrum per row.
`degree`	An integer specifying the degree of the polynomial fitting function. The default value is 4.
`tol`	A numeric value representing the tolerance for the difference between iterations. The default value is 1e-3.
`max.iter`	An integer specifying the maximum number of iterations for the algorithm. The default value is 100.
`method`	The algorithm: `"imodpoly"` (default) or `"modpoly"`. See Details.

Details

Two iterative polynomial fits are available:

- `"modpoly"`: the modified polynomial fit of Lieber and Mahadevan-Jansen (2003). A polynomial is fitted to the spectrum, the points of the spectrum above the fit are replaced by the fit, and the fit is repeated, so that the peaks are progressively removed. The first fits are pulled up by strong peaks, and the ends of the spectrum, where nothing pulls the polynomial back, can then fall far below the background.
- `"imodpoly"` (default): the improved modified polynomial fit of Zhao *et al.* (2007). The points more than one standard deviation of the residuals above the first fit (the peaks) are left out of the following fits, and the remaining points are clipped at the fit plus the standard deviation of the residuals, rather than at the fit. The iterations stop when this standard deviation changes by less than `tol` (relative). The baseline follows the background up to the ends of the spectrum and passes through its noise.

Value

A list with two elements:

- `correction`: The baseline-corrected spectral matrix.
- `background`: The fitted background emission.

Author(s)

Christian L. Goueguel

References

- Lieber, C.A., Mahadevan-Jansen, A., (2003). Automated method for subtraction of fluorescence from biological Raman spectra. Applied Spectroscopy, 57(11):1363-1367
- Zhao, J., Lui, H., McLean, D.I., Zeng, H., (2007). Automated autofluorescence background subtraction algorithm for biomedical Raman spectroscopy. Applied Spectroscopy, 61(11):1225-1232.

Examples

```
data(forageLIBS)
spectrum <- forageLIBS[1, -(1:14)]
wl <- as.numeric(names(spectrum))
in_region <- wl > 240 & wl < 300
region <- spectrum[in_region]
res <- baseline_lsp(region, degree = 4)
oldpar <- par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))
# the spectrum and its fitted baseline
plot(wl[in_region], unlist(region), type = "l", col = "grey40", ylim = c(800, 3000),
     xlab = "Wavelength (nm)", ylab = "Counts", main = "Spectrum and baseline")
lines(wl[in_region], unlist(res$background), col = "red")
# the corrected spectrum: the background is now around zero
plot(wl[in_region], unlist(res$correction), type = "l", col = "grey40",
     ylim = c(-200, 2000), xlab = "Wavelength (nm)", ylab = "Counts",
```

```
     main = "Corrected spectrum")
abline(h = 0, col = "red", lty = 2)
par(oldpar)
```

`biweight_location` *Biweight Location*

Description

This function computes the biweight location, a robust measure of central tendency for a numeric vector. The biweight location is less sensitive to outliers than the sample mean.

Usage

```
biweight_location(
  x,
  loc = NULL,
  c = 6,
  tol = 1e-06,
  max_iter = 50,
  drop.na = FALSE
)
```

Arguments

`x`	A numeric vector.
`loc`	Initial guess for the location (default: median of `x`).
`c`	A numeric value specifying the tuning constant for the biweight estimator (`c = 6` by default).
`tol`	Convergence tolerance for the iterative computation (default: 1e-6).
`max_iter`	Maximum number of iterations (default: 50).
`drop.na`	A logical value indicating whether to remove missing values (`NA`) from the calculations. If `TRUE`, missing values will be removed. If `FALSE` (the default), the result is `NA` when `x` contains missing values.

Details

Starting from the initial location M (the median by default), the biweight location is

$$\zeta = M + \frac{\sum_{|u_i|<1}(x_i - M)(1-u_i^2)^2}{\sum_{|u_i|<1}(1-u_i^2)^2}, \quad u_i = \frac{x_i - M}{c \cdot \mathrm{MAD}}$$

where MAD is the (unscaled) median absolute deviation. The estimate is iterated, replacing M by ζ, until the change is smaller than `tol` or `max_iter` iterations are reached.

Value

The biweight location of x.

Author(s)

Christian L. Goueguel

References

- Mosteller, F., and Tukey, J. W. (1977). Data Analysis and Regression: A Second Course in Statistics. Addison-Wesley, pp. 203-209.
- Beers, T.C., Flynn, K., Gebhardt, K., (1990). Measures of location and scale for velocities in clusters of galaxies - A robust approach. The Astronomical Journal, 100:32-46.

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
c(mean = mean(fe), median = stats::median(fe), biweight = biweight_location(fe))
```

---

biweight_midcorrelation *Biweight Midcorrelation*

Description

This function computes the biweight midcorrelation between two numeric vectors. The biweight midcorrelation is a robust measure of correlation that is less sensitive to outliers than the traditional Pearson's correlation coefficient.

Usage

```
biweight_midcorrelation(x, y)
```

Arguments

x	A numeric vector.
y	A numeric vector of the same length as x.

Details

The biweight midcorrelation is calculated using the biweight midvariances and biweight midcovariance, as described by Wilcox (1994). It is bounded between -1 and 1.

Value

The biweight midcorrelation between x and y.

Author(s)

Christian L. Goueguel

References

- Wilcox, R. R. (1994). The Biweight Midcorrelation: A Robust Correlation Technique for Two Samples. Journal of Statistical Computation and Simulation, 48(2):103-110.

Examples

```
data(forageLIBS)
ok <- !is.na(forageLIBS$Fe)
c(pearson = stats::cor(forageLIBS$Fe[ok], forageLIBS$Mn[ok]),
  biweight = biweight_midcorrelation(forageLIBS$Fe[ok], forageLIBS$Mn[ok]))
```

---

biweight_midcovariance *Biweight Midcovariance*

Description

This function computes the biweight midcovariance, a robust measure of covariance between two numerical vectors. The biweight midcovariance is less sensitive to outliers than the traditional covariance.

Usage

```
biweight_midcovariance(x, y)
```

Arguments

x	A numeric vector.
y	A numeric vector of the same length as x.

Value

The biweight midcovariance between x and y.

Author(s)

Christian L. Goueguel

References

- Wilcox, R., (1997). Introduction to Robust Estimation and Hypothesis Testing. Academic Press

Examples

```
data(forageLIBS)
# iron and manganese contents (mg/kg), with a few iron-rich samples
ok <- !is.na(forageLIBS$Fe)
c(covariance = stats::cov(forageLIBS$Fe[ok], forageLIBS$Mn[ok]),
  biweight = biweight_midcovariance(forageLIBS$Fe[ok], forageLIBS$Mn[ok]))
```

---

biweight_midvariance *Biweight Midvariance*

Description

This function calculates the biweight midvariance of a numeric vector, which is a robust measure of scale that can be used to estimate the variability of the data while being resistant to the influence of outliers.

Usage

```
biweight_midvariance(x, drop.na = FALSE)
```

Arguments

x	A numeric vector.
drop.na	A logical value indicating whether to remove missing values (NA) from the calculations. If TRUE, missing values will be removed. If FALSE (the default), the result is NA when x contains missing values.

Details

For scale estimators, the standard deviation (or variance) is the optimal estimator for Gaussian data. However, it is not resistant and it does not have robustness of efficiency. In robust statistics, the median absolute deviation (MAD) is a resistant estimate, but it has only modest robustness of efficiency, while the biweight midvariance estimator is both resistant and robust of efficiency.

Value

The biweight midvariance of the input vector.

Author(s)

Christian L. Goueguel

References

- Wilcox, R., (1997). Introduction to Robust Estimation and Hypothesis Testing. Academic Press

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
c(variance = stats::var(fe), biweight = biweight_midvariance(fe))
```

---

biweight_scale *Biweight Scale*

Description

This function computes the biweight scale, a robust measure of scale or dispersion for a numeric vector. The biweight scale is less sensitive to outliers than the sample standard deviation.

Usage

```
biweight_scale(x, loc = NULL, c = 9, reduced = FALSE, drop.na = FALSE)
```

Arguments

x	A numeric vector.
loc	The location about which the scale is computed (default: median of x).
c	A numeric value specifying the tuning constant for the biweight estimator (c = 9 by default).
reduced	A logical value specifying whether the sample size, *n*, should be reduced to the number of non-rejected values. If TRUE, *n* is reduced to the number of observations that pass a rejection criteria ($\lvert u_i \rvert < 1$). If FALSE (default), *n* is equal to the length of x (the input data).
drop.na	A logical value indicating whether to remove missing values (NA) from the calculations. If TRUE, missing values will be removed. If FALSE (the default), the result is NA when x contains missing values.

Details

The biweight scale is the square root of the biweight midvariance:

$$\zeta = \frac{\sqrt{n \sum_{|u_i|<1} (x_i - M)^2 (1 - u_i^2)^4}}{\left| \sum_{|u_i|<1} (1 - u_i^2)(1 - 5u_i^2) \right|}, \quad u_i = \frac{x_i - M}{c \cdot \mathrm{MAD}}$$

where M is the location (the median by default) and MAD is the (unscaled) median absolute deviation. For normally distributed data the biweight scale is a consistent estimator of the standard deviation.

Value

The biweight scale of `x`.

Author(s)

Christian L. Goueguel

References

- Mosteller, F., and Tukey, J. W. (1977). Data Analysis and Regression: A Second Course in Statistics. Addison-Wesley, pp. 203-209.
- Beers, T.C., Flynn, K., Gebhardt, K., (1990). Measures of location and scale for velocities in clusters of galaxies - A robust approach. The Astronomical Journal, 100:32-46.

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
c(sd = stats::sd(fe), mad = stats::mad(fe), biweight = biweight_scale(fe))
```

`boltzmann` *Plasma Temperature from a Boltzmann Plot*

Description

Estimates the excitation temperature of a plasma in local thermodynamic equilibrium (LTE) from the intensities of several lines of the same species, by a Boltzmann plot.

Usage

```
boltzmann(lines, units = "energy")
```

Arguments

`lines`	A data frame with one row per line and columns `intensity` (integrated line intensity), `wavelength` (nm), `Aki` (transition probability, s^{-1}), `gk` (upper-level degeneracy) and `Ek` (upper-level energy, eV).
`units`	The units of `intensity`: `"energy"` (default; radiance or energy-calibrated counts) or `"photons"` (photon counts).

Details

In LTE, the integrated intensity of a line emitted from an upper level of energy E_k and degeneracy g_k, with transition probability A_{ki} and wavelength λ, satisfies

$$\ln \frac{I\lambda}{g_k A_{ki}} = -\frac{E_k}{k_B T} + C$$

for intensities in energy units, or $\ln(I/g_k A_{ki})$ for photon counts (`units = "photons"`). A least-squares line through the points gives the temperature from its slope, $T = -1/(k_B \,\mathrm{slope})$, and its standard error from that of the slope.

The intensities must be corrected for the spectral response of the instrument, and the lines must be optically thin (see `self_absorption()`). A wide range of upper-level energies gives a more precise temperature. Atomic data (A_{ki}, g_k, E_k) can be taken from the NIST Atomic Spectra Database (https://physics.nist.gov/PhysRefData/ASD/lines_form.html).

Value

An object of class `specproc_boltzmann`, a list with

- `temperature` and `temperature_se`: the temperature and its standard error, in K;
- `r_squared`: the coefficient of determination of the fit;
- `points`: a tibble with the abscissa `x` (eV) and ordinate `y` of each line;
- `fit`: the `lm` fit of `y` on `x`.

Use `plot_boltzmann()` to draw the plot.

Author(s)

Christian L. Goueguel

References

- Cristoforetti, G., De Giacomo, A., Dell'Aglio, M., Legnaioli, S., Tognoni, E., Palleschi, V., Omenetto, N. (2010). Local thermodynamic equilibrium in laser-induced breakdown spectroscopy: beyond the McWhirter criterion. Spectrochimica Acta Part B, 65(1):86-95.

See Also

`saha_boltzmann()`, `plot_boltzmann()`, `mcwhirter_criterion()`

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
# calcium lines, with their atomic data from the NIST database
atomic <- data.frame(
  species = c(rep("Ca I", 6), rep("Ca II", 4)),
  stage = c(rep(1, 6), rep(2, 4)),
  wavelength = c(428.301, 430.253, 431.865, 445.478, 612.222, 616.217,
                 315.887, 317.933, 370.603, 373.690),
```

```
  Aki = c(4.34e7, 1.36e8, 7.40e7, 8.70e7, 2.87e7, 4.77e7, 3.10e8, 3.60e8, 8.80e7, 1.70e8),
  gk = c(5, 5, 3, 7, 3, 3, 4, 6, 2, 2),
  Ek = c(4.780, 4.780, 4.769, 4.681, 3.910, 3.910, 7.047, 7.050, 6.468, 6.468)
)
lines <- line_intensities(mean_spectrum, atomic, baseline = TRUE)
fit <- boltzmann(lines[lines$stage == 1, ])
fit
plot_boltzmann(fit)
```

`calibration_curve` *Univariate Calibration Curve*

Description

Fits a univariate calibration curve (signal against concentration), with its figures of merit, limits of detection and quantification, and tests of linearity. The curve predicts the concentration of new samples from their signal, with confidence intervals.

Usage

```
calibration_curve(
  data,
  signal,
  concentration,
  model = "linear",
  weights = NULL,
  blank = NULL,
  lod_method = NULL,
  level = 0.95
)
```

Arguments

`data`	A data frame with the calibration samples.
`signal`, `concentration`	The columns of `data` holding the signal and the reference concentration, unquoted or as strings.
`model`	`"linear"` (default) or `"quadratic"`.
`weights`	`NULL` (default, unweighted), `"1/x"`, `"1/x2"`, or a numeric vector of weights, one per row of `data`.
`blank`	An optional numeric vector of replicate signals of a blank sample.
`lod_method`	The estimate of the blank standard deviation: `"residual"`, `"intercept"` or `"blank"` (see details). By default, `"blank"` when `blank` is given, `"residual"` otherwise.
`level`	The confidence level of the intervals of the coefficients. Default is 0.95.

Details

The signal, such as a line intensity from `line_intensities()` or a line ratio, is modeled as a straight line or a second-degree polynomial of the concentration, fitted by (weighted) least squares. Weights `"1/x"` or `"1/x2"` suit signals whose variance grows with concentration, which is common in LIBS.

Figures of merit. The sensitivity is the slope of the curve at zero concentration. The limits of detection and quantification are $LOD = 3.3\,\sigma/S$ and $LOQ = 10\,\sigma/S$ (ICH, 2005), where S is the sensitivity and σ the standard deviation of the signal of a blank, estimated from

- `"residual"`: the residual standard deviation of the fit (default without `blank`);
- `"intercept"`: the standard error of the intercept;
- `"blank"`: the standard deviation of replicate `blank` signals (default when given).

With weights, the residual standard deviation depends on the scale of the weights, so prefer `"blank"` or `"intercept"`.

Linearity. For a straight line, Mandel's test compares its residual variance with that of a quadratic fit; a small p-value means that the curvature is significant (as with self-absorption or detector saturation). When concentrations are replicated, the lack-of-fit test compares the residuals with the pure error of the replicates.

Intervals. The confidence band of the curve shows where the mean signal lies at each concentration; the prediction band, wider, where a single new measurement is expected to fall, since it adds the noise of the measurement. `plot_calibration()` draws either or both, and predict() computes them for given concentrations (`type = "signal"`).

Inverse prediction. The concentration of a sample is the solution of the calibration equation for its signal (for a quadratic curve, the root within or nearest to the calibration range). Its standard error, by the delta method, combines the uncertainty of the curve with that of the new signal, the mean of `replicates` measurements.

Value

An object of class `specproc_calibration`, a list with

- `coefficients`: a tibble of the estimates, standard errors and confidence intervals (`lower`, `upper`) at `level`;
- `figures_of_merit`: a tibble with the number of standards `n`, the `sensitivity`, `r_squared`, the residual standard deviation `sigma`, the blank standard deviation used (`sigma_blank`), `lod` and `loq` (in concentration units);
- `linearity`: a tibble with the statistic, degrees of freedom and p-value of Mandel's test and of the lack-of-fit test (when they apply);
- `fit`: the `lm` fit, and `data`, the calibration data.

Use predict() for new samples and `plot_calibration()` to draw the curve.

Author(s)

Christian L. Goueguel

References

- ICH (2005). Validation of analytical procedures: text and methodology Q2(R1). International Conference on Harmonisation.
- Mandel, J. (1964). The Statistical Analysis of Experimental Data. Interscience, New York.
- Miller, J.N., Miller, J.C. (2018). Statistics and Chemometrics for Analytical Chemistry, 7th ed. Pearson, Harlow.

See Also

predict.specproc_calibration(), plot_calibration(), line_intensities(), nas()

Examples

```
data(forageLIBS)
# the K I 769.90 nm line, normalized to the C I 247.86 nm line of the matrix
lines <- line_intensities(forageLIBS[-(1:14)], c(C = 247.856, K = 769.896), baseline = TRUE)
standards <- data.frame(
  K = forageLIBS$K,
  signal = lines$intensity[lines$line == "K"] / lines$intensity[lines$line == "C"]
)
cal <- calibration_curve(standards[1:300, ], signal, K)
cal
# the concentrations of new samples, from their signals
head(predict(cal, standards$signal[301:368]))
```

---

cellpca — *Robust PCA by Casewise and Cellwise Weighting (cellPCA)*

Description

Robust PCA that handles outlying observations (casewise outliers), outlying cells (cellwise outliers) and missing values by minimizing a single objective function: the cellPCA method of Centofanti, Hubert and Rousseeuw. Each cell and each observation gets a weight between 0 and 1 that reflects its outlyingness, and regular cells and observations are not downweighted, which makes cellPCA more efficient than macropca(). The iterations are computed in C++. The results have the same form as those of robpca(), so that predict(), plot_outlier_map() and plot_cell_map() work the same way.

Usage

```
cellpca(
  x,
  k = NULL,
  alpha = 0.5,
  kmax = 10,
  var_explained = 0.8,
```

```
  scale = FALSE,
  ndir = 250,
  maxiter = 1000,
  tol = 1e-06,
  max_col_frac = 0.5,
  od_cutoff = c("simulated", "chisq")
)
```

Arguments

`x`	A numeric matrix or data frame, with one observation per row. Missing values are allowed.
`k`	The number of principal components. If `NULL` (default), it is chosen from `var_explained` and `kmax`.
`alpha`	The robustness parameter, between 0.5 and 1: the fraction of observations used in the fit. Default is 0.5.
`kmax`	The maximum number of components. Default is 10.
`var_explained`	The fraction of variance used to choose `k` when it is not given. Default is 0.8.
`scale`	A logical: scale the variables by their robust scale (`FALSE`, default: they are only centered, so that intense emission lines are not outweighed by noise and continuum channels).
`ndir`	The number of random directions of the outlyingness, or `"all"`. Default is 250.
`maxiter`, `tol`	The maximum number of iterations, and the tolerance on the relative change of the fit. Defaults are 1000 and 1e-6.
`max_col_frac`	The largest fraction of cells of a variable that can get a zero weight. Default is 0.5.
`od_cutoff`	How to compute the cut-off of `od`: `"simulated"` (default) or `"chisq"`. See Details.

Details

cellPCA approximates the data by a fit $\hat{X} = 1_n \mu^T + UV^T$ of rank `k` that minimizes

$$\hat{\sigma}_2^2 \frac{1}{n} \sum_{i=1}^{n} m_i \rho_2 \left(\frac{t_i}{\hat{\sigma}_2} \right), \quad t_i = \sqrt{\frac{1}{m_i} \sum_{j=1}^{p} m_{ij} \hat{\sigma}_{1,j}^2 \rho_1 \left(\frac{x_{ij} - \hat{x}_{ij}}{\hat{\sigma}_{1,j}} \right)}$$

where m_{ij} is 0 for a missing cell and 1 otherwise and m_i is the number of observed cells of observation i. The bounded function ρ_1 limits the effect of the outlying cells (the residuals of variable j divided by their scale $\hat{\sigma}_{1,j}$), and ρ_2 that of the outlying observations (the casewise total deviations t_i divided by their scale $\hat{\sigma}_2$). Both are hyperbolic tangent functions (Hampel et al., 1981): ρ_1 with $b = 1.5$ and $c = 4$, and ρ_2 with b and c the 0.70 and 0.99 quantiles of the standardized total deviations of simulated Gaussian residuals. Their weights $\psi(z)/z$ are 1 in the central region and 0 beyond c.

The algorithm follows the reference code of the authors:

1. **Initial fit.** A `macropca()` fit (with `alpha`, `kmax`, `var_explained`, `scale` and `ndir`), which also chooses `k` when it is `NULL`, with its scores computed from the data whose flagged and missing cells are imputed by the fit.
2. **Scales.** $\hat{\sigma}_{1,j}$ is the M-scale of the residuals of variable j, and $\hat{\sigma}_2$ that of the total deviations (with $\rho_{1.5,4}$: 50% breakdown, consistent at the normal distribution). They are kept fixed.
3. **Iteratively reweighted least squares.** The loadings (one variable at a time, with the casewise and cellwise weights) and the scores (one observation at a time, with the cellwise weights) are updated by weighted least squares, the loadings are orthonormalized, the center is updated, and so are the weights, until the fit UV^T changes by less than `tol` (relative), at most `maxiter` times. Each iteration decreases the objective. If more than `max_col_frac` of the cells of a variable get a zero weight, the previous iteration is kept.
4. **Principal directions.** The center and the eigenvectors within the subspace are estimated by the deterministic MCD of the scores of the observations with a non-zero casewise weight (the exact MCD when `k = 1`), and the sign of each loading vector is set so that its largest element is positive.

The residuals of each variable are then standardized by their M-scale, and the cells beyond $\sqrt{\chi^2_{1,0.99}}$ are flagged. As in the enhanced outlier map of the paper, `od` is the norm of the standardized residuals of each observation, and `sd` the score distance of its projection on the subspace (of its robust scores when it has missing cells). The cut-off of `sd` is $\sqrt{\chi^2_{k,0.99}}$. That of `od` is, by default (`od_cutoff = "simulated"`), the 0.99 quantile of the `od` of a cellPCA fit to clean data simulated from the fit, as in the reference code (this needs a second fit); `"chisq"` uses $\sqrt{\chi^2_{p,0.99}}$ instead.

New observations, with missing or outlying cells, are projected by the robust regression of their observed cells on the loadings, also in C++.

Value

An object of class `specproc_cellpca` (inheriting from `specproc_robpca`), with the components described in `robpca()` and `macropca()`, and:

- `cell_weights`: the cellwise weights (0 for missing cells).
- `case_weights`: the casewise weights.
- `deviation`: the standardized casewise total deviations.
- `fitted`: the fitted values $\hat{X}$.
- `imputed`: the data in which the outlying cells are moved toward the fit in proportion to their weights, and the missing cells are replaced by the fit, so that the projection of each observation on the subspace is its fitted value.
- `sigma1`, `sigma2`: the scales of the cellwise residuals and of the casewise total deviations; `resid_scale`: the scales of the final residuals.
- `objective`: the objective at each iteration; `iterations` and `converged`.

Author(s)

Christian L. Goueguel

References

- Centofanti, F., Hubert, M., Rousseeuw, P.J. (2026). Robust principal components by casewise and cellwise weighting. Technometrics. doi:10.1080/00401706.2026.2643216
- Hampel, F.R., Rousseeuw, P.J., Ronchetti, E. (1981). The change-of- variance curve and optimal redescending M-estimators. Journal of the American Statistical Association, 76(375):643-648.
- Hubert, M., Rousseeuw, P.J., Van den Bossche, W. (2019). MacroPCA: an all-in-one PCA method allowing for missing values as well as cellwise and rowwise outliers. Technometrics, 61(4):459-473.

See Also

`macropca()`, `robpca()`, `step_cellpca()`, `plot_outlier_map()`, `plot_cell_map()`

Examples

```
data(forageLIBS)
# the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]
set.seed(1)
fit <- cellpca(spectra, k = 2, od_cutoff = "chisq")
fit
plot_outlier_map(fit, shade = TRUE, relative = TRUE, log = TRUE, colour_by = "distance")
plot_cell_map(fit, order = "od")
```

---

center *Data Centering*

Description

Function to perform mean-centering or median-centering on a numeric matrix or data frame.

Usage

```
center(x, method = "mean", drop.na = FALSE)
```

Arguments

`x`	A numeric matrix or data frame.
`method`	A character string specifying the centering method, either "mean" or "median".
`drop.na`	A logical value indicating whether to ignore missing values (NA) when computing the column means or medians. Missing values are kept in the output.

Details

Mean-centering calculates the mean of each column and subtracts this from the column. Median-centering is very similar to mean-centering except that the reference point is the median of each column rather than the mean.

Value

A numeric matrix with the same dimensions as the input, with columns centered according to the specified method. The column centers are stored in the `"center"` attribute.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
center(spectra)[1:3, 1:4]
center(spectra, method = "median")[1:3, 1:4]
```

---

cf_libs *Calibration-Free LIBS Quantification*

Description

Estimates the elemental composition of a sample from the intensities of its emission lines, without calibration standards, by the calibration-free LIBS (CF-LIBS) method of Ciucci et al. (1999).

Usage

```
cf_libs(
  lines,
  electron_density = NULL,
  temperature = NULL,
  method = NULL,
  partition = NULL,
  ionization_energy = NULL,
  reference = NULL,
  units = "energy",
  timeout = 120
)
```

Arguments

`lines` A data frame with one row per line and columns `species` (such as `"Ca I"` or `"Ca II"`; stages I and II), `intensity` (integrated line intensity), `wavelength` (nm), `Aki` (s^{-1}), `gk` and `Ek` (eV), such as the lines of `nist_lines()` with measured intensities.

`electron_density` The electron density, in cm^{-3}. Needed for the elements observed in one ionization stage only; without it, only the observed stage of these elements is counted, with a warning.

`temperature` The plasma temperature, in K. By default, it is estimated from the lines.

`method` `"boltzmann"` or `"saha-boltzmann"` (see details). By default, `"saha-boltzmann"` when `electron_density` is given, `"boltzmann"` otherwise.

`partition` The partition functions: `NULL` (default) to compute them from the levels of the NIST Atomic Spectra Database (see `nist_levels()`; needs an internet connection), a data frame of levels with columns `species`, `g` and `energy` (eV), or a function of the species and the temperature returning the partition functions.

`ionization_energy` The ionization energies of the neutral atoms, in eV, named by element (`c(Ca = 6.113)`) or neutral species (`c("Ca I" = 6.113)`): for all elements with the Saha-Boltzmann method, and otherwise for the elements observed in one stage only and to truncate the partition functions computed from levels. By default, they are taken from the NIST database (see `nist_ionization_energy()`).

`reference` An optional known mass fraction of one element, named by the element, such as `c(Ca = 0.35)`, used instead of closure.

`units` The units of `intensity`: `"energy"` (default) or `"photons"`, as in `boltzmann()`.

`timeout` The timeout of the NIST downloads, in seconds. Default is 120.

Details

In a plasma in local thermodynamic equilibrium (LTE), optically thin and of the same composition as the sample, the intensity of a line of species s satisfies

$$\ln \frac{I\lambda}{g_k A_{ki}} = -\frac{E_k}{k_B T} + \ln \frac{F C_s}{U_s(T)}$$

where C_s is the concentration of the species, $U_s(T)$ its partition function and F an experimental factor common to all lines. The Boltzmann plots of all species are therefore parallel lines:

1. The temperature is estimated from their common slope, by a least-squares fit with one intercept per plot (unless `temperature` is given).
2. The relative number density of each species follows from the intercept q_s of its plot, $FC_s = U_s(T)e^{q_s}$.
3. The density of an ionization stage that is not observed is computed with the Saha equation, from the electron density (see `electron_density()`) and the ionization energy.
4. The factor F is eliminated by closure (the concentrations of the elements sum to 1), or, with `reference`, by the known mass fraction of one element (internal reference), which is useful when some elements of the sample are not measured.

Two methods place the lines on the plots:

- "boltzmann" (Ciucci et al., 1999): one Boltzmann plot per species, with its own intercept. The temperature rests on the range of upper-level energies within each species, often narrow (1 to 2 eV), so it can be imprecise.
- "saha-boltzmann" (the default when electron_density is given): one Saha-Boltzmann plot per element, on which the lines of the ion are placed with the Saha equation, as in saha_boltzmann(). The energy range is extended by the ionization energy, which gives a much more precise temperature, and the ionization balance of each element follows from the Saha equation. It needs the ionization energies of all elements.

At least one plot needs two lines or more of different upper-level energies.

When the partition functions are computed from energy levels, the levels of the neutral atoms above their ionization energy, lowered by the Debye-Hückel correction when electron_density is given, are left out (see partition_function()). This needs the ionization energies of all elements observed as neutral atoms.

The mass fractions are computed from the atomic fractions with the standard atomic weights. The result is only as good as its assumptions: all major elements must be measured (with closure), the lines must be optically thin (see self_absorption()), the intensities corrected for the spectral response of the instrument, and the atomic data accurate (prefer lines of accuracy B or better in nist_lines()).

Value

An object of class specproc_cflibs, a list with

- composition: a tibble with, for each element, the atomic_fraction and mass_fraction, and the stages observed (a stage computed by the Saha equation is noted "Saha");
- species: a tibble with, for each species, the number of lines, the intercept of its own Boltzmann plot, its partition function, its relative number density and whether it was observed;
- temperature, temperature_se (K) and electron_density;
- points: the points of the plots (for the Saha-Boltzmann method, the abscissa of ions includes the ionization energy and the ordinate the Saha correction), intercept, the intercept of each plot, and fit, the lm fit;
- method and the number of iterations of the Saha-Boltzmann fit.

Use plot_boltzmann() to draw the Boltzmann plots.

Author(s)

Christian L. Goueguel

References

- Ciucci, A., Corsi, M., Palleschi, V., Rastelli, S., Salvetti, A., Tognoni, E. (1999). New procedure for quantitative elemental analysis by laser-induced plasma spectroscopy. Applied Spectroscopy, 53(8):960-964.
- Tognoni, E., Cristoforetti, G., Legnaioli, S., Palleschi, V. (2010). Calibration-free laser-induced breakdown spectroscopy: state of the art. Spectrochimica Acta Part B, 65(1):1-14.

See Also

saha_boltzmann(), electron_density(), nist_lines(), nist_levels(), plot_boltzmann()

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
# calcium and magnesium lines, with their atomic data from the NIST database
atomic <- data.frame(
  species = c("Ca I", "Ca I", "Ca I", "Ca II", "Ca II", "Mg I", "Mg I", "Mg I", "Mg II", "Mg II"),
   wavelength = c(428.301, 430.253, 445.478, 315.887, 317.933,
                  516.732, 517.268, 518.360, 279.078, 279.800),
  Aki = c(4.34e7, 1.36e8, 8.70e7, 3.10e8, 3.60e8, 1.13e7, 3.37e7, 5.61e7, 4.01e8, 4.79e8),
   gk = c(5, 5, 7, 4, 6, 3, 3, 3, 4, 6),
   Ek = c(4.780, 4.780, 4.681, 7.047, 7.050, 5.108, 5.108, 5.108, 8.864, 8.864)
)
# partition functions, interpolated from a table (see partition_function())
partition <- function(species, temperature) {
  grid <- c(6000, 8000, 10000, 12000)
  table <- rbind(`Ca I` = c(1.407, 2.401, 4.499, 8.356), `Ca II` = c(2.389, 2.917, 3.560, 4.262),
             `Mg I` = c(1.049, 1.207, 1.597, 2.463), `Mg II` = c(2.001, 2.010, 2.036, 2.087))
  vapply(species, function(s) exp(stats::approx(grid, log(table[s, ]), xout = temperature)$y),
          numeric(1))
}
fit <- cf_libs(line_intensities(mean_spectrum, atomic, baseline = TRUE),
               electron_density = 1.9e17, partition = partition,
               ionization_energy = c(Ca = 6.113, Mg = 7.646))
fit
plot_boltzmann(fit)
```

---

contributions *Q and T-squared Contributions of a PCA Model*

Description

Computes how much each variable (wavelength) contributes to the Q residual or to Hotelling's T^2 of each sample, keeping the sign of the deviation, to find the variables that make a sample an outlier. Relative contributions, computed against reference samples, show what differs between a sample and the reference ones.

Usage

```
contributions(
  model,
  k = NULL,
  statistic = c("q", "t2"),
  data = NULL,
  samples = NULL,
```

```
  reference = NULL
)
```

Arguments

`model`	A `stats::prcomp()` fit, or an object returned by `robpca()`, `rospca()` or `macropca()`.
`k`	The number of components. Required for a `stats::prcomp()` fit; for a robust fit, at most its number of components (the default).
`statistic`	`"q"` (default) for the Q contributions, or `"t2"` for the T^2 contributions.
`data`	The samples whose contributions are computed: a numeric matrix or data frame with the variables of the model (other columns are ignored when the columns are named), preprocessed like the data the model was fitted on (for example, with `center()` if the model was fitted on `center(spectra)`). Default is the calibration data of a `stats::prcomp()` fit (which must keep all its components), or the imputed data of a `macropca()` fit; it is required for `robpca()` and `rospca()` fits, which do not keep their data.
`samples`	Optional indices or names of the rows of `data` to return. Default is all.
`reference`	Optional indices or names of rows of `data` whose mean contribution is subtracted (relative contributions), or `"regular"` for all the regular samples.

Details

For a sample x (centered and scaled as in the model), with scores $t = xP$ on the first k components (loadings P, variances of the scores λ):

- the **Q contributions** are the residuals $e = x - tP^T$, whose sum of squares is Q (the squared orthogonal distance of robust fits);
- the T^2 **contributions** are $t\Lambda^{-1/2}P^T$, the scaled scores projected back onto the variables, whose sum of squares is T^2 (the squared score distance of robust fits). For `rospca()` fits, whose loadings are not exactly orthogonal, the sum of squares is close to, but not exactly, T^2.

Q contributions show which variables the model does not describe for a sample: large contributions grouped on a few emission lines point to a systematic deviation (a contamination, a matrix effect, a saturated or shifted line), small ones spread over all channels to random noise. T^2 contributions show which variables place a sample far from the center in the score space.

Relative contributions. Normally, contributions are relative to the model (its center, for T^2). With `reference`, the mean contribution of the reference samples is subtracted from the contribution of each sample, which shows what differs between them: for example, whether two samples have large Q residuals for the same reason, or what moves a sample from the regular samples to where it lies in the score space. `reference = "regular"` uses all the regular samples: of the robust fit, or at the highest confidence level of `q_residuals()` for a `stats::prcomp()` fit.

The contributions are in the units of the preprocessed data of the model: centered (and scaled, if the model scaled the variables).

Value

A tibble of class `specproc_contributions` with one row per sample: `sample` (the row name, or number) and one column per variable. The attribute `total` holds the statistic of each sample (Q or T^2, before any reference is subtracted). Draw it with `plot_contributions()`.

Author(s)

Christian L. Goueguel

References

- Westerhuis, J.A., Gurden, S.P., Smilde, A.K. (2000). Generalized contribution plots in multivariate statistical process monitoring. Chemometrics and Intelligent Laboratory Systems, 51(1):95-114.

See Also

plot_contributions(), q_residuals(), plot_outlier_map()

Examples

```
spectra_id <- names(forageLIBS)[1:2]
minerals <- names(forageLIBS)[3:14]
spectra <- forageLIBS |> dplyr::select(-dplyr::all_of(c(spectra_id, minerals)))
set.seed(1)
fit <- robpca(spectra, k = 3)

# Q contributions of two outlying samples, relative to the regular ones
q <- contributions(fit, data = spectra, samples = c(49, 127), reference = "regular")
q[, 1:5]
```

---

correct_self_absorption

Self-Absorption Correction with an Internal Reference Line

Description

Corrects the intensities of emission lines for self-absorption by the internal reference method of Sun and Yu (2009): for each species, the self-absorption coefficient of every line is measured against a line of the same species that is little self-absorbed.

Usage

```
correct_self_absorption(
  lines,
  temperature = NULL,
  electron_density = NULL,
  ionization_energy = NULL,
  reference = NULL,
  units = "energy",
  tol = 1e-08,
  timeout = 120
)
```

Arguments

`lines` A data frame with one row per line and columns `species`, `intensity`, `wavelength` (nm), `Aki` (s^{-1}), `gk`, `Ek` (upper-level energy, eV) and, for the default reference, `Ei` (lower-level energy, eV).

`temperature` The plasma temperature, in K. Without it, it is estimated from the corrected lines, which needs `electron_density`.

`electron_density` The electron density, in cm^{-3}, to estimate the temperature (see details).

`ionization_energy` The ionization energies of the neutral atoms, in eV, named by element, to estimate the temperature. By default, they are taken from the NIST database (see `nist_ionization_energy()`).

`reference` The reference lines: row numbers of `lines`, one per species, or `NULL` (default) to choose them by optical depth.

`units` The units of `intensity`: `"energy"` (default) or `"photons"`, as in `boltzmann()`.

`tol` The relative tolerance on the estimated temperature. Default is 1e-8.

`timeout` The timeout of the NIST downloads, in seconds.

Details

In an optically thin plasma in LTE, the intensity ratio of two lines of the same species depends only on their atomic data and on the temperature. With a reference line r assumed optically thin, the self-absorption coefficient of a line j is

$$SA_j = \frac{I_j}{I_r}\frac{g_r A_r \lambda_j}{g_j A_j \lambda_r}\exp\left(\frac{E_j - E_r}{k_B T}\right)$$

(for intensities in energy units; without the wavelengths for photon counts), where g, A and E are the statistical weight, transition probability and energy of the upper level. The corrected intensity is I_j/SA_j.

Self-absorption grows with the optical depth of the line, roughly proportional to $g_k A_{ki} \lambda^4 \exp(-E_i/k_B T)$, where E_i is the energy of the lower level: resonance lines ($E_i = 0$) and strong lines are the most absorbed. By default, the reference of each species is the line of smallest optical depth, which needs the lower-level energies (column `Ei`, as returned by `nist_lines()`); otherwise, give the reference lines with `reference`. The reference line must be well measured: its noise propagates to every coefficient.

The corrected lines of a species lie, by construction, on the Boltzmann plot of its reference line at the temperature used, so they cannot give the temperature by themselves. Give `temperature`, measured on optically thin lines (for example with `saha_boltzmann()` or `cf_libs()`), or give `electron_density`: the temperature is then the one at which the corrected lines of the neutral atom and of the ion of each element lie on a common Saha-Boltzmann plot (as in `cf_libs()`), which needs lines of both stages for at least one element. This is the iterative scheme of Sun and Yu (2009), solved directly for the temperature.

Coefficients above 1 mean that the line is stronger than expected from the reference, because of noise, errors of the atomic data, or a self-absorbed reference; they are kept, with a warning.

Value

lines, as a tibble, with the corrected intensity, the measured_intensity, the self-absorption coefficient SA and a logical reference, ready for boltzmann() or cf_libs(). The temperature used is in the attribute temperature.

Author(s)

Christian L. Goueguel

References

- Sun, L., Yu, H. (2009). Correction of self-absorption effect in calibration-free laser-induced breakdown spectroscopy by an internal reference method. Talanta, 79(2):388-395.

See Also

self_absorption() for the coefficient from line widths, cf_libs(), line_intensities()

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
# potassium lines: the resonance doublet at 766.49 and 769.90 nm ends on
# the ground state (Ei = 0) and is strongly reabsorbed
k_lines <- data.frame(
  species = "K I", wavelength = c(404.414, 404.721, 691.108, 693.877, 766.490, 769.896),
  Aki = c(1.150e6, 1.070e6, 2.500e6, 4.956e6, 3.779e7, 3.734e7),
  gk = c(4, 2, 2, 2, 4, 2),
  Ek = c(3.065, 3.063, 3.403, 3.403, 1.617, 1.610),
  Ei = c(0, 0, 1.610, 1.617, 0, 0)
)
corrected <- correct_self_absorption(line_intensities(mean_spectrum, k_lines), temperature = 8000)
corrected[c("wavelength", "measured_intensity", "SA", "reference")]
```

---

correlation — *Correlation Coefficients: Pearson, Spearman, Kendall, Chatterjee, and Biweight Midcorrelation*

Description

Computes various correlation coefficients between one or several response variables and each of the remaining variables in a given data frame or tibble. The available correlation methods are Pearson's product-moment correlation (parametric), Spearman's rank correlation, Kendall's tau correlation (non-parametric), Chatterjee's new correlation coefficient, and the biweight midcorrelation (a robust correlation measure).

Usage

```
correlation(
  x,
  var,
  method = "pearson",
  plot = FALSE,
  color = c("#1f4e79", "#c0392b"),
  interactive = FALSE,
  top = NULL,
  cluster = FALSE
)
```

Arguments

`x`	A data frame or tibble containing the variables of interest.
`var`	The response variable(s): one or several columns, given unquoted, as strings, or with a tidyselect helper, such as `c(K, Ca)` or `dplyr::all_of(minerals)`. The other columns are the variables correlated with each response.
`method`	A character string indicating the correlation method to use. Allowed values are "pearson", "spearman", "kendall", "chatterjee", or "bicor" (for biweight midcorrelation). The default is "pearson".
`plot`	A logical value indicating whether to produce a visualization of the correlations. Default is FALSE (no plot).
`color`	The colors of the plot: one color, or two for positive and negative correlations (the two ends of the color scale of a heatmap). Default is `c("#1f4e79", "#c0392b")`.
`interactive`	A logical value indicating whether to create an interactive plot using plotly. Default is FALSE (static ggplot2 plot).
`top`	For a bar chart (or a heatmap of variables that are not wavelengths), the number of variables with the largest absolute correlations to show. Default is `NULL` (all).
`cluster`	With several responses, a logical: order the rows of the heatmap by a hierarchical clustering of the responses on their correlations, and draw its dendrogram (`FALSE`, default). Ignored with a single response.

Details

The Pearson correlation coefficient measures the linear relationship between two continuous variables and is suitable when the data follows a bivariate normal distribution. The Spearman and Kendall correlations are non-parametric measures of monotonic association, making them suitable for non-linear relationships and when the data deviates from normality. The Chatterjee correlation coefficient $\xi_n(X, Y)$ measures how much the response `var` is a (possibly non-monotonic) function of each other variable; it lies between 0 and 1 (asymptotically) and is not symmetric. The biweight midcorrelation is a robust correlation measure that downweights the influence of outliers and is recommended when the data contains extreme values or deviates significantly from normality.

Missing values are handled pairwise: each coefficient uses the observations where both the response and the other variable are available. With several responses, each one uses its own observations, so a response with many missing values does not reduce the data of the others.

Value

- If `plot = FALSE`, a tibble with columns `variable`, `.correlation` and `method`, sorted by decreasing correlation. With several responses, it starts with a column `outcome`, and is sorted within each response.
- If `plot = TRUE`, a list containing the tibble (`correlation`) and a `ggplot2` object (`plot`; a patchwork object with `cluster = TRUE`), and with `cluster = TRUE` the clustering (`clustering`, an `stats::hclust()` object).
- If `plot = TRUE` and `interactive = TRUE`, a `plotly` object.

Plots

When the variables are named by wavelengths (the channels of spectra), the plot is a correlation spectrum: the correlation at each wavelength, with the thresholds of significance at the 5% level (for the Pearson, Spearman and biweight coefficients, from the t distribution; not corrected for multiple testing). Otherwise, it is a chart of the correlation of each variable, sorted, colored by sign and labeled with its value. The interactive versions show the variable and its correlation on hover; the correlation spectrum uses WebGL, so it stays fast with thousands of channels.

With several responses, the plot is a heatmap: one row per response, in the order given, and one column per wavelength (each tile as wide as the spacing of the channels, with gaps between detectors) or per variable. The color scale is fixed, from -1 to 1 (0 to 1 for Chatterjee's coefficient), so that a color means the same correlation in every row and in every plot.

With `cluster = TRUE`, the responses are clustered on their correlation profiles, the correlations with all the variables (as a function of wavelength, for spectra): the distance between two responses is $1 - r$, where r is the Pearson correlation between their profiles, and the clusters are merged by average linkage. Responses whose correlations rise and fall at the same wavelengths, such as elements with lines in the same regions or that vary together in the samples, are then adjacent, and the dendrogram on the left shows how similar they are. The static plot then needs the patchwork package; the interactive one is only reordered.

Author(s)

Christian L. Goueguel

References

- Chatterjee, S. (2021). A new coefficient of correlation. Journal of the American Statistical Association, 116(536):2009-2022.
- Wilcox, R. (2012). Introduction to robust estimation and hypothesis testing (3rd ed.). Academic Press. (ISBN 978-0123869838).

Examples

```
# LIBS spectra of forage samples and their mineral contents
data(forageLIBS)
spectra_id <- names(forageLIBS)[1:2]
minerals <- names(forageLIBS)[3:14]

# potassium against the other mineral contents
```

```
correlation(forageLIBS[minerals], K)
correlation(forageLIBS[minerals], "K", method = "bicor")
correlation(forageLIBS[minerals], K, plot = TRUE)$plot

spectra <- forageLIBS[setdiff(names(forageLIBS), spectra_id)]

# a correlation spectrum: potassium against every channel
k <- correlation(spectra[setdiff(names(spectra), setdiff(minerals, "K"))], K, plot = TRUE)
k$plot


# a heatmap: every mineral against every channel
all_minerals <- correlation(spectra, dplyr::all_of(minerals), plot = TRUE)
all_minerals$plot

# the same, with the minerals clustered by their correlation profiles
if (requireNamespace("patchwork", quietly = TRUE)) {
  correlation(spectra, dplyr::all_of(minerals), plot = TRUE, cluster = TRUE)$plot
}
```

`direct_orthogonal` *Direct Orthogonalization*

Description

This function implements the Direct Orthogonalization (DO) algorithm, as proposed by Andersson (1999), to filter out variation from predictor variables, **X**, that is orthogonal to the response variable(s), **Y**.

Usage

```
direct_orthogonal(x, y, ncomp = 2, center = TRUE, scale = FALSE)
```

Arguments

`x`	A matrix or data frame of the predictor variables
`y`	A vector, matrix or data frame of the response variable(s)
`ncomp`	An integer specifying the number of principal components to retain for orthogonal processing. Default is 2. Values larger than the rank of the orthogonalized matrix are reduced accordingly.
`center`	A logical value specifying whether to center the data. Default is `TRUE`.
`scale`	A logical value specifying whether to scale the data. Default is `FALSE`.

Details

Contrary to the Orthogonal Signal Correction (OSC) algorithm, Wold *et al.* (1998), which uses inverse Partial Least Squares (PLS) regression to filter out the orthogonal signal, DO filters out the orthogonal signal directly by orthogonalization of the **X** matrix:

$$\mathbf{Z} = \mathbf{X} - \mathbf{Y}(\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T\mathbf{X}$$

Principal Components Analysis (PCA) is performed on **Z** to obtain the loadings **P**, and the corrected matrix is $\mathbf{X}_{DO} = \mathbf{X} - \mathbf{XPP}^T$. Direct Orthogonalization is typically simpler and faster than OSC.

To correct new data, preprocess it with the returned `center` and `scale` vectors and apply $\mathbf{X}_{new} - \mathbf{X}_{new}\mathbf{PP}^T$.

Value

An object of class `specproc_direct_orthogonal` (a list), which predict() applies to new spectra, with the following components:

- `correction`: The corrected matrix.
- `loading`: The loadings matrix **P**.
- `score`: The scores matrix **XP**.
- `center`, `scale`: The column centers and scales applied to `x`.

Author(s)

Christian L. Goueguel

References

- Andersson, C.A., (1999). Direct orthogonalization. Chemometrics Intell. Lab. Syst., 47(1):51-63
- Pierna, J.A.F., Massart, D.L., de Noord, O.E., Ricoux, P., (2001). Direct orthogonalization: some case studies. Chemometrics Intell. Lab. Syst., 55(1-2):101-108
- Wold, S., Antti, H., Lindgren, F., Ohman, J. (1998). Orthogonal signal correction of near-infrared spectra. Chemometrics Intell. Lab. Syst., 44(1):175-185.

See Also

`predict.specproc_filter()` to correct new spectra with the fitted filter, `step_direct_orthogonal()` to use it in a tidymodels recipe. Before specProc 0.4.0, `nas()` returned the same correction; it now computes the net analyte signal and figures of merit.

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
res <- direct_orthogonal(spectra, forageLIBS$K, ncomp = 2)
dim(res$correction)
```

`direct_osc` *Direct Orthogonal Signal Correction*

Description

This function implements the Direct Orthogonal Signal Correction (DOSC) algorithm, as proposed by Westerhuis *et al.* (2001), to remove systematic variation from predictor variables, $\mathbf{X}$, that is orthogonal to the response variable(s), $\mathbf{Y}$.

Usage

```
direct_osc(x, y, ncomp = 10, center = TRUE, scale = FALSE, tol = 0.001)
```

Arguments

`x`	A matrix or data frame of the predictor variables
`y`	A vector, matrix or data frame of the response variable(s)
`ncomp`	An integer specifying the number of orthogonal components to remove. Default is 10; it is reduced if larger than the rank of the orthogonalized matrix.
`center`	A logical value specifying whether to center the data. Default is `TRUE`.
`scale`	A logical value specifying whether to scale the data. Default is `FALSE`.
`tol`	A numeric value giving the relative tolerance used to compute the pseudo-inverse of $\mathbf{X}$; singular values smaller than `tol` times the largest one are discarded, which regularizes the weights. Default is 1e-3.

Details

Different from the Orthogonal Signal Correction (OSC) algorithm, Wold *et al.* (1998), the DOSC algorithm is non-iterative:

1. $\mathbf{Y}$ is projected onto the column space of $\mathbf{X}$: $\hat{\mathbf{Y}} = \mathbf{X}\mathbf{X}^{+}\mathbf{Y}$.
2. $\mathbf{X}$ is orthogonalized with respect to $\hat{\mathbf{Y}}$: $\mathbf{Z} = \mathbf{X} - \hat{\mathbf{Y}}\hat{\mathbf{Y}}^{+}\mathbf{X}$.
3. PCA of $\mathbf{Z}$ gives the orthogonal scores $\mathbf{T}$.
4. The weights $\mathbf{W} = \mathbf{X}^{+}\mathbf{T}$ express the scores as a linear combination of $\mathbf{X}$, the loadings are $\mathbf{P} = \mathbf{X}^{T}\mathbf{T}(\mathbf{T}^{T}\mathbf{T})^{-1}$, and the corrected matrix is $\mathbf{X} - \mathbf{T}\mathbf{P}^{T}$.

New data are corrected with $\mathbf{X}_{new} - \mathbf{X}_{new}\mathbf{W}\mathbf{P}^{T}$ after applying the returned `center` and `scale`.

Value

An object of class `specproc_direct_osc` (a list), which predict() applies to new spectra, with the following components:

- `correction`: The corrected matrix.
- `loading`: The loadings matrix $\mathbf{P}$.

- score: The scores matrix **T**.
- weight: The weights matrix **W**.
- center, scale: The column centers and scales applied to x.

Author(s)

Christian L. Goueguel

References

- Westerhuis, J.A., Jong, S.D., Smilde, A.K., (2001). Direct orthogonal signal correction. Chemometrics Intell. Lab. Syst., 56(1):13-25
- Wold, S., Antti, H., Lindgren, F., Ohman, J. (1998). Orthogonal signal correction of near-infrared spectra. Chemometrics Intell. Lab. Syst., 44(1):175-185.

See Also

predict.specproc_filter() to correct new spectra with the fitted filter, and step_direct_osc() to use it in a tidymodels recipe.

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
res <- direct_osc(spectra, forageLIBS$K, ncomp = 2)
dim(res$correction)
```

---

directional_outlyingness

Directional Outlyingness for Skewed Distribution

Description

This function computes the directional outlyingness of a numeric vector, as proposed by Rousseeuw *et al.* (2018), which is a measure of outlyingness of data point that takes the skewness of the underlying distribution into account.

Usage

```
directional_outlyingness(
  x,
  cutoff.quantile = 0.995,
  rmZeroes = FALSE,
  maxRatio = NULL,
  precScale = 1e-10
)
```

Arguments

`x`	A numeric vector
`cutoff.quantile`	A numeric value between 0 and 1 specifying the quantile for outlier detection (default: 0.995).
`rmZeroes`	A logical value. If `TRUE`, removes values close to zero (default: `FALSE`).
`maxRatio`	A numeric value greater than 2. If provided, constrains the ratio between positive and negative scales (default: `NULL`).
`precScale`	A numeric value specifying the precision scale for near-zero comparisons (default: 1e-10).

Details

Directional outlyingness takes the potential skewness of the underlying distribution into account, by splitting the univariate dataset in two half samples around the median. And then apply one-step M-estimator with Huber ρ-function for scaling each part.

Value

A tibble with columns:

- `data`: The original numeric values.
- `score`: The calculated outlyingness score.
- `flag`: A logical vector indicating whether each value is a potential outlier or not.

Author(s)

Christian L. Goueguel

References

- Rousseeuw, P.J., Raymaekers, J., Hubert, M., (2018). A Measure of Directional Outlyingness With Applications to Image Data and Video. Journal of Computational and Graphical Statistics, 27(2):345–359.

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
head(directional_outlyingness(fe))
```

dmodx *Distance to the Model (DModX) of a PCA Model*

Description

Computes the distance of each sample to a principal component analysis (PCA) model in the space of the variables (DModX): the residual standard deviation of the sample, normalized by that of the calibration samples, with its limits at one or more confidence levels, and Hotelling's T^2 for the influence plot.

Usage

```
dmodx(
  model,
  k,
  newdata = NULL,
  conf_level = 0.975,
  normalized = TRUE,
  df = "effective",
  t2_method = "f",
  center = TRUE,
  scale = FALSE
)
```

Arguments

model	A stats::prcomp() fit that kept all its components (the default of prcomp()), or a numeric matrix or data frame, on which a PCA is fitted with center and scale.
k	The number of components of the model.
newdata	Optional new samples (a matrix or data frame with the variables of the model), whose distances are computed instead of those of the calibration samples.
conf_level	The confidence level(s) of the limits: one or more values between 0 and 1. Default is 0.975.
normalized	A logical: DModX relative to the residual standard deviation of the calibration samples (TRUE, default), or absolute.
df	The degrees of freedom of the limit: "effective" (default) or "nominal" (see details).
t2_method	The distribution of the T^2 limit of the samples of the model: "f" (default) or "beta" (see hotelling_t2()).
center, scale	Passed to stats::prcomp() when model is data. Default is TRUE and FALSE.

Details

With the residuals e_{ij} of the K variables after k components, for N calibration samples (and $A_0 = 1$ if the data are centered),

$$s_i = \sqrt{\frac{\sum_j e_{ij}^2}{K-k}} \sqrt{\frac{N}{N-k-A_0}}, \qquad s_0 = \sqrt{\frac{\sum_i \sum_j e_{ij}^2}{(N-k-A_0)(K-k)}}$$

(the correction factor $\sqrt{N/(N-k-A_0)}$ applies to the calibration samples only). DModX is s_i/s_0 (normalized, default) or s_i. Its squared ratio follows an F distribution with ν and $(N-k-A_0)\nu$ degrees of freedom, so the limit of the normalized DModX is $\sqrt{F_{1-\alpha}}$ (times s_0 for the absolute DModX).

The nominal degrees of freedom are $\nu = K - k$ (`df = "nominal"`), as if the residuals of the variables were independent. For spectra, with far more (correlated) channels than samples, this gives a limit close to 1 that flags a large share of ordinary samples. With `df = "effective"` (default), ν is the effective number of residual dimensions, θ_1^2/θ_2 from the eigenvalues λ_a of the components left out ($\theta_j = \sum_{a>k} \lambda_a^j$, as in Box's approximation of Q), which is $K - k$ when the residuals are independent with equal variances. DModX is then consistent with the Q residuals of `q_residuals()`, of which it is a scaled square root.

Value

A tibble of class `specproc_influence`, with one row per sample: `sample`, `t2` and its limits (as for `q_residuals()`), `dmodx`, its limits at each confidence level (`dmodx_limit_97.5`, ...), and `outlier`, the type of the sample at the highest confidence level. Draw it with `plot_influence()`.

Author(s)

Christian L. Goueguel

References

- Eriksson, L., Johansson, E., Kettaneh-Wold, N., Trygg, J., Wikström, C., Wold, S. (2006). Multi- and Megavariate Data Analysis, Part I, 2nd ed. Umetrics Academy, Umeå.
- Box, G.E.P. (1954). Some theorems on quadratic forms applied in the study of analysis of variance problems, I. Annals of Mathematical Statistics, 25(2):290-302.

See Also

`q_residuals()`, `plot_influence()`, `hotelling_t2()`

Examples

```
if (rlang::is_installed("HotellingEllipse", version = "1.3.0")) {
  data(forageLIBS)
  # the 380-430 nm window (Ca II H and K lines)
  wl <- suppressWarnings(as.numeric(names(forageLIBS)))
  spectra <- forageLIBS[which(wl > 380 & wl < 430)]
  pca <- stats::prcomp(spectra)
  d <- dmodx(pca, k = 3)
```

```
  d[d$outlier != "regular", ]
  plot_influence(d, label = forageLIBS$Measurement)
}
```

`electron_density` *Electron Density from Stark Broadening*

Description

Estimates the electron density of a plasma from the Stark width of an emission line: from the Stark broadening parameters of the line (STARK-B, or a reference width), or from the width of the hydrogen Hα line at 656.28 nm.

Usage

```
electron_density(
  width,
  method = "stark",
  stark = NULL,
  wavelength = NULL,
  temperature = NULL,
  reference_width = NULL,
  reference_density = 1e+17,
  tolerance = 0.5
)
```

Arguments

`width`	The measured Stark full width at half maximum, in nm (see Details). A vector is allowed.
`method`	`"stark"` (default) or `"halpha"`.
`stark`	For `method = "stark"`: a tibble returned by `starkb_lines()`. Alternatively, give `reference_width`.
`wavelength`	For `method = "stark"` with `stark`: the wavelength of the line, in nm.
`temperature`	For `method = "stark"` with `stark`: the plasma temperature, in K.
`reference_width`	For `method = "stark"`: the electron-impact Stark width (nm) at `reference_density`, when `stark` is not given.
`reference_density`	The density of `reference_width`, in cm^{-3}. Default is 10^{17}.
`tolerance`	For `method = "stark"` with `stark`: the largest difference between `wavelength` and the tabulated wavelength, in nm (see `stark_width()`). Default is 0.5.

Details

`width` is the Stark full width at half maximum, in nm, with the instrumental and Doppler broadening removed. The line must be optically thin: self-absorption broadens lines and overestimates the density (see `self_absorption()`).

- For isolated lines (`method = "stark"`), the Stark profile is Lorentzian: use the Lorentzian width `wL` of a Voigt fit with `peak_fit()` or `multipeak_fit()`, whose Gaussian part absorbs the instrumental and Doppler broadening.
- The Stark profile of Hα (`method = "halpha"`) is not Lorentzian: use the FWHM of the whole line, for example `voigt_fwhm()` of the fitted widths, corrected for the instrumental width when it is not negligible (for Gaussian broadening, approximately $\sqrt{w^2 - w_{inst}^2}$).
- `method = "stark"`: in the impact approximation the Stark width is proportional to the electron density,
$$N_e = N_{ref} \frac{w}{w_{ref}(T)}$$
where $w_{ref}(T)$ is the electron-impact width at density N_{ref}. It is taken from `stark` (a table returned by `starkb_lines()`, interpolated at `temperature` with `stark_width()`), or given directly as `reference_width` (nm) at `reference_density`. Tables from other sources can be built with `stark_table()` or read from saved STARK-B files with `read_starkb()`. Ion broadening is neglected, which is usually justified for lines of ions and at LIBS densities.
- `method = "halpha"`: the computer-simulation fit of Gigosos, González and Cardeñoso (2003) for the Hα line,
$$w = 1.098\,(N_e/10^{17})^{0.67823}\ \mathrm{nm},$$
which includes the ion dynamics and depends only weakly on temperature.

Value

A numeric vector of electron densities, in cm^{-3}.

References

- Gigosos, M.A., González, M.Á., Cardeñoso, V. (2003). Computer simulated Balmer-alpha, -beta and -gamma Stark line profiles for non-equilibrium plasmas diagnostics. Spectrochimica Acta Part B, 58(8):1489-1504.
- Konjević, N. (1999). Plasma broadening and shifting of non-hydrogenic spectral lines: present status and applications. Physics Reports, 316(6):339-401.

See Also

`starkb_lines()`, `stark_width()`, `peak_fit()`, `voigt_fwhm()`

Examples

```
# From the H-alpha line: a Stark FWHM of 0.5 nm
electron_density(0.5, method = "halpha")

# From a line with known Stark width 0.012 nm at 1e17 cm-3
electron_density(c(0.006, 0.012, 0.024), reference_width = 0.012)
```

emsc *Extended Multiplicative Signal Correction*

Description

This function implements extended multiplicative signal correction (EMSC), as proposed by Martens and Stark (1991). EMSC extends `msc()` by modeling, in addition to the multiplicative and additive effects, a smooth polynomial baseline and, optionally, known interferent spectra.

Usage

```
emsc(
  x,
  xref = NULL,
  degree = 2,
  interferents = NULL,
  wavelength = NULL,
  robust = TRUE
)
```

Arguments

`x`	A numeric matrix or data frame, with one spectrum per row.
`xref`	An optional numeric vector giving the reference spectrum. If `NULL` (default), the median (`robust = TRUE`) or mean spectrum of `x` is used.
`degree`	A non-negative integer giving the degree of the polynomial baseline. Default is 2. With 0, only a constant offset is modeled.
`interferents`	An optional numeric vector (a single spectrum), matrix or data frame of interferent spectra, one per row, with the same number of columns as `x`.
`wavelength`	An optional numeric vector of wavelengths used to build the polynomials. If `NULL` (default), the column names of `x` are used when they are numeric, and the column indices otherwise.
`robust`	A logical value indicating whether the median (`TRUE`, default) or the mean spectrum is used as the reference.

Details

Each spectrum $\mathbf{x}_i$ is modeled as a linear combination of a reference spectrum $\mathbf{m}$, polynomials of the wavelength λ and interferent spectra $\mathbf{k}_j$:

$$\mathbf{x}_i = b_i\mathbf{m} + a_i + d_{i1}\lambda + d_{i2}\lambda^2 + \cdots + \sum_j h_{ij}\mathbf{k}_j + \mathbf{e}_i$$

The coefficients are estimated by least squares, and the corrected spectrum is

$$\mathbf{x}_i^{corr} = (\mathbf{x}_i - a_i - d_{i1}\lambda - \cdots - \sum_j h_{ij}\mathbf{k}_j)/b_i$$

With `degree = 0` and no interferents, EMSC is identical to `msc()` with `drop.offset = TRUE`.

The wavelengths are rescaled to $[-1, 1]$ before the polynomials are formed, for numerical stability. Interferent spectra describe variation that should be removed, for example the spectrum of a contaminant or the dominant directions of repeated measurements of the same samples (see `epo()`).

The reference spectrum is estimated from `x` unless `xref` is given, so new spectra should be corrected with predict(), which uses the reference and design of the calibration data.

Value

An object of class `specproc_emsc` (a list), which predict() applies to new spectra, with the following components:

- `correction`: The corrected spectra.
- `coefficients`: A tibble with the estimated coefficients of each spectrum: `slope` (b_i), `offset` (a_i), `poly1`, ... (d_{ik}) and `interferent1`, ... (h_{ij}).
- `reference`: The reference spectrum.
- `design`: The design matrix (one column per model term).

Author(s)

Christian L. Goueguel

References

- Martens, H., Stark, E. (1991). Extended multiplicative signal correction and spectral interference subtraction: new preprocessing methods for near infrared spectroscopy. Journal of Pharmaceutical and Biomedical Analysis, 9(8):625-635.
- Afseth, N.K., Kohler, A. (2012). Extended multiplicative signal correction in vibrational spectroscopy, a tutorial. Chemometrics and Intelligent Laboratory Systems, 117:92-99.

See Also

`msc()`, `step_emsc()` to use EMSC in a tidymodels recipe.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
fit <- emsc(spectra[1:300, ], degree = 2)
head(fit$coefficients)
# new spectra are corrected with the calibration reference
corrected <- predict(fit, spectra[301:368, ])
dim(corrected)
```

epo *External Parameter Orthogonalization*

Description

This function implements the External Parameter Orthogonalization (EPO) algorithm as proposed by Roger *et al.* (2003). The EPO algorithm aims to remove interferences or clutter present in the spectral matrix, effectively separating the signal of interest from unwanted perturbations.

Usage

```
epo(x, ncomp = 2, clutter = NULL)
```

Arguments

`x`	A numeric matrix, data frame or tibble.
`ncomp`	An integer specifying the number of singular vectors to orthogonalize against. Default is 2.
`clutter`	An optional numeric matrix or data frame, with the same number of columns as `x`, describing the external (clutter) variation. If `NULL` (default), the clutter directions are estimated from `x` itself, i.e. the `ncomp` dominant directions of `x` are removed.

Details

The EPO algorithm works by detecting the dominant directions (right singular vectors) of a clutter matrix $\mathbf{D}$ that describe the external variations, e.g. the differences between spectra of the same samples measured under different temperatures or moisture levels. It then projects the spectral matrix onto the subspace orthogonal to these variations, effectively removing the unwanted perturbations and extracting the signal of interest.

Let $\mathbf{X}$ be the spectral matrix. The EPO algorithm aims to split $\mathbf{X}$ into:

$$\mathbf{X} = \mathbf{XP} + \mathbf{XQ} + \mathbf{R}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are, respectively, the projection matrices of $\mathbf{X}$ onto the useful and perturbation (clutter) subspaces. $\mathbf{R}$ is the residual matrix. With $\mathbf{V}$ the first `ncomp` right singular vectors of $\mathbf{D}$, $\mathbf{Q} = \mathbf{VV}^T$ and the corrected matrix is $\mathbf{X} - \mathbf{XVV}^T$.

The singular value decomposition is computed in C++ (Eigen), and the $p \times p$ projection matrix is never formed explicitly.

Value

An object of class `specproc_epo` (a list), which predict() applies to new spectra, with four components:

- `correction`: The orthogonalized matrix, representing the signal of interest.

- `clutter`: The clutter part of `x`, $\mathbf{XVV}^T$.
- `loadings`: The singular vectors $\mathbf{V}$ used for the orthogonalization.
- `singular_values`: The corresponding singular values of the clutter matrix.

Author(s)

Christian L. Goueguel

References

- Roger, J.-M., Chauchard, F., Bellon-Maurel, V. (2003). EPO-PLS external parameter orthogonalization of PLS application to temperature-independent measurement of sugar content of intact fruits. Chemometrics and Intelligent Laboratory Systems, 66(2):191-204.

See Also

`predict.specproc_filter()` to correct new spectra with the fitted filter, and `step_epo()` to use it in a tidymodels recipe.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
# the three samples measured twice
twice <- forageLIBS$Sample[duplicated(forageLIBS$Sample)]
first <- match(twice, forageLIBS$Sample)
second <- vapply(twice, function(s) max(which(forageLIBS$Sample == s)), integer(1))
# their differences describe the variation between repeated measurements
clutter <- as.matrix(spectra[second, ]) - as.matrix(spectra[first, ])
res <- epo(spectra, ncomp = 1, clutter = clutter)
dim(res$correction)
```

`flagged_regions` *Flagged Regions of a MacroPCA or cellPCA Fit*

Description

Lists the wavelength regions where many observations have cells flagged by `macropca()` or `cellpca()`: runs of adjacent channels (on the same detector segment) whose share of flagged observations is at least `threshold`. These are the channels that persistently deviate from the PCA fit, for example emission lines affected by self-absorption, saturation or matrix effects, and candidates to exclude or down-weight. `plot_cell_map()` labels the regions with the largest share.

Usage

```
flagged_regions(
  object,
  threshold = 0.1,
  rows = NULL,
  columns = NULL,
  lines = NULL,
  tol = 0.1
)
```

Arguments

object	An object returned by macropca() or cellpca().
threshold	The share of flagged observations above which channels form a flagged region. Default is 0.1.
rows, columns	Optional indices or names of the rows and columns to show. Default is all.
lines	Optional line list returned by libs_lines(), to label the regions with the emission lines they match.
tol	The largest distance, in nm, between a region and a line it matches. Default is 0.1.

Details

Runs separated by at most two channels below the threshold are merged. A region is "higher" when at least two thirds of its flagged cells are higher than the fit, "lower" when at most one third are, and "mixed" otherwise. With lines, the lines between start - tol and end + tol are the candidates, from the nearest to the peak.

Value

A tibble with one row per region, sorted by decreasing share, and columns start, end and peak (the wavelength, or variable number, of the largest share), channels (the number of channels), share (the largest share of flagged observations), mean_share and direction. With lines, it also has the columns species and line_wavelength of the nearest line (NA when none) and candidates.

Author(s)

Christian L. Goueguel

See Also

plot_cell_map(), macropca(), libs_lines()

Examples

```
spectra_id <- forageLIBS |> dplyr::select(1:2) |> names()
minerals <- forageLIBS |> dplyr::select(3:14) |> names()
set.seed(1)
```

```
fit <- forageLIBS |>
  dplyr::select(-dplyr::all_of(c(spectra_id, minerals))) |>
  center() |>
  macropca(k = 3)

flagged_regions(fit)
```

---

forageLIBS *LIBS Spectra of Forage Samples*

Description

Laser-induced breakdown spectroscopy (LIBS) spectra of 365 forage samples (368 measurements), together with their reference contents of 12 elements measured by conventional laboratory analysis.

Usage

```
forageLIBS
```

Format

A tibble with 368 rows and 7166 columns:

Measurement Measurement identifier.

Sample Sample identifier (365 samples).

Ca, Cl, Mg, P, K, Na, S Macro-element content, in percent.

Cu, Fe, Mn, Mo, Zn Trace-element content, in mg/kg.

199.3771616, ..., 822.1848577 Emission intensity (counts) at each wavelength, in nm. The column names are the wavelengths.

Details

Each measurement is the mean of 8 laser shots, rounded to integer counts; the shot-level spectra are not included, to keep the package small. Three samples were measured twice, so there are 368 measurements of 365 samples. Keep the measurements of a sample together when splitting the data into calibration and validation sets.

The spectra cover 199.4 to 822.2 nm in 7152 channels (about 0.084 nm per channel) and are raw detector counts (no background subtraction or normalization). They are recorded by several spectrometers whose ranges are concatenated: the channels are in increasing wavelength order except near 766 nm, where two spectrometers overlap by about 0.43 nm, and there are gaps between 781.5 and 789.2 nm and between 800.7 and 813.9 nm. The detector saturates at 65535 counts: the strongest lines (Ca II 393.37/396.85 nm, K I 766.49/769.90 nm, Mg II 279.55/280.27 nm) are saturated in many shots and are nonlinear in concentration.

Reference values are missing for some elements (see the examples). Cu and Mo are available for only a minority of samples.

Source

Laboratory LIBS measurements provided by Christian L. Goueguel. The script `data-raw/forageLIBS.R` in the package source builds the data set from the shot-level export.

Examples

```
data(forageLIBS)
dim(forageLIBS)
colSums(!is.na(forageLIBS[3:14])) # available reference values per element

# K I doublet of the first three measurements
wl <- as.numeric(names(forageLIBS)[-(1:14)])
keep <- names(forageLIBS)[-(1:14)][wl > 403.5 & wl < 405.5]
plot_spectra(forageLIBS[1:3, c("Measurement", keep)], id = Measurement)
```

---

gaussian_profile *Gaussian Function*

Description

Computes the values of a Gaussian function (also known as the normal distribution) given specific parameters. The Gaussian profile is a bell-shaped curve that is symmetric around its center, making it distinct from other profiles like the Lorentzian profile, which has heavier tails.

Usage

```
gaussian_profile(x, y0, xc, wG, A)
```

Arguments

`x`	A numeric vector representing the independent variable (e.g., wavelength, frequency).
`y0`	A numeric value specifying the baseline offset.
`xc`	A numeric value representing the center of the peak.
`wG`	A numeric value specifying the full width at half maximum (FWHM).
`A`	A numeric value representing the peak area.

Details

The Gaussian function is defined as:

$$y = y_0 + \frac{A}{w_G\sqrt{\pi/(4\log(2))}} \exp\left(-\frac{4\log(2)(x-x_c)^2}{w_G^2}\right)$$

where:

- A is the peak area
- x_c is the center of the peak
- w_G is the full width at half maximum (FWHM)
- y_0 is the baseline offset

In spectroscopy, the Gaussian function is widely used to model the shape of spectral lines broadened by Doppler broadening, which arises from the thermal motion of atoms or molecules emitting or absorbing the radiation. However, in some cases, other broadening mechanisms, such as pressure broadening or natural broadening, may contribute to the overall line shape, leading to deviations from a pure Gaussian profile. In these situations, more complex line shape models, such as the Voigt profile (a convolution of Gaussian and Lorentzian profiles), may be required.

Additionally, instrumental broadening, caused by the finite resolution of the spectrometer or other experimental apparatus, is another important factor that can contribute to the observed spectral line shape, and it's often modeled using a Gaussian profile as well.

Value

A numeric vector containing the values of the Gaussian function evaluated at the provided `x` values.

Author(s)

Christian L. Goueguel

Examples

```
x <- seq(-10, 10, length.out = 100)
y <- gaussian_profile(x, y0 = 0, xc = 0, wG = 2, A = 1)
plot(x, y, type = "l", col = "red", main = "Gaussian Profile")
```

---

generalized_boxplot *Generalized Boxplot*

Description

This function implements the generalized boxplot, a robust data visualization technique designed to effectively represent skewed and heavy-tailed distributions, as proposed by Bruffaerts *et al*. (2014).

Usage

```
generalized_boxplot(
  x,
  alpha = 0.05,
  p = 0.9,
  plot = TRUE,
  xlabels.angle = 90,
  xlabels.vjust = 1,
```

```
  xlabels.hjust = 1,
  box.width = 0.5,
  notch = FALSE,
  notchwidth = 0.5,
  staplewidth = 0.5
)
```

Arguments

x	A numeric data frame or tibble.
alpha	A scalar, between 0 and 1 that specifies the desired detection rate of atypical values.
p	A scalar, between 0.5 and 1 that specifies the quantile order for estimating g and h.
plot	Logical value indicating whether to plot the boxplot or return the boxplot statistics.
xlabels.angle	A numeric value specifying the angle (in degrees) for x-axis labels (default is 90).
xlabels.vjust	A numeric value specifying the vertical justification of x-axis labels (default is 1).
xlabels.hjust	A numeric value specifying the horizontal justification of x-axis labels (default is 1).
box.width	A numeric value specifying the width of the boxplot (default is 0.5).
notch	A logical value indicating whether to display a notched boxplot (default is FALSE).
notchwidth	A numeric value specifying the width of the notch relative to the body of the boxplot (default is 0.5).
staplewidth	A numeric value specifying the width of staples at the ends of the whiskers.

Details

This method extends the adjusted boxplot method by leveraging the flexible Tukey's g-and-h parametric distribution to model the underlying data structure, particularly for asymmetric or long-tailed datasets, providing a more nuanced and informative summary of the data's central tendency, spread, and potential outliers.

Value

- If plot = TRUE, returns a ggplot2 object containing the generalized boxplot.
- If plot = FALSE, returns a list of tibbles: stats, with the whisker ends (lower, upper: the most extreme observations within the fences), quartiles, median, fences and the estimated g and h parameters of each variable, and outliers, with the potential outliers (out gives the tail).

Author(s)

Christian L. Goueguel

References

- Bruffaerts, C., Verardi, V., Vermandele, C. (2014). A generalized boxplot for skewed and heavy-tailed distributions. Statistics and Probability Letters 95(C):110–117

Examples

```
data(forageLIBS)
# mineral contents (%) of the forage samples
generalized_boxplot(forageLIBS[c("Ca", "Mg", "P", "K", "S")])
generalized_boxplot(forageLIBS[c("Ca", "Mg", "P", "K", "S")], plot = FALSE)
```

---

glsw *Generalized Least Squares Weighting*

Description

The Generalized Least Squares Weighting (GLSW) algorithm, proposed by Martens *et al.* (2003), is a technique used to mitigate the effects of external interferences in datasets. It constructs a filter to remove these interferences, allowing for more accurate data analysis and processing.

Usage

```
glsw(x1, x2, alpha = 0.01)
```

Arguments

x1	A numeric matrix, data frame or tibble representing the first set of data.
x2	A numeric matrix, data frame or tibble representing the second set of data.
alpha	A positive numeric value specifying the weighting parameter. Typical values range from 1 to 0.0001. Default is 0.01.

Details

The algorithm works by first calculating a covariance matrix from the differences between two spectral datasets that should ideally be similar. These differences are considered to be the interferences or clutter present in the data. For example, if two sets of measurements have been taken under similar conditions, the differences between them could be attributed to external factors such as sensor noise, environmental conditions, or other sources of interference. Once the covariance matrix is calculated, GLSW applies a filtering matrix to down-weight the contributions of the identified interferences or clutter. This filtering matrix is constructed using a regularization parameter, denoted as alpha (α).

With $\mathbf{X}_d$ the difference between the mean-centered datasets, $\mathbf{C} = \mathbf{X}_d^T\mathbf{X}_d = \mathbf{V}\mathbf{S}^2\mathbf{V}^T$, the filter is

$$\mathbf{G} = \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^T, \quad \mathbf{D} = \sqrt{\mathbf{S}^2/\alpha + \mathbf{I}}$$

where $\mathbf{S}^2$ holds the eigenvalues of $\mathbf{C}$.

The value of α determines how strongly the algorithm down-weights the clutter components in the data. In cases where the interferences are well-characterized and distinct from the desired signal, a small α value may be appropriate to achieve effective clutter removal. However, if the interferences are more subtle or intertwined with the desired signal, a larger α value may be preferred to avoid over-suppression of the signal itself.

Let $\mathbf{X}$ be a data matrix. The GLSW filter is applied as follows:

$$\mathbf{X}_{new} = \mathbf{X} \cdot \mathbf{G}$$

where, $\mathbf{X}_{new}$ is the filtered matrix and $\mathbf{G}$ the filtering matrix.

Value

A tibble containing the $p \times p$ filtering matrix $\mathbf{G}$.

Author(s)

Christian L. Goueguel

References

- Martens, H., Hoy, M., Wise, B.M., Bro, R., Brockhoff, P.B., (2003). Pre-whitening of data by covariance-weighted preprocessing. Journal of Chemometrics, 17(3):153-165

See Also

`step_glsw()` to use the filter in a tidymodels recipe.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
x <- spectra[wl > 760 & wl < 780]  # the K I resonance lines
# the three samples measured twice
twice <- forageLIBS$Sample[duplicated(forageLIBS$Sample)]
first <- match(twice, forageLIBS$Sample)
second <- vapply(twice, function(s) max(which(forageLIBS$Sample == s)), integer(1))
G <- glsw(x[first, ], x[second, ], alpha = 0.01)
filtered <- as.matrix(x) %*% as.matrix(G)
dim(filtered)
```

glsw_alpha — *Relative GLSW Weighting Parameter*

Description

A dials parameter for the alpha argument of step_glsw() and step_y_gradient_glsw(), on a log10 scale.

Usage

```
glsw_alpha(range = c(-4, 0), trans = scales::transform_log10())
```

Arguments

range	A two-element vector with the range of alpha, in log10 units. Default is c(-4, 0), i.e. 0.0001 to 1.
trans	A transformation object from the scales package. Default is log10.

Value

A quant_param object.

Examples

```
glsw_alpha()
dials::value_seq(glsw_alpha(), 5)
```

hotelling_t2 — *Hotelling's T-squared Statistic of Samples in an Embedding*

Description

Computes Hotelling's T^2 statistic of each sample from its scores on the first k components of an embedding (such as PCA or PLS scores), with its limits at one or more confidence levels, for all the samples together or within groups, with HotellingEllipse::ellipseParam(). Samples beyond a limit are far from the center of the data (or of their group) given its covariance: candidate outliers.

Usage

```
hotelling_t2(
  data,
  columns = NULL,
  k = 2,
  group = NULL,
  conf_level = 0.975,
  method = "f"
)
```

Arguments

`data`	The embedding: a data frame, a matrix, a `stats::prcomp()` fit, or an object of `robpca()`, `rospca()` or `macropca()`.
`columns`	The components used: a character vector of column names. By default, the first `k` embedding coordinates (columns named like `PC1`, `UMAP1`, ...), or else the first `k` numeric columns.
`k`	The number of components, when `columns` is not given. Default is 2.
`group`	An optional grouping: a column of `data` (unquoted or as a string), or a vector with one value per sample.
`conf_level`	The confidence level(s) of the limits: one or more values between 0 and 1. Default is 0.975.
`method`	The distribution of the limits: `"f"` (default) or `"beta"`.

Details

T^2 is the squared Mahalanobis distance of a sample to the mean of the scores, with their covariance. Its limit at the confidence level $1-\alpha$, for n samples, is

- `method = "f"` (default): $k(n-1)/(n-k)\,F_{1-\alpha}(k, n-k)$, the limit for a new sample, more conservative;
- `method = "beta"`: $(n-1)^2/n\,B_{1-\alpha}(k/2, (n-k-1)/2)$, the exact distribution for the samples that estimated the mean and covariance (Tracy, Young and Mason, 1992); the F limit can even exceed the largest T^2 a sample can reach, $(n-1)^2/n$, for small n.

Within groups (`group`), each group gets its own mean, covariance and limits, which tells whether a sample is typical of its own group rather than of the whole data set; each group needs more than `k + 1` samples.

The statistic uses the `k` components given, not only the two shown on a plot: a sample can lie inside the ellipse of two components and still exceed the limit in `k` dimensions. Classical estimates are themselves sensitive to outliers; with many outliers, prefer the robust distances of `robpca()`.

Value

A tibble with one row per sample: its row number `sample`, the `group` (with `group`), `t2`, then for each confidence level (in %, e.g. 97.5) its limit `limit_97.5` and whether the sample exceeds it, `outlier_97.5`, and the number of samples `n` of its group (or of the data).

Author(s)

Christian L. Goueguel

References

- Hotelling, H. (1931). The generalization of Student's ratio. The Annals of Mathematical Statistics, 2(3):360-378.
- Tracy, N.D., Young, J.C., Mason, R.L. (1992). Multivariate control charts for individual observations. Journal of Quality Technology, 24(2):88-95.
- Jackson, J.E. (1991). A User's Guide to Principal Components. Wiley, New York.

See Also

plot_embedding(), q_residuals(), robpca()

Examples

```
if (rlang::is_installed("HotellingEllipse", version = "1.3.0")) {
  data(forageLIBS)
  # the 380-430 nm window (Ca II H and K lines)
  wl <- suppressWarnings(as.numeric(names(forageLIBS)))
  spectra <- forageLIBS[which(wl > 380 & wl < 430)]
  pca <- stats::prcomp(spectra)
  t2 <- hotelling_t2(pca, k = 3)
  t2[t2$outlier_97.5, ]
  # two limits, with the exact distribution of the calibration samples
  hotelling_t2(pca, k = 3, conf_level = c(0.95, 0.99), method = "beta")
}
```

---

iqr_outliers *Interquartile Range Method*

Description

This function identifies potential outliers in a numeric vector using the interquartile range (IQR) method.

Usage

```
iqr_outliers(x, k = 1.5, skew = FALSE, drop.na = FALSE)
```

Arguments

x	A numeric vector.
k	A numeric value specifying the fence factor. k = 1.5 is the default as it strikes a balance between sensitivity to mild outliers and robustness against extreme outliers. k = 3 is more lenient and is sometimes used when a higher tolerance for outliers is desired.
skew	A logical value indicating whether to calculate the version of the fences that accounts for moderate skewness in the underlying data distribution. By default, skew = FALSE, which calculates the fences assuming a symmetric distribution. However, if skew = TRUE, the lower and upper fences incorporate the medcouple, to account for potential asymmetry in the underlying data distribution. These formulas are explicitly derived and optimized for the scenario where k = 1.5 (Hubert and Vandervieren, 2008). Consequently, if the user attempts to use a value of k other than 1.5, the code will issue a warning message indicating that the formula is only defined for k = 1.5. In such cases, the code will automatically reset k to 1.5 and proceed with the calculations using the appropriate formulas and constants.

`drop.na` A logical value indicating whether to remove missing values (`NA`) from the calculations. If `TRUE`, missing values will be removed. If `FALSE` (the default), missing values are kept in the output (with a missing flag) and ignored when computing the fences.

Details

For symmetric distributions, observations that fall outside the range defined by the lower fence (Q1 - k * IQR) and upper fence (Q3 + k * IQR) are considered as potential outliers, where Q1 and Q3 are the 25th and 75th percentiles, respectively. The fence factor can be adjusted to make the method more or less robust (often 1.5 or 3). Optionally, the method can account for moderate skewness in data distributions by incorporating the medcouple. In such a case, the lower and upper fences are expressed in terms of the medcouple, adjusting the fences asymmetrically to better accommodate skewed distributions. Note that the implemented method does not explicitly account for tail heaviness. While the medcouple can provide some robustness against heavy tails, the method may still struggle to accurately identify potential outliers in distributions with extreme kurtosis or long-tailed behavior.

Value

A tibble with two columns:

- `data`: The original numeric values.
- `flag`: A logical vector indicating whether each value is a potential outlier or not.

Author(s)

Christian L. Goueguel

References

- Everitt, B. S., and Skrondal, A. (2010). The Cambridge Dictionary of Statistics. Cambridge University Press.
- Tukey, J. (1977). Exploratory Data Analysis. Addison-Wesley
- Hubert M., and Vandervieren E. (2008). An Adjusted Boxplot for Skewed Distributions. Computational Statistics & Data Analysis, 52(12):5186-5201

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
iqr_outliers(fe)
iqr_outliers(fe, skew = TRUE)   # adjusted for the skewness of the contents
```

libs_lines *Candidate Emission Lines for LIBS Spectra*

Description

Lists the lines of the selected atoms and ions that are expected to be the strongest in a plasma at a given temperature, from the NIST Atomic Spectra Database, to help identify the emission lines of a spectrum.

Usage

```
libs_lines(
  species,
  wavelength = c(200, 900),
  temperature = 10000,
  top = 20,
  min_relative = 0,
  timeout = 120
)
```

Arguments

species	A character vector of emitters in spectroscopic notation, such as `c("Ca I", "Ca II", "Mg I")`.
wavelength	A numeric vector of length 2: the wavelength range, in nm. Default is 200 to 900 nm.
temperature	The plasma temperature, in K. Default is 10000.
top	The maximum number of lines kept per species. Default is 20.
min_relative	The minimum relative intensity of the lines kept, between 0 and 1. Default is 0.
timeout	The download timeout, in seconds. Default is 120.

Details

For each species, the lines with a transition probability are retrieved with `nist_lines()` (and cached for the R session). Their relative intensities in a plasma in local thermodynamic equilibrium at temperature T are

$$I \propto \frac{g_k A_{ki}}{\lambda} \exp\left(-\frac{E_k}{k_B T}\right)$$

normalized to the strongest line of the species in the range. The intensities are relative within each species: comparing species would need their concentrations and, between ionization stages, the Saha equation. Self-absorption, which weakens resonance lines, is ignored.

The `top` strongest lines of each species with a relative intensity of at least `min_relative` are kept. `plot_lines()` overlays them on a spectrum, and `line_finder()` does so interactively.

Value

A tibble with one row per line, sorted by wavelength, and columns `species`, `element`, `stage` (1 for neutral atoms, 2 for singly charged ions, ...), `wavelength` (nm), `relative_intensity`, `Aki`, `Ek`, `gk`, `accuracy`, `lower` and `upper`. Species without lines in the range contribute no rows.

Author(s)

Christian L. Goueguel

See Also

`plot_lines()`, `line_finder()`, `nist_lines()`

Examples

```
# needs an internet connection
lines <- try(libs_lines(c("Ca I", "Ca II"), wavelength = c(390, 450), top = 5))
if (!inherits(lines, "try-error")) lines
```

---

line_finder *Interactive Identification of Emission Lines*

Description

Launches a Shiny app to identify the emission lines of LIBS spectra. Elements are selected on a periodic table, and the strongest lines of their selected ionization stages, from the NIST Atomic Spectra Database, are overlaid on a spectrum in an interactive plotly graph, with one color per ionization stage.

Usage

```
line_finder(spectra, launch.browser = interactive(), show_table = TRUE)
```

Arguments

`spectra`	Spectra, one per row: a numeric matrix, or a data frame whose wavelength columns are named by their wavelengths (other columns are used as labels).
`launch.browser`	Passed to `shiny::runApp()`. Default is `TRUE` when R is interactive.
`show_table`	A logical: show the periodic table when the app starts (`TRUE`, default). The **Hide table** button of the Elements panel hides or shows it at any time; hiding it gives the spectrum the whole height of the window.

Details

The app offers:

- a clickable periodic table; elements without lines in the wavelength range of the spectra are grayed out once queried. It can be hidden (**Hide table**) to enlarge the spectrum, and the selected elements stay listed in the panel header;
- the ionization stages to show (I, II, III);
- the plasma temperature and the number of lines per species, which set the lines kept and the height of their markers (relative intensities in local thermodynamic equilibrium, see `libs_lines()`);
- the wavelength range, a wavelength shift to correct the calibration of the spectrometer, and the spectrum to show: the mean of all spectra, the mean of a group (from the first character or factor column of `spectra`), or a single spectrum;
- a download of the displayed lines as CSV, and buttons to save and reload the lines fetched so far, so that a session can continue offline.

Each species is downloaded from NIST once per R session (see `nist_lines()`); later changes of the settings are computed locally. The zoom of the plot is kept when the lines change.

The app needs the shiny, plotly and bslib packages, and an internet connection for the species not yet fetched.

Value

Called for its side effect: runs the app. Use `libs_lines()` and `plot_lines()` for the same results in scripts.

Author(s)

Christian L. Goueguel

See Also

`libs_lines()`, `plot_lines()`, `nist_lines()`

Examples

```
if (interactive() && rlang::is_installed(c("shiny", "plotly", "bslib"))) {
  data(forageLIBS)
  line_finder(forageLIBS[-(1:14)])
}
```

line_intensities *Intensities of Emission Lines*

Description

Measures the intensity of emission lines in spectra: for each line, the peak is searched near its tabulated wavelength, and its area or height is measured. The result feeds `boltzmann()`, `saha_boltzmann()`, `cf_libs()` and `calibration_curve()`.

Usage

```
line_intensities(
  spectra,
  lines,
  half_width = 0.15,
  search = 0.2,
  method = "area",
  baseline = FALSE,
  fit_width = 3 * half_width,
  limit = NULL,
  min_snr = 3
)
```

Arguments

`spectra`	The spectra: a numeric vector named by wavelength (one spectrum), or a matrix or data frame with one spectrum per row and the wavelengths as column names. Other columns of a data frame (such as sample names) are kept in the result.
`lines`	The lines: a numeric vector of wavelengths (nm), whose names are used as labels, or a data frame with a `wavelength` column, such as returned by `nist_lines()`, whose columns are kept in the result.
`half_width`	The half-width of the integration window around the peak, in nm. Default is 0.15.
`search`	The half-width of the window in which the peak is searched around the tabulated wavelength, in nm. Default is 0.2.
`method`	`"area"` (default), `"height"` or `"voigt"`.
`baseline`	A logical: subtract a linear baseline under each line (`FALSE`, default).
`fit_width`	The half-width of the window of the Voigt fit, in nm. Default is 3 times `half_width`.
`limit`	The saturation limit of the detector, in counts, or `NULL` (default) not to check saturation.
`min_snr`	The signal-to-noise ratio above which a line is detected. Default is 3.

Details

For each spectrum and line, the peak is the channel of highest intensity within `search` nm of the tabulated wavelength, which absorbs small errors of the wavelength calibration. The intensity is then measured on the channels within `half_width` nm of the peak:

- `method = "area"` (default): the integrated intensity (trapezoidal rule), in intensity units times nm;
- `method = "height"`: the peak intensity;
- `method = "voigt"`: the area of a Voigt profile fitted with `peak_fit()` to the channels within `fit_width` nm of the peak. It separates the line from a flat background and is less sensitive to the window, but it is slower and needs a line well sampled by the detector.

With `baseline = TRUE`, a straight line through the intensities at both ends of the window is subtracted before the area or height is measured (the Voigt fit always includes a constant background).

Each line is also checked:

- `snr`: the height above the baseline divided by the noise of the spectrum, estimated robustly from its channel-to-channel differences; `detected` is `snr >= min_snr`.
- `saturated`: with `limit`, whether a channel of the window reaches the saturation limit of the detector (see `saturation_summary()`).

Value

A tibble with one row per spectrum and line: the other columns of `spectra` (or `spectrum`, the row number), the columns of `lines` (or `line` and `wavelength`), and `peak_wavelength`, `shift` (nm), `height`, `intensity`, `snr`, `detected` and `saturated` (which replace the columns of `lines` of the same name, such as the relative intensity listed by `nist_lines()`). For a single spectrum, it has one row per line, like `lines`, with the measured `intensity`.

Author(s)

Christian L. Goueguel

See Also

`step_line_intensities()`, `nist_lines()`, `cf_libs()`, `calibration_curve()`, `peak_fit()`

Examples

```
data(forageLIBS)
spectra <- forageLIBS[1:8, -(1:14)]
k <- c(`K I 404.41` = 404.414, `K I 404.72` = 404.721, `Mg I 518.36` = 518.360)
line_intensities(spectra, k, baseline = TRUE)

# one spectrum and a table of lines, ready for a Boltzmann plot
lines <- data.frame(wavelength = c(428.30, 430.25, 443.50, 445.48),
                    Aki = c(4.34e7, 1.36e8, 6.70e7, 8.70e7), gk = c(5, 5, 5, 7),
                    Ek = c(4.78, 4.78, 4.68, 4.68))
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
line_intensities(mean_spectrum, lines, baseline = TRUE)
```

loading_peaks *Peaks of the Loadings of a PCA*

Description

Finds the wavelengths that contribute most to the components of a PCA: the largest positive and negative local extrema of the loadings, or the largest maxima of the variance explained by wavelength, optionally matched to emission lines. `plot_loadings()` labels the same peaks.

Usage

```
loading_peaks(
  model,
  components = NULL,
  type = c("loadings", "contribution"),
  top = 10,
  lines = NULL,
  tol = 0.1,
  span = 5
)
```

Arguments

`model`	A `stats::prcomp()` fit or an object returned by `robpca()`, `rospca()` or `macropca()`. The variable names must be the wavelengths; otherwise, the variables are numbered.
`components`	The components to show (numbers). Default is the first three (or all of them, if fewer).
`type`	`"loadings"` (default) for one panel of loadings per component, or `"contribution"` for the variance of each wavelength explained by `components`.
`top`	The number of peaks labeled per component (per panel). Default is 10; use 0 for no labels.
`lines`	Optional line list returned by `libs_lines()`, to label the peaks with the emission lines they match.
`tol`	The largest distance, in nm, between a peak and a line it matches. Default is 0.1.
`span`	The half-width, in channels, of the window in which a peak is a local extremum. Default is 5.

Details

See `plot_loadings()` for the definitions of the peaks, of the explained variance, of the orientation of the components and of the line matching. A peak is flagged as part of a derivative shape (`derivative = TRUE`) when a local extremum of the opposite sign, at least half as large, lies within `2 * span` channels on the same detector segment: this usually means a shift or a broadening of the line between spectra.

Value

A tibble with one row per peak, sorted by component and by decreasing absolute value, and columns `component`, `variable` (the column name), `wavelength`, `value` (the loading, or the explained variance), `sign` (`"positive"` or `"negative"`) and, for loadings, `derivative`. With `lines`, it also has the columns `species` and `line_wavelength` of the matched line (`NA` when none is within `tol`) and `candidates`, all the lines within `tol`, from the nearest.

Author(s)

Christian L. Goueguel

See Also

`plot_loadings()`, `libs_lines()`

Examples

```
spectra_id <- forageLIBS |> dplyr::select(1:2) |> names()
minerals <- forageLIBS |> dplyr::select(3:14) |> names()
spectra <- forageLIBS |> dplyr::select(-dplyr::all_of(c(spectra_id, minerals)))
set.seed(1)
fit <- robpca(center(spectra))
loading_peaks(fit, top = 5)
```

`lorentzian_profile` *Lorentzian Function*

Description

Computes the Lorentzian function, which is a peaked function with a distinctive bell-shaped curve and wider tails than the Gaussian function. The Lorentzian function is commonly used to model spectral line shapes.

Usage

```
lorentzian_profile(x, y0, xc, wL, A)
```

Arguments

`x`	A numeric vector representing the independent variable (e.g., wavelength, frequency).
`y0`	A numeric value specifying the baseline offset.
`xc`	A numeric value representing the center of the peak.
`wL`	A numeric value specifying the full width at half maximum (FWHM).
`A`	A numeric value representing the peak area.

Details

The Lorentzian function is defined as:

$$y = y_0 + \frac{A}{\pi} \frac{\gamma}{(x - x_c)^2 + \gamma^2}$$

where:

- A is the peak area
- x_c is the center of the peak
- $\gamma = w_L/2$ is the half-width at half maximum (HWHM)
- y_0 is the baseline offset

The Lorentzian function arises naturally in various physical processes, such as the decay of excited states in atoms or molecules, and is commonly used to model phenomena that exhibit homogeneous broadening, where all atoms or molecules experience the same broadening mechanism.

Value

A numeric vector containing the values of the Lorentzian function evaluated at the provided `x` values.

Author(s)

Christian L. Goueguel

Examples

```
x <- seq(-10, 10, length.out = 100)
y <- lorentzian_profile(x, y0 = 0, xc = 0, wL = 2, A = 1)
plot(x, y, type = "l", col = "red", main = "Lorentzian Profile")
```

`macropca` *Robust PCA for Cellwise and Casewise Outliers (MacroPCA)*

Description

Robust PCA that handles both outlying observations (casewise outliers), outlying cells (cellwise outliers) and missing values, by the MacroPCA algorithm of Hubert, Rousseeuw and Van den Bossche (2019), which combines the detection of deviating cells (DDC) with the steps of ROBPCA (Hubert, Rousseeuw and Vanden Branden, 2005). The results have the same form as those of `robpca()`, so that predict() and `plot_outlier_map()` work the same way, and `plot_cell_map()` shows the outlying cells. `cellpca()` starts from a MacroPCA fit.

Usage

```
macropca(
  x,
  k = NULL,
  alpha = 0.5,
  kmax = 10,
  var_explained = 0.8,
  scale = FALSE,
  ndir = 250,
  maxiter = 20,
  tol = 0.005
)
```

Arguments

x	A numeric matrix or data frame, with one observation per row. Missing values are allowed.
k	The number of principal components. If NULL (default), it is chosen from var_explained and kmax.
alpha	The robustness parameter, between 0.5 and 1: the fraction of observations used in the fit. Default is 0.5.
kmax	The maximum number of components. Default is 10.
var_explained	The fraction of variance used to choose k when it is not given. Default is 0.8.
scale	A logical: scale the variables by their robust scale (FALSE, default: they are only centered, so that intense emission lines are not outweighed by noise and continuum channels).
ndir	The number of random directions of the outlyingness, or "all". Default is 250.
maxiter, tol	The maximum number of iterations of step 4, and the tolerance on the largest angle between successive subspaces (as a fraction of a right angle). Defaults are 20 and 0.005, as in the paper.

Details

The algorithm follows Hubert, Rousseeuw and Van den Bossche (2019) and the MacroPCA code of the cellWise package, whose results it reproduces (with scale = FALSE): 0. **Deviating cells.** The cells that deviate from the values predicted by the most correlated variables, and the outlying observations, are detected by the DDC of Rousseeuw and Van den Bossche (2018), computed in C++ as in cellWise (for more than 750 variables, the neighbors are those with the largest wrapped correlations, found exactly by blocks of variables). DDC also imputes the flagged and missing cells. Of the observations flagged by DDC, at most the $n - h$ most outlying are set aside.

1. **Standardization.** With scale = TRUE, the variables are divided by their robust scale (1-step M-estimator).
2. **Projection pursuit.** As in robpca(), the outlyingness of each observation is its largest standardized distance (with the univariate MCD) over ndir directions through pairs of observations (all pairs when there are few), on the data in which only the h observations with the

fewest flagged cells have their flagged cells imputed. The h least outlying observations not set aside form H_0.

3. **Subspace dimension.** A classical PCA of the observations of H_0, with their flagged and missing cells imputed, gives the eigenvalues: when k is NULL, it is the smallest number of components that explain var_explained of their variance (at most kmax).
4. **Iterative subspace estimation.** The flagged and missing cells are imputed by the fitted values of the current PCA, and the PCA of H_0 is refitted, until the largest angle between the old and the new subspace (as a fraction of a right angle) is below tol (at most maxiter times).
5. **Reweighting.** The observations whose orthogonal distance is below the cut-off, and not set aside, form H^*, and the PCA is refitted on them (with their flagged cells imputed).
6. **Robust basis.** The center and the eigenvectors within the subspace are estimated by concentration steps on the scores of H^* followed by the deterministic MCD (DetMCD), so that good leverage observations do not tilt the loadings.
7. **Distances.** The scores and the distances of all the observations are computed from the data with only the missing cells imputed. The cut-off of the orthogonal distances is computed on the data whose observations of H^* have their flagged cells imputed.
8. **Residuals.** The residuals of the observed cells are standardized by their 1-step M scale, and the cells beyond $\sqrt{\chi^2_{1,0.99}}$ are flagged.

As in the paper, the cut-offs of the score distances ($\sqrt{\chi^2_{k,0.99}}$) and of the orthogonal distances (the Wilson-Hilferty approximation with the univariate MCD and the 0.99 quantile) are at the 99% level. New observations are analyzed as in MacroPCApredict of cellWise: their deviating cells are detected with the DDC model of the fit, and their flagged and missing cells are imputed iteratively by the fit before their distances are computed.

Value

An object of class specproc_macropca (inheriting from specproc_robpca), with the components described in robpca(), and:

- std_resid: the standardized cell residuals (NA for missing cells).
- flagged_cells: a logical matrix of the flagged cells.
- flagged_rows: the observations flagged by DDC.
- imputed: the data with the missing values imputed by the PCA fit (the flagged cells keep their values).
- resid_scale: the robust scale of the residuals of each variable, used to standardize those of new data.

Author(s)

Christian L. Goueguel

References

- Hubert, M., Rousseeuw, P.J., Van den Bossche, W. (2019). MacroPCA: an all-in-one PCA method allowing for missing values as well as cellwise and rowwise outliers. Technometrics, 61(4):459-473. doi:10.1080/00401706.2018.1562989
- Hubert, M., Rousseeuw, P.J., Vanden Branden, K. (2005). ROBPCA: a new approach to robust principal component analysis. Technometrics, 47(1):64-79.
- Rousseeuw, P.J., Van den Bossche, W. (2018). Detecting deviating data cells. Technometrics, 60(2):135-145.
- Hubert, M., Rousseeuw, P.J., Verdonck, T. (2012). A deterministic algorithm for robust location and scatter. Journal of Computational and Graphical Statistics, 21(3):618-637.

See Also

`cellpca()`, `robpca()`, `step_macropca()`, `plot_outlier_map()`, `plot_cell_map()`

Examples

```
set.seed(1)
# LIBS spectra of forage samples
minerals <- c("Ca", "Cl", "Cu", "Fe", "Mg", "Mn", "Mo", "P", "K", "Na", "S", "Zn")
forageLIBS |>
  dplyr::select(-Measurement, -Sample, -dplyr::all_of(minerals)) |>
  center() |>
  macropca() |>
  print()
```

---

mcwhirter_criterion *McWhirter Criterion for Local Thermodynamic Equilibrium*

Description

Computes the minimum electron density required for local thermodynamic equilibrium (LTE) by the McWhirter criterion, and checks a measured density against it.

Usage

```
mcwhirter_criterion(temperature, delta_e, electron_density = NULL)
```

Arguments

`temperature`	The plasma temperature, in K.
`delta_e`	The largest energy gap between adjacent levels, in eV.
`electron_density`	An optional measured electron density, in cm^{-3}.

Details

$$N_e \geq 1.6 \times 10^{12}\, T^{1/2} (\Delta E)^3 \text{ cm}^{-3}$$

with T in K and ΔE (eV) the largest energy gap between adjacent levels of interest. The criterion is necessary but not sufficient: in transient and inhomogeneous plasmas such as those of LIBS, LTE also requires that equilibration be faster than the plasma evolution and diffusion (Cristoforetti et al., 2010).

Value

A tibble with the `temperature`, `delta_e`, `minimum_density` (cm^{-3}) and, if `electron_density` is given, the density and whether the criterion is `satisfied`. The inputs are recycled to a common length.

References

- McWhirter, R.W.P. (1965). Spectral intensities. In Huddlestone, R.H., Leonard, S.L. (eds.), Plasma Diagnostic Techniques, Academic Press, New York, pp. 201-264.
- Cristoforetti, G., et al. (2010). Local thermodynamic equilibrium in laser-induced breakdown spectroscopy: beyond the McWhirter criterion. Spectrochimica Acta Part B, 65(1):86-95.

Examples

```
mcwhirter_criterion(temperature = 10000, delta_e = 3.12, electron_density = 1e17)
```

`medcouple_weight` *Medcouple Tail Weight Measure*

Description

This function calculates the left medcouple (LMC) and the right medcouple (RMC) for a given numeric vector. The left and right medcouples provide a robust measure of the distribution's tail behavior in the presence of outliers or contaminated data.

Usage

```
medcouple_weight(x, drop.na = FALSE)
```

Arguments

`x`	A numeric vector.
`drop.na`	Logical value indicating whether to remove missing values (NA). If `FALSE` (default) and `x` contains missing values, `NA` is returned.

Details

The left and right medcouples are a robust measure of tail weight based on the median and the medcouple (Brys *et al.* 2004): the left medcouple is minus the medcouple of the observations below the median, and the right medcouple is the medcouple of the observations above the median. The left and right medcouples are robust to outliers and have a breakdown value of 25%. Specifically, the left medcouple (LMC) measures the skewness in the lower tail of the distribution, while the right medcouple (RMC) measures the skewness in the upper tail.

The interpretation of LMC and RMC is as follows:

- At the normal distribution, LMC = RMC = 0.2 (approximately).
- Larger values indicate heavier tails than the normal distribution, and smaller values lighter tails.

Value

A tibble with two numeric columns:

- `LMC`: Left medcouple.
- `RMC`: Right medcouple.

Author(s)

Christian L. Goueguel

References

- Brys, G., Hubert, M., and Struyf, A. (2006). Robust measures of tail weight. Computational Statistics & Data Analysis, 50(3):733-759
- Brys, G., Hubert, M., and Struyf, A. (2004). A robust measure of skewness. Journal of Computational and Graphical Statistics, 13(4):996-1017

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
moments::kurtosis(fe)   # classical
medcouple_weight(fe)    # robust, left and right tails
```

minmax *Min-Max Normalization*

Description

This function rescales a numeric vector using the min-max normalization technique, which linearly transforms the values to a new range specified by the a and b arguments. The minimum value in the original vector is mapped to a, and the maximum value is mapped to b.

Usage

```
minmax(x, a = 0, b = 1, drop.na = TRUE)
```

Arguments

x	A numeric vector.
a	The minimum value of the new range (default: 0).
b	The maximum value of the new range (default: 1).
drop.na	A logical value indicating whether to remove missing values (NA). If TRUE (the default), missing values are removed from the output. If FALSE, they are kept (as NA) and ignored when computing the range.

Value

A numeric vector with values rescaled to the new range [a, b].

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
head(minmax(forageLIBS$K))
head(minmax(forageLIBS$S, a = -1, b = 1, drop.na = FALSE))
```

msc *Multiplicative Signal Correction*

Description

This function performs multiplicative scatter/signal correction (MSC) on an input data of spectra. It corrects for multiplicative and additive effects in the spectral data by regressing against a reference spectrum.

Usage

```
msc(
  x,
  xref = NULL,
  drop.offset = TRUE,
  robust = TRUE,
  window = NULL,
  drop.na = TRUE
)
```

Arguments

`x`	A numeric matrix or data frame containing the input spectra. Each row represents a sample, and each column represents a spectral variable.
`xref`	An optional numeric vector representing the reference spectrum. If `NULL` (default), the mean or median of `x` is used as the reference.
`drop.offset`	A logical value indicating whether the additive offset a_i should be removed from the spectra (default is `TRUE`). If `FALSE`, only the multiplicative effect is corrected: $\mathbf{x}_i/b_i$.
`robust`	A logical value indicating whether the median (`TRUE`, default) or the mean (`FALSE`) of `x` is used as `xref`.
`window`	An optional list of numeric vectors specifying the column indices of spectral windows. If provided, MSC is performed separately for each window.
`drop.na`	A logical value indicating whether to remove spectra (rows) containing missing values. If `TRUE` (the default), such rows are removed.

Details

Each spectrum $\mathbf{x}_i$ is regressed against the reference spectrum, $\mathbf{x}_i = a_i + b_i\mathbf{x}_{ref} + \mathbf{e}_i$, and corrected as $(\mathbf{x}_i - a_i)/b_i$. When `window` is given, the regression and correction are carried out separately in each spectral window (piecewise MSC); variables outside all windows are left unchanged.

To correct new spectra consistently, pass the returned `reference` as `xref`.

Value

A list with the following components:

`correction`	The corrected spectra.
`offset`	The intercepts/offsets a_i (a matrix with one column per window when `window` is given).
`slope`	The multiplicative scatter factors/slopes b_i (a matrix with one column per window when `window` is given).
`reference`	The reference spectrum used.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
res <- msc(spectra)
res$correction[1:3, 1:4]
```

---

multipeak_fit *Multiple Peaks Fitting*

Description

Fitting of multiple spectral lines by the same or different lineshape functions with variable parameters.

Usage

```
multipeak_fit(
  x,
  peaks,
  profiles,
  wL = NULL,
  wG = NULL,
  A = NULL,
  wlgth.min = NULL,
  wlgth.max = NULL,
  id = NULL,
  max.iter = 200
)
```

Arguments

x	A data frame or tibble of spectra, one spectrum per row. Column names are the wavelengths; an optional identifier column can be given with id.
peaks	A numeric vector of the (approximate) peak center wavelengths.
profiles	A character vector of the lineshape functions for fitting, one per peak or a single one used for all peaks: "lorentzian", "gaussian", "voigt" (exact) or "pseudo_voigt" (case insensitive).
wL	A numeric (single value or one per peak) of the Lorentzian full width at half maximum (initial guess)
wG	A numeric (single value or one per peak) of the Gaussian full width at half maximum (initial guess)
A	A numeric (single value or one per peak) of the peak area (initial guess)
wlgth.min	A numeric of the lower bound of the wavelength subset
wlgth.max	A numeric of the upper bound of the wavelength subset
id	A character specifying the name of the column holding the spectra id (optional)
max.iter	A numeric specifying the maximum number of iteration (200 by default)

Details

The function uses minpack.lm::nlsLM, which is based on the Levenberg-Marquardt algorithm for searching the minimum value of the square of the sum of the residuals. All peaks are fitted simultaneously with a common baseline offset:

$$y = y_0 + \sum_i A_i \cdot f_i(x; x_{c,i}, w_i)$$

The parameters of peak i are named with the suffix _i (e.g. xc_1, wG_1, A_1). Initial values that are not supplied are estimated from the data around each peak center. Each peak center is constrained to lie within the fitted wavelength range.

Value

A tibble with one row per spectrum and the columns id (or spectrum), data, fit, tidied and augmented (see peak_fit()). augmented additionally contains one column per peak (.peak_1, ...) with the contribution of each fitted line.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
wl <- as.numeric(names(mean_spectrum))
# the overlapping lines of the K I doublet
doublet <- tibble::as_tibble(as.list(mean_spectrum[wl > 404.1 & wl < 405.0]))
res <- multipeak_fit(doublet, peaks = c(404.414, 404.721), profiles = "gaussian")
res$tidied[[1]]
```

nas *Net Analyte Signal and Figures of Merit*

Description

This function computes the net analyte signal (NAS) of each sample for a multivariate inverse calibration model (PLS or PCR), following Lorber (1997) and Faber (1998), and the figures of merit derived from it: sensitivity, selectivity and, when the instrumental noise is known, analytical sensitivity, limits of detection and quantification, and signal-to-noise ratios.

Usage

```
nas(
  x,
  y,
  ncomp = 5,
  method = "pls",
  center = TRUE,
  scale = FALSE,
  noise = NULL
)
```

Arguments

`x`	A numeric matrix or data frame of calibration spectra (one per row).
`y`	A numeric vector (or one-column matrix or data frame) of analyte concentrations.
`ncomp`	A positive integer giving the number of latent variables of the calibration model. Default is 5.
`method`	The inverse calibration model: `"pls"` (default, SIMPLS) or `"pcr"`.
`center`	A logical value indicating whether to mean-center `x` and `y`. Default is `TRUE`.
`scale`	A logical value indicating whether to scale the columns of `x` to unit variance. Default is `FALSE`. The figures of merit are then expressed in scaled units.
`noise`	An optional positive number: the standard deviation of the instrumental noise, in the units of `x`. Needed for the analytical sensitivity, LOD, LOQ and signal-to-noise ratios.

Details

The net analyte signal is the part of a spectrum that is unique to the analyte, i.e. orthogonal to the spectral contributions of all other constituents (the interferents). In an inverse calibration model $\hat{y} = \bar{y} + (\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{b}$ with A latent variables, the interferent space is spanned by the calibration spectra reconstructed from the model, from which the part explained by $\hat{\mathbf{y}}$ has been removed. The regression

vector **b** is orthogonal to this space, so the NAS space within the model is one-dimensional and spanned by **b** (Faber, 1998; Bro and Andersen, 2003). The NAS vector of sample i is therefore

$$\mathbf{r}_i^* = \frac{(\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{b}}{\|\mathbf{b}\|^2}\,\mathbf{b}$$

and its signed length, the scalar NAS, is

$$\mathrm{NAS}_i = \frac{(\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{b}}{\|\mathbf{b}\|} = \frac{\hat{y}_i - \bar{y}}{\|\mathbf{b}\|}$$

The figures of merit follow Olivieri *et al.* (2006):

- **Sensitivity**, $\mathrm{SEN} = 1/\|\mathbf{b}\|$: the NAS produced by a unit change in concentration, in signal units per concentration unit.
- **Selectivity** of sample i, $\mathrm{SEL}_i = |\mathrm{NAS}_i|/\|\mathbf{x}_i - \bar{\mathbf{x}}\|$: the fraction of the (centered) signal that is used for prediction, between 0 and 1.
- With the standard deviation of the instrumental noise σ_x (`noise`): the **analytical sensitivity** $\gamma = \mathrm{SEN}/\sigma_x$, whose inverse is the smallest concentration difference that can be distinguished; the **limit of detection** $\mathrm{LOD} = 3.3\,\sigma_x/\mathrm{SEN}$; the **limit of quantification** $\mathrm{LOQ} = 10\,\sigma_x/\mathrm{SEN}$; and the **signal-to-noise ratio** of each sample, NAS_i/σ_x.

The LOD and LOQ above account for the instrumental noise only, not for the uncertainty of the calibration model or of the reference values, so they are lower bounds. The noise level must be estimated independently, for example as the standard deviation of the differences between repeated spectra of the same sample divided by $\sqrt{2}$, in the same units (and after the same preprocessing) as `x`.

All quantities depend on the number of latent variables `ncomp`, which should be chosen by validation beforehand.

Before specProc 0.4.0, `nas()` returned spectra filtered by direct orthogonalization; that correction is available from `direct_orthogonal()`.

Value

An object of class `specproc_nas`, a list with:

- `nas`: A tibble with one row per calibration sample: the scalar `nas`, the `selectivity`, the fitted concentration `fitted` and, if `noise` is given, the signal-to-noise ratio `snr`.
- `figures_of_merit`: A named vector with the `sensitivity`, the mean selectivity (`selectivity`) and, if `noise` is given, `analytical_sensitivity`, `lod` and `loq`.
- `nas_vectors`: A tibble of the NAS vectors $\mathbf{r}_i^*$ of the calibration samples.
- `coefficients`: The regression vector **b**.
- `ncomp`, `method`, `noise`, `center`, `scale` and `y_center`: the model settings and preprocessing parameters.

Use predict() to compute the NAS, selectivity and predicted concentration of new samples.

Author(s)

Christian L. Goueguel

References

- Lorber, A. (1986). Error propagation and figures of merit for quantification by solving matrix equations. Analytical Chemistry, 58(6):1167-1172.
- Lorber, A., Faber, K., Kowalski, B.R. (1997). Net analyte signal calculation in multivariate calibration. Analytical Chemistry, 69(8):1620-1626.
- Faber, N.M. (1998). Efficient computation of net analyte signal vector in inverse multivariate calibration models. Analytical Chemistry, 70(23):5108-5110.
- Bro, R., Andersen, C.M. (2003). Theory of net analyte signal vectors in inverse regression. Journal of Chemometrics, 17(12):646-652.
- Olivieri, A.C., Faber, N.M., Ferré, J., Boqué, R., Kalivas, J.H., Mark, H. (2006). Uncertainty estimation and figures of merit for multivariate calibration (IUPAC Technical Report). Pure and Applied Chemistry, 78(3):633-661.

See Also

`direct_orthogonal()` for the filter formerly returned by `nas()`.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
# the three samples measured twice
twice <- forageLIBS$Sample[duplicated(forageLIBS$Sample)]
first <- match(twice, forageLIBS$Sample)
second <- vapply(twice, function(s) max(which(forageLIBS$Sample == s)), integer(1))
# the noise of the spectra, from the differences between repeated measurements
noise <- stats::sd(as.matrix(spectra[second, ]) - as.matrix(spectra[first, ])) / sqrt(2)

cal <- 1:300
fit <- nas(spectra[cal, ], forageLIBS$K[cal], ncomp = 7, noise = noise)
fit
head(fit$nas)
# the net analyte signal and selectivity of new spectra
head(predict(fit, spectra[-cal, ]))
```

`nist_ionization_energy`

Ionization Energies from the NIST Atomic Spectra Database

Description

Retrieves ionization energies from the NIST Atomic Spectra Database, for example the ionization energy of the neutral atom needed by `saha_boltzmann()`.

Usage

```
nist_ionization_energy(species, timeout = 120)
```

Arguments

species	A character vector of emitters in spectroscopic notation: "Ca I" gives the energy needed to ionize neutral calcium, "Ca II" that needed to ionize Ca^+.
timeout	The download timeout, in seconds. Default is 120.

Value

A named numeric vector of ionization energies, in eV.

See Also

nist_lines(), saha_boltzmann()

Examples

```
# needs an internet connection
try(nist_ionization_energy(c("Ca I", "Mg I")))
```

---

nist_levels — *Energy Levels and Partition Functions from the NIST Atomic Spectra Database*

Description

nist_levels() retrieves the energy levels of an atom or ion from the NIST Atomic Spectra Database, and partition_function() computes the internal partition function of a species from its levels, as needed by cf_libs().

Usage

```
nist_levels(species, timeout = 120)

partition_function(levels, temperature, max_energy = Inf)
```

Arguments

species	A character vector of species in spectroscopic notation, such as "Ca I".
timeout	The download timeout, in seconds. Default is 120.
levels	A data frame of levels with columns species, g and energy (eV), such as returned by nist_levels().
temperature	The temperature(s), in K.
max_energy	The highest level energy included, in eV: one value, or a vector named by species. Default is Inf (all levels).

Details

All the levels listed by the ASD are returned, including the autoionizing levels above the first ionization limit, as in the partition functions computed by the ASD; levels without a statistical weight are dropped. The partition function

$$U(T) = \sum_i g_i \exp\left(-\frac{E_i}{k_B T}\right)$$

is summed over the levels given, up to `max_energy`. In a plasma, the ionization energy is lowered by the surrounding charges, and the levels above the lowered limit do not exist: truncating the sum there matters for atoms with many high Rydberg levels close to the limit, such as the alkali metals (see `cf_libs()`, which does it). Without truncation, the result equals the partition function computed by the ASD.

The results of a query are cached for the R session. Cite the database when you use these data (see `nist_lines()`).

Value

`nist_levels()`: a tibble with one row per level and columns `species`, `configuration`, `term`, `J`, `g` (statistical weight) and `energy` (eV).

`partition_function()`: a tibble with the `species`, the `temperature` and the `partition` function, for each species and temperature.

Author(s)

Christian L. Goueguel

See Also

`cf_libs()`, `nist_lines()`, `nist_ionization_energy()`

Examples

```
# needs an internet connection
ca <- try(nist_levels("Ca I"))
if (!inherits(ca, "try-error")) partition_function(ca, temperature = c(8000, 10000))
```

---

nist_lines — *Atomic Line Data from the NIST Atomic Spectra Database*

Description

Retrieves the lines of an atom or ion in a wavelength range from the NIST Atomic Spectra Database (ASD): wavelengths, transition probabilities, level energies and statistical weights, as needed by `boltzmann()` and `saha_boltzmann()`. The data are downloaded on demand; they are not distributed with specProc.

Usage

```
nist_lines(species, wavelength, with_aki = TRUE, timeout = 120)
```

Arguments

species	A character string naming the emitter in spectroscopic notation, such as "Ca I" or "Ca II".
wavelength	A numeric vector of length 2: the wavelength range, in nm.
with_aki	A logical: keep only the lines with a transition probability (TRUE, default).
timeout	The download timeout, in seconds. Default is 120.

Details

Wavelengths are in air between 200 and 2000 nm, and in vacuum outside this range, as in the ASD. The observed wavelength is used when there is one, the Ritz wavelength otherwise. Energies in brackets or with other qualifiers are returned as they can be read as numbers; check the ASD for such levels. The accuracy grade of the transition probability (accuracy: AAA, AA, A+, A, B+, B, C+, C, D+, D, E) matters for plasma diagnostics: prefer lines of grade B or better.

The results of a query are cached for the R session. Cite the database when you use these data:

- Kramida, A., Ralchenko, Yu., Reader, J., and NIST ASD Team. NIST Atomic Spectra Database, https://physics.nist.gov/asd. National Institute of Standards and Technology, Gaithersburg, MD.

Value

A tibble with one row per line and columns species, wavelength (nm), Aki (s^{-1}), fik (oscillator strength), accuracy, Ei and Ek (lower and upper level energies, eV), gi and gk (statistical weights), lower and upper (configuration, term and J of the levels) and intensity (the relative intensity listed by the ASD, as text).

Author(s)

Christian L. Goueguel

See Also

nist_ionization_energy(), boltzmann(), saha_boltzmann(), starkb_lines()

Examples

```
# needs an internet connection
ca <- try(nist_lines("Ca II", wavelength = c(310, 400)))
if (!inherits(ca, "try-error")) ca
```

`normalize` *Spectra Normalization*

Description

This function implements normalization methods based on background, total area, internal standard, and the L1, L2 and maximum norms of each spectrum.

Usage

```
normalize(x, method = "area", bkg = NULL, wlength = NULL, drop.na = TRUE)
```

Arguments

`x`	A numeric matrix or data frame containing the spectra.
`method`	A character vector specifying the normalization method to apply. Available methods are: "area", "l1", "l2", "max", "background", and "internal". Spectra whose L1, L2 or maximum norm is zero are set to `NA`, with a warning.
`bkg`	A numeric matrix or data frame of the same dimension as `x`, specifying the intensity of the continuum radiation (background emission) used for normalizing `x`. Required for "background" method.
`wlength`	A character vector of the selected wavelength(s) (column names). Required for the "internal" method, where the intensities at these wavelengths are summed to give the internal standard. Optional for the "background" method, where only these wavelengths are normalized and returned.
`drop.na`	A logical value indicating whether to remove spectra (rows) containing missing values before normalizing. Default is `TRUE`.

Details

The normalization methods:

- **Normalization to the background:** Spectra are divided by the intensity of the background emission. Note that it is recommended that the detector dark current be subtracted prior to the normalization.
- **Normalization to the total area:** Each spectrum is divided by the total area of the spectrum over the whole spectral range. The detector dark current must be subtracted prior to this normalization. The total area is calculated as the sum of all intensity levels.
- **L1 normalization** (`"l1"`): each spectrum is divided by the sum of the absolute values of its intensities. It equals the total area for non-negative spectra, and also suits derivative spectra, whose values change sign.
- **L2 (vector) normalization** (`"l2"`): each spectrum is divided by its Euclidean norm, so that its sum of squares is 1. Unlike SNV, the spectrum is not centered.
- **Maximum normalization** (`"max"`): each spectrum is divided by its largest absolute intensity, so that its strongest feature is 1 (or -1).

- **Normalization to an internal standard:** The peak intensity (or area) of the emission line related to the analyte is divided by the peak intensity (or area) of a selected emission line related to the internal standard. The internal standard concentration is assumed constant or known.

Value

A tibble of normalized spectra.

Author(s)

Christian L. Goueguel

References

- De Giacomo, A., Dell'Aglio, M., De Pascale, O., Gaudiuso, R., Santagata, A., Teghil, R., (2008). Laser-induced breakdown spectroscopy methodology for the analysis of copper based alloys used in ancient artworks. Spectrochimica Acta Part B, 63(5):585-590
- Body, D., Chadwick, B.L., (2001). Optimization of the spectral data processing in a LIBS simultaneous elemental analysis system. Spectrochimica Acta Part B, 56(6):725-736.
- Rinnan, A., Van den Berg, F., Balling Engelsen, S., (2009). Review of the most common pre-processing techniques for near-infrared spectra, Trends in Analytical Chemistry, 28(10):1201-1222.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
normalize(spectra[1:3, ], method = "area")[, 1:4]
normalize(spectra[1:3, ], method = "l2")[, 1:4]
# internal standard: the C I 247.86 nm line of the organic matrix
carbon <- names(spectra)[which.min(abs(as.numeric(names(spectra)) - 247.856))]
normalize(spectra[1:3, ], method = "internal", wlength = carbon)[, 1:4]
```

`o2pls` *Modified Orthogonal Projections to Latent Structures*

Description

This function implements the modified orthogonal projections to latent structures (O2PLS) algorithm as proposed by Trygg (2002). The OPLS and O2PLS methods differ as follows: OPLS is unidirectional ($X \Rightarrow Y$), meaning that only orthogonal variations in the X-space are filtered out. Whilst O2PLS is bi-directional ($X \Leftrightarrow Y$), meaning that orthogonal variations in both the X- and Y-space are filtered out.

Usage

```
o2pls(x, y, ncomp = 1, nx = 1, ny = 0, center = TRUE, scale = FALSE)
```

Arguments

`x`	A numeric matrix or data frame representing the predictor variables.
`y`	A numeric vector, matrix or data frame representing the response variables.
`ncomp`	An integer giving the number of joint (predictive) components. Default is 1.
`nx`	An integer giving the number of Y-orthogonal components removed from `x`. Default is 1.
`ny`	An integer giving the number of X-orthogonal components removed from `y`. Default is 0.
`center`	A logical value indicating whether to mean-centered `x` and `y`. Default is `TRUE`.
`scale`	A logical value indicating whether to scale `x` and `y`. Default is `FALSE`.

Details

The O2PLS method handles situations where systematic X-orthogonal variation in Y exists, and it is predictive in both ways, $(X \Rightarrow Y)$ and $(Y \Rightarrow X)$. The systematic part in X and Y is divided into two parts, one which is related to both X and Y (joint/covarying) and one that is not (orthogonal).

The algorithm follows Trygg and Wold (2003):

1. The joint weights $\mathbf{W}$ and $\mathbf{C}$ are the dominant singular vectors of $\mathbf{Y}^T\mathbf{X}$.
2. For each Y-orthogonal component of X, the weight $\mathbf{w}_o$ is the dominant singular vector of $\mathbf{E}_{XY}^T\mathbf{T}$ (with $\mathbf{T} = \mathbf{XW}$ and $\mathbf{E}_{XY} = \mathbf{X} - \mathbf{TW}^T$), and $\mathbf{X}$ is deflated by $\mathbf{t}_o\mathbf{p}_o^T$.
3. The same procedure is applied to Y for the X-orthogonal components.
4. The joint components are recomputed from the filtered data.

For a single response variable, `ny` must be 0 and the method is equivalent to OPLS with `nx` orthogonal components.

Value

An object of class `o2pls`, a list containing:

- `correction`: The filtered `x` (Y-orthogonal variation removed).
- `correction_y`: The filtered `y` (X-orthogonal variation removed).
- `scores`: A list with the joint scores `x` ($\mathbf{T}$) and `y` ($\mathbf{U}$), and the orthogonal scores `x_ortho` and `y_ortho`.
- `loadings`: A list with the joint loadings `x` and `y`, and the orthogonal loadings `x_ortho` and `y_ortho`.
- `weights`: A list with the joint weights `x` ($\mathbf{W}$) and `y` ($\mathbf{C}$), and the orthogonal weights `x_ortho` and `y_ortho`.
- `center`, `scale`: The column centers and scales applied to `x`.

Author(s)

Christian L. Goueguel

References

- Trygg, J., (2002). O2-PLS for qualitative and quantitative analysis in multivariate calibration. J. Chemom. 16(1):283–293.
- Trygg, J., Wold, S., (2003). O2-PLS, a two-block (X–Y) latent variable regression (LVR) method with an integral OSC filter. J. Chemom. 17(1):53–64.

See Also

`predict.specproc_filter()` to correct new spectra with the fitted filter, and `step_o2pls()` to use it in a tidymodels recipe, with one or several outcomes.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
# potassium and calcium at once
fit <- o2pls(spectra, forageLIBS[c("K", "Ca")], ncomp = 2, nx = 2)
fit
```

---

opls — *Orthogonal Projections to Latent Structures*

Description

This function fits an Orthogonal Projections to Latent Structures (OPLS) model to the provided x (predictor) and y (response) data.

Usage

```
opls(
  x,
  y,
  ncomp = NULL,
  center = TRUE,
  scale = FALSE,
  crossval = 7,
  permutation = 0,
  ncomp.ortho = deprecated()
)
```

Arguments

`x`	A numeric matrix or data frame of the predictor variables.
`y`	A numeric vector, or a matrix or data frame with one column, of the response variable.
`ncomp`	The number of orthogonal components removed. If `NULL` (default), it is determined automatically by cross-validation.

`center`	A logical value indicating whether to mean-center `x` and `y`. Default is `TRUE`.
`scale`	A logical value indicating whether to scale `x` and `y` to unit variance. Default is `FALSE`.
`crossval`	An integer giving the number of cross-validation groups (default 7), between 2 and the number of observations. With `crossval = 0`, the model is not cross-validated (Q^2 is `NA`), which requires a fixed `ncomp` and no permutation.
`permutation`	An integer giving the number of permutations for the permutation test. Default is 0 (no test).
`ncomp.ortho`	**[Deprecated]** Use `ncomp`.

Details

OPLS is a supervised modeling technique used to find the multidimensional direction in the x-space that explains the maximum multidimensional variance in the y-space. It separates the systematic variation in x into two parts: one that is linearly related to y (predictive components) and one that is statistically uncorrelated to the response variable y (orthogonal components).

The model has one predictive component and `ncomp` orthogonal components, fitted with the NIPALS algorithm of Trygg and Wold (2002). For each orthogonal component:

1. The weight $\mathbf{w} = \mathbf{X}^T\mathbf{y}/\|\mathbf{X}^T\mathbf{y}\|$, the score $\mathbf{t} = \mathbf{X}\mathbf{w}$ and the loading $\mathbf{p} = \mathbf{X}^T\mathbf{t}/(\mathbf{t}^T\mathbf{t})$ are computed.
2. The orthogonal weight is the part of $\mathbf{p}$ orthogonal to $\mathbf{w}$, $\mathbf{w}_o = \mathbf{p} - (\mathbf{w}^T\mathbf{p})\mathbf{w}$, normalized, with score $\mathbf{t}_o = \mathbf{X}\mathbf{w}_o$ and loading $\mathbf{p}_o = \mathbf{X}^T\mathbf{t}_o/(\mathbf{t}_o^T\mathbf{t}_o)$.
3. $\mathbf{X}$ is deflated by $\mathbf{t}_o\mathbf{p}_o^T$.

The predictive component is then computed from the filtered $\mathbf{X}$. The filtered data are the same as those of `projected_osc()` with the same `ncomp`, and of `o2pls()` with `nx = ncomp`.

Preprocessing. x and y are centered (`center`) and scaled to unit variance (`scale`) with their means and standard deviations, as in the other orthogonalization methods. For Pareto scaling, common in metabolomics, apply `pareto_scale()` to x first (or `step_pareto_scale()` in a recipe): it gives the same filter, scores and R^2 and Q^2 values.

Cross-validation. The observations are split into `crossval` interleaved groups (observation `i` is in group `(i - 1) %% crossval + 1`). Each group is predicted by a model fitted to the others, with the same number of orthogonal components, giving the predictive residual sum of squares (PRESS) and $Q^2 = 1 - \mathrm{PRESS}/\mathrm{SS}_y$, where SS_y is the total sum of squares of the preprocessed y. The data are preprocessed once, with the centers and scales of all the observations.

Number of orthogonal components. If `ncomp = NULL`, components are added (up to `min(10, n, p) - 1`) while each is significant: a component is significant if it increases R^2Y by at least 0.01 and Q^2 by at least 0.01. If the predictive component alone is not significant, no model is built. If the first orthogonal component is not significant, the model has no orthogonal component (a one-component PLS model), with a warning.

Permutation test. The response is permuted `permutation` times and the model refitted with the same number of components. `pR2Y` and `pQ2` are the proportions of permuted models whose R^2Y and Q^2 are at least those of the model, $(1 + \#\{\text{perm} \geq \text{model}\})/\text{permutation}$. Use `set.seed()` for reproducible p-values. The test is skipped by default (`permutation = 0`), as it refits the model as many times.

The results reproduce those of `ropls::opls()` with `predI = 1` and the same preprocessing (`scaleC`), `crossvalI`, `orthoI` and `permI` (apart from rounding): specProc no longer depends on ropls. Unlike ropls, variables with zero variance are kept (their centered values are zero), and when `ncomp = NULL` a model without orthogonal components is returned instead of no model when only the predictive component is significant.

Value

An object of class `specproc_opls` (a list), which predict() applies to new data, with the following components:

x_scores The predictive scores **t** (one column, `p1`).

x_loadings The predictive loadings **p**.

x_weights The predictive weights **w**.

orthoScores The orthogonal scores $\mathbf{T}_o$ (columns `o1`, `o2`, ...).

orthoLoadings The orthogonal loadings $\mathbf{P}_o$.

orthoWeights The orthogonal weights $\mathbf{W}_o$.

y_weights The y-weight c of the predictive component.

y_scores The y-scores $\mathbf{u} = \mathbf{y}/c$.

correction The OPLS-filtered x, $\mathbf{X} - \mathbf{T}_o\mathbf{P}_o^T$ (preprocessed).

fitted The fitted response, in the units of y.

coefficients The regression coefficients of the preprocessed filtered x, $\mathbf{w}c$.

vip, ortho_vip The predictive and orthogonal variable importance in projection (Galindo-Prieto et al., 2014), one value per variable.

components A tibble with one row per component: `R2X`, `R2Y` and `Q2` (the increase due to the component), their cumulative values, and `significance` (`"R1"` significant, `"NS"` Q^2 increase below 0.01, `"N4"` R^2Y increase below 0.01).

summary A one-row data frame with `R2X(cum)`, `R2Y(cum)`, `Q2(cum)`, `RMSEE` (root mean squared error of estimation), the numbers of predictive (`pre`) and orthogonal (`ort`) components and, with a permutation test, `pR2Y` and `pQ2`.

permutation With a permutation test, a tibble of `R2Y(cum)`, `Q2(cum)` and `sim` (the correlation between the permuted and the original response) of the model (first row) and of each permuted model.

center, scale The column centers and scales applied to x.

y_center, y_scale The center and scale applied to y.

Author(s)

Christian L. Goueguel

References

- Trygg, J., and Wold, S., (2002). Orthogonal projections to latent structures (O-PLS). Journal of Chemometrics, 16(3):119-128.
- Galindo-Prieto, B., Eriksson, L., and Trygg, J., (2014). Variable influence on projection (VIP) for orthogonal projections to latent structures (OPLS). Journal of Chemometrics, 28(8):623-632.
- Thévenot, E.A., Roux, A., Xu, Y., Ezan, E., and Junot, C., (2015). Analysis of the human adult urinary metabolome variations with age, body mass index, and gender by implementing a comprehensive workflow for univariate and OPLS statistical analyses. Journal of Proteome Research, 14(8):3322-3335.

See Also

`predict.specproc_opls()`, and `step_opls()` to use the OPLS filter in a tidymodels recipe.

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
cal <- 1:300
fit <- opls(spectra[cal, ], forageLIBS$K[cal], permutation = 5)
fit
head(predict(fit, spectra[-cal, ], type = "response"))

# Pareto scaling of the spectra
opls(pareto_scale(spectra[cal, ]), forageLIBS$K[cal], ncomp = 2)
```

---

osc *Orthogonal Signal Correction*

Description

This function implements three orthogonal signal correction (OSC) algorithms, which are a class of preprocessing techniques designed to minimize, in a set of spectral data, the systematic variability or noise not directly related to or correlated with the response vector or property of interest.

Usage

```
osc(
  x,
  y,
  method = "sjoblom",
  center = TRUE,
  scale = FALSE,
  ncomp = 10,
  tol = 0.001,
```

```
  max.iter = 10,
  pls.ncomp = 5
)
```

Arguments

`x`	A matrix or data frame of the predictor variables.
`y`	A vector (or one-column matrix/data frame) of the response variable.
`method`	A character string indicating the OSC method to use. Accepted values are `"wold"`, `"sjoblom"` and `"fearn"`. Default is `"sjoblom"`.
`center`	A logical value indicating whether to mean-centered `x` and `y`. Default is `TRUE`.
`scale`	A logical value indicating whether to scale `x` and `y`. Default is `FALSE`.
`ncomp`	An integer representing the number of orthogonal components to remove. Default value is 10; it is reduced to `min(n - 1, p)` if larger.
`tol`	A numeric value representing the tolerance for convergence. The default value is 1e-3.
`max.iter`	An integer representing the maximum number of iterations. The default value is 10.
`pls.ncomp`	An integer giving the number of PLS components used to compute the weights in Wold's method. Default is 5.

Details

The OSC algorithm identifies and removes the orthogonal variation in the input spectral matrix, $\mathbf{X}$, by iteratively deflating $\mathbf{X}$ with respect to the response vector $\mathbf{y}$. The resulting $\mathbf{X}$-matrix contains only the variation that is relevant to the $\mathbf{y}$-vector, which can then be used for further modeling or analysis. This function implements three different methods for OSC:

- `"wold"`: the original method of Wold *et al.* (1998). The first principal component score is orthogonalized to $\mathbf{y}$ and the weights are obtained by a PLS regression of the orthogonalized score on $\mathbf{X}$, iterating until convergence.
- `"sjoblom"`: the method of Sjöblom *et al.* (1998), where the weights are obtained by a simple least-squares regression instead of PLS.
- `"fearn"`: the non-iterative method of Fearn (2000), which finds the directions of maximum variance of $\mathbf{X}$ subject to the scores being exactly orthogonal to $\mathbf{y}$.

For the iterative methods, $\mathbf{X}$ is deflated after each component. New data can be corrected by applying the returned `center` and `scale` and then, for each component i, computing $\mathbf{t}_i = \mathbf{X}_{new}\mathbf{w}_i$ and $\mathbf{X}_{new} \leftarrow \mathbf{X}_{new} - \mathbf{t}_i\mathbf{p}_i^T$.

Value

An object of class `specproc_osc` (a list), which predict() applies to new spectra, with the following components:

- `correction`: The corrected matrix.
- `scores`: The orthogonal scores matrix.

- `loadings`: The orthogonal loadings matrix.
- `weights`: The orthogonal weights matrix.
- `R2`: The percentage of the (preprocessed) variance of `x` remaining after correction.
- `angle`: The mean angle (in degrees) between the orthogonal scores and `y`; values close to 90 indicate orthogonality.
- `center`, `scale`: The column centers and scales applied to `x`.

Author(s)

Christian L. Goueguel

References

- Sjöblom, J., Svensson, O., Josefson, M., Kullberg, H., Wold, S., (1998). An evaluation of orthogonal signal correction applied to calibration transfer of near infrared spectra. Chemometrics Intell. Lab. Syst., 44(1):229-244.
- Fearn, T., (2000). On orthogonal signal correction. Chemometrics Intell. Lab. Syst., 50(1):47-52.
- Wold, S., Antti, H., Lindgren, F., Ohman, J. (1998). Orthogonal signal correction of near-infrared spectra. Chemometrics Intell. Lab. Syst., 44(1):175-185.
- Svensson, O., Kourti, T. and MacGregor, J.F., (2002). An investigation of orthogonal correction algorithms and their characteristics. Journal of Chemometrics, 16(1):176-188.

See Also

`predict.specproc_filter()` to correct new spectra with the fitted filter, and `step_osc()` to use it in a tidymodels recipe.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
x <- spectra[wl > 760 & wl < 780]  # the K I resonance lines
res <- osc(x, forageLIBS$K, method = "fearn", ncomp = 2)
res$angle
```

`pareto_scale` *Pareto Scaling*

Description

This function performs Pareto scaling on a numeric matrix or data frame. Pareto scaling scales each variable (column) by dividing it by the square root of its standard deviation.

Usage

```
pareto_scale(x, drop.na = FALSE)
```

Arguments

x	A numeric matrix or data frame to be scaled.
drop.na	A logical value indicating whether to ignore missing values (NA) when computing the standard deviations. Default is FALSE.

Value

A numeric matrix (or a tibble if x is a data frame) with the same dimensions as x, but with each variable scaled by the square root of its standard deviation.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
pareto_scale(spectra)[1:3, 1:4]
```

---

pds — *Piecewise Direct Standardization*

Description

The pds function performs Piecewise Direct Standardization (PDS), a method proposed by Wang *et al.* (1991) to transfer spectra from one instrument to another. Optionally, the local models can be blended with a global PLSR model (direct standardization), which can lead to better transfer performance when the instrumental differences are not purely local.

Usage

```
pds(x1, x2, win = 5, ncomp = 2, alpha = 0)
```

Arguments

x1	A matrix or data frame containing spectra acquired with the standard (master) instrument.
x2	A matrix or data frame containing spectra of the same samples acquired with the instrument to be standardized (slave).
win	An integer specifying the half size of the moving window used for PLSR. A larger value may improve the transfer but will increase computational time. Default is 5.

`ncomp` An integer specifying the number of components to be used in PLSR. Typically, a small number (e.g., 2-5) is sufficient. Default is 2.

`alpha` A numeric value between 0 and 1 specifying the weight for the global PLSR model. A value of 0 (default) corresponds to the original PDS method, while a value of 1 corresponds to using only the global PLSR model.

Details

For each wavelength j of the standard (master) instrument, a PLSR model is fitted between the response of the master instrument at j and the responses of the slave instrument in the window $[j - win, j + win]$ (truncated at the edges of the spectrum). The regression coefficients are stored in a banded transfer matrix $\mathbf{F}$, and the intercepts in a vector $\mathbf{b}_0$. Spectra measured on the slave instrument are then standardized as

$$\mathbf{X}_{2,std} = \mathbf{X}_2\mathbf{F} + \mathbf{1}\mathbf{b}_0^T$$

When `alpha` > `0`, a global PLSR model mapping the whole slave spectrum to the whole master spectrum is also fitted, and the transfer matrix and intercept are $(1 - \alpha)$ times the local ones plus α times the global ones.

Value

A list with two components:

`transfer_matrix` The $p \times p$ transfer matrix $\mathbf{F}$.

`intercept` A vector of length p containing the intercepts.

Author(s)

Christian L. Goueguel

References

- Wang, Y., Veltkamp, D.J., Kowalski, B.R., (1991). Multivariate instrument standardization. Analytical Chemistry, 63(23):2750-2756.
- Bouveresse, E., Massart, D.L., (1996). Improvement of the piecewise direct standardization procedure for the transfer of NIR spectra for multivariate calibration. Chemometrics and Intelligent Laboratory Systems, (32)2:201-213.

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
# a second instrument, with a shifted and less sensitive response
x1 <- as.matrix(spectra[1:30, ])
x2 <- 0.8 * cbind(x1[, -1], x1[, ncol(x1)]) + 50
res <- pds(x1, x2, win = 3, ncomp = 2)
# spectra of the second instrument transferred to the first one
```

```
transferred <- sweep(x2 %*% res$transfer_matrix, 2, res$intercept, "+")
max(abs(transferred - x1)) / max(x1)
```

---

peak_fit *Peak Fitting*

Description

Fitting of a single spectral line by lineshape functions with variable parameters.

Usage

```
peak_fit(
  x,
  profile = "voigt",
  wL = NULL,
  wG = NULL,
  A = NULL,
  wlgth.min = NULL,
  wlgth.max = NULL,
  id = NULL,
  max.iter = 200
)
```

Arguments

x	A data frame or tibble of spectra, one spectrum per row. Column names are the wavelengths (e.g. "396.15"); an optional identifier column can be given with id.
profile	A character specifying the lineshape function to be used: "lorentzian", "gaussian", "voigt" (exact) or "pseudo_voigt".
wL	A numeric specifying the Lorentzian full width at half maximum (initial guess)
wG	A numeric specifying the Gaussian full width at half maximum (initial guess)
A	A numeric specifying the peak area (initial guess)
wlgth.min	A numeric specifying the lower bound of the wavelength subset
wlgth.max	A numeric specifying the upper bound of the wavelength subset
id	A character specifying the name of the column holding the spectra id (optional)
max.iter	A numeric specifying the maximum number of iteration (200 by default)

Details

The function uses `minpack.lm::nlsLM`, which is based on the Levenberg-Marquardt algorithm for searching the minimum value of the square of the sum of the residuals. Each spectrum (row of `x`) is fitted separately with the model

$$y = y_0 + A \cdot f(x; x_c, w)$$

where f is a unit-area Gaussian, Lorentzian, Voigt or pseudo-Voigt profile (see `gaussian_profile()`, `lorentzian_profile()`, `voigt_profile()` and `pseudo_voigt_profile()`). The fitted parameters are `y0`, `xc`, `A` and the width(s) `wG` and/or `wL` (full widths at half maximum). The Voigt profile is evaluated exactly in C++; the pseudo-Voigt (Thompson-Cox-Hastings) approximation is faster but accurate to about 1\

Initial values that are not supplied are estimated from the data: the peak center from the position of the maximum, the width from the full width at half maximum, and the area from the integrated intensity above the minimum. The center is constrained to the fitted wavelength range and the widths and area to positive values. If the fit of a spectrum fails, a warning is issued and the corresponding `fit`, `tidied` and `augmented` entries are `NULL`.

Value

A tibble with one row per spectrum and the columns:

- `id` column (or `spectrum`, the row number, when `id = NULL`),
- `data`: the fitted data (`x`, `y`),
- `fit`: the `nls` model object,
- `tidied`: the estimated parameters with their standard errors,
- `augmented`: the data with fitted values (`.fitted`) and residuals (`.resid`).

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
wl <- as.numeric(names(mean_spectrum))
# the H-alpha line of the hydrogen of the samples, Stark-broadened
halpha <- tibble::as_tibble(as.list(mean_spectrum[wl > 653.5 & wl < 659.5]))
res <- peak_fit(halpha, profile = "lorentzian")
res$tidied[[1]]
```

plot_boltzmann *Draw a Boltzmann or Saha-Boltzmann Plot*

Description

Plots the points and the fitted line of boltzmann() or saha_boltzmann(), with the estimated temperature, or the parallel Boltzmann plots of the species of cf_libs().

Usage

```
plot_boltzmann(object, title = NULL)
```

Arguments

object	An object returned by boltzmann(), saha_boltzmann() or cf_libs().
title	The plot title. By default, the method and temperature.

Value

A ggplot object.

See Also

boltzmann(), saha_boltzmann()

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
# calcium lines, with their atomic data from the NIST database
atomic <- data.frame(
  species = c(rep("Ca I", 6), rep("Ca II", 4)),
  stage = c(rep(1, 6), rep(2, 4)),
  wavelength = c(428.301, 430.253, 431.865, 445.478, 612.222, 616.217,
                 315.887, 317.933, 370.603, 373.690),
  Aki = c(4.34e7, 1.36e8, 7.40e7, 8.70e7, 2.87e7, 4.77e7, 3.10e8, 3.60e8, 8.80e7, 1.70e8),
  gk = c(5, 5, 3, 7, 3, 3, 4, 6, 2, 2),
  Ek = c(4.780, 4.780, 4.769, 4.681, 3.910, 3.910, 7.047, 7.050, 6.468, 6.468)
)
lines <- line_intensities(mean_spectrum, atomic, baseline = TRUE)
plot_boltzmann(saha_boltzmann(lines, ionization_energy = 6.113, electron_density = 1.9e17))
```

plot_calibration *Plot a Calibration Curve*

Description

Draws the calibration standards, the fitted curve with its confidence and prediction bands, the limits of detection and quantification, and optionally the concentrations predicted for new samples.

Usage

```
plot_calibration(
  object,
  interval = "both",
  level = 0.95,
  newdata = NULL,
  replicates = 1,
  title = NULL
)
```

Arguments

object	A curve fitted with calibration_curve().
interval	The bands to draw: "both" (default), "confidence", "prediction" or "none".
level	The confidence level of the bands and intervals. Default is 0.95.
newdata	Optional signals of new samples (a numeric vector, or a data frame with the signal column) to show with their predicted concentrations.
replicates	The number of replicate measurements averaged in each new signal. Default is 1.
title	The plot title. By default, the model and the figures of merit.

Details

The confidence band shows where the mean signal lies; the prediction band, where a single new measurement (or the mean of replicates) is expected to fall (see calibration_curve()). With newdata, each new signal is drawn on the curve at its predicted concentration, with guide lines to the axes and a horizontal bar for the interval of the concentration from predict().

Value

A ggplot object.

See Also

calibration_curve(), predict.specproc_calibration()

Examples

```
# the calibration curve of potassium of vignette("calibration")
data(forageLIBS)
spectra_id <- names(forageLIBS)[1:2]
minerals <- names(forageLIBS)[3:14]
spectra <- forageLIBS[setdiff(names(forageLIBS), c(spectra_id, minerals))]
corrected <- baseline_arpls(spectra, lambda = 1e5, max.iter = 20)$correction

# the K I 769.90 nm line, normalized to the C I 247.86 nm line of the matrix
lines <- line_intensities(corrected, c(C = 247.856, K = 769.896))
data <- forageLIBS[c(spectra_id, "K")]
data$K_signal <- lines$intensity[lines$line == "K"] / lines$intensity[lines$line == "C"]

# a quarter of the samples are set aside to test the calibration
set.seed(1)
test_samples <- sample(unique(data$Sample), round(0.25 * length(unique(data$Sample))))
calibration <- data[!data$Sample %in% test_samples, ]

k_curve <- calibration_curve(calibration, K_signal, K)
plot_calibration(k_curve)
```

`plot_cell_map` *Cell Map of a MacroPCA or cellPCA Fit*

Description

Shows which cells of the data deviate from a `macropca()` or `cellpca()` fit: one row per observation, the variables (wavelengths) along the horizontal axis, and the flagged cells colored red when the observed value is higher than the fit and blue when it is lower. A strip on the right shows the outlier type of each observation, and a panel on top the share of flagged cells at each wavelength over the mean spectrum, with the most flagged regions labeled, which shows which emission lines hold the cellwise outliers.

Usage

```
plot_cell_map(
  object,
  rows = NULL,
  columns = NULL,
  resolution = c(200, 400),
  order = c("data", "od", "cluster"),
  profile = TRUE,
  spectra = NULL,
  threshold = 0.1,
  labels = 5,
  lines = NULL,
  tol = 0.1,
  title = NULL
)
```

Arguments

`object`	An object returned by `macropca()` or `cellpca()`.
`rows, columns`	Optional indices or names of the rows and columns to show. Default is all.
`resolution`	The maximum numbers of rows and columns of blocks. Default is `c(200, 400)`.
`order`	The order of the rows: `"data"` (default), `"od"` for decreasing orthogonal distance, which puts the most outlying observations at the top, or `"cluster"` to group the observations with similar flagged cells.
`profile`	If `TRUE` (default), add the panel of the share of flagged cells by variable.
`spectra`	Optional spectra whose mean is drawn behind the profile (a data frame or matrix with the variables of the model, other columns being ignored, or a single named spectrum). Default is the data imputed by the fit.
`threshold`	The share of flagged observations above which channels form a flagged region. Default is 0.1.
`labels`	The number of flagged regions labeled in the profile. Default is 5; use 0 for none.
`lines`	Optional line list returned by `libs_lines()`, to label the regions with the emission lines they match.
`tol`	The largest distance, in nm, between a region and a line it matches. Default is 0.1.
`title`	The plot title. Default is "MacroPCA cell map" or "cellPCA cell map".

Details

Resolution. Spectra have many more cells than a plot has pixels, so adjacent cells are combined into blocks: the numbers of rows and columns of blocks are at most `resolution`, and a map smaller than that shows every cell. Channels are combined only within a detector segment, and the gaps between segments stay empty. The map fills the plot area, whatever the numbers of observations and variables.

Colors. A flagged cell with standardized residual r has an intensity that grows with $\log(|r|/c)$, from 0.25 at the flagging cut-off c to 1 at $10c$ and beyond. The color of a block is the mean of these signed intensities over its cells (0 for cells that are not flagged), on a square-root scale so that blocks with a few flagged cells remain visible: pale blocks have few or mixed flagged cells, saturated blocks many cells that deviate strongly in the same direction.

Order. With `order = "cluster"`, the observations are sorted by a hierarchical clustering (Ward's method) of their flagged cells, at the column resolution of the map, so that observations that deviate in the same regions form bands (for example, a batch or a type of matrix).

Profile. The top panel shows, for each variable, the share of the (shown) observations whose cell is flagged: above zero when higher than the fit, below zero when lower. The mean spectrum is drawn in grey behind it, rescaled, to show whether the flagged channels are on emission lines, on the continuum or in noise. It is the mean of `spectra` or, by default, of the data imputed by the fit; it is left out when this mean is close to zero, as for centered data (then give the raw spectra in `spectra`). The dashed lines are at `threshold`, and the `labels` flagged regions with the largest share (see `flagged_regions()`) are labeled with their peak wavelength, or with the emission line they match.

Value

A patchwork (ggplot2) object.

Author(s)

Christian L. Goueguel

See Also

flagged_regions(), macropca(), plot_outlier_map()

Examples

```
spectra_id <- forageLIBS |> dplyr::select(1:2) |> names()
minerals <- forageLIBS |> dplyr::select(3:14) |> names()
set.seed(1)
fit <- forageLIBS |>
  dplyr::select(-dplyr::all_of(c(spectra_id, minerals))) |>
  center() |>
  macropca(k = 3)

if (requireNamespace("patchwork", quietly = TRUE)) {
  plot_cell_map(fit, order = "cluster")
}
```

---

plot_contributions *Plot Q and T-squared Contributions*

Description

Plots the contributions of the variables to the Q residual or to the T^2 of samples (see contributions()) against wavelength, one panel per sample, and labels the wavelengths that contribute most, with the emission lines they match when a line list is given.

Usage

```
plot_contributions(
  x,
  samples = NULL,
  top = 10,
  lines = NULL,
  tol = 0.1,
  spectra = NULL,
  span = 5,
  interactive = FALSE,
  title = NULL
)
```

Arguments

x	An object returned by contributions().
samples	The samples to show (row numbers of x, or names in its sample column). Default is the three with the largest statistic.
top	The number of peaks labeled per component (per panel). Default is 10; use 0 for no labels.
lines	Optional line list returned by libs_lines(), to label the peaks with the emission lines they match.
tol	The largest distance, in nm, between a peak and a line it matches. Default is 0.1.
spectra	Optional spectra (a data frame or matrix with the variables of the model, other columns being ignored) or a single spectrum (a named numeric vector), whose mean is drawn in grey behind each panel, rescaled to it, to show whether a peak is on an emission line.
span	The half-width, in channels, of the window in which a peak is a local extremum. Default is 5.
interactive	If TRUE, the plot is made with plotly, with the peaks and their candidate lines shown on hover. Default is FALSE.
title	The plot title.

Details

Each contribution is drawn red above zero (the sample is higher than the model, or than the reference samples) and blue below. The top largest positive and negative local extrema are labeled, as in plot_loadings(): in bold with the nearest line of lines within tol nm. The mean spectrum can be drawn in grey behind each panel, rescaled to it.

Value

A ggplot object, or a plotly object if interactive = TRUE.

Author(s)

Christian L. Goueguel

See Also

contributions(), plot_loadings(), libs_lines()

Examples

```
spectra_id <- forageLIBS |> dplyr::select(1:2) |> names()
minerals <- forageLIBS |> dplyr::select(3:14) |> names()
spectra <- forageLIBS |> dplyr::select(-dplyr::all_of(c(spectra_id, minerals)))
set.seed(1)
fit <- robpca(spectra, k = 3)
q <- contributions(fit, data = spectra, samples = c(49, 127), reference = "regular")
plot_contributions(q, spectra = spectra)
```

plot_embedding *Plot a Two-Dimensional Embedding*

Description

Draws the samples in two dimensions of an embedding, such as UMAP coordinates from `embed::step_umap()`, principal component scores from `recipes::step_pca()`, `robpca()` or `stats::prcomp()`, colored by a variable.

Usage

```
plot_embedding(
  data,
  x = NULL,
  y = NULL,
  colour = NULL,
  size = 2,
  alpha = 0.8,
  ellipse = FALSE,
  conf_level = 0.975,
  robust = FALSE,
  distribution = "normal",
  hotelling = "none",
  k = 2,
  t2_method = "f",
  flag = FALSE,
  label = NULL,
  biplot = FALSE,
  biplot_top = 10,
  aspect_ratio = 0.7,
  title = NULL
)
```

Arguments

`data`	The embedding: a data frame (such as a baked recipe), a matrix, a `stats::prcomp()` fit, or an object of `robpca()`, `rospca()` or `macropca()`.
`x, y`	The columns of the axes, unquoted or as strings. By default, the first two embedding coordinates (see details).
`colour`	The variable coloring the points: a column of `data` (unquoted or as a string), or a vector with one value per sample.
`size`	The size of the points: a number (default 2), or a numeric column of `data` or vector with one value per sample to vary it.
`alpha`	The opacity of the points.
`ellipse`	A logical: draw confidence ellipses (`FALSE`, default). Needs the ConfidenceEllipse package.

conf_level	The confidence level(s) of the ellipses, confidence and T^2: one or more values between 0 and 1. Default is 0.975. The samples beyond a T^2 limit are flagged at the highest level.
robust	A logical: robust ellipses (FALSE, default).
distribution	The quantile of the ellipses: "normal" (default, chi-square) or "hotelling".
hotelling	Hotelling's T^2 ellipses and outliers: "none" (default), "all" (all samples) or "group" (within the groups of a discrete colour). Needs the HotellingEllipse package (1.3.0 or later).
k	The number of components of T^2: the two axes, then the next embedding coordinates. Default is 2.
t2_method	The distribution of the T^2 limits: "f" (default) or "beta" (see hotelling_t2()).
flag	A logical: circle in red the samples beyond the T^2 limit (FALSE, default).
label	The labels of the samples beyond the T^2 limit, whether or not they are circled: NULL or FALSE (default) for none, TRUE for their row numbers, or a column of data or a vector with one value per sample.
biplot	A logical: draw the loadings over the scores (FALSE, default).
biplot_top	The number of loadings drawn as labeled arrows. Default is 10.
aspect_ratio	The ratio of the height to the width of the panel. Default is 0.7; NULL lets the panel fill the plot.
title	The plot title.

Details

By default, the axes are the first two columns named like embedding coordinates (UMAP1, PC1, Comp1, ...; the name followed by a number), or else the first two numeric columns. A numeric colour uses a continuous viridis scale, other types a discrete palette.

With ellipse = TRUE, a confidence ellipse is drawn for each group of a discrete colour (or for all the samples otherwise), with ConfidenceEllipse::confidence_ellipse(), at each level of conf_level (0.975 by default). It covers the region expected to hold that share of the samples of the group if they follow a bivariate normal distribution, from their mean and covariance, or from robust estimates (MCD) with robust = TRUE, which resist outlying samples. distribution = "hotelling" uses the quantile of Hotelling's T^2 distribution, which accounts for the uncertainty of the estimates and suits small groups. Robust estimates need larger groups (the MCD fits a subset of about three quarters of the samples): with fewer than about 10 samples per group, their ellipses can be flat or leave out several samples. Groups with fewer than 4 samples get no ellipse.

With hotelling = "all", the ellipses of Hotelling's T^2 at each level of conf_level (97.5% by default) are drawn for all the samples, and the samples beyond the limit at the highest level of T^2 on k components are counted in the subtitle, circled in red with flag = TRUE and labeled with label (each independently of the other): the classical outlier limits of a score plot. For a PCA model (stats::prcomp(), robpca(), rospca() or macropca()), T^2 is that of the model: the scores divided by the variances of the components (the robust eigenvalues of a robust fit), so the ellipses are centered at 0 with the axes of the components, and semi-axes $\sqrt{\lambda_a L}$ for the limit L. The limit is the F (or Beta) limit of hotelling_t2() for a stats::prcomp() fit, and the chi-square quantile of ROBPCA for a robust fit, whose flagged samples are then the leverage points of plot_outlier_map() (with k its number of components and conf_level = 0.975). The scores

of the outlying samples then do not inflate or tilt the ellipses. For other embeddings, T^2 is computed from the mean and covariance of the samples (with `hotelling_t2()` and `HotellingEllipse::ellipseCoord()`). With `hotelling = "group"`, each group of a discrete `colour` gets its own ellipses and limits, from its mean and covariance, which flags the samples atypical of their own group. The ellipses are labeled with their level (instead of a legend), and drawn for the two components shown, while the limits use `k` components, so with `k > 2` a flagged sample can lie inside the ellipses. With `ellipse = TRUE`, the confidence ellipses are always computed from the samples shown. Without groups, `ellipse = TRUE` and `hotelling = "all"` would draw two ellipses of all the samples: only the T^2 ellipse is drawn, with a warning. With groups, both are drawn: the confidence ellipses of the groups and the T^2 limits of all the samples. T^2 suits linear scores such as PCA or PLS; on a UMAP map, whose distances are not meaningful, prefer `ellipse`.

Style. The panel is grey outside the outermost ellipse (of T^2, or of each group) and white inside, so that the samples beyond the limits stand out, with no grid and a fixed `aspect_ratio` (0.7 by default; `NULL` lets the plot fill the space), and thin light grey lines through the origin (when it lies in the range of the samples, as for centered scores). Without ellipses, the panel is white.

Axes. For a `stats::prcomp()` fit or a robust PCA, the axis titles give the share of the variance of each component: of the total variance for `stats::prcomp()`, and of the variance of the `k` components of the model for `robpca()`, `rospca()` and `macropca()`, which estimate only these. With groups, the ellipses (confidence, or T^2 with `hotelling = "group"`) are filled with the color of their group.

Point size. `size` is a number, or a numeric variable (a column of `data` or a vector with one value per sample), such as the concentration of an element, which sets the size of each point, with a legend.

Biplot. With `biplot = TRUE` (for a `stats::prcomp()` fit or a robust PCA), the loadings of the two components are drawn over the scores: all the variables as faint points, and the `biplot_top` longest as labeled arrows. For spectra, the arrows are the local peaks of the loading length along the wavelength, one per emission line, labeled with their wavelength. Each axis of loadings is scaled to the range of its scores, and read on the top and right axes. A sample lies toward the arrows of the variables in which it is high.

In a UMAP embedding, only the neighborhoods are meaningful: the sizes of the clusters and the distances between them are not, and they change with `neighbors` and `min_dist`. Read the plot as a map of which samples are similar, not as a quantitative projection. When both axes are UMAP coordinates (named `UMAP1`, `UMAP2`, ..., as by `embed::step_umap()`), the lines through the origin are left out, and `hotelling` (`"all"` or `"group"`) is ignored, with a warning: its T^2 limits assume linear scores.

Value

A ggplot object.

Author(s)

Christian L. Goueguel

See Also

`hotelling_t2()`, `plot_outlier_map()`, `plot_loadings()`, `robpca()`

Examples

```
# robust PCA of the forage spectra, with the Hotelling's T-squared limit
if (rlang::is_installed("HotellingEllipse", version = "1.3.0")) {
  data(forageLIBS)
  # the 380-430 nm window (Ca II H and K lines)
  wl <- suppressWarnings(as.numeric(names(forageLIBS)))
  set.seed(1)
  forageLIBS[which(wl > 380 & wl < 430)] |>
    center() |>
    robpca(k = 2) |>
    plot_embedding(hotelling = "all", flag = FALSE, label = TRUE)
}


# UMAP map of the iris flowers, from embed::step_umap(): no lines through
# the origin, whose position means nothing on such a map
if (rlang::is_installed(c("recipes", "embed"))) {
  set.seed(1)
  umap <- recipes::recipe(Species ~ ., data = iris) |>
    recipes::step_normalize(recipes::all_predictors()) |>
    embed::step_umap(recipes::all_predictors(), neighbors = 15) |>
    recipes::prep() |>
    recipes::bake(new_data = NULL)
  plot_embedding(umap, colour = Species, title = "UMAP of the iris flowers")
}
```

`plot_fit` *Plotting of Fitted Spectral Line*

Description

Plots the data and the fitted lineshape returned by `peak_fit()` or `multipeak_fit()`, together with the residuals. The fitted curve is drawn on a fine wavelength grid, so it shows the fitted profile between the measured channels. For multi-peak fits, the individual peak contributions are drawn as dashed lines. When several spectra were fitted, one panel is drawn per spectrum.

Usage

```
plot_fit(
  data,
  title = NULL,
  pt.size = 3,
  pt.colour = "black",
  pt.shape = 21,
  pt.fill = "black",
  line.size = 1,
  line.colour = "red",
```

```
  linetype = "solid",
  resid.shape = 21,
  resid.size = 2,
  resid.fill = "blue",
  resid.colour = "black"
)
```

Arguments

`data`	Data to be displayed from the `peak_fit` or `multipeak_fit` function
`title`	Plot title
`pt.size`	Size of original data point
`pt.colour`	Colour of original data points
`pt.shape`	Shape of original data points
`pt.fill`	Colour fill of original data points
`line.size`	Size of the fitted data line
`line.colour`	Colour of the fitted data line
`linetype`	Type of the fitted data line
`resid.shape`	Shape of the residuals data points
`resid.size`	Size of the residuals data points
`resid.fill`	Colour fill of the residuals data points
`resid.colour`	Colour of the residuals data points

Value

A `patchwork` (ggplot2) object.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
wl <- as.numeric(names(mean_spectrum))
halpha <- tibble::as_tibble(as.list(mean_spectrum[wl > 653.5 & wl < 659.5]))
plot_fit(peak_fit(halpha, profile = "lorentzian"), title = "H-alpha 656.28 nm")
```

plot_influence *Influence Plot of a PCA Model*

Description

Plots the residual distance of each sample to a PCA model (Q residual or DModX) against its Hotelling's T^2, with their limits at each confidence level, from `q_residuals()` or `dmodx()`. It is drawn like the outlier map of a robust PCA, `plot_outlier_map()`.

Usage

```
plot_influence(
  x,
  label = NULL,
  labels = 3,
  relative = FALSE,
  shade = FALSE,
  log = FALSE,
  colour_by = c("type", "distance"),
  title = NULL,
  ...
)
```

Arguments

`x`	The result of `q_residuals()` or `dmodx()`.
`label`	The names of the samples: a vector with one value per sample. By default, their row numbers.
`labels`	The number of most outlying samples to label (beyond a limit at the highest confidence level). Default is 3; use 0 for no labels.
`relative`	If `TRUE`, plot the distances divided by their limits at the highest confidence level. Default is `FALSE`.
`shade`	If `TRUE`, shade the three outlying regions in grey, darker for more harmful samples (extreme, residual, both), and name them in their corners. Default is `FALSE`.
`log`	If `TRUE`, use logarithmic axes, which spread out the regular samples when a few are far away. Zero distances are drawn at the smallest positive distance. Default is `FALSE`.
`colour_by`	The colors of the points: `"type"` (default), by type, or `"distance"`, by their distance relative to the limits, $\max(T^2/T^2_{lim}, Q/Q_{lim})$, on a rainbow scale from dark red to blue, as in `plot_outlier_map()`.
`title`	The plot title.
`...`	Further arguments passed to `ggplot2::geom_point()` to style the points, such as `alpha` (default 0.85), `size` (2.2), `stroke` (0.4) or `colour` (the outline, `"black"`). `size` can also be a numeric vector with one value per sample, such as the concentration of an element, to vary the size of the points, with a legend.

Details

The limits at the highest confidence level (dashed lines) divide the plot into four types of samples:

- **regular** samples (bottom left);
- **extreme** samples (bottom right): a high T^2 but well described by the model, for example a high concentration;
- **residual** samples (top left): not described by the model (another matrix, a contamination, an instrumental problem);
- samples beyond **both** limits, the most harmful.

The limits at lower confidence levels are dotted, dot-dashed, ... lines.

With `relative = TRUE`, each distance is divided by its limit at the highest confidence level, so both limits are at 1 whatever the model, which puts plots of different models on the same scale.

Value

A ggplot object.

Author(s)

Christian L. Goueguel

See Also

`q_residuals()`, `dmodx()`, `plot_outlier_map()`

Examples

```
data(forageLIBS)
pca <- stats::prcomp(forageLIBS[-(1:14)])
influence <- q_residuals(pca, k = 3, conf_level = c(0.95, 0.99))
plot_influence(influence, label = forageLIBS$Measurement)
# the same options as plot_outlier_map(): distances relative to the limits,
# shaded outlying regions, logarithmic axes and colors by distance
plot_influence(influence, relative = TRUE, shade = TRUE, log = TRUE, colour_by = "distance")

# a single limit at 97.5%, with the outlying regions shaded
influence_975 <- q_residuals(pca, k = 3, conf_level = 0.975)
plot_influence(influence_975, label = forageLIBS$Measurement, shade = TRUE)
```

plot_lines *Overlay Emission Lines on a Spectrum*

Description

Plots a spectrum with vertical markers at the wavelengths of candidate emission lines, colored by ionization stage, to identify the lines of the spectrum.

Usage

```
plot_lines(
  spectrum,
  lines,
  shift = 0,
  scale_markers = TRUE,
  interactive = TRUE,
  title = NULL
)
```

Arguments

spectrum	A spectrum: a numeric vector named by wavelength, or a one-row matrix or data frame whose column names are the wavelengths. Average several spectra first.
lines	A tibble returned by libs_lines().
shift	A wavelength shift, in nm, added to the line wavelengths to correct the wavelength calibration of the spectrometer. Default is 0.
scale_markers	A logical: scale the markers by the relative intensity of the lines (TRUE, default) or draw them full height.
interactive	A logical: plotly (TRUE, default) or ggplot2.
title	An optional plot title.

Details

With interactive = TRUE, the plot is made with plotly: the spectrum is drawn with WebGL, the lines of each ionization stage form one trace that can be shown or hidden from the legend, and hovering over a line shows its species, wavelength, transition probability, upper-level energy and accuracy grade. Otherwise, a ggplot is returned.

With scale_markers = TRUE, the height of each marker is its relative intensity (see libs_lines()) times the maximum of the spectrum; otherwise all markers span the full height.

Value

A plotly or ggplot object.

See Also

libs_lines(), line_finder()

Examples

```
data(forageLIBS)
spectrum <- colMeans(forageLIBS[-(1:14)])
spectrum <- spectrum[as.numeric(names(spectrum)) > 403 & as.numeric(names(spectrum)) < 406]
# the K I doublet, with the atomic data of the NIST database
lines <- tibble::tibble(
  species = "K I", element = "K", stage = 1L, wavelength = c(404.414, 404.721),
  relative_intensity = c(1, 0.5), Aki = c(1.15e6, 1.07e6), Ek = c(3.065, 3.063),
  gk = c(4, 2), accuracy = "A", lower = "4s 2S", upper = c("5p 2P* 3/2", "5p 2P* 1/2")
)
plot_lines(spectrum, lines, interactive = FALSE)
```

---

plot_loadings *Loadings of a PCA as Spectra*

Description

Plots the loadings of a PCA against wavelength, one panel per component, and labels the wavelengths that contribute most to each component, with the emission lines they match when a line list is given. With type = "contribution", plots instead the variance of each wavelength explained by the components, which shows in one panel which lines drive the variance of the spectra.

Usage

```
plot_loadings(
  model,
  components = NULL,
  type = c("loadings", "contribution"),
  top = 10,
  lines = NULL,
  tol = 0.1,
  spectra = NULL,
  span = 5,
  interactive = FALSE,
  title = NULL
)
```

Arguments

model	A stats::prcomp() fit or an object returned by robpca(), rospca() or macropca(). The variable names must be the wavelengths; otherwise, the variables are numbered.

`components`	The components to show (numbers). Default is the first three (or all of them, if fewer).
`type`	`"loadings"` (default) for one panel of loadings per component, or `"contribution"` for the variance of each wavelength explained by `components`.
`top`	The number of peaks labeled per component (per panel). Default is 10; use 0 for no labels.
`lines`	Optional line list returned by `libs_lines()`, to label the peaks with the emission lines they match.
`tol`	The largest distance, in nm, between a peak and a line it matches. Default is 0.1.
`spectra`	Optional spectra (a data frame or matrix with the variables of the model, other columns being ignored) or a single spectrum (a named numeric vector), whose mean is drawn in grey behind each panel, rescaled to it, to show whether a peak is on an emission line.
`span`	The half-width, in channels, of the window in which a peak is a local extremum. Default is 5.
`interactive`	If `TRUE`, the plot is made with plotly, with the peaks and their candidate lines shown on hover. Default is `FALSE`.
`title`	The plot title.

Details

Loadings. Each loading is drawn red above zero and blue below. The wavelengths on the same side of zero increase together along the component; those on opposite sides vary in opposite directions, as with matrix effects or competing emitters. The sign of a component is arbitrary: each component is oriented so that its largest absolute loading is positive, which keeps the plots stable between fits.

Peaks. The `top` largest positive and negative local extrema of each loading are labeled with their wavelength (see `loading_peaks()`). A local extremum is the largest (or smallest) value within `span` channels on each side, on the same detector segment. Labels that would overlap are moved aside, with a leader line to their peak. A positive and a negative peak side by side on the same line (a derivative shape) usually comes from a shift or a broadening of the line between spectra rather than from a change in its intensity; `wavelength_calibration()` corrects shifts.

Emission lines. With a line list from `libs_lines()`, each peak within `tol` nm of lines is labeled with the nearest one (in bold). All the candidates are listed by `loading_peaks()`. Uncalibrated spectra can be shifted by more than `tol` from the tabulated lines: correct them with `wavelength_calibration()` or increase `tol`, and check the candidates. Only the lines of the species in the list can be matched.

Contribution. The variance of wavelength j explained by the components K is $s_j^2 \sum_{k \in K} \lambda_k p_{jk}^2$, with λ_k the variance of the scores of component k, p_{jk} the loading and s_j the scale of the variable (1 without scaling), in the squared units of the data.

Preprocessing. The loadings reflect the preprocessing. With centered spectra, the variance of a channel grows with its intensity, so strong lines dominate the loadings. With autoscaled spectra, every channel has the same variance, and weak lines and noise weigh as much as strong lines.

Explained variance. For a `stats::prcomp()` fit, the percentages in the panel titles are shares of the total variance. For `robpca()`, `rospca()` and `macropca()` fits, which estimate only the `k` components, they are shares of the variance of these `k` components.

Value

A ggplot object, or a plotly object if `interactive = TRUE`. The room left for the labels suits plots about 8 inches high; make the plot taller if labels are clipped.

Author(s)

Christian L. Goueguel

See Also

`loading_peaks()`, `libs_lines()`, `robpca()`, `plot_outlier_map()`

Examples

```
spectra_id <- forageLIBS |> dplyr::select(1:2) |> names()
minerals <- forageLIBS |> dplyr::select(3:14) |> names()
spectra <- forageLIBS |> dplyr::select(-dplyr::all_of(c(spectra_id, minerals)))
set.seed(1)
fit <- robpca(center(spectra))
plot_loadings(fit, spectra = spectra)
plot_loadings(fit, type = "contribution", spectra = spectra)

# label the peaks with emission lines from the NIST database (needs an
# internet connection)
lines <- try(libs_lines(c("Ca I", "Ca II", "K I", "Mg I", "Mg II", "Na I", "C I", "H I")))
if (!inherits(lines, "try-error")) plot_loadings(fit, lines = lines, spectra = spectra)
```

---

plot_outlier_map *Outlier Map of a Robust PCA*

Description

Plots the orthogonal distance of each observation against its score distance, for a robust PCA fitted by `robpca()`, `rospca()`, `macropca()` or `cellpca()` (Hubert, Rousseeuw and Vanden Branden, 2005). For `cellpca()`, the vertical axis is the norm of the standardized residuals, as in the enhanced outlier map of Centofanti, Hubert and Rousseeuw.

Usage

```
plot_outlier_map(
  object,
  newdata = NULL,
  labels = 3,
  relative = FALSE,
  shade = FALSE,
  log = FALSE,
```

```
  colour_by = c("type", "distance"),
  title = NULL,
  ...
)
```

Arguments

`object`	An object returned by `robpca()`, `rospca()`, `macropca()` or `cellpca()`.
`newdata`	Optional new observations to add to the map (a numeric matrix or data frame with the calibration variables).
`labels`	The number of most outlying observations to label (by their row names, or row numbers). Default is 3; use 0 for no labels.
`relative`	If `TRUE`, plot the reduced distances (divided by their cut-offs). Default is `FALSE`.
`shade`	If `TRUE`, shade the three outlying regions in grey, darker for more harmful observations (good leverage, orthogonal outliers, bad leverage), and name them in their corners. The region of regular observations stays white. Default is `FALSE`.
`log`	If `TRUE`, use logarithmic axes, which spread out the regular observations when a few are far away. Zero distances are drawn at the smallest positive distance. Default is `FALSE`.
`colour_by`	The colors of the points: `"type"` (default), by outlier type, or `"distance"`, by their reduced distance from the origin, $\max(SD/c_{SD}, OD/c_{OD})$, on a rainbow scale from dark red (the closest) to blue (the farthest). With `relative = TRUE`, where both cut-offs are at 1, yellow marks the cut-offs: dark red through orange for the regular observations, then green and blue for the outlying ones. Otherwise, the colors spread evenly over the distances.
`title`	The plot title.
`...`	Further arguments passed to `ggplot2::geom_point()` to style the points, such as `alpha` (default 0.85), `size` (2.2), `stroke` (0.4) or `colour` (the outline, `"black"`). `size` can also be a numeric vector with one value per sample (the calibration samples, then those of `newdata`), such as the concentration of an element, to vary the size of the points, with a legend.

Details

The dashed lines are the cut-offs of the two distances. They divide the map into four types of observations:

- **regular** observations (bottom left);
- **good leverage** points (bottom right): far from the center, but close to the PCA subspace, so they follow the structure of the data;
- **orthogonal outliers** (top left): close to the center once projected, but far from the PCA subspace;
- **bad leverage** points (top right): far on both counts, the most harmful outliers.

New observations (`newdata`) are projected onto the model with predict() and shown with the calibration cut-offs, which is how new spectra are screened before prediction.

With `relative = TRUE`, each distance is divided by its cut-off (the reduced score and orthogonal distances), so both cut-offs are at 1 whatever the model. This puts maps of different models (for example, with different numbers of components) on the same scale. A relative distance tells where an observation falls with respect to the cut-off, not how unlikely it is: twice the cut-off is not equally rare in every model.

Value

A ggplot object.

Author(s)

Christian L. Goueguel

See Also

`robpca()`, `rospca()`, `macropca()`, `plot_cell_map()`

Examples

```
spectra_id <- forageLIBS |> dplyr::select(1:2) |> names()
minerals <- forageLIBS |> dplyr::select(3:14) |> names()
set.seed(1)
fit <- forageLIBS |>
  dplyr::select(-dplyr::all_of(c(spectra_id, minerals))) |>
  center() |>
  robpca()

plot_outlier_map(fit, relative = TRUE, shade = TRUE, log = TRUE)
plot_outlier_map(fit, relative = TRUE, shade = TRUE, log = TRUE,
labels = 10, colour_by = "distance")
```

---

plot_outliers *Univariate Representation of Multivariate Outliers*

Description

This function creates a visual representation of multivariate outliers using a univariate plot.It uses robust covariance estimation methods to identify outliers and provides options for displaying the results through various plotting styles.

Usage

```
plot_outliers(
  x,
  quan = 1/2,
  alpha = 0.025,
  show.outlier = TRUE,
```

```
  show.mahal = FALSE
)
```

Arguments

x	A matrix or data frame.
quan	A numeric value, between 0.5 and 1, that specifies the amount of observations which are used for MCD estimations. Default is 0.5.
alpha	A numeric value specifying the amount of observations used for calculating the adjusted quantile. Default is 0.025.
show.outlier	A logical value, if TRUE (default), outliers are highlighted in the plot.
show.mahal	A logical value, if FALSE (default), robust Mahalanobis distances are not color-coded in the plot.

Details

The function uses the Minimum Covariance Determinant (MCD) method to compute robust estimates of location and scatter. It then applies an adaptive reweighting step to further improve the outlier detection. The results are visualized using ggplot2, with options to highlight outliers and/or color-code points based on their Mahalanobis distances.

Value

Depending on the combination of show.outlier and show.mahal:

- A ggplot object with outliers highlighted (if show.outlier = TRUE)
- A ggplot object with Mahalanobis distances color-coded (if show.mahal = TRUE)
- A ggplot object combining both outlier highlighting and Mahalanobis distance color-coding (if both show.outlier and show.mahal are TRUE)
- A tibble containing standardized scores, outlier flags, and robust multivariate Mahalanobis distances (if both show.outlier and show.mahal are FALSE)

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
# mineral contents (%) of the forage samples
contents <- forageLIBS[c("Ca", "Mg", "P", "K")]
plot_outliers(contents)
plot_outliers(contents, show.outlier = FALSE, show.mahal = TRUE)
# a data frame instead of a plot
head(plot_outliers(contents, show.outlier = FALSE, show.mahal = FALSE))
```

plot_som *Plot a Self-Organizing Map*

Description

Draws a map fitted by `som()`: the number of spectra per unit, the distances between neighboring prototypes, the component planes, the spectra on the map, or the prototype spectra.

Usage

```
plot_som(
  object,
  type = c("counts", "umatrix", "quality", "component", "mapping", "prototypes"),
  variables = NULL,
  colour = NULL,
  newdata = NULL,
  units = NULL,
  title = NULL
)
```

Arguments

`object`	A map fitted by `som()`.
`type`	The plot: `"counts"` (default), `"umatrix"`, `"quality"`, `"component"`, `"mapping"` or `"prototypes"`.
`variables`	For `"component"`: the variables, as wavelengths (the nearest channel is used) or column names.
`colour`	For `"mapping"`: a vector with one value per training spectrum, such as a concentration or a class.
`newdata`	For `"mapping"`: optional new spectra to place on the map.
`units`	For `"prototypes"`: the units whose prototypes are drawn.
`title`	The plot title.

Details

The types of plot are:

- `"counts"`: the number of training spectra on each unit (empty units in white).
- `"umatrix"`: the unified distance matrix, the mean distance of each prototype to those of its neighbors: high values (light) are the boundaries between groups of similar spectra, low values (dark) the groups themselves.
- `"quality"`: the mean quantization error of the spectra of each unit.
- `"component"`: the component planes, the value of the prototypes at each of the `variables` (one panel per variable): which units have intense emission lines. Each plane is scaled from its lowest (0) to its highest (1) unit, so that weak lines are as readable as strong ones.

- "mapping": the training spectra on their units (spread within the unit), colored by colour, and, with newdata, the new spectra as triangles, the novel ones circled in red.
- "prototypes": the prototype spectra of the units against wavelength. By default, six prototypes spread over the map: the unit with the most spectra, then, in turn, the unit whose prototype is the farthest from those already chosen.

Value

A ggplot object.

See Also

som(), predict.specproc_som()

Examples

```
data(forageLIBS)
# the Na I and K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which((wl > 585 & wl < 595) | (wl > 760 & wl < 780))]
fit <- som(spectra)
plot_som(fit, type = "umatrix")
# component planes of the K I and Na I lines
plot_som(fit, type = "component", variables = c(769.90, 589.59))
plot_som(fit, type = "mapping", colour = forageLIBS$K)
plot_som(fit, type = "prototypes")
```

---

plot_spectra *Plotting of Spectra*

Description

Spectrum plots are commonly x–y plots in which the x-axis represents the wavelength and the y-axis represents intensity of a spectrum's signal. The function allows to plot a spectrum or several spectra in a single plot, identified either by an id (for example, the samples or spectra id) or by a target variable (for example, the concentration of a chemical element).

Usage

```
plot_spectra(
  x,
  id = NULL,
  colvar = NULL,
  .interactive = FALSE,
  drop_na = FALSE,
  offset = NULL
)
```

Arguments

`x`	data frame or tibble of the spectra.
`id`	optional (`NULL` by default). Column name of a factor variable that identified each spectrum.
`colvar`	optional (`NULL` by default). Column name of a numeric variable to be display in color scale.
`.interactive`	optional (`FALSE` by default). When set to `TRUE` enables interactive plot.
`drop_na`	Optional (`FALSE` by default). Remove rows with NA intensity if drop_na is `TRUE`.
`offset`	Optional (`NULL` by default). The offsets between successive spectra, to separate them: a number (vertical offset, in intensity units), or two numbers `c(horizontal, vertical)` (in nm and intensity units). The spectrum of row `i` is shifted by `(i - 1)` times the offsets, so that the first one stays in place.

Details

This function is based on the ggplot2 package, thus allowing users to easily add or modify different components of the plot. Each row of `x` is drawn as a separate line. All columns other than `id` and `colvar` must be named by their wavelength.

Value

Object of class ggplot or of class plotly if `.interactive = TRUE`.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
wl <- as.numeric(names(forageLIBS)[-(1:14)])
k_lines <- names(forageLIBS)[-(1:14)][wl > 764 & wl < 772]
# the K I resonance lines, colored by potassium content
plot_spectra(forageLIBS[1:20, c("K", k_lines)], colvar = K)

# five spectra stacked, each shifted up by 20000 counts and right by 0.5 nm
plot_spectra(forageLIBS[1:5, c("Measurement", k_lines)], id = Measurement,
             offset = c(0.5, 20000))
```

---

plot_wavelength_calibration

Plot a Wavelength Calibration

Description

Plots the offset of each reference line (reference minus measured wavelength) against its measured wavelength, with the fitted correction of each detector segment. Lines not used are drawn as open circles and labeled with the reason.

Usage

```
plot_wavelength_calibration(object, title = NULL)
```

Arguments

object	The result of wavelength_calibration().
title	The plot title.

Value

A ggplot object.

See Also

wavelength_calibration()

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
first_detector <- spectra[seq_len(which(diff(wl) < 0)[1])]
reference <- c(`Mg I` = 285.213, `Ca II` = 317.933, `Ca II` = 393.366, `K I` = 404.414,
               `Ca I` = 422.673, `Mg I` = 518.360, `Na I` = 588.995, `Ca I` = 643.907)
cal <- wavelength_calibration(first_detector, reference)
plot_wavelength_calibration(cal)
```

---

poisson_scale *Poisson Scaling*

Description

This function performs Poisson scaling on a numeric matrix or data frame. Poisson scaling scales each variable by its square root mean value, with an optional scaling offset to avoid over-scaling when a variable has a near-zero mean.

Usage

```
poisson_scale(x, sc = NULL, drop.na = TRUE, options = list())
```

Arguments

x	A numeric matrix or data frame.
sc	A vector of previously calculated scales. If provided, these scales will be applied to the data.
drop.na	A logical value indicating whether to remove rows containing missing values. If TRUE (the default), such rows are removed. If FALSE, missing values are kept and ignored when computing the means.
options	A list of options for Poisson scaling. See the 'Options' section below.

Value

If sc is not provided, the function returns a list with the following components:

xs	The Poisson-scaled data.
sc	A vector of scales calculated for the given data.

If sc is provided, the function returns the Poisson-scaled data xs.

Options

The options argument is a list with the following fields:

- offset: A numeric value representing the percentage of the maximum mean value to be used as an offset on all scales. Avoids division by near-zero means. Default is 3.
- mode: An integer value (1 or 2) specifying the dimension of the data on which to calculate the mean value for scaling. 1 = mean over rows (to scale variables); 2 = mean over columns (to scale samples). Default is 1.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
res <- poisson_scale(spectra[1:300, ])
head(res$sc)
# the same scales for new spectra
poisson_scale(spectra[301:368, ], sc = res$sc)[1:3, 1:4]
```

---

predict.specproc_calibration

Predict from a Calibration Curve

Description

Computes the concentrations of new samples from their signal, by inverse prediction from a curve fitted with calibration_curve(), or the expected signal at given concentrations, with confidence or prediction intervals.

Usage

```
## S3 method for class 'specproc_calibration'
predict(
  object,
  newdata,
  replicates = 1,
  level = 0.95,
  type = "concentration",
  interval = "prediction",
  ...
)
```

Arguments

object	A curve fitted with calibration_curve().
newdata	For type = "concentration", the signals of the new samples: a numeric vector, or a data frame with the signal column used for the calibration. For type = "signal", their concentrations: a numeric vector or a data frame with the concentration column.
replicates	The number of replicate measurements averaged in each signal. Default is 1.
level	The confidence level of the intervals. Default is 0.95.
type	"concentration" (default, inverse prediction) or "signal".
interval	For type = "signal": "prediction" (default) or "confidence".
...	Not used.

Details

With type = "concentration" (default), the interval of each concentration combines the uncertainty of the curve with the noise of the new signal, the mean of replicates measurements (see calibration_curve()).

With type = "signal", newdata holds concentrations, and the interval is either the confidence interval of the mean signal (interval = "confidence") or the prediction interval of a new signal, the mean of replicates measurements (interval = "prediction", default). For a weighted curve, the noise of a new signal follows the weights at its concentration (numeric weights give new samples a weight of 1).

Value

For type = "concentration", a tibble with the signal, the predicted concentration, its standard error se, the interval (lower, upper) and whether it is below the limit of detection (below_lod) or of quantification (below_loq). For type = "signal", a tibble with the concentration, the predicted signal, its standard error se and the interval (lower, upper).

See Also

calibration_curve(), plot_calibration()

Examples

```
data(forageLIBS)
# the K I 769.90 nm line, normalized to the C I 247.86 nm line of the matrix
lines <- line_intensities(forageLIBS[-(1:14)], c(C = 247.856, K = 769.896), baseline = TRUE)
standards <- data.frame(
  K = forageLIBS$K,
  signal = lines$intensity[lines$line == "K"] / lines$intensity[lines$line == "C"]
)
cal <- calibration_curve(standards[1:300, ], signal, K)
predict(cal, standards$signal[301:305])
# the signal expected at given contents (% K)
predict(cal, c(1, 2, 3), type = "signal")
predict(cal, c(1, 2, 3), type = "signal", interval = "confidence")
```

---

predict.specproc_emsc *Apply EMSC to New Spectra*

Description

Corrects new spectra with the reference spectrum, polynomial basis and interferents of a model fitted by `emsc()`.

Usage

```
## S3 method for class 'specproc_emsc'
predict(object, newdata, ...)
```

Arguments

`object`	An object returned by `emsc()`.
`newdata`	A numeric matrix or data frame of new spectra, with the same variables as the calibration data.
`...`	Not used.

Value

A tibble of corrected spectra.

See Also

`emsc()`, `predict.specproc_filter()`

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
fit <- emsc(spectra[1:300, ], degree = 2)
corrected <- predict(fit, spectra[301:368, ])
dim(corrected)
```

---

predict.specproc_filter

Apply an Orthogonalization Filter to New Spectra

Description

Corrects new spectra with a filter estimated by epo(), osc(), direct_orthogonal(), direct_osc(), projected_osc() or o2pls(). The new spectra are preprocessed with the centers and scales of the calibration data, and the components estimated on the calibration data are removed. Estimating the filter on calibration data and applying it to validation data with predict() avoids the optimistic bias that arises when a supervised filter is fitted to all the data before validation.

Usage

```
## S3 method for class 'specproc_epo'
predict(object, newdata, ...)

## S3 method for class 'specproc_osc'
predict(object, newdata, ...)

## S3 method for class 'specproc_direct_orthogonal'
predict(object, newdata, ...)

## S3 method for class 'specproc_direct_osc'
predict(object, newdata, ...)

## S3 method for class 'specproc_projected_osc'
predict(object, newdata, ...)

## S3 method for class 'o2pls'
predict(object, newdata, ...)
```

Arguments

object	A filter returned by epo(), osc(), direct_orthogonal(), direct_osc(), projected_osc() or o2pls().
newdata	A numeric matrix or data frame of new spectra, with the same variables as the calibration data. If both have column names, the columns of newdata are matched by name.
...	Not used.

Details

For epo() the spectra are not centered, and the correction is $\mathbf{X}_{new} - \mathbf{X}_{new}\mathbf{V}\mathbf{V}^T$. For the other methods, the corrected spectra are centered (and scaled, if scale = TRUE was used), like the correction component of the fitted object:

- osc() with method = "wold" or "sjoblom", and o2pls(), remove the orthogonal components one at a time, because their weights refer to the deflated matrix.
- osc() with method = "fearn", direct_orthogonal(), direct_osc() and projected_osc() remove all components at once: $\mathbf{X}_{new} - \mathbf{X}_{new}\mathbf{W}\mathbf{P}^T$.

Applied to the calibration spectra, predict() returns the correction component of the fitted object.

Value

A tibble of corrected spectra, with the same columns as newdata.

Author(s)

Christian L. Goueguel

See Also

The recipe steps step_osc() and related functions, which apply these filters within a tidymodels workflow.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
x <- spectra[wl > 760 & wl < 780]  # the K I resonance lines
cal <- 1:300
fit <- osc(x[cal, ], forageLIBS$K[cal], method = "fearn", ncomp = 2)
corrected <- predict(fit, x[-cal, ])
dim(corrected)

# applied to the calibration spectra, predict() gives the correction
all.equal(predict(fit, x[cal, ]), fit$correction)
```

---

predict.specproc_macropca

Scores and Distances of New Observations

Description

Projects new observations onto a robust PCA model fitted by robpca(), rospca() or macropca(), and computes their score and orthogonal distances with the cut-offs of the calibration data.

Usage

```
## S3 method for class 'specproc_macropca'
predict(object, newdata, ...)

## S3 method for class 'specproc_cellpca'
predict(object, newdata, ...)

## S3 method for class 'specproc_robpca'
predict(object, newdata, ...)
```

Arguments

object	An object returned by robpca(), rospca() or macropca().
newdata	A numeric matrix or data frame with the same variables as the calibration data. For macropca(), missing values are allowed.
...	Not used.

Value

A tibble with the scores (PC1, ..., PCk), the score distance sd, the orthogonal distance od and the outlier_type of each new observation.

See Also

plot_outlier_map(), which can display new observations.

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
set.seed(1)
fit <- robpca(spectra[1:300, ])
head(predict(fit, spectra[301:368, ]))
```

---

predict.specproc_nas *Net Analyte Signal of New Samples*

Description

Computes the net analyte signal, selectivity and predicted concentration of new samples with a model fitted by nas().

Usage

```
## S3 method for class 'specproc_nas'
predict(object, newdata, ...)
```

Arguments

`object`	An object returned by `nas()`.
`newdata`	A numeric matrix or data frame of new spectra, with the same variables as the calibration spectra.
`...`	Not used.

Value

A tibble with one row per new sample and columns `nas`, `selectivity`, `predicted` and, if the model has a `noise` level, `snr`.

See Also

`nas()`

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
cal <- 1:300
fit <- nas(spectra[cal, ], forageLIBS$K[cal], ncomp = 3)
head(predict(fit, spectra[-cal, ]))
```

`predict.specproc_opls` *Predict with an OPLS Model*

Description

Applies an OPLS model fitted by `opls()` to new data: removes the orthogonal components (the OPLS filter), or predicts the response or the scores.

Usage

```
## S3 method for class 'specproc_opls'
predict(object, newdata, type = c("correction", "response", "scores"), ...)
```

Arguments

`object`	A model returned by `opls()`.
`newdata`	A numeric matrix or data frame of new data, with the same variables as the calibration data. If both have column names, the columns of `newdata` are matched by name.
`type`	The prediction: `"correction"` (default) the filtered data, like the `correction` component of the model; `"response"` the predicted response; `"scores"` the predictive (`p1`) and orthogonal (`o1`, ...) scores.
`...`	Not used.

Details

The new data are preprocessed with the centers and scales of the calibration data, and the orthogonal components are removed one at a time ($\mathbf{t}_o = \mathbf{X}\mathbf{w}_o$, $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{t}_o\mathbf{p}_o^T$). The predicted response is $\mathbf{X}\mathbf{w}c$, of the filtered data, back in the units of y.

Value

A tibble of filtered data or of scores, or a numeric vector of predicted responses.

See Also

opls(), step_opls()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
cal <- 1:300
fit <- opls(spectra[cal, ], forageLIBS$K[cal], ncomp = 2)
head(predict(fit, spectra[-cal, ], type = "response"))
```

---

predict.specproc_som *Map New Spectra onto a Self-Organizing Map*

Description

Places new spectra on the best-matching unit of a map fitted by som(), with their quantization error, and flags the spectra that fit no unit.

Usage

```
## S3 method for class 'specproc_som'
predict(object, newdata, ...)
```

Arguments

object	A map fitted by som().
newdata	A numeric matrix or data frame of spectra, with the variables of the training data.
...	Not used.

Value

A tibble with one row per spectrum: its best-matching unit, the coordinates x and y of the unit on the map, the quantization error qe, and novel, TRUE when the error exceeds the cut-off of the training spectra (the spectrum fits no known group).

See Also

som(), plot_som()

Examples

```
data(forageLIBS)
# the Na I and K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which((wl > 585 & wl < 595) | (wl > 760 & wl < 780))]
fit <- som(spectra[1:300, ])
head(predict(fit, spectra[301:368, ]))
```

---

predict.specproc_wavelength_calibration

Correct Wavelengths with a Wavelength Calibration

Description

Converts measured wavelengths to calibrated ones with the correction fitted by wavelength_calibration().

Usage

```
## S3 method for class 'specproc_wavelength_calibration'
predict(object, wavelength, ...)
```

Arguments

object	The result of wavelength_calibration().
wavelength	Measured wavelengths, in nm.
...	Not used.

Value

The corrected wavelengths. Each wavelength gets the correction of the detector segment whose range contains it (the first one where detectors overlap), or of the nearest segment.

See Also

wavelength_calibration(), apply_calibration()

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
first_detector <- spectra[seq_len(which(diff(wl) < 0)[1])]
reference <- c(`Mg I` = 285.213, `Ca II` = 317.933, `Ca II` = 393.366, `K I` = 404.414,
               `Ca I` = 422.673, `Mg I` = 518.360, `Na I` = 588.995, `Ca I` = 643.907)
cal <- wavelength_calibration(first_detector, reference)
# the corrected wavelengths of the Ca II H and K lines
predict(cal, c(393.3, 396.8))
```

---

projected_osc *Projected Orthogonal Signal Correction*

Description

The projected orthogonal signal correction (POSC) method is a preprocessing technique used to remove systematic variation from predictor variables that is orthogonal to the response variable. This function implements the POSC algorithm for model fitting and prediction.

Usage

```
projected_osc(
  x,
  y,
  ncomp = 4,
  center = TRUE,
  scale = FALSE,
  tol = 1e-10,
  newdata = NULL
)
```

Arguments

x	A matrix or data frame of the predictor variables.
y	A vector of the response variable.
ncomp	An integer specifying the number of orthogonal components removed (at least 1), as in the other orthogonalization methods. The underlying PLS model has ncomp + 1 components. Default is 4.
center	A logical value indicating whether to mean-center x and y. Default is TRUE.
scale	A logical value indicating whether to scale x and y. Default is FALSE.
tol	A numeric value; orthogonal components whose singular value is smaller than tol times the largest one are discarded. The default value is 1e-10.
newdata	An optional matrix or data frame of new predictor variables to be corrected using the POSC model. It is preprocessed with the centers and scales of x.

Details

POSC obtains OPLS-filtered data directly from an ordinary (non-orthogonalized) PLS1 model (Kemsley and Tapp, 2009):

1. A PLS1 model with `ncomp + 1` components is fitted, giving the score matrix $\mathbf{T}$ and the fitted response $\hat{\mathbf{y}}$.
2. The part of the score space orthogonal to $\hat{\mathbf{y}}$, $\mathbf{T} - \hat{\mathbf{y}}(\hat{\mathbf{y}}^T\hat{\mathbf{y}})^{-1}\hat{\mathbf{y}}^T\mathbf{T}$, spans `ncomp` orthogonal components with scores $\mathbf{T}_o$.
3. The orthogonal loadings are $\mathbf{P}_o = \mathbf{X}^T\mathbf{T}_o(\mathbf{T}_o^T\mathbf{T}_o)^{-1}$ and the filtered data are $\mathbf{X} - \mathbf{T}_o\mathbf{P}_o^T$.

The filtered data are identical to those obtained from an OPLS model with one predictive and `ncomp` orthogonal components (`opls()` with the same `ncomp`).

Value

An object of class `specproc_projected_osc` (a list), which predict() applies to new spectra, with the following components:

- `correction`: The corrected `x`.
- `scores`: The orthogonal scores matrix $\mathbf{T}_o$.
- `loadings`: The orthogonal loadings matrix $\mathbf{P}_o$.
- `weights`: The orthogonal weights $\mathbf{W}_o$, such that $\mathbf{T}_o = \mathbf{X}\mathbf{W}_o$.
- `center`, `scale`: The column centers and scales applied to `x`.
- `newdata`: If `newdata` is provided, a list with the corrected new data (`correction`) and its orthogonal scores (`scores`).

Author(s)

Christian L. Goueguel

References

- Kemsley, E.K., Tapp, H.S., (2009). OPLS filtered data can be obtained directly from non-orthogonalized PLS1. Journal of Chemometrics, 23(5):263-264.
- Trygg, J., Wold, S., (2002). Orthogonal projections to latent structures (O-PLS). Journal of Chemometrics, 16(3):119-128.

See Also

`predict.specproc_filter()` to correct new spectra with the fitted filter, and `step_projected_osc()` to use it in a tidymodels recipe.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
cal <- 1:300
res <- projected_osc(spectra[cal, ], forageLIBS$K[cal], ncomp = 2, newdata = spectra[-cal, ])
dim(res$newdata$correction)
```

`pseudo_voigt_profile` *Pseudo-Voigt Function*

Description

The Pseudo-Voigt function is an approximation for the Voigt function. It is defined as a linear combination of the Gaussian and Lorentzian functions, weighted by a mixing parameter η (`eta`) that determines the relative contribution of each component.

Usage

```
pseudo_voigt_profile(x, y0, xc, wG, wL, A, eta = NULL)
```

Arguments

`x`	A numeric vector representing the independent variable (e.g., wavelength or frequency).
`y0`	A numeric value specifying the baseline offset.
`xc`	A numeric value representing the center of the peak.
`wG`	A numeric value specifying the Gaussian full width at half maximum (FWHM).
`wL`	A numeric value specifying the Lorentzian FWHM.
`A`	A numeric value representing the peak area.
`eta`	A numeric value between 0 and 1, representing the mixing parameter. If `eta = NULL` (default), the mixing parameter is approximated by a polynomial function of `wG` and `wL`. This approximation (accuracy of 1%), proposed by Ida et al. (2000), provides an analytical expression for the mixing parameter based on the relative FWHM of the Gaussian and Lorentzian components. The polynomial approximation is useful when the precise value of `eta` is unknown.

Details

The Pseudo-Voigt function is given by:

$$y = y_0 + A \cdot [\eta \cdot L(x, x_c, w_L) + (1 - \eta) \cdot G(x, x_c, w_G)]$$

where:

- A is the peak area
- y_0 is the baseline offset
- $G(x, x_c, w_G)$ is the Gaussian component with center x_c and the full width at half maximum w_G
- $L(x, x_c, w_L)$ is the Lorentzian component with center x_c and the full width at half maximum w_L

- η is the mixing parameter, with $0 \leq \eta \leq 1$. The mixing parameter η allows the pseudo-Voigt function to describe a wide range of line shapes, from pure Gaussian ($\eta = 0$) to pure Lorentzian ($\eta = 1$), and various intermediate shapes in between.

Both G and L have unit area, so A is the area of the peak.

When `eta` is not given, the Thompson-Cox-Hastings approximation of the Voigt profile is used (Ida et al. 2000): both components share the total FWHM

$$f = (w_G^5 + 2.69269 w_G^4 w_L + 2.42843 w_G^3 w_L^2 + 4.47163 w_G^2 w_L^3 + 0.07842 w_G w_L^4 + w_L^5)^{1/5}$$

and the mixing parameter is

$$\eta = 1.36603(w_L/f) - 0.47719(w_L/f)^2 + 0.11116(w_L/f)^3$$

For comparison, the Olivero and Longbothum (1977) approximation, and its subsequent refinements (Belafhal 2000; Zdunkowski et al. 2007), estimate the FWHM of a Voigt line as $w_V = \frac{1}{2}[c_1 w_L + \sqrt{c_2 w_L^2 + 4 w_G^2}]$ with $c_1 = 1.0692$, $c_2 = 0.86639$ (accuracy of 0.02%).

Value

A numeric vector of the same length as `x`, containing the computed pseudo-Voigt function values. If `eta = NULL`, the function returns a list with the following components:

- `$y`: A numeric vector of the same length as `x`, containing the computed pseudo-Voigt function values using the approximated mixing parameter.
- `$eta`: The approximate mixing parameter calculated using the polynomial approximation based on the `wG` and `wL` values.

References

- Zdunkowski, W., Trautmann, T., Bott, A., (2007). Radiation in the Atmosphere — A Course in Theoretical Meteorology. Cambridge University Press.
- Belafhal, A., (2000). The shape of spectral lines: Widths and equivalent widths of the Voigt profile. Opt. Commun., 177(1-6):111–118.
- Olivero, J., Longbothum, R., (1977). Empirical fits to the Voigt line width: A brief review. J. Quant. Spectrosc. Radiat. Transfer, 17(2):233–236.
- Ida, T., Ando, M., Toraya, H., (2000). Extended pseudo-Voigt function for approximating the Voigt profile. Journal of Applied Crystallography. 33(6):1311–1316.

See Also

`voigt_profile()` for the exact Voigt profile.

Examples

```
x <- seq(-2, 2, length.out = 100)
y1 <- pseudo_voigt_profile(x, y0 = 0, xc = 0, wG = 1, wL = 0.5, A = 2, eta = 0)
y2 <- pseudo_voigt_profile(x, y0 = 0, xc = 0, wG = 1, wL = 0.5, A = 2, eta = 0.5)
y3 <- pseudo_voigt_profile(x, y0 = 0, xc = 0, wG = 1, wL = 0.5, A = 2, eta = 1)
y4 <- pseudo_voigt_profile(x, y0 = 0, xc = 0, wG = 1, wL = 0.5, A = 2)
```

```
plot(x, y1, type = "l", col = "red", main = "Pseudo-Voigt Profile", ylim = c(0, 2.5))
lines(x, y2, col = "blue")
lines(x, y3, col = "green")
lines(x, y4$y, col = "black")
legend("topright", legend = c("eta = 0", "eta = 0.5", "eta = 1", "eta = NULL"),
col = c("red", "blue", "green", "black"), lty = 1)

# The approximated mixing parameter:
y4$eta
```

---

q_residuals *Hotelling's T-squared and Q Residuals of a PCA Model*

Description

Computes, for each sample, the two distances used to monitor a principal component analysis (PCA) model: Hotelling's T^2, the distance within the model plane of the first k components, and the Q residual (squared prediction error, SPE), the squared distance to that plane, with their limits at one or more confidence levels.

Usage

```
q_residuals(
  model,
  k,
  newdata = NULL,
  conf_level = 0.975,
  method = "jackson",
  t2_method = "f",
  center = TRUE,
  scale = FALSE
)
```

Arguments

model	A stats::prcomp() fit that kept all its components (the default of prcomp()), or a numeric matrix or data frame, on which a PCA is fitted with center and scale.
k	The number of components of the model.
newdata	Optional new samples (a matrix or data frame with the variables of the model), whose distances are computed instead of those of the calibration samples.
conf_level	The confidence level(s) of the limits: one or more values between 0 and 1. Default is 0.975.
method	The limit of Q: "jackson" (default, Jackson-Mudholkar) or "box".
t2_method	The distribution of the T^2 limit of the samples of the model: "f" (default) or "beta" (see hotelling_t2()).
center, scale	Passed to stats::prcomp() when model is data. Default is TRUE and FALSE.

Details

With scores t_{ia} and eigenvalues λ_a,

$$T_i^2 = \sum_{a=1}^{k} \frac{t_{ia}^2}{\lambda_a}, \qquad Q_i = \|x_i - \hat{x}_i\|^2$$

where $\hat{x}_i$ is the reconstruction of the (centered and scaled) sample from the `k` components. A high T^2 is an extreme but well-modeled sample (for example, a high concentration); a high Q is a sample the model does not describe (another matrix, a contamination, an instrumental problem).

For the samples of the model, T^2 and its limits come from `hotelling_t2()` (F or Beta distribution, `t2_method`). For new samples, the limit is that of a new observation, $k(n+1)(n-1)/(n(n-k))\,F(k, n-k)$. The limit of Q is that of Jackson and Mudholkar (1979), from the eigenvalues of the components left out, or Box's (1954) scaled chi-square approximation (`method = "box"`); both tend to be slightly conservative. The Jackson-Mudholkar limit depends on a power h_0 of Q computed from these eigenvalues; when h_0 is close to zero (a few large eigenvalues followed by a long tail of small ones, common for spectra), its limit as $h_0 \to 0$, a lognormal approximation, is used. Samples are classified at the highest confidence level.

These are classical estimates, themselves affected by outliers: see `robpca()` and `plot_outlier_map()` for robust score and orthogonal distances. `dmodx()` gives the residual distance as a standard deviation, relative to that of the calibration samples.

Value

A tibble of class `specproc_influence`, with one row per sample: `sample` (row number), `t2`, its limits at each confidence level (in %, e.g. `t2_limit_97.5`), `q`, its limits (`q_limit_97.5`, ...) and `outlier`, the type of the sample at the highest confidence level: `"regular"`, `"extreme"` (high T^2 only), `"residual"` (high Q only) or `"both"`. Draw it with `plot_influence()`.

Author(s)

Christian L. Goueguel

References

- Jackson, J.E., Mudholkar, G.S. (1979). Control procedures for residuals associated with principal component analysis. Technometrics, 21(3):341-349.
- Box, G.E.P. (1954). Some theorems on quadratic forms applied in the study of analysis of variance problems, I. Annals of Mathematical Statistics, 25(2):290-302.
- Nomikos, P., MacGregor, J.F. (1995). Multivariate SPC charts for monitoring batch processes. Technometrics, 37(1):41-59.

See Also

`dmodx()`, `plot_influence()`, `hotelling_t2()`, `robpca()`

Examples

```
if (rlang::is_installed("HotellingEllipse", version = "1.3.0")) {
  data(forageLIBS)
  # the 380-430 nm window (Ca II H and K lines)
  wl <- suppressWarnings(as.numeric(names(forageLIBS)))
  spectra <- forageLIBS[which(wl > 380 & wl < 430)]
  pca <- stats::prcomp(spectra)
  influence <- q_residuals(pca, k = 3)
  influence[influence$outlier != "regular", ]
  plot_influence(influence, label = forageLIBS$Measurement)
  # several limits
  plot_influence(q_residuals(pca, k = 3, conf_level = c(0.95, 0.99)))
}
```

---

quantile_weight *Quantile Tail Weight Measure*

Description

This function calculates the left quantile weight (LQW) and the right quantile weight (RQW) for a given numeric vector. These weights serve as robust measures of tail heaviness, providing insights into the distribution's behavior in the left and right tails, respectively.

Usage

```
quantile_weight(x, p = 0.125, q = 0.875, drop.na = FALSE)
```

Arguments

x	A numeric vector.
p	A numeric value between 0 and 0.5 (p = 0.125 by default).
q	A numeric value between 0.5 and 1 (q = 0.875 by default).
drop.na	Logical value indicating whether to remove missing values (NA). If FALSE (default) and x contains missing values, NA is returned.

Details

The quantile weights, comprising the left quantile weight (LQW) and the right quantile weight (RQW), are robust measures of tail weight in probability distributions. They have a breakdown value of 12.5%, meaning that they are resistant to the influence of up to 12.5% of outliers or contaminated data.

The concept of quantile weights is derived from quartile skewness, introduced by D.V. Hinkley in 1975. Quartile skewness measures the skewness or asymmetry of a distribution by comparing the differences between the quartiles, which are robust measures of location and scale.

Specifically, the quantile weights are obtained by applying quartile skewness to either the left half or the right half of the probability mass, divided at the median of the univariate distribution (Brys *et al.* 2006):

$$LQW(p) = -\frac{Q((1-p)/2) + Q(p/2) - 2Q(0.25)}{Q((1-p)/2) - Q(p/2)}$$

$$RQW(q) = \frac{Q((1+q)/2) + Q(1-q/2) - 2Q(0.75)}{Q((1+q)/2) - Q(1-q/2)}$$

where Q is the quantile function of the data. The defaults $p = 0.125$ and $q = 0.875$ give a breakdown value of 12.5%.

Interpretation of Quantile Weights:

- At the normal distribution, LQW = RQW = 0.2 (approximately).
- Larger values indicate heavier tails than the normal distribution, and smaller values lighter tails.

Value

A tibble with two numeric columns:

- `LQW`: Left quantile weight.
- `RQW`: Right quantile weight.

Author(s)

Christian L. Goueguel

References

- Brys, G., Hubert, M., and Struyf, A. (2006). Robust measures of tail weight. Computational Statistics & Data Analysis, 50(3):733-759
- Hinkley, D.V., (1975). On power transformations to symmetry. Biometrika, 62(1):101–111.

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
moments::kurtosis(fe)   # classical
quantile_weight(fe)     # robust, left and right tails
```

read_starkb *Read Saved STARK-B Data*

Description

Reads Stark broadening parameters from an XSAMS file of the STARK-B database saved earlier: for example, the result of a query to the STARK-B VAMDC service downloaded in a web browser or through the VAMDC portal. This gives the same table as `starkb_lines()` without an internet connection, and makes an analysis reproducible with archived data.

Usage

```
read_starkb(file, wavelength = NULL, perturber = NULL)
```

Arguments

`file`	The path to an XSAMS (XML) file from STARK-B.
`wavelength`	An optional numeric vector of length 2: the wavelength range to keep, in nm.
`perturber`	An optional character vector of perturbers to keep, such as `"electron"`, `"H II"` (protons) or `"He II"`. By default all are kept.

Details

A query for one species can be downloaded from `http://stark-b.obspm.fr/12.07/vamdc/tap/sync?LANG=VSS2&REQUES` (with the element and the ion charge, 0 for neutral atoms, adapted). Cite the STARK-B database and the original publications, listed in the `source` column, as for `starkb_lines()`.

Value

A tibble as returned by `starkb_lines()`.

See Also

`starkb_lines()`, `stark_table()` for data from other sources.

Examples

```
# a minimal file in the STARK-B format, with made-up values
file <- tempfile(fileext = ".xml")
writeLines(c(
  '<XSAMSData xmlns="http://vamdc.org/xml/xsams/1.0">',
 '<Sources><Source sourceID="B1"><Title>Example</Title><Year>2000</Year></Source></Sources>',
  '<Environments><Environment envID="E1"><Temperature><Value units="K">10000</Value>',
 '</Temperature><TotalNumberDensity><Value units="1/cm3">1e17</Value></TotalNumberDensity>',
 '<Composition><Species name="electron" speciesRef="XP1"/></Composition></Environment>',
 '</Environments><Species><Atoms><Atom><ChemicalElement><ElementSymbol>Ca</ElementSymbol>',
 '</ChemicalElement><Isotope><Ion speciesID="X1"><IonCharge>1</IonCharge></Ion></Isotope>',
  '</Atom></Atoms></Species><Processes><Radiative><RadiativeTransition id="P1">',
```

```
  '<SourceRef>B1</SourceRef><EnergyWavelength><Wavelength><Value units="A">3950</Value>',
  '</Wavelength></EnergyWavelength><SpeciesRef>X1</SpeciesRef>',
  '<Broadening name="pressure" envRef="E1"><Lineshape name="Lorentzian">',
  '<LineshapeParameter name="gammaL"><Value units="A">0.2</Value></LineshapeParameter>',
  '</Lineshape></Broadening></RadiativeTransition></Radiative></Processes></XSAMSData>'
), file)
read_starkb(file)
```

`reject_shots` *Rejection of Outlying Laser Shots*

Description

Flags the outlying shots (spectra) of each sample of a LIBS dataset, such as shots with a weak or missed plasma, or shots on an inclusion or a contaminated spot, before the shots are averaged.

Usage

```
reject_shots(
  data,
  sample,
  method = c("intensity", "correlation"),
  cutoff = 3.5,
  wavelength = NULL,
  drop = FALSE
)
```

Arguments

`data`	A data frame with one shot per row: a sample column and spectral columns named by their wavelengths (other columns are kept but not used).
`sample`	The column identifying the sample of each shot, unquoted or as a string.
`method`	The criteria: one or more of `"intensity"`, `"correlation"` (both by default) and `"distance"`.
`cutoff`	The robust z-score above which a shot is rejected. Default is 3.5.
`wavelength`	An optional wavelength range (nm) in which the criteria are computed, for example to avoid saturated lines.
`drop`	A logical: return only the kept shots, without the added columns (`FALSE`, default).

Details

Each shot is compared with the other shots of its sample, by robust z-scores (deviation from the median of the sample divided by the MAD) of one or more criteria:

- "intensity": the total intensity of the spectrum. Both weak and unusually strong shots are flagged.
- "correlation": the Pearson correlation of the spectrum with the median spectrum of the sample, which detects changes of the shape of the spectrum (other lines, other line ratios) whatever its intensity. Only shots with a low correlation are flagged.
- "distance": the Euclidean distance of the spectrum to the median spectrum of the sample, which combines both. Only far shots are flagged.

A shot is rejected when any criterion exceeds cutoff (3.5 by default, following Iglewicz and Hoaglin, 1993). Samples with fewer than 3 shots are not checked. When all shots of a sample vary by the same amount, the MAD can be zero; the mean absolute deviation is then used.

To average the kept shots per sample, use average() on the result with drop = TRUE. In a recipe, use step_reject_shots().

Value

A tibble: data with a logical column .rejected, the criteria that rejected each shot in .reason (NA for kept shots) and the robust z-score of each criterion (.intensity_z, .correlation_z, .distance_z). With drop = TRUE, the kept rows of data.

Author(s)

Christian L. Goueguel

References

- Iglewicz, B., Hoaglin, D.C. (1993). How to Detect and Handle Outliers. ASQC Quality Press, Milwaukee.

See Also

step_reject_shots(), average(), saturation_summary()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
# five shots of each of four samples, one of them with a weak plasma
set.seed(1)
shots <- spectra[rep(1:4, each = 5), ] * stats::runif(20, 0.95, 1.05)
shots[7, ] <- 0.2 * shots[7, ]
shots <- cbind(sample = rep(c("A", "B", "C", "D"), each = 5), shots)
res <- reject_shots(shots, sample)
res[res$.rejected, c("sample", ".reason", ".intensity_z")]
# the kept shots only
nrow(reject_shots(shots, sample, drop = TRUE))
```

robpca *Robust Principal Component Analysis (ROBPCA)*

Description

Robust PCA by the ROBPCA algorithm of Hubert, Rousseeuw and Vanden Branden (2005), which combines projection pursuit with the minimum covariance determinant (MCD) estimator. It resists up to `n - h` outlying spectra and works when there are more variables than observations. The computational kernels (outlyingness, FAST-MCD) are written in C++.

Usage

```
robpca(
  x,
  k = NULL,
  kmax = 10,
  alpha = 0.75,
  ndir = 250,
  var_explained = 0.8,
  nsamp = 500
)
```

Arguments

`x`	A numeric matrix or data frame, with one observation per row.
`k`	The number of principal components. If `NULL` (default), it is chosen from `var_explained` and `kmax`.
`kmax`	The maximum number of components. Default is 10.
`alpha`	The robustness parameter: the fraction of observations the estimates are based on, between 0.5 and 1. Default is 0.75. The subset size is $h = \max(\lfloor \alpha n \rfloor, \lfloor (n + k_{max} + 1)/2 \rfloor)$.
`ndir`	The number of random directions used for the outlyingness. Default is 250; use `"all"` for all directions through pairs of observations.
`var_explained`	The fraction of variance used to choose `k` when it is not given. Default is 0.8.
`nsamp`	The number of random subsets of FAST-MCD. Default is 500.

Details

The algorithm follows the published description:

1. The data are reduced by a singular value decomposition to the affine subspace they span (at most $n - 1$ dimensions).
2. The Stahel-Donoho outlyingness of every observation is computed over directions through pairs of observations, using the univariate MCD location and scale on each direction. The `h` least outlying observations form the subset H_0.

3. The number of components k is chosen (if not given) as the smallest number that explains var_explained of the variance of H_0, with at most kmax. Observations whose orthogonal distance to the k-dimensional PCA subspace of H_0 is below the cut-off form H_1, and the subspace is re-estimated from H_1.
4. All observations are projected onto this subspace, and the reweighted FAST-MCD estimator of the scores gives the final center, loadings and eigenvalues.

Each observation then has a score distance (SD), its robust Mahalanobis distance within the PCA subspace, and an orthogonal distance (OD) to the subspace. The cut-off for the SD is $\sqrt{\chi^2_{k,0.975}}$; the cut-off for the OD uses the Wilson-Hilferty approximation, with the univariate MCD of $OD^{2/3}$. plot_outlier_map() plots both.

This is an independent implementation of the published algorithm, not a port of the rospca or rrcov code, so its results can differ slightly from theirs (random directions and subsets, and small-sample correction factors, which are not applied here). Use set.seed() for reproducible results.

Value

An object of class specproc_robpca, a list with:

- loadings: $p \times k$ matrix of robust loadings.
- eigenvalues: robust eigenvalues (variances of the scores).
- center: robust center.
- scores: $n \times k$ matrix of scores.
- sd, od: score and orthogonal distances of each observation.
- cutoff_sd, cutoff_od: their cut-offs.
- outlier_type: a factor classifying each observation as "regular", "good leverage" (high SD only), "orthogonal outlier" (high OD only) or "bad leverage" (both).
- k, h, alpha: the settings used.
- H0, H1: logical vectors of the observations in the subsets H_0 and H_1.

Use predict() for the scores and distances of new observations.

Author(s)

Christian L. Goueguel

References

- Hubert, M., Rousseeuw, P.J., Vanden Branden, K. (2005). ROBPCA: a new approach to robust principal component analysis. Technometrics, 47(1):64-79.
- Rousseeuw, P.J., Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator. Technometrics, 41(3):212-223.

See Also

rospca() for sparse loadings, macropca() for data with cellwise outliers or missing values, step_robpca() for a recipe step, plot_outlier_map().

Examples

```
set.seed(1)
# the 380-430 nm window (Ca II H and K lines)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
forageLIBS[which(wl > 380 & wl < 430)] |>
  center() |>
  robpca() |>
  print()
```

---

robust_bcyj *Robust Box-Cox and Yeo-Johnson Transformation*

Description

Transforms each variable in a dataset toward central normality using re-weighted maximum likelihood to robustly fit the Box-Cox or Yeo-Johnson transformation.

Usage

```
robust_bcyj(x, var = NULL, type = "bestObj", quantile = 0.99, nbsteps = 2)
```

Arguments

x	A data frame or tibble containing the variables to be transformed.
var	A vector of character or numeric variable names to be transformed. If NULL (default), all columns are selected.
type	A character string specifying the transformation method(s) to use. Allowed values are "BC", "YJ", or "bestObj" (default).
quantile	A numeric value between 0 and 1 specifying the quantile to use for determining the weights in the re-weighting step. Default is 0.99.
nbsteps	An integer specifying the number of re-weighting steps to perform. Default is 2.

Details

The Box-Cox and Yeo-Johnson transformations are power transformations aimed at making the data distribution more normal-like. The Box-Cox transformation is suitable for strictly positive values, while the Yeo-Johnson transformation can handle both positive and negative values. The transformations are fitted robustly (Raymaekers and Rousseeuw, 2021), so that outlying observations do not drive the estimate of λ:

1. Each variable is pre-standardized: divided by its median for Box-Cox, which needs strictly positive values, and centered by its median and divided by its MAD for Yeo-Johnson.
2. An initial λ (between -4 and 6) minimizes a robust distance (Tukey's biweight) between the sorted, robustly standardized transformed values and the quantiles of the normal distribution. For this step, the transformation is continued linearly on the side that it compresses (the upper side for $\lambda < 1$, the lower side for $\lambda > 1$), beyond the point where the transformed value is 1.5 times that of the quartile, so that outliers on that side cannot dictate λ.

3. The values whose standardized transformed value exceeds the `quantile` of the normal distribution are given weight zero (first with the rectified transformation), and λ is re-estimated by maximum likelihood on the others, `nbsteps` times.
4. The transformed variable is standardized by the mean and standard deviation of these inliers.

Variables with fewer than 5 values or no spread (zero MAD) are left unchanged (method `"none"`), as are variables with non-positive values when `type = "BC"`.

The `type` parameter controls which transformation method(s) to use:

- "BC": Only applies the Box-Cox transformation to strictly positive variables.
- "YJ": Only applies the Yeo-Johnson transformation to all variables.
- "bestObj" (default): For strictly positive variables, both BC and YJ are applied, and the solution with the lowest objective function value is kept. For variables with negative values, only YJ is applied.

Value

A list containing two data frames:

- `summary`:
  - `variable`: the variable(s) name
  - `lambda`: the estimated lambda parameter
  - `method`: the method used ('BC' for Box-Cox, 'YJ' for Yeo-Johnson, or 'none')
  - `objective`: the objective function value
- `transformation`:
  - the transformed variable(s)

Author(s)

Christian L. Goueguel

References

- Raymaekers, J., Rousseeuw, P.J., (2021). Transforming variables to central normality. Machine Learning, https://doi.org/10.1007/s10994-021-05960-5.
- Box, G. E. P., Cox, D. R. (1964). An analysis of transformations. Journal of the Royal Statistical Society, Series B, 26:211–252.

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
# two lines, transformed to central normality
res <- robust_bcyj(spectra[, c(10, 50)])
res$summary
```

rospca *Robust Sparse Principal Component Analysis (ROSPCA)*

Description

Robust sparse PCA by the ROSPCA approach of Hubert, Reynkens, Schmitt and Verdonck (2016): outliers are detected as in `robpca()`, and sparse loadings are computed from the clean observations by a grid-search sparse PCA (Croux, Filzmoser and Fritz, 2013). Sparse loadings are zero for most variables, which makes the components easier to interpret, for example as a few emission lines. The computational kernels are written in C++.

Usage

```
rospca(
  x,
  k = 2,
  lambda = 1,
  alpha = 0.75,
  ndir = 250,
  stand = FALSE,
  ngrid = 10,
  maxiter = 10
)
```

Arguments

`x`	A numeric matrix or data frame, with one observation per row.
`k`	The number of principal components. Default is 2.
`lambda`	A non-negative number: the sparsity parameter. Default is 1.
`alpha`	The robustness parameter: the fraction of observations the estimates are based on, between 0.5 and 1. Default is 0.75. The subset size is $h = \max(\lfloor \alpha n \rfloor, \lfloor (n + k_{max} + 1)/2 \rfloor)$.
`ndir`	The number of random directions used for the outlyingness. Default is 250; use `"all"` for all directions through pairs of observations.
`stand`	A logical value: standardize the variables robustly (median and Q_n) before the analysis (`FALSE`, default) or only center them by their median.
`ngrid`	The number of angles in the grid search. Default is 10.
`maxiter`	The maximum number of grid refinements per component. Default is 10.

Details

The algorithm follows the published description in three steps:

1. **Outlier detection.** The variables are robustly standardized (median and Q_n) if `stand = TRUE`. As in `robpca()`, the `h` least outlying observations form H_0, and those whose orthogonal distance to the k-dimensional PCA subspace of H_0 is below the cut-off form H_1

2. **Sparsification.** The observations of H_1 are standardized and the sparse loadings are computed by maximizing, component by component, the variance of the scores minus `lambda` times the L_1 norm of the loadings. Variables with zero loadings on all components are set aside; the observations whose orthogonal distance to the sparse subspace is below the cut-off form H_2, and the sparse loadings are recomputed from H_2, standardized in turn.
3. **Eigenvalues and center.** The eigenvalues are first estimated by the squared Q_n of the scores of H_2. The `h` observations of H_2 with the smallest score distances form H_3; the center is their mean, and the final eigenvalues are the variances of their scores. The components are sorted by decreasing eigenvalue.

Larger `lambda` values give sparser loadings. The loadings apply to the data standardized by `center` and `scale`; they are unit vectors, but those of different components are not exactly orthogonal. The eigenvalues are the variances of the scores in these standardized units. `lambda` is best chosen by validation, for example with `step_rospca()` and tune.

This is an independent implementation of the published description, not a port of the rospca code, so results can differ from those of `rospca::rospca()`.

Value

An object of class `specproc_rospca` (inheriting from `specproc_robpca`), with the components described in `robpca()`, and:

- `scale`: the scales that standardize the variables before projection: the robust scale (Q_n, if `stand = TRUE`) times the standard deviation in H_2.
- `lambda`: the sparsity parameter.
- `H2`, `H3`: logical vectors of the observations in these subsets.

Author(s)

Christian L. Goueguel

References

- Hubert, M., Reynkens, T., Schmitt, E., Verdonck, T. (2016). Sparse PCA for high-dimensional data with outliers. Technometrics, 58(4):424-434.
- Croux, C., Filzmoser, P., Fritz, H. (2013). Robust sparse principal component analysis. Technometrics, 55(2):202-214.

See Also

`robpca()`, `step_rospca()`, `plot_outlier_map()`

Examples

```
set.seed(1)
# the 380-430 nm window (Ca II H and K lines)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
forageLIBS[, which(wl > 380 & wl < 430)] |>
  center() |>
```

```
rospca() |>
print()
```

---

rousseeuw_croux *Rousseeuw-Croux Scale Estimators*

Description

This function calculates either the Rousseeuw-Croux Sn or Qn scale estimator.

Usage

```
rousseeuw_croux(x, estimator = c("Sn", "Qn"), drop.na = FALSE)
```

Arguments

x	A numeric vector of data values.
estimator	A character string indicating whether to calculate the "Sn" or "Qn" estimator.
drop.na	A logical value indicating whether to remove missing values (NA) from the input vector. If FALSE (default) and x contains missing values, NA is returned.

Details

The Sn and Qn estimators, proposed by Rousseeuw and Croux (1993), are robust measures of scale (or dispersion) that are designed to have a high breakdown point, which is a desirable property for estimators in the presence of outliers or heavy-tailed distributions. Specifically, both estimators have a breakdown point of 50%. The Sn estimator is based on the median, while the Qn estimator is based on a weighted combination of order statistics. Both estimators are consistent estimators of the population scale parameter under normality. Although, the function uses the robustbase package for estimating Sn and Qn, the bias-correction factors used in the calculations have been, however, refined according to Akinshin, A., (2022).

Value

A numeric value representing the calculated Sn or Qn scale estimator.

Author(s)

Christian L. Goueguel

References

- Rousseeuw, P.J. and Croux, C. (1993). Alternatives to the median absolute deviation. Journal of the American Statistical Association, 88(424):1273-1283.
- Akinshin, A., (2022). Finite-sample Rousseeuw-Croux scale estimators. arxiv:2209.12268v1.

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
c(sd = stats::sd(fe), mad = stats::mad(fe),
  Sn = rousseeuw_croux(fe, estimator = "Sn"), Qn = rousseeuw_croux(fe, estimator = "Qn"))
```

---

saha_boltzmann *Plasma Temperature from a Saha-Boltzmann Plot*

Description

Estimates the temperature of a plasma in LTE from lines of the neutral atom and of the singly charged ion of the same element, by a Saha-Boltzmann plot.

Usage

```
saha_boltzmann(
  lines,
  ionization_energy,
  electron_density,
  units = "energy",
  max_iter = 100,
  tol = 1e-08
)
```

Arguments

lines	A data frame as for boltzmann(), with an additional column stage: 1 for lines of the neutral atom, 2 for lines of the singly charged ion.
ionization_energy	The ionization energy of the neutral atom, in eV.
electron_density	The electron density, in cm^{-3}, for example from electron_density().
units	The units of intensity: "energy" (default) or "photons".
max_iter	The maximum number of iterations. Default is 100.
tol	The relative tolerance on the temperature. Default is 1e-8.

Details

Lines of the ion are placed on the Boltzmann plot of the neutral atom with the Saha equation. For ionic lines, the abscissa becomes $E_k + E_{ion}$ and the ordinate

$$y^* = \ln \frac{I\lambda}{g_k A_{ki}} - \ln \left(\frac{2}{N_e} \left(\frac{2\pi m_e k_B T}{h^2} \right)^{3/2} \right)$$

where E_{ion} is the ionization energy of the neutral atom and N_e the electron density (Aguilera and Aragón, 2004). Because the correction depends on T, the fit is iterated until the temperature converges. The lowering of the ionization energy in the plasma is neglected. The much wider energy range than in a Boltzmann plot of a single species gives a more precise temperature.

Value

An object of class `specproc_boltzmann`, as for `boltzmann()`, with the number of iterations in `iterations`.

References

- Aguilera, J.A., Aragón, C. (2004). Characterization of a laser-induced plasma by spatially resolved spectroscopy of neutral atom and ion emissions: comparison of local and spatially integrated measurements. Spectrochimica Acta Part B, 59(12):1861-1876.

See Also

`boltzmann()`, `electron_density()`, `plot_boltzmann()`

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
# calcium lines, with their atomic data from the NIST database
atomic <- data.frame(
  species = c(rep("Ca I", 6), rep("Ca II", 4)),
  stage = c(rep(1, 6), rep(2, 4)),
  wavelength = c(428.301, 430.253, 431.865, 445.478, 612.222, 616.217,
                 315.887, 317.933, 370.603, 373.690),
  Aki = c(4.34e7, 1.36e8, 7.40e7, 8.70e7, 2.87e7, 4.77e7, 3.10e8, 3.60e8, 8.80e7, 1.70e8),
  gk = c(5, 5, 3, 7, 3, 3, 4, 6, 2, 2),
  Ek = c(4.780, 4.780, 4.769, 4.681, 3.910, 3.910, 7.047, 7.050, 6.468, 6.468)
)
lines <- line_intensities(mean_spectrum, atomic, baseline = TRUE)
# with the ionization energy of Ca I (eV) and the electron density (cm-3)
fit <- saha_boltzmann(lines, ionization_energy = 6.113, electron_density = 1.9e17)
fit
plot_boltzmann(fit)
```

`saturation_summary` *Detect Saturated Channels in Spectra*

Description

Finds the channels at the saturation limit of the detector, for each spectrum and for each wavelength. Saturated lines are clipped, so their intensity and width are wrong and they respond nonlinearly to concentration.

Usage

```
saturation_summary(x, limit = 65535, tolerance = 0)
```

Arguments

x	A numeric matrix or data frame of spectra, one per row, with the wavelengths as column names.
limit	The saturation limit of the detector, in counts. Default is 65535 (16-bit detectors).
tolerance	Channels within tolerance counts of limit are counted as saturated. Default is 0.

Value

A list with two tibbles:

- spectra: for each spectrum (row), the number n_saturated and fraction of saturated channels;
- channels: for each channel saturated in at least one spectrum, its wavelength (from the column names; NA if they are not numeric), column index, and the number and fraction of saturated spectra.

Examples

```
data(forageLIBS)
sat <- saturation_summary(forageLIBS[-(1:14)], limit = 65535)
sat$channels
```

---

savgol_derivative *Derivative Order of a Savitzky-Golay Filter*

Description

A dials parameter for the derivative of step_savgol(): 0 (smoothing), 1 or 2.

Usage

```
savgol_derivative(range = c(0L, 2L), trans = NULL)
```

Arguments

range	The range of derivative orders. Default is 0 to 2.
trans	Not used.

Value

A dials quant_param object.

See Also

step_savgol()

Examples

```
savgol_derivative()
dials::value_seq(savgol_derivative(), 3)
```

---

savitzky_golay *Savitzky-Golay Smoothing and Derivatives*

Description

Smooths spectra, or computes their first or second derivative, with the Savitzky-Golay filter: a polynomial of degree `order` fitted by least squares in a moving window of `window` channels.

Usage

```
savitzky_golay(x, window = 11, order = 2, derivative = 0, segments = TRUE)
```

Arguments

`x`	A numeric matrix or data frame of spectra, one per row, or a numeric vector (one spectrum). Their names are the wavelengths.
`window`	The width of the window, an odd number of channels. Default is 11.
`order`	The degree of the polynomial, smaller than `window`. Default is 2.
`derivative`	The derivative: 0 (smoothing, default), 1 or 2. It must not exceed `order`.
`segments`	A logical: filter the segments between gaps of the wavelength axis separately (`TRUE`, default). Needs wavelengths as names.

Details

Smoothing reduces noise, at the cost of broadening narrow lines when the window is wide compared with the line width (a higher `order` preserves the line shape better, and smooths less). Derivatives remove constant (first derivative) or linear (second derivative) baselines and separate overlapping features, but amplify noise, which the polynomial fit counteracts.

Every channel is kept: at the ends of a spectrum, the values come from the polynomial fitted to the first (or last) `window` channels. When `segments = TRUE`, the spectrum is split between the detectors of a multi-spectrometer system (where the wavelengths step back, or jump by more than 5 times the median spacing), and each segment is filtered on its own; segments shorter than `window` are returned as `NA`, with a warning.

Derivatives are per channel. For derivatives per nm, divide the first derivative by the channel spacing (and the second by its square).

Value

The filtered spectra, with the same shape and names as `x` (a tibble for a data frame).

Author(s)

Christian L. Goueguel

References

- Savitzky, A., Golay, M.J.E. (1964). Smoothing and differentiation of data by simplified least squares procedures. Analytical Chemistry, 36(8):1627-1639.
- Rinnan, A., van den Berg, F., Engelsen, S.B. (2009). Review of the most common pre-processing techniques for near-infrared spectra. Trends in Analytical Chemistry, 28(10):1201-1222.

See Also

`step_savgol()`

Examples

```
data(forageLIBS)
spectrum <- unlist(forageLIBS[1, -(1:14)])
smooth <- savitzky_golay(spectrum, window = 7)
first <- savitzky_golay(spectrum, window = 11, derivative = 1)
wl <- as.numeric(names(spectrum))
keep <- wl > 400 & wl < 410
plot(wl[keep], spectrum[keep], type = "l", col = "grey", xlab = "Wavelength (nm)",
     ylab = "Intensity")
lines(wl[keep], smooth[keep], col = "blue")
```

`self_absorption` *Self-Absorption Coefficient from Line Widths*

Description

Estimates the self-absorption coefficient of an emission line from the ratio of its measured Stark width to the width it would have if it were optically thin (El Sherbini et al., 2005).

Usage

```
self_absorption(width, thin_width, alpha = -0.54)
```

Arguments

`width`	The measured Lorentzian full width at half maximum of the line, in nm.
`thin_width`	The width of the line if it were optically thin, in nm.
`alpha`	The exponent of the relation. Default is -0.54.

Details

Self-absorption flattens and broadens a line. The self-absorption coefficient, the ratio of the measured peak intensity to the intensity without absorption, is

$$SA = \left(\frac{\Delta\lambda}{\Delta\lambda_0}\right)^{1/\alpha}, \quad \alpha = -0.54$$

where $\Delta\lambda$ is the measured Lorentzian (Stark) width and $\Delta\lambda_0$ the width of the optically thin line. The latter follows from the electron density measured on an optically thin line (such as Hα, see `electron_density()`) and the Stark width of the line at that density (`stark_width()`). $SA = 1$ means no self-absorption; the measured intensity divided by SA corrects it.

Value

A tibble with the self-absorption coefficient `SA` and the `intensity_correction` factor $1/SA$.

References

- El Sherbini, A.M., El Sherbini, Th.M., Hegazy, H., Cristoforetti, G., Legnaioli, S., Palleschi, V., Pardini, L., Salvetti, A., Tognoni, E. (2005). Evaluation of self-absorption coefficients of aluminum emission lines in laser-induced breakdown spectroscopy measurements. Spectrochimica Acta Part B, 60(12):1573-1579.

See Also

`electron_density()`, `stark_width()`

Examples

```
# a line twice as wide as expected from its Stark width
self_absorption(width = 0.04, thin_width = 0.02)
```

`snv` *Standard Normal Variate*

Description

This function performs Standard Normal Variate (SNV) scaling on the input spectral data. SNV scaling scales each row of the input data to have zero mean and unit standard deviation. This is equivalent to autoscaling the transpose of the input data.

Usage

```
snv(x, drop.na = TRUE)
```

Arguments

`x`	A numeric matrix or data frame.
`drop.na`	A logical value indicating whether to remove spectra (rows) containing missing values. If `TRUE` (the default), such rows are removed.

Value

A list with the following components:

`correction`	The SNV-scaled data.
`means`	A vector of row means.
`stds`	A vector of row standard deviations.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]  # the spectral channels
corrected <- snv(spectra)$correction
corrected[1:3, 1:4]
```

---

som — *Self-Organizing Map of Spectra*

Description

Maps spectra onto a two-dimensional grid of units with a self-organizing map (SOM; Kohonen, 1982), so that similar spectra land on the same or on neighboring units. Each unit holds a prototype (codebook vector), which is itself a spectrum. The map is useful for sorting samples quickly (scrap metals, mineral phases), and it follows nonlinear relations between spectra (matrix effects, self-absorption, changes of the plasma) that a linear projection such as PCA can miss. New spectra are placed on their best-matching unit with predict(), and a large quantization error flags a spectrum that fits no known group.

Usage

```
som(
  x,
  grid = NULL,
  topology = c("hexagonal", "rectangular"),
  epochs = 50,
  radius = NULL,
  init = c("pca", "random"),
  center = TRUE,
```

```
  scale = FALSE,
  robust = FALSE
)
```

Arguments

x	A numeric matrix or data frame of spectra, one per row.
grid	The numbers of units along the two sides of the map, c(xdim, ydim). If NULL (default), chosen from the number of spectra (see details).
topology	"hexagonal" (default) or "rectangular".
epochs	The number of training epochs. Default is 50.
radius	The radius of the neighborhood kernel (in units) at the first and the last epoch. If NULL (default), c(max(grid) / 3, 1).
init	"pca" (default) or "random".
center, scale	Logical values: center the variables (TRUE, default) and scale them to unit variance (FALSE, default) before training.
robust	A logical: down-weight the outlying spectra (FALSE, default).

Details

Algorithm. The batch SOM (Kohonen, 2013) is computed in C++. At each epoch, every spectrum is assigned to its best-matching unit (BMU), the unit with the closest prototype, and every prototype becomes the mean of the spectra weighted by a Gaussian kernel of the grid distance between its unit and their BMU. The radius of the kernel decreases linearly over the epochs, from radius[1] (a global ordering of the map) to radius[2] (local fine-tuning). The distances between spectra and prototypes are computed in one matrix product per epoch, so thousands of channels are handled quickly.

Initialization. With init = "pca" (default), the prototypes start on the plane of the first two principal components, spread over two standard deviations of the scores, with the long side of the grid along the first component. The training is then deterministic, and the map is oriented like a PCA score plot. init = "random" starts from randomly chosen spectra; use set.seed() for reproducibility.

Grid. By default, the map has about $5\sqrt{n}$ units (Vesanto et al., 2000), with the ratio of its sides set to the square root of the ratio of the first two eigenvalues of the data. The "hexagonal" topology gives each unit six equidistant neighbors, which displays the clusters better than the four of "rectangular".

Robust SOM. With robust = TRUE, the spectra are weighted at each epoch by Huber weights of their quantization errors (robustly standardized, with cut-off $\sqrt{\chi^2_{1,0.99}}$): the outlying spectra (a misfired shot, a contaminated sample) pull the prototypes less, as in the robust SOMs of Allende et al. (2004).

Quality and novelty. The mean quantization error measures how well the prototypes represent the spectra, and the topographic error (the share of spectra whose two best units are not neighbors) how well the map keeps the neighborhoods. The quantization errors of the training spectra give a robust cut-off (the Wilson-Hilferty transformation with the univariate MCD, as for the orthogonal

distances of robpca()): spectra beyond it fit no unit well, and new spectra beyond it are flagged as novel by predict().

Stability. The map depends on its grid, radius and initialization. Use som_stability() to check that the neighborhoods are stable over resampled data.

Value

An object of class specproc_som, a list with

- codebook: the prototypes, one row per unit, in the units of x;
- grid: a tibble with the unit, its row and col, and its coordinates x and y on the map;
- unit, qe: the best-matching unit and the quantization error of each training spectrum, and weights (with robust = TRUE);
- quantization_error, topographic_error: the quality of the map;
- cutoff: the quantization error beyond which a spectrum is novel;
- center, scale and the training settings.

Author(s)

Christian L. Goueguel

References

- Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43(1):59-69.
- Kohonen, T. (2013). Essentials of the self-organizing map. Neural Networks, 37:52-65.
- Vesanto, J., Himberg, J., Alhoniemi, E., Parhankangas, J. (2000). SOM Toolbox for Matlab 5. Report A57, Helsinki University of Technology.
- Allende, H., Moreno, S., Rogel, C., Salas, R. (2004). Robust self-organizing maps. In Progress in Pattern Recognition, Image Analysis and Applications (CIARP 2004), Lecture Notes in Computer Science 3287:179-186.

See Also

predict.specproc_som(), plot_som(), som_stability()

Examples

```
data(forageLIBS)
# the Na I and K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which((wl > 585 & wl < 595) | (wl > 760 & wl < 780))]
fit <- som(spectra)
fit
# where the potassium-rich samples are, and how many samples per unit
plot_som(fit, type = "mapping", colour = forageLIBS$K)
plot_som(fit, type = "counts")
```

som_stability *Stability of a Self-Organizing Map*

Description

Fits a self-organizing map with `som()` on resampled data several times, maps all the spectra on each map, and measures how consistently the spectra keep their neighbors.

Usage

```
som_stability(x, runs = 10, ...)
```

Arguments

`x`	A numeric matrix or data frame of spectra, one per row.
`runs`	The number of resampled fits. Default is 10.
`...`	Further arguments passed to `som()`, such as `grid` or `robust`.

Details

Each run fits the map on a bootstrap sample of the spectra (with the same grid), and maps all of them. For each run, the grid distances between the units of every pair of spectra form a matrix; the stability is the mean correlation between these matrices over the pairs of runs. Values close to 1 mean that the same spectra are neighbors on every map, whatever the sample: the structure of the map is not an artifact of the training data. Low values suggest a smaller grid, a larger final radius or more epochs.

Value

A list with `runs`, a tibble with the quantization and topographic errors of each run, `stability`, the mean correlation between the runs, and `correlations`, the matrix of the correlations between runs.

See Also

`som()`

Examples

```
data(forageLIBS)
set.seed(1)
som_stability(forageLIBS[-(1:14)], runs = 5)$stability
```

spectral_norm_method *Norm of a Spectral Normalization*

Description

A dials parameter for the method of step_spectral_norm().

Usage

```
spectral_norm_method(values = c("l1", "area", "l2", "max"))
```

Arguments

values	The norms to try. Default is all of "l1", "area", "l2" and "max".

Value

A dials qual_param object.

See Also

step_spectral_norm()

Examples

```
spectral_norm_method()
spectral_norm_method(c("l1", "max"))
```

stark_table *Build a Table of Stark Broadening Parameters*

Description

Builds a table of Stark widths and shifts from user-supplied values, in the format of starkb_lines(), so that stark_width() and electron_density() can use Stark parameters from any source: values copied from the STARK-B website, from a publication, or from the fitted temperature laws that STARK-B provides.

Usage

```
stark_table(
  wavelength,
  temperature,
  width = NULL,
  units,
  density = 1e+17,
  shift = 0,
  perturber = "electron",
  species = NA_character_,
  upper = "",
  lower = "",
  coefficients = NULL,
  shift_coefficients = NULL,
  log_base = 10,
  source = "user-supplied"
)
```

Arguments

`wavelength`	The wavelength of the line (or multiplet), in `units`.
`temperature`	The temperatures, in K.
`width`	The Stark full widths at half maximum at each temperature, in `units`. Leave `NULL` when `coefficients` are given.
`units`	The units of `wavelength`, `width` and `shift`: `"nm"` or `"A"` (Ångström). Required.
`density`	The perturber density of the widths, in cm^{-3}. Default is 10^{17}.
`shift`	The Stark shifts, in `units`. Default is 0 (unknown).
`perturber`	The perturber. Default is `"electron"`.
`species`, `upper`, `lower`	Optional labels of the emitter and of the upper and lower levels.
`coefficients`	Optional coefficients (a_0, a_1, a_2) of the fitted width law.
`shift_coefficients`	Optional coefficients (b_0, b_1, b_2) of the fitted law for the ratio d/w; used with `coefficients`.
`log_base`	The base of the logarithms of the fitted laws. Default is 10.
`source`	A description of where the data come from.

Details

Give either the tabulated `width` at each `temperature`, or the coefficients of a fitted temperature law. STARK-B fits the widths and shifts at a given perturber density as

$$\log w = a_0 + a_1 \log T + a_2 (\log T)^2$$

$$d/w = b_0 + b_1 \log T + b_2 (\log T)^2$$

with decimal logarithms and w in the units of its tables (Å); the fits are only valid within the tabulated temperature range (Sahal-Bréchot, Dimitrijević and Ben Nessib, 2011). With `coefficients`, the table is evaluated at `temperature`, which should span that range. Check the base of the logarithm and the units of any other source.

`units` is required, because Stark widths are reported in both Å and nm, and a wrong guess changes the electron density tenfold. It applies to `wavelength`, `width` and `shift` (and to the fitted widths); the table is returned in nm.

Arguments of length 1 are recycled. Tables for several lines or perturbers can be combined with `rbind()` or `dplyr::bind_rows()`.

Value

A tibble with the columns of `starkb_lines()`.

References

- Sahal-Bréchot, S., Dimitrijević, M.S., Ben Nessib, N. (2011). Widths and shifts of isolated lines of neutral and ionized atoms perturbed by collisions with electrons and ions: an outline of the semiclassical perturbation (SCP) method and of the approximations used for the calculations. Baltic Astronomy, 20:523-530.

See Also

`stark_width()`, `electron_density()`, `read_starkb()`

Examples

```
# made-up widths of a hypothetical line, in Angstrom at 1e17 cm-3
x <- stark_table(
  wavelength = 5000, temperature = c(5000, 10000, 20000),
  width = c(0.20, 0.16, 0.13), units = "A", species = "X II"
)
x
electron_density(0.012, stark = x, wavelength = 500, temperature = 8000)

# the same line from the coefficients of a fitted temperature law
# log10(w) = a0 + a1 log10(T) + a2 log10(T)^2 (here fitted to the table above)
lt <- log10(c(5000, 10000, 20000))
a <- unname(coef(lm(log10(c(0.20, 0.16, 0.13)) ~ lt + I(lt^2))))
stark_table(wavelength = 5000, temperature = c(5000, 10000, 20000),
            coefficients = a, units = "A")
```

`stark_width` *Stark Width of a Line at Given Plasma Conditions*

Description

Computes the Stark width (and shift) of a line at a given temperature and electron density, from the Stark broadening parameters returned by `starkb_lines()`.

Usage

```
stark_width(
  stark,
  wavelength,
  temperature,
  density = 1e+17,
  perturber = "electron",
  tolerance = 0.5
)
```

Arguments

stark	A tibble returned by starkb_lines().
wavelength	The wavelength of the line, in nm.
temperature	The temperature, in K (a vector is allowed).
density	The perturber (electron) density, in cm^{-3}. Default is 10^{17}.
perturber	The perturber. Default is "electron".
tolerance	The largest allowed difference between wavelength and the tabulated wavelength, in nm. Default is 0.5.

Details

The transition whose wavelength is closest to wavelength is used; its wavelength must lie within tolerance nm. STARK-B often tabulates a multiplet at its mean wavelength λ_m; the width and shift of a line of the multiplet at λ are then $w_\lambda = w_m \lambda^2/\lambda_m^2$ (and likewise for the shift), as the database recommends. This scaling is applied to every match. Within the impact approximation, the width and shift are proportional to the perturber density, so the tabulated values at the density closest to density are scaled linearly to density. Between the tabulated temperatures, the width is interpolated linearly in $\log w$ against $\log T$, and the shift linearly against $\log T$; temperatures outside the tabulated range give an error.

Value

A tibble with one row per temperature and columns wavelength (the requested wavelength, nm), tabulated_wavelength, upper, lower, perturber, temperature, density, width (FWHM, nm) and shift (nm).

See Also

starkb_lines(), electron_density()

Examples

```
# A table in the format of starkb_lines(), for illustration
stark <- data.frame(
  species = "X II", wavelength = 500, upper = "u", lower = "l",
  perturber = "electron", temperature = c(5000, 10000, 20000),
  density = 1e17, width = c(0.020, 0.014, 0.010), shift = 0, source = ""
```

```
)
stark_width(stark, wavelength = 500, temperature = 8000, density = 5e16)
```

---

starkb_lines *Stark Broadening Parameters from the STARK-B Database*

Description

Retrieves the Stark broadening parameters (full widths at half maximum and shifts) of the lines of an atom or ion from STARK-B, the database of Stark broadening parameters of isolated lines of atoms and ions in the impact approximation (Sahal-Bréchot, Dimitrijević and Moreau, Observatoire de Paris and Astronomical Observatory of Belgrade). The data are downloaded on demand from the database's Virtual Atomic and Molecular Data Centre (VAMDC) service; they are not distributed with specProc.

Usage

```
starkb_lines(species, wavelength = NULL, perturber = NULL, timeout = 120)
```

Arguments

species	A character string naming the emitter in spectroscopic notation, such as "Ca II" (singly ionized calcium) or "Na I" (neutral sodium).
wavelength	An optional numeric vector of length 2: the wavelength range to keep, in nm.
perturber	An optional character vector of perturbers to keep, such as "electron", "H II" (protons) or "He II". By default all are kept.
timeout	The download timeout, in seconds. Default is 120.

Details

For each transition, STARK-B tabulates the Stark width w (full width at half maximum) and shift d for collisions with electrons and with ions (protons, singly ionized helium, ...), at several temperatures and perturber densities. In the impact approximation, the width and shift are proportional to the perturber density. Many entries are multiplets, given at the mean wavelength of the multiplet; the upper and lower configurations and terms identify them.

The results of a query are cached for the R session, so repeated calls with the same species do not download the data again.

When you use these data, cite the STARK-B database and the original publications, listed in the source column:

- Sahal-Bréchot, S., Dimitrijević, M.S., Moreau, N. STARK-B database, stark-b.obspm.fr. Observatoire de Paris and Astronomical Observatory of Belgrade (see the References).

Value

A tibble with one row per transition, perturber, temperature and density, and columns:

- `species`, `wavelength` (nm, as given by STARK-B), `upper`, `lower` (configuration and term of the upper and lower levels);
- `perturber`, `temperature` (K), `density` (perturber density, cm^{-3});
- `width` (Stark full width at half maximum, nm) and `shift` (nm; positive towards the red);
- `source`: the publications the data come from.

Author(s)

Christian L. Goueguel

References

- Sahal-Bréchot, S., Dimitrijević, M.S., Moreau, N., Ben Nessib, N. (2015). The STARK-B database VAMDC node: a repository for spectral line broadening and shifts due to collisions with charged particles. Physica Scripta, 90(5):054008. doi:10.1088/00318949/90/5/054008

See Also

`stark_width()` to interpolate the width of a line at a given temperature and density, `electron_density()`, `read_starkb()` for saved STARK-B files, and `stark_table()` for data from other sources.

Examples

```
# needs an internet connection
ca <- try(starkb_lines("Ca II", wavelength = c(390, 400), perturber = "electron"))
if (!inherits(ca, "try-error")) head(ca)
```

---

step_baseline *Baseline Correction Recipe Step*

Description

`step_baseline()` creates a *specification* of a recipe step that subtracts a fitted baseline from each spectrum, with `baseline_arpls()`, `baseline_als()` or `baseline_lsp()`.

Usage

```
step_baseline(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  method = "arpls",
```

```
  lambda = 1000,
  degree = 4,
  options = list(),
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("baseline")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
method	The baseline algorithm: "arpls" (default), "als" or "lsp".
lambda	The smoothing parameter of the "arpls" and "als" methods. Default is 1000.
degree	The polynomial degree of the "lsp" method. Default is 4.
options	A list of further arguments passed to the baseline function: ratio and max.iter for baseline_arpls(), p and max.iter for baseline_als(), tol and max.iter for baseline_lsp().
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns, in the order in which they appear in the data, form one spectrum per row, so select all the channels of the spectrum and only them. The baseline is fitted to each spectrum separately, so nothing is estimated from the training data and the step gives the same result whether it is applied before or after the data are split. The selected columns are replaced by the corrected values.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

Tuning

lambda (methods "arpls" and "als") can be tuned with tune::tune(), using baseline_lambda(), and degree (method "lsp") using dials::degree_int() with values 1 to 8.

Tidying

tidy() returns a tibble with columns `terms`, `method` and `id`.

Author(s)

Christian L. Goueguel

See Also

`baseline_arpls()`, `baseline_als()`, `baseline_lsp()`

Examples

```
library(recipes)
data(forageLIBS)
spectra <- forageLIBS[c(2, 15:406)]  # sample id and spectral channels

rec <- recipe(~ ., data = spectra[1:16, ]) |>
  update_role(Sample, new_role = "id") |>
  step_baseline(all_predictors(), method = "arpls", lambda = 1e5) |>
  step_snv(all_predictors())
prepped <- prep(rec)
bake(prepped, new_data = spectra[17:24, ])[, 1:6]

# five spectra between 240 and 300 nm, before and after the baseline correction
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
region <- forageLIBS[1:5, c(2, which(wl > 240 & wl < 300))]
corrected <- recipe(~ ., data = region) |>
  update_role(Sample, new_role = "id") |>
  step_baseline(all_predictors(), method = "arpls", lambda = 1e5) |>
  prep() |>
  bake(new_data = NULL)
plot_spectra(region, id = Sample) +
  ggplot2::coord_cartesian(ylim = c(500, 3000))
plot_spectra(corrected, id = Sample) +
  ggplot2::coord_cartesian(ylim = c(-200, 2500))
```

---

step_cellpca *cellPCA Recipe Step*

Description

`step_cellpca()` creates a *specification* of a recipe step that converts the selected variables into robust principal component scores with `cellpca()`, which weights the outlying cells and observations and handles missing values.

Usage

```
step_cellpca(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  options = list(),
  prefix = "CPC",
  distances = FALSE,
  keep_original_cols = FALSE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("cellpca")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See `recipes::selections()`.
role	The role of the new score columns. Default is "predictor".
trained	A logical indicating whether the step has been trained.
num_comp	The number of principal components. Default is 2.
options	A list of further arguments passed to `cellpca()`, such as alpha, maxiter or tol.
prefix	The prefix of the new column names. Default is "CPC".
distances	A logical: add the score and orthogonal distances as columns. Default is FALSE.
keep_original_cols	A logical: keep the selected columns. Default is FALSE.
res	The fitted robust PCA model, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by `recipes::bake()`? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

As `step_robpca()`. Missing values are allowed in the selected columns, both when the recipe is prepped and when it is baked. New observations are projected by the robust regression of their observed cells on the loadings, so that their outlying cells do not distort their scores.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

See Also

`cellpca()`, `step_macropca()`, `step_robpca()`

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
set.seed(1)
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_cellpca(recipes::all_predictors(), num_comp = 2, distances = TRUE,
               options = list(od_cutoff = "chisq"))
recipes::bake(recipes::prep(rec), new_data = dat[301:368, ])
```

---

step_direct_orthogonal

Direct Orthogonalization Recipe Step

Description

`step_direct_orthogonal()` creates a *specification* of a recipe step that removes response-orthogonal variation from the selected predictors with `direct_orthogonal()`.

Usage

```
step_direct_orthogonal(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  num_comp = 2,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("direct_orthogonal")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variable, as a bare name or a selector. If NULL (default), the single outcome of the recipe is used.
num_comp	The number of orthogonal components to remove.
options	A list of further arguments passed to direct_orthogonal(), such as scale.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns are replaced by the corrected values, which are centered (and scaled, if options = list(scale = TRUE)). Because the filter uses the outcome, it must be estimated on training data only. Within a workflows::workflow() and tune::tune_grid(), this happens automatically for every resample. The outcome is not needed when new data are baked.

The related steps step_direct_orthogonal(), step_direct_osc(), step_projected_osc(), step_opls() and step_o2pls() (which also handles several outcomes) remove response-orthogonal variation with other algorithms. step_epo() and step_glsw() remove variation described by external clutter, and step_y_gradient_glsw() down-weights variation between samples with similar outcomes.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

direct_orthogonal(), predict.specproc_filter()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_direct_orthogonal(recipes::all_predictors(), num_comp = 2)
```

```
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_direct_osc *Direct Orthogonal Signal Correction Recipe Step*

Description

step_direct_osc() creates a *specification* of a recipe step that removes response-orthogonal variation from the selected predictors with direct_osc().

Usage

```
step_direct_osc(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  num_comp = 2,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("direct_osc")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variable, as a bare name or a selector. If NULL (default), the single outcome of the recipe is used.
num_comp	The number of orthogonal components to remove.
options	A list of further arguments passed to direct_osc(), such as scale or tol.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns are replaced by the corrected values, which are centered (and scaled, if `options = list(scale = TRUE)`). Because the filter uses the outcome, it must be estimated on training data only. Within a `workflows::workflow()` and `tune::tune_grid()`, this happens automatically for every resample. The outcome is not needed when new data are baked.

The related steps `step_direct_orthogonal()`, `step_direct_osc()`, `step_projected_osc()`, `step_opls()` and `step_o2pls()` (which also handles several outcomes) remove response-orthogonal variation with other algorithms. `step_epo()` and `step_glsw()` remove variation described by external clutter, and `step_y_gradient_glsw()` down-weights variation between samples with similar outcomes.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

See Also

`direct_osc()`, `predict.specproc_filter()`

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_direct_osc(recipes::all_predictors(), num_comp = 2)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

`step_emsc` *Extended Multiplicative Signal Correction Recipe Step*

Description

`step_emsc()` creates a *specification* of a recipe step that corrects each spectrum for multiplicative effects, a polynomial baseline and, optionally, known interferents, with `emsc()`.

Usage

```
step_emsc(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  degree = 2,
```

```
  interferents = NULL,
  wavelength = NULL,
  robust = TRUE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("emsc")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
degree	The degree of the polynomial baseline. Default is 2.
interferents	An optional numeric matrix or data frame of interferent spectra, one per row. If it has column names, the columns matching the selected predictors are used; otherwise it must have one column per selected predictor, in the same order.
wavelength	An optional numeric vector of wavelengths, one per selected column, used to build the polynomials. If NULL, the column names are used when they are numeric, and the column positions otherwise.
robust	A logical: use the median (TRUE, default) or the mean of the training spectra as the reference.
res	The fitted EMSC model, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The reference spectrum is estimated from the training spectra when the recipe is prepped, and new spectra are corrected with the same reference, polynomials and interferents when they are baked. The selected columns form one spectrum per row and are replaced by the corrected values.

degree can be tuned with tune::tune(), using dials::degree_int() with values 0 to 4. tidy() returns the selected terms, the reference value of each (NA before the step is trained) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

emsc(), step_msc()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_emsc(recipes::all_predictors(), degree = 2)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_epo *External Parameter Orthogonalization Recipe Step*

Description

step_epo() creates a *specification* of a recipe step that projects the selected predictors onto the space orthogonal to the dominant directions of a clutter matrix, with epo().

Usage

```
step_epo(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  clutter = NULL,
  num_comp = 2,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("epo")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.

clutter	A numeric matrix or data frame of clutter spectra. If it has column names, the columns matching the selected predictors are used; otherwise it must have one column per selected predictor, in the same order.
num_comp	The number of orthogonal components to remove.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The clutter describes variation that should not affect the outcome, for example the differences between repeated measurements of the same samples. It is external information, supplied when the step is specified, and is not estimated from the training data. If clutter = NULL, the dominant directions of the training data themselves are removed, as in epo(). The outcome is not used.

The selected columns are replaced by the corrected values (not centered). num_comp can be tuned with tune::tune(). tidy() returns the selected terms, num_comp and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

epo(), predict.specproc_filter()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
# the three samples measured twice
twice <- forageLIBS$Sample[duplicated(forageLIBS$Sample)]
first <- match(twice, forageLIBS$Sample)
second <- vapply(twice, function(s) max(which(forageLIBS$Sample == s)), integer(1))
# their differences describe the variation between repeated measurements
clutter <- as.matrix(dat[second, -1]) - as.matrix(dat[first, -1])
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_epo(recipes::all_predictors(), clutter = clutter, num_comp = 1)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

step_glsw *Generalized Least Squares Weighting Recipe Step*

Description

step_glsw() creates a *specification* of a recipe step that down-weights the directions of the selected predictors that vary in a clutter matrix, with the generalized least squares weighting filter of glsw().

Usage

```
step_glsw(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  clutter,
  alpha = 0.01,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("glsw")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
clutter	A numeric matrix or data frame of difference spectra. It is matched to the selected predictors as in step_epo().
alpha	A positive number: the weighting parameter, relative to the largest eigenvalue of the clutter. Default is 0.01.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

clutter holds differences between spectra that should be identical, such as x2 - x1 for the paired spectra of glsw(). It is centered, so the filter equals glsw(x1, x2, alpha = a), where the absolute a is alpha times the largest eigenvalue of the centered clutter cross-product. Setting alpha relative to that eigenvalue makes it independent of the scale of the spectra, which simplifies tuning: small values remove the clutter directions almost completely (like step_epo()), large values leave the data nearly unchanged.

The filter is stored in factored form, so its memory use grows with the number of predictors rather than its square. The selected columns are replaced by the filtered values (not centered). alpha can be tuned with tune::tune(), using glsw_alpha(). tidy() returns the selected terms, alpha (relative) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

glsw(), step_y_gradient_glsw()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
# the three samples measured twice
twice <- forageLIBS$Sample[duplicated(forageLIBS$Sample)]
first <- match(twice, forageLIBS$Sample)
second <- vapply(twice, function(s) max(which(forageLIBS$Sample == s)), integer(1))
# their differences describe the variation between repeated measurements
clutter <- as.matrix(dat[second, -1]) - as.matrix(dat[first, -1])
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_glsw(recipes::all_predictors(), clutter = clutter, alpha = 0.01)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_line_intensities *Emission Line Intensities Recipe Step*

Description

step_line_intensities() creates a *specification* of a recipe step that replaces the spectral columns by the intensities of selected emission lines, measured with line_intensities().

Usage

```
step_line_intensities(
  recipe,
  ...,
  lines,
  half_width = 0.15,
  search = 0.2,
  method = "area",
  baseline = FALSE,
  prefix = "line_",
  keep_original_cols = FALSE,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("line_intensities")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See `recipes::selections()`.
lines	The lines: a numeric vector of wavelengths (nm), preferably named, such as `c(Ca = 393.37, Mg = 279.55)`.
half_width	The half-width of the integration window around the peak, in nm. Default is 0.15.
search	The half-width of the window in which the peak is searched around the tabulated wavelength, in nm. Default is 0.2.
method	`"area"` (default) or `"height"`.
baseline	A logical: subtract a linear baseline under each line (`FALSE`, default).
prefix	The prefix of the new column names. Default is `"line_"`.
keep_original_cols	A logical: keep the spectral columns (`FALSE`, default).
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by `recipes::bake()`? Keep the default `FALSE`, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns form one spectrum per row, and their names must be their wavelengths (in nm). Each line becomes a new column, named by `prefix` and the name of the line (or its wavelength). Nothing is estimated from the training data. The Voigt method of `line_intensities()` is not offered, as it is too slow for resampling. tidy() returns the `line` names, their `wavelength`, the `method` and `id`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

See Also

`line_intensities()`, `step_line_ratio()`

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  rec <- recipes::recipe(K ~ ., data = forageLIBS[-c(1:10, 12:14)]) |>
    step_line_intensities(recipes::all_predictors(),
                          lines = c(K = 769.90, Mg = 285.21, Ca = 317.93)) |>
    recipes::prep()
  head(recipes::bake(rec, new_data = NULL))
}
```

---

step_line_ratio *Internal Standard Normalization Recipe Step*

Description

`step_line_ratio()` creates a *specification* of a recipe step that divides each spectrum by the intensity of a reference emission line, such as a line of a matrix element of constant concentration (internal standard). This compensates for shot-to-shot changes of the ablated mass and of the plasma conditions.

Usage

```
step_line_ratio(
  recipe,
  ...,
  reference,
  window = 0.1,
  method = "area",
  baseline = FALSE,
  role = NA,
  trained = FALSE,
  columns = NULL,
```

```
  channels = NULL,
  skip = FALSE,
  id = recipes::rand_id("line_ratio")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
reference	The wavelength(s) of the reference line(s), in nm.
window	The half-width of the window around each reference wavelength, in nm. Default is 0.1.
method	The intensity of the reference line: "area" (default) or "height".
baseline	A logical: subtract a linear baseline under the reference line (FALSE, default).
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
channels	The channels of each reference window, found when the recipe is prepped. Not to be set by the user.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns form one spectrum per row, and their names must be their wavelengths (in nm). The intensity of the reference line is measured in each spectrum over the channels within window nm of reference:

- method = "area" (default): the integrated intensity (trapezoidal rule), more robust to noise and to small wavelength shifts;
- method = "height": the maximum intensity.

With baseline = TRUE, a straight line through the intensities at both ends of the window is subtracted first, so that the continuum under the line does not contribute. When several reference wavelengths are given, their intensities are summed.

Nothing is estimated from the training data: the channels of the window are found when the recipe is prepped, and each spectrum is normalized independently when it is baked. The selected columns are replaced by the normalized values. Spectra whose reference intensity is not positive are set to NA, with a warning. tidy() returns the reference wavelengths, the window, the method and id.

Choose a line that is well resolved, not self-absorbed and not saturated (see saturation_summary() and self_absorption()), and from an upper level close to that of the analyte lines, so that the ratio depends little on the plasma temperature.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

Author(s)

Christian L. Goueguel

References

- Body, D., Chadwick, B.L., (2001). Optimization of the spectral data processing in a LIBS simultaneous elemental analysis system. Spectrochimica Acta Part B, 56(6):725-736.

See Also

`normalize()`, `step_reject_shots()`

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  # normalize to the C I 247.86 nm line of the organic matrix
  rec <- recipes::recipe(~ ., data = forageLIBS[-(1:14)]) |>
    step_line_ratio(recipes::all_numeric(), reference = 247.856, window = 0.15) |>
    recipes::prep()
  recipes::tidy(rec, number = 1)
}
```

---

step_macropca *MacroPCA Recipe Step*

Description

`step_macropca()` creates a *specification* of a recipe step that converts the selected variables into robust principal component scores with `macropca()`, which handles cellwise outliers and missing values as well as outlying observations.

Usage

```
step_macropca(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  options = list(),
  prefix = "MPC",
  distances = FALSE,
  keep_original_cols = FALSE,
```

```
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("macropca")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	The role of the new score columns. Default is "predictor".
trained	A logical indicating whether the step has been trained.
num_comp	The number of principal components. Default is 2.
options	A list of further arguments passed to macropca(), such as alpha or MacroPCA parameters.
prefix	The prefix of the new column names. Default is "MPC".
distances	A logical: add the score and orthogonal distances as columns. Default is FALSE.
keep_original_cols	A logical: keep the selected columns. Default is FALSE.
res	The fitted robust PCA model, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

As step_robpca(). Missing values are allowed in the selected columns, both when the recipe is prepped and when it is baked.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

macropca(), step_robpca()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
set.seed(1)
rec <- recipes::recipe(K ~ ., data = dat) |>
  step_macropca(recipes::all_predictors(), num_comp = 2)
recipes::bake(recipes::prep(rec), new_data = NULL)
```

---

step_msc *Multiplicative Scatter Correction Recipe Step*

Description

step_msc() creates a *specification* of a recipe step that corrects each spectrum for multiplicative and additive effects relative to a reference spectrum, with msc().

Usage

```
step_msc(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  robust = TRUE,
  drop.offset = TRUE,
  window = NULL,
  reference = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("msc")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
robust	A logical: use the median (TRUE, default) or the mean of the training spectra as the reference.

drop.offset	A logical: remove the additive offset (TRUE, default) or only the multiplicative effect.
window	An optional list of column index vectors (within the selected columns) for piecewise MSC. See msc().
reference	The reference spectrum, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The reference spectrum (the median or mean of the training spectra) is estimated when the recipe is prepped, and new spectra are corrected against it when they are baked. The selected columns form one spectrum per row. They are replaced by the corrected values. tidy() returns the selected terms, the reference value of each (NA before the step is trained) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

msc(), step_emsc()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_msc(recipes::all_predictors())
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_o2pls *O2PLS Filter Recipe Step for One or Several Outcomes*

Description

step_o2pls() creates a *specification* of a recipe step that removes num_comp outcome-orthogonal components from the selected predictors with o2pls(). Unlike the other orthogonalization steps, the filter can be estimated against several outcomes at once.

Usage

```
step_o2pls(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  num_comp = 2,
  joint_comp = 1,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("o2pls")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See `recipes::selections()`.
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variables, as bare names or selectors. If `NULL` (default), all the outcomes of the recipe are used.
num_comp	The number of outcome-orthogonal components to remove (`nx` in `o2pls()`).
joint_comp	The number of joint (predictive) components (`ncomp` in `o2pls()`), at most the number of outcomes. Default is 1.
options	A list of further arguments passed to `o2pls()`: `center` and `scale`.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by `recipes::bake()`? Keep the default `FALSE`, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The joint (predictive) directions are the `joint_comp` dominant singular vectors of $\mathbf{Y}^T\mathbf{X}$, and the removed components are the systematic variation of the predictors orthogonal to them (Trygg and Wold, 2003). Only the predictors are filtered: the outcomes are left unchanged, so that models are fitted to, and assessed on, the measured values. (The outcome-side filtering of `o2pls()`, its `ny` argument, does not change the filtered predictors.)

With a single outcome and `joint_comp = 1`, the filtered data are those of `step_opls()` and `step_projected_osc()` with the same `num_comp`.

The selected columns are replaced by the filtered values, which are centered (and scaled, if `options = list(scale = TRUE)`). The filter uses the outcomes, so it is estimated on the training data only (on the analysis set of each resample within `tune::tune_grid()`); the outcomes are not needed when new data are baked.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

Several outcomes

Declare the outcomes in the recipe formula; a model that handles several outcomes, such as `parsnip::pls()` with the mixOmics engine, then predicts them all, with one column per outcome (`.pred_K`, `.pred_Ca`, ...):

```
rec <- recipe(K + Ca ~ ., data = spectra) |>
  step_o2pls(all_predictors(), num_comp = 2, joint_comp = 2)
model <- parsnip::pls(num_comp = 2) |>
  set_mode("regression") |>
  set_engine("mixOmics", scale = FALSE)
fitted <- fit(workflow(rec, model), data = spectra)
predict(fitted, new_data = new_spectra)
```

Use `outcome` to estimate the filter against some of the outcomes only.

Tuning

`num_comp` and `joint_comp` can be tuned with `tune::tune()` when the recipe has a single outcome. Their default ranges are `dials::num_comp()` with values 1 to 4 and 1 to 3; `joint_comp` cannot exceed the number of outcomes.

`tune::tune_grid()` does not support several outcomes. Instead, create one workflow per outcome with the workflowsets package, and tune them all with `workflowsets::workflow_map("tune_grid", ...)`. Each recipe has a single outcome, for the model, but the filter can still be estimated against all the outcomes: give the others a role of their own (`"reference"` here), not needed to bake new data, and select them with `outcome`. Each outcome gets its own number of filter and model components:

```
outcomes <- c("K", "Ca", "Mg")
recipes <- purrr::map(outcomes, \(y) {
  recipe(reformulate(".", response = y), data = spectra) |>
    update_role(all_of(setdiff(outcomes, y)), new_role = "reference") |>
    update_role_requirements("reference", bake = FALSE) |>
   step_o2pls(all_predictors(), outcome = c(all_outcomes(), has_role("reference")),
               num_comp = tune("filter"), joint_comp = length(outcomes))
}) |>
  purrr::set_names(outcomes)
model <- parsnip::pls(num_comp = tune()) |>
```

```
  set_mode("regression") |>
  set_engine("mixOmics", scale = FALSE)

wf_set <- workflow_set(preproc = recipes, models = list(pls = model))
res <- workflow_map(wf_set, "tune_grid", resamples = vfold_cv(spectra, v = 5),
                    grid = 10, metrics = metric_set(rmse), seed = 1)

# best settings and final model of each outcome
fits <- purrr::map(purrr::set_names(res$wflow_id), \(id) {
  best <- select_best(extract_workflow_set_result(res, id), metric = "rmse")
  extract_workflow(res, id) |>
    finalize_workflow(best) |>
    fit(data = spectra)
})
purrr::map(fits, \(f) predict(f, new_data = new_spectra))
```

Select the outcomes by role (`all_outcomes()`, `has_role()`) rather than with `all_of(outcomes)`, which refers to a variable outside the recipe. To filter each outcome against itself only, drop the other outcomes from its recipe instead; the step is then equivalent to `step_opls()`. Read the results outcome by outcome: `workflowsets::rank_results()` would rank workflows of different outcomes, whose RMSE are in different units.

Tidying

tidy() returns a tibble with columns `terms` (the selected predictors), `num_comp`, `joint_comp` and `id`.

References

- Trygg, J., Wold, S., (2003). O2-PLS, a two-block (X–Y) latent variable regression (LVR) method with an integral OSC filter. J. Chemom. 17(1):53–64.

See Also

`o2pls()`, predict.o2pls()

Examples

```
data(forageLIBS)
# potassium, calcium and the spectral channels
dat <- forageLIBS[-c(1:2, 4:10, 12:14)]
rec <- recipes::recipe(K + Ca ~ ., data = dat[1:300, ]) |>
  step_o2pls(recipes::all_predictors(), num_comp = 2, joint_comp = 2)
prepped <- recipes::prep(rec)
recipes::tidy(prepped, number = 1)
dim(recipes::bake(prepped, new_data = dat[301:368, ]))
```

step_opls *Orthogonal Projections to Latent Structures (OPLS) Recipe Step*

Description

step_opls() creates a *specification* of a recipe step that removes num_comp orthogonal components from the selected predictors with the OPLS model of opls(): the selected columns are replaced by the OPLS-filtered data.

Usage

```
step_opls(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  num_comp = 2,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("opls")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variable, as a bare name or a selector. If NULL (default), the single outcome of the recipe is used.
num_comp	The number of orthogonal components to remove.
options	A list of further arguments passed to opls(): center and scale.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The filtered data are the same as those of step_projected_osc() with the same num_comp. For Pareto scaling, add step_pareto_scale() before this step. The model is fitted without cross-validation or permutation test, which do not change the filter. The number of orthogonal components is not selected automatically: tune num_comp instead. As for step_osc(), the filter uses the outcome and is estimated on the training data only; the outcome is not needed when new data are baked.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

Tuning

num_comp can be tuned with tune::tune(); its default range is dials::num_comp() with values 1 to 4.

Tidying

tidy() returns a tibble with columns terms (the selected predictors), num_comp and id.

See Also

opls(), predict.specproc_opls()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
# Pareto scaling, then the OPLS filter
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_pareto_scale(recipes::all_predictors()) |>
  step_opls(recipes::all_predictors(), num_comp = 2)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_osc *Orthogonal Signal Correction Recipe Step*

Description

step_osc() creates a *specification* of a recipe step that removes the variation of the selected predictors that is orthogonal to the outcome, with osc(). The filter is estimated on the training data when the recipe is prepped and applied to new data when it is baked.

Usage

```
step_osc(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  method = "sjoblom",
  num_comp = 2,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("osc")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variable, as a bare name or a selector. If NULL (default), the single outcome of the recipe is used.
method	The OSC algorithm: "wold", "sjoblom" (default) or "fearn". See osc().
num_comp	The number of orthogonal components to remove.
options	A list of further arguments passed to the underlying function, such as scale, tol or max.iter for osc().
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns are replaced by the corrected values, which are centered (and scaled, if options = list(scale = TRUE)). Because the filter uses the outcome, it must be estimated on training data only. Within a workflows::workflow() and tune::tune_grid(), this happens automatically for every resample. The outcome is not needed when new data are baked.

The related steps step_direct_orthogonal(), step_direct_osc(), step_projected_osc(), step_opls() and step_o2pls() (which also handles several outcomes) remove response-orthogonal

variation with other algorithms. step_epo() and step_glsw() remove variation described by external clutter, and step_y_gradient_glsw() down-weights variation between samples with similar outcomes.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

Tuning

num_comp can be tuned with tune::tune(); its default range is dials::num_comp() with values 1 to 4. When the model also has a num_comp argument (for example, parsnip::pls()), give the step's parameter its own id, such as num_comp = tune("filter_comp"), so that the two can be tuned together:

```
rec <- recipe(K ~ ., data = spectra) |>
  step_osc(all_predictors(), method = "fearn", num_comp = tune("filter_comp"))
model <- parsnip::pls(num_comp = tune()) |>
  set_mode("regression") |>
  set_engine("mixOmics", scale = FALSE)  # center only, like pls::plsr()
wf <- workflow(rec, model)
tune_grid(wf, resamples = group_vfold_cv(spectra, group = Sample), grid = 9)
```

Tidying

tidy() returns a tibble with columns terms (the selected predictors), num_comp and id.

Author(s)

Christian L. Goueguel

See Also

osc(), predict.specproc_filter()

Examples

```
library(recipes)
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipe(K ~ ., data = dat[1:300, ]) |>
  step_osc(all_predictors(), method = "fearn", num_comp = 2)
prepped <- prep(rec)
tidy(prepped, number = 1)
# new spectra are corrected with the filter estimated on the training data
bake(prepped, new_data = dat[301:368, -1])
```

step_pareto_scale *Pareto Scaling Recipe Step*

Description

step_pareto_scale() creates a *specification* of a recipe step that divides each selected column by the square root of its standard deviation, as pareto_scale() does.

Usage

```
step_pareto_scale(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  scales = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("pareto_scale")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
scales	The scale of each column, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The standard deviations are estimated from the training data and applied unchanged to new data. Columns with zero standard deviation are not scaled. The data are not centered; add recipes::step_center() if needed. tidy() returns the selected terms, the scale each column is divided by (NA before the step is trained) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

pareto_scale(), step_poisson_scale()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_pareto_scale(recipes::all_predictors())
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_poisson_scale *Poisson Scaling Recipe Step*

Description

step_poisson_scale() creates a *specification* of a recipe step that divides each selected column by the square root of its mean plus an offset, as poisson_scale() does (column mode).

Usage

```
step_poisson_scale(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  offset = 3,
  scales = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("poisson_scale")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
offset	The offset, in percent of the largest column mean. Default is 3.

scales	The scale of each column, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

Poisson scaling suits count data, whose variance grows with the mean. The scales $\sqrt{\bar{x}_j + c}$ are estimated from the training data, where the offset c is offset percent of the largest column mean, and are applied unchanged to new data. tidy() returns the selected terms, the scale each column is divided by (NA before the step is trained) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

poisson_scale(), step_pareto_scale()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_poisson_scale(recipes::all_predictors())
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_projected_osc *Projected Orthogonal Signal Correction (OPLS Filter) Recipe Step*

Description

step_projected_osc() creates a *specification* of a recipe step that removes num_comp response-orthogonal components from the selected predictors with projected_osc(). The result is the OPLS-filtered data (equivalently, o2pls() with a single outcome).

Usage

```
step_projected_osc(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  num_comp = 2,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("projected_osc")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variable, as a bare name or a selector. If NULL (default), the single outcome of the recipe is used.
num_comp	The number of orthogonal components to remove (ncomp of projected_osc()).
options	A list of further arguments passed to projected_osc(), such as scale or tol.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns are replaced by the corrected values, which are centered (and scaled, if options = list(scale = TRUE)). Because the filter uses the outcome, it must be estimated on training data only. Within a workflows::workflow() and tune::tune_grid(), this happens automatically for every resample. The outcome is not needed when new data are baked.

The related steps step_direct_orthogonal(), step_direct_osc(), step_projected_osc(), step_opls() and step_o2pls() (which also handles several outcomes) remove response-orthogonal variation with other algorithms. step_epo() and step_glsw() remove variation described by external clutter, and step_y_gradient_glsw() down-weights variation between samples with similar outcomes.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

See Also

`projected_osc()`, `predict.specproc_filter()`

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_projected_osc(recipes::all_predictors(), num_comp = 2)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

---

step_reject_shots *Shot Rejection Recipe Step*

Description

`step_reject_shots()` creates a *specification* of a recipe step that removes the outlying laser shots of each sample from the training data, with `reject_shots()`.

Usage

```
step_reject_shots(
  recipe,
  ...,
  sample,
  method = c("intensity", "correlation"),
  cutoff = 3.5,
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = TRUE,
  id = recipes::rand_id("reject_shots")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose the spectral columns (named by their wavelengths, see `reject_shots()`).

sample	The column identifying the sample of each shot, as a bare name or a selector. Give it the "id" role (see recipes::update_role()) so that it is not used as a predictor.
method	The criteria: one or more of "intensity", "correlation" (both by default) and "distance" (see reject_shots()).
cutoff	The robust z-score above which a shot is rejected. Default is 3.5.
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical: skip the step when new data are baked (TRUE, default). See details.
id	A character string that is unique to this step.

Details

Each shot is compared with the other shots of the same sample (the sample column), so nothing is estimated from the training data as a whole. Because the step removes rows, skip = TRUE by default: shots are rejected when the recipe is prepped, but new data are not filtered when they are baked (predictions must be made for every row). Use a resampling scheme grouped by sample, such as rsample::group_vfold_cv(), so that shots of the same sample are not split between analysis and assessment sets.

To model sample means instead of single shots, average the shots before the recipe, with reject_shots() and average().

tidy() returns the spectral terms, the sample column, the criteria (method), the cutoff and id.

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

See Also

reject_shots(), average(), step_line_ratio()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 760 & wl < 780)]  # the K I resonance lines
set.seed(1)
shots <- spectra[rep(1:4, each = 5), ] * stats::runif(20, 0.95, 1.05)
shots[7, ] <- 0.2 * shots[7, ]
shots <- cbind(sample = rep(c("A", "B", "C", "D"), each = 5), shots)
rec <- recipes::recipe(~ ., data = shots) |>
  step_reject_shots(recipes::all_numeric(), sample = "sample")
prepped <- recipes::prep(rec)
# the rejected shot is removed from the training data
nrow(recipes::bake(prepped, new_data = NULL))
```

step_robpca *Robust PCA (ROBPCA) Recipe Step*

Description

step_robpca() creates a *specification* of a recipe step that converts the selected variables into robust principal component scores, with robpca(). It is the robust counterpart of recipes::step_pca().

Usage

```
step_robpca(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  options = list(),
  prefix = "RPC",
  distances = FALSE,
  keep_original_cols = FALSE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("robpca")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	The role of the new score columns. Default is "predictor".
trained	A logical indicating whether the step has been trained.
num_comp	The number of principal components. Default is 2.
options	A list of further arguments passed to robpca(), such as alpha or ndir.
prefix	The prefix of the new column names. Default is "RPC".
distances	A logical: add the score and orthogonal distances as columns. Default is FALSE.
keep_original_cols	A logical: keep the selected columns. Default is FALSE.
res	The fitted robust PCA model, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The robust PCA model is estimated on the training data when the recipe is prepped, and new data are projected onto it when they are baked. The new columns are named with `prefix` followed by the component number. With `distances = TRUE`, two more columns hold the score distance (`<prefix>_SD`) and orthogonal distance (`<prefix>_OD`) of each observation, which can be used to screen outliers. The selected columns are removed unless `keep_original_cols = TRUE`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

Tuning

`num_comp` can be tuned with `tune::tune()`, using `dials::num_comp()` with values 1 to 4.

Tidying

tidy() returns the loadings as a tibble with columns `terms`, `value`, `component` and `id`.

See Also

`robpca()`, `step_rospca()`, `step_macropca()`

Examples

```
library(recipes)
data(forageLIBS)
# the 380-430 nm window (Ca II H and K lines)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
rec <- recipe(~ ., data = spectra) |>
  step_robpca(all_predictors(), num_comp = 3, distances = TRUE)
bake(prep(rec), new_data = NULL)
```

---

step_robust_bcyj *Robust Box-Cox and Yeo-Johnson Transformation Recipe Step*

Description

`step_robust_bcyj()` creates a *specification* of a recipe step that transforms the selected variables toward central normality with the robust Box-Cox or Yeo-Johnson transformation of `robust_bcyj()`, like `recipes::step_BoxCox()` or `recipes::step_YeoJohnson()` but robust to outliers.

Usage

```
step_robust_bcyj(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  type = "bestObj",
  quantile = 0.99,
  nbsteps = 2,
  standardize = TRUE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("robust_bcyj")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See `recipes::selections()`.
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
type	The transformation: `"bestObj"` (default; Box-Cox or Yeo-Johnson, whichever fits best, for positive variables), `"BC"` or `"YJ"`. See `robust_bcyj()`.
quantile	The quantile used for the weights of the re-weighting steps. Default is 0.99.
nbsteps	The number of re-weighting steps. Default is 2.
standardize	A logical: robustly standardize the transformed variables (`TRUE`, default).
res	The fitted transformation, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by `recipes::bake()`? Keep the default `FALSE`, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The transformation parameter λ of each variable is estimated on the training data by re-weighted maximum likelihood (Raymaekers and Rousseeuw, 2021), as in `robust_bcyj()`, and applied unchanged to new data. With `standardize = TRUE`, the transformed variables are also centered and scaled with the mean and standard deviation of the training inliers. Variables that cannot be transformed (for example, constant ones) are left unchanged.

tidy() returns a tibble with columns `terms`, `lambda`, `method` (`"BC"` or `"YJ"`) and `id`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

Author(s)

Christian L. Goueguel

See Also

`robust_bcyj()`

Examples

```
library(recipes)
data(forageLIBS)
contents <- forageLIBS[c("Ca", "Mg", "P", "K", "Mn")]
rec <- recipe(~ ., data = contents[1:300, ]) |>
  step_robust_bcyj(all_numeric_predictors())
prepped <- prep(rec)
tidy(prepped, number = 1)
bake(prepped, new_data = contents[301:368, ])
```

---

step_rospca — *Robust Sparse PCA (ROSPCA) Recipe Step*

Description

`step_rospca()` creates a *specification* of a recipe step that converts the selected variables into robust sparse principal component scores, with `rospca()`.

Usage

```
step_rospca(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  lambda = 1,
  options = list(),
  prefix = "RSPC",
  distances = FALSE,
  keep_original_cols = FALSE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("rospca")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	The role of the new score columns. Default is "predictor".
trained	A logical indicating whether the step has been trained.
num_comp	The number of principal components. Default is 2.
lambda	The sparsity parameter of rospca(). Default is 1.
options	A list of further arguments passed to rospca(), such as alpha, stand or ndir.
prefix	The prefix of the new column names. Default is "RSPC".
distances	A logical: add the score and orthogonal distances as columns. Default is FALSE.
keep_original_cols	A logical: keep the selected columns. Default is FALSE.
res	The fitted robust PCA model, stored once the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

As step_robpca(), with sparse loadings. num_comp and lambda can be tuned with tune::tune(); lambda uses dials::penalty() with a range of 0.01 to 100.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

rospca(), step_robpca()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
set.seed(1)
rec <- recipes::recipe(K ~ ., data = dat) |>
  step_rospca(recipes::all_predictors(), num_comp = 2, distances = TRUE)
recipes::bake(recipes::prep(rec), new_data = NULL)
```

step_savgol *Savitzky-Golay Recipe Step*

Description

step_savgol() creates a *specification* of a recipe step that smooths the spectra, or computes their derivative, with savitzky_golay().

Usage

```
step_savgol(
  recipe,
  ...,
  window = 11,
  order = 2,
  derivative = 0,
  segments = TRUE,
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("savgol")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
window	The width of the window, an odd number of channels. Default is 11.
order	The degree of the polynomial, smaller than window. Default is 2.
derivative	The derivative: 0 (smoothing, default), 1 or 2. It must not exceed order.
segments	A logical: filter the segments between gaps of the wavelength axis separately (TRUE, default). Needs wavelengths as names.
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns form one spectrum per row, and are replaced by the filtered values. Nothing is estimated from the training data. The `window`, `order` and `derivative` can be tuned with `tune::tune()` (dials parameters `window_size()`, `degree_int()` and `savgol_derivative()`); an even `window`, as a tuning grid may propose, is increased to the next odd number. tidy() returns the `terms`, `window`, `order`, `derivative` and `id`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

See Also

`savitzky_golay()`, `step_snv()`

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  rec <- recipes::recipe(K ~ ., data = forageLIBS[-c(1:10, 12:14)]) |>
    step_savgol(recipes::all_predictors(), window = 11, derivative = 1) |>
    recipes::prep()
  recipes::tidy(rec, number = 1)
}
```

`step_snv` *Standard Normal Variate Recipe Step*

Description

`step_snv()` creates a *specification* of a recipe step that centers and scales each spectrum by its own mean and standard deviation, with `snv()`.

Usage

```
step_snv(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("snv")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns form one spectrum per row; select all the channels of the spectrum and only them. Nothing is estimated from the training data. The selected columns are replaced by the SNV-transformed values. tidy() returns the selected terms and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

snv()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_snv(recipes::all_predictors())
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

step_spectral_norm *Spectral Normalization Recipe Step*

Description

step_spectral_norm() creates a *specification* of a recipe step that divides each spectrum by one of its norms, with normalize(): its total area, L1 norm, L2 (Euclidean) norm, or maximum absolute intensity.

Usage

```
step_spectral_norm(
  recipe,
  ...,
  method = "l1",
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("spectral_norm")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
method	The norm: "l1" (default), "area", "l2" or "max".
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns form one spectrum per row, and are replaced by the normalized values. Unlike recipes::step_normalize(), which scales each column (variable), this step scales each row (spectrum), to remove multiplicative variation between spectra such as shot-to-shot changes of the ablated mass. Nothing is estimated from the training data. Spectra whose norm is zero are set to NA, with a warning.

The methods are those of normalize(): "l1" (sum of absolute values, default), "area" (sum of values, equal to "l1" for non-negative spectra), "l2" (Euclidean norm, also called vector normalization) and "max" (largest absolute value). The method can be tuned with tune::tune() (dials parameter spectral_norm_method()). tidy() returns the terms, the method and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

Author(s)

Christian L. Goueguel

See Also

normalize(), step_snv(), step_line_ratio()

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  # potassium and the K I resonance lines
  wl <- suppressWarnings(as.numeric(names(forageLIBS)))
  dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
  rec <- recipes::recipe(K ~ ., data = dat) |>
    step_baseline(recipes::all_predictors()) |>
    step_spectral_norm(recipes::all_predictors(), method = "l2") |>
    recipes::prep()
  baked <- recipes::bake(rec, new_data = NULL)
  spectra <- baked[setdiff(names(baked), "K")]
  rowSums(spectra[1:3, ]^2)   # 1
}
```

---

step_wavelet *Wavelet Features Recipe Step*

Description

step_wavelet() creates a *specification* of a recipe step that replaces the spectral columns by their wavelet coefficients, computed with wavelet_features(), optionally keeping only the coefficients of largest variance in the training data.

Usage

```
step_wavelet(
  recipe,
  ...,
  wavelet = "d4",
  level = 3,
```

```
  coefficients = "approximation",
  num_coef = NULL,
  prefix = "wav_",
  keep_original_cols = FALSE,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  res = NULL,
  skip = FALSE,
  id = recipes::rand_id("wavelet")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().
wavelet	The wavelet: "haar", "d4" (default), "d6", "d8" or "la8".
level	The number of levels of the decomposition. Default is 3.
coefficients	The coefficients returned: "approximation" (default) or "all".
num_coef	The number of coefficients to keep, those of largest variance in the training data, or NULL (default) to keep them all.
prefix	The prefix of the new column names. Default is "wav_".
keep_original_cols	A logical: keep the spectral columns (FALSE, default).
role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
res	The coefficients kept, stored once the step has been trained. Not to be set by the user.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

The selected columns form one spectrum per row. With num_coef, the variance of each coefficient is computed on the training data when the recipe is prepped, and the num_coef coefficients of largest variance are kept (Trygg and Wold, 1998); new data get the same coefficients. The level and num_coef can be tuned with tune::tune() (dials parameters wavelet_level() and num_terms()). tidy() returns the names of the kept coefficients (terms), their variance in the training data (NA before the step is trained) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

wavelet_features(), step_savgol()

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  rec <- recipes::recipe(K ~ ., data = forageLIBS[-c(1:10, 12:14)]) |>
    step_wavelet(recipes::all_predictors(), wavelet = "la8", level = 4, num_coef = 50) |>
    recipes::prep()
  dim(recipes::bake(rec, new_data = NULL))
}
```

---

step_y_gradient_glsw *y-Gradient Generalized Least Squares Weighting Recipe Step*

Description

step_y_gradient_glsw() creates a *specification* of a recipe step that down-weights variation between samples with similar outcomes, with the filter of y_gradient_glsw().

Usage

```
step_y_gradient_glsw(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  alpha = 0.01,
  window = 5,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("y_gradient_glsw")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose the predictors (for example, the spectral channels). See recipes::selections().

role	Not used by this step, since no new variables are created.
trained	A logical indicating whether the step has been trained.
outcome	The outcome variable, as a bare name or a selector. If NULL (default), the single outcome of the recipe is used.
alpha	A positive number: the weighting parameter, relative to the largest eigenvalue of the clutter. Default is 0.01.
window	An odd integer giving the width of the Savitzky-Golay window used to compute the gradients. Default is 5.
res	The fitted filter, stored once the step has been trained.
columns	The names of the selected predictors, stored once the step has been trained.
skip	A logical. Should the step be skipped when the recipe is baked by recipes::bake()? Keep the default FALSE, so that new data are corrected with the same filter as the training data.
id	A character string that is unique to this step.

Details

As in step_glsw(), alpha is relative to the largest eigenvalue of the (weighted) gradient matrix, and the filter is stored in factored form. The filter uses the outcome, so it is estimated on training data only; the outcome is not needed when new data are baked. The selected columns are replaced by the filtered values (not centered). alpha can be tuned with tune::tune(), using glsw_alpha(). tidy() returns the selected terms, alpha (relative) and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

y_gradient_glsw(), step_glsw()

Examples

```
data(forageLIBS)
# potassium and the K I resonance lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "K"), which(wl > 760 & wl < 780))]
rec <- recipes::recipe(K ~ ., data = dat[1:300, ]) |>
  step_y_gradient_glsw(recipes::all_predictors(), alpha = 0.01)
prepped <- recipes::prep(rec)
recipes::bake(prepped, new_data = dat[301:368, ])
```

summary_stats *Classical or Robust Descriptive Statistics*

Description

This function calculates various descriptive statistics (robust and non-robust) for a specified variable or all variables in a given data frame or tibble.

Usage

```
summary_stats(x, var = NULL, digits = 2, robust = FALSE, drop.na = TRUE)
```

Arguments

x	A data frame or tibble.
var	A character vector of names (or a numeric vector of column positions) specifying the variable(s) for which to calculate the summary statistics. If left as NULL (the default), summary statistics will be calculated for all numeric variables in the data frame/tibble.
digits	An integer specifying the number of significant digits to display after the decimal point in the output.
robust	A logical value indicating whether to compute robust descriptive statistics. If FALSE (the default), computes the classical descriptive statistics for describing the distribution of a univariate variable.
drop.na	A logical value indicating whether to remove missing values (NA) from the calculations. If TRUE (the default), missing values will be removed. If FALSE, the statistics of variables containing missing values are NA.

Value

A tibble with one row per variable. The classical statistics are the mean, mode, median, IQR, standard deviation, variance, coefficient of variation (cv, in \ the median, MAD (scaled to be consistent at the normal distribution), Qn and Sn estimators, medcouple, left/right medcouples, biweight location, scale and midvariance, robust coefficient of variation (rcv = MAD / median, in \ Robust statistics that cannot be computed (e.g. for a constant variable) are NA.

Author(s)

Christian L. Goueguel

Examples

```
data(forageLIBS)
# mineral contents (%) of the forage samples
summary_stats(forageLIBS[c("Ca", "Mg", "P", "K", "S")])
summary_stats(forageLIBS, var = c("Ca", "K"), robust = TRUE)
```

tukey_gh *Tukey g-and-h Parametric Distribution*

Description

The Tukey g-and-h (TGH) distribution is a flexible family of parametric distributions that can accommodate a wide range of skewness and kurtosis values, making it useful for modeling non-normal data.

Usage

```
tukey_gh(
  x,
  type = "d",
  location = 0,
  scale = 1,
  g = 0,
  h = 0,
  log = FALSE,
  log.p = FALSE,
  n = NULL
)
```

Arguments

x	A numeric vector: quantiles for type = "d" or "p", probabilities for type = "q". Ignored for type = "r".
type	A character string specifying the function to be called (d for density, p for cumulative distribution, q for quantile, or r for random number generation).
location	The location parameter of the TGH distribution.
scale	The scale parameter of the TGH distribution.
g	The skewness parameter of the TGH distribution.
h	The kurtosis parameter of the TGH distribution.
log	If TRUE, the log-density is returned for the density function. Default is FALSE.
log.p	If TRUE, the log-probability is returned for the cumulative distribution and quantile functions. Default is FALSE.
n	For random number generation, the number of random values to be generated. Default is NULL.

Details

The Tukey g-and-h distribution, introduced by John W. Tukey in 1977, is defined by two transformation functions, g and h, which are applied to a standard normal distribution. The g transformation controls the skewness of the distribution, while the h transformation controls the kurtosis (or heaviness of the tails). The TGH distribution is given by:

$$T_{g,h}(Z) = \frac{1}{g}(e^{gZ} - 1)e^{\frac{1}{2}hZ^2}$$

Where Z is a random variable with standard normal distribution. The parameters g and h stand for the bias and elongation of the tails, respectively, of Tukey's g-and-h distribution.

Value

A numeric vector.

Author(s)

Christian L. Goueguel

References

- Tukey, J.W., (1977). Modern techniques in data analysis. NSF-sponsored regional research conference at Southeastern Massachusetts University, North Dartmouth, MA.
- Martinez, J., Iglewicz, B., (1984). Some properties of the Tukey g and h family of distributions. Communications in Statistics: Theory and Methods, 13(3):353–369.
- Hoaglin, D.C., (1985). Summarizing shape numerically: The g-and-h distributions. In: Hoaglin, D.C., Mosteller, F., Tukey, J.W., (eds), Data Analysis for Tables, Trends, and Shapes. New York:Wiley.

Examples

```
x <- seq(-5, 5, length.out = 100)
y1 <- tukey_gh(x, type = "d", location = 0, scale = 1, g = 0, h = 0)
y2 <- tukey_gh(x, type = "d", location = 0, scale = 1, g = 0.5, h = 0)
y3 <- tukey_gh(x, type = "d", location = 0, scale = 1, g = -0.5, h = 0)
y4 <- tukey_gh(x, type = "d", location = 0, scale = 1, g = 0, h = 0.1)
y5 <- tukey_gh(x, type = "d", location = 0, scale = 1, g = 0.5, h = 0.1)
y6 <- tukey_gh(x, type = "d", location = 0, scale = 1, g = -0.5, h = 0.1)
plot(x, y1, type = "b", col = "red", pch = 16,
main = "Tukey g-and-h distribution", ylim = c(0, 0.6))
lines(x, y2, type = "b", col = "blue", pch = 15)
lines(x, y3, type = "b", col = "green", pch = 17)
lines(x, y4, type = "b", col = "black", pch = 6)
lines(x, y5, type = "b", col = "gold", pch = 18)
lines(x, y6, type = "b", col = "orange", pch = 8)
legend(
"topright",
legend = c(
"g = 0, h = 0", "g = 0.5, h = 0", "g = -0.5, h = 0",
"g = 0, h = 0.1", "g = 0.5, h = 0.1", "g = -0.5, h = 0.1"),
col = c("red", "blue", "green", "black", "gold", "orange"),
lty = 1
)
```

umad *Unbiased Median Absolute Deviation*

Description

This function calculates the Median Absolute Deviation (MAD) scale estimator for a numeric vector, using the Park-Kim-Wang approach. The method adds a small sample correction factor to make MAD unbiased at the normal distribution.

Usage

```
umad(x, method = "hayes", drop.na = TRUE)
```

Arguments

`x`	A numeric vector.
`method`	A character string specifying the method to use for calculating the correction factor when the number of sample is more than 100. The available options are "hayes" (default) and "williams".
`drop.na`	A logical value indicating whether to remove missing values (NA) from the calculations. If `TRUE` (the default), missing values will be removed. If `FALSE`, missing values will be included.

Details

The correction factor, C, is calculated differently based on the sample size n:

- For $n > 100$, C is calculated using an analytical approximation proposed by either Hayes (2014) or Williams (2011).
- For $n \leq 100$, C is obtained from a pre-computed table of values proposed by Park *et al.* (2020).

Value

The MAD scale estimate for the input vector `x`.

Author(s)

Christian L. Goueguel

References

- Park, C., Kim, H., Wang, M., (2020). Investigation of finite-sample properties of robust location and scale estimators. Communications in Statistics - Simulation and Computation, 51(5):2619–2645.
- Hayes, K., (2014). Finite-Sample Bias-Correction Factors for the Median Absolute Deviation.Communications in Statistics - Simulation and Computation, 43(10):2205–2212.
- Williams, D.C., (2011). Finite sample correction factors for several simple robust estimators of normal standard deviation. Journal of Statistical Computation and Simulation, 81(11):1697–1702.

Examples

```
set.seed(1)
x <- c(stats::rnorm(50), 10, 12)  # two outliers
umad(x)
stats::mad(x)
```

`voigt_fwhm` *Full Width at Half Maximum of a Voigt Profile*

Description

Computes the full width at half maximum (FWHM) of a Voigt profile from its Gaussian and Lorentzian widths, with the approximation of Olivero and Longbothum (1977), accurate to about 0.02%:

$$f_V \approx 0.5346\, f_L + \sqrt{0.2166\, f_L^2 + f_G^2}$$

Usage

```
voigt_fwhm(wG, wL)
```

Arguments

`wG`, `wL`	The Gaussian and Lorentzian full widths at half maximum, for example the `wG` and `wL` parameters of a Voigt fit with `peak_fit()`. Vectors are recycled.

Details

The total width of a fitted line is often better determined than its split into Gaussian and Lorentzian parts, which trade off against each other when a line is noisy or sampled by few channels. It is the width to use for the Hα line in `electron_density()`, whose Stark profile is not Lorentzian.

Value

A numeric vector of Voigt FWHM, in the units of `wG` and `wL`.

Author(s)

Christian L. Goueguel

References

- Olivero, J.J., Longbothum, R.L. (1977). Empirical fits to the Voigt line width: a brief review. Journal of Quantitative Spectroscopy and Radiative Transfer, 17(2):233-236.

See Also

`voigt_profile()`, `peak_fit()`, `electron_density()`

Examples

```
voigt_fwhm(wG = 0.1, wL = 0)    # Gaussian limit
voigt_fwhm(wG = 0, wL = 0.1)    # Lorentzian limit
voigt_fwhm(wG = 0.1, wL = 0.1)
```

`voigt_profile` *Voigt Function*

Description

Computes the exact Voigt profile, the convolution of a Gaussian and a Lorentzian profile. In laser-induced plasmas, the Gaussian component typically reflects Doppler and instrumental broadening, and the Lorentzian component Stark (pressure) and natural broadening.

Usage

```
voigt_profile(x, y0, xc, wG, wL, A)
```

Arguments

`x`	A numeric vector representing the independent variable (e.g., wavelength).
`y0`	A numeric value specifying the baseline offset.
`xc`	A numeric value representing the center of the peak.
`wG`	A non-negative numeric value specifying the Gaussian full width at half maximum (FWHM).
`wL`	A non-negative numeric value specifying the Lorentzian FWHM.
`A`	A numeric value representing the peak area.

Details

The Voigt function is defined as:

$$y = y_0 + A \int_{-\infty}^{\infty} G(t; w_G)\, L(x - x_c - t; w_L)\, dt = y_0 + A \frac{\mathrm{Re}[w(z)]}{\sigma\sqrt{2\pi}}, \quad z = \frac{x - x_c + i\gamma}{\sigma\sqrt{2}}$$

where $w(z) = e^{-z^2}\mathrm{erfc}(-iz)$ is the Faddeeva function, $\sigma = w_G/(2\sqrt{2\ln 2})$ is the standard deviation of the Gaussian component and $\gamma = w_L/2$ is the half width at half maximum of the Lorentzian component. Both components have unit area, so A is the area of the line.

The Faddeeva function is evaluated in C++ with Weideman's (1994) rational approximation using 32 terms. Its relative error is below 10^{-10} over the range of widths met in emission spectroscopy. A width of zero gives the pure Gaussian or pure Lorentzian profile. See `pseudo_voigt_profile()` for the faster pseudo-Voigt approximation.

Value

A numeric vector containing the values of the Voigt function evaluated at the provided x values.

Author(s)

Christian L. Goueguel

References

- Weideman, J.A.C., (1994). Computation of the complex error function. SIAM Journal on Numerical Analysis, 31(5):1497-1518.
- Armstrong, B.H., (1967). Spectrum line profiles: the Voigt function. J. Quant. Spectrosc. Radiat. Transfer, 7(1):61-88.

See Also

pseudo_voigt_profile(), gaussian_profile(), lorentzian_profile(), peak_fit()

Examples

```
x <- seq(-3, 3, length.out = 200)
v <- voigt_profile(x, y0 = 0, xc = 0, wG = 1, wL = 0.5, A = 1)
pv <- pseudo_voigt_profile(x, y0 = 0, xc = 0, wG = 1, wL = 0.5, A = 1)$y
plot(x, v, type = "l", main = "Voigt vs pseudo-Voigt")
lines(x, pv, col = "red", lty = 2)
max(abs(v - pv)) / max(v) # pseudo-Voigt relative error
```

---

wavelength_calibration

Wavelength Calibration from Reference Lines

Description

Measures the positions of known emission lines in spectra and fits a correction of the wavelength axis, so that the lines fall at their reference wavelengths (for example, from the NIST Atomic Spectra Database). apply_calibration() then relabels the wavelengths of the spectra, without changing their intensities.

Usage

```
wavelength_calibration(
  spectra,
  lines,
  degree = 1,
  search = 0.3,
  min_snr = 20,
```

```
  segments = TRUE,
  reject = TRUE
)
```

Arguments

`spectra`	The spectra: a numeric vector named by wavelength, or a matrix or data frame with one spectrum per row and the wavelengths as column names (other columns are ignored). The lines are located in their mean.
`lines`	The reference lines: a numeric vector of wavelengths (nm), preferably named, or a data frame with a `wavelength` column (and optionally a `species` column), such as returned by `nist_lines()`. Choose isolated, strong but not saturated lines, spread over the range.
`degree`	The degree of the correction polynomial: 0 (shift), 1 (default) or 2.
`search`	The half-width of the window in which each line is searched, in nm. Default is 0.3.
`min_snr`	The minimum height of a line, in units of the noise of the spectrum. Default is 20.
`segments`	A logical: fit each detector segment separately (`TRUE`, default).
`reject`	A logical: remove the outlying lines after the fit (`TRUE`, default).

Details

The lines are located in the mean of the spectra. For each reference line, the peak is the channel of highest intensity within `search` nm of the reference wavelength, and its position is refined to a fraction of a channel by the vertex of the parabola through the logarithms of the peak channel and its two neighbors, above the lowest channel of the window (Gaussian interpolation, exact for Gaussian lines; Caruana et al., 1986). A line is not used when

- `"edge"`: the highest channel is at the border of the search window (the peak is not in the window, or a stronger line is nearby);
- `"weak"`: its height above the lowest channel of the window is less than `min_snr` times the noise of the spectrum;
- `"flat"`: three channels or more are within 1% of its maximum, or its curvature is not that of a peak, as for saturated or strongly self-absorbed lines;
- `"outlier"`: its residual after the fit exceeds 3 robust standard deviations of the residuals (for example, a blend, or a line identified wrongly), when `reject = TRUE`.

The correction is a polynomial of degree `degree` of the measured wavelength, $\lambda_{true} = \lambda + \sum_{j=0}^{d} c_j(\lambda - \lambda_0)^j$, where λ_0 is the mean position of the lines: a constant shift for `degree = 0`, linear for `degree = 1` (default). It needs at least `degree + 1` usable lines, and more to estimate its residual error. Beyond the range of the lines, the polynomial is extrapolated: spread the reference lines over the spectral range, and prefer low degrees.

Instruments made of several spectrometers have one wavelength calibration per detector. With `segments = TRUE` (default), the axis is split where the wavelengths step back (overlapping detectors) or jump (gaps of more than 5 times the median channel spacing), and each segment gets its own correction. A segment with fewer usable lines than `degree + 1` gets a constant shift if it has one line, and no correction otherwise, with a warning.

Value

An object of class specproc_wavelength_calibration, a list with

- lines: a tibble with, for each reference line, its line label, reference and measured wavelengths, the offset (reference minus measured), its segment, snr, whether it was used (otherwise the reason), the correction of the fit at the line and the residual;
- segments: a tibble with, for each segment, its wavelength range (from, to), the degree fitted, the number of lines n_lines and the root mean square residual rmse (nm);
- wavelength: the wavelengths of the calibrated axis, in column order.

Use apply_calibration() to correct spectra, predict() to correct any wavelength, and plot_wavelength_calibration() to draw the fit.

Author(s)

Christian L. Goueguel

References

- Caruana, R.A., Searle, R.B., Heller, T., Shupack, S.I. (1986). Fast algorithm for the resolution of spectra. Analytical Chemistry, 58(6):1162-1167.

See Also

apply_calibration(), plot_wavelength_calibration(), nist_lines(), line_intensities(), line_finder()

Examples

```
# the first detector (199 to 766 nm) of the forage spectra
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
first_detector <- spectra[seq_len(which(diff(wl) < 0)[1])]
reference <- c(`C I` = 247.856, `Mg I` = 285.213, `Ca II` = 317.933, `Ca II` = 370.603,
               `Ca II` = 373.690, `K I` = 404.414, `Ca I` = 430.253, `Ca I` = 445.478,
               `Mg I` = 516.732, `Mg I` = 517.268, `Mg I` = 518.360, `Na I` = 588.995,
               `Na I` = 589.592, `Ca I` = 616.217, `Ca I` = 643.907)
cal <- wavelength_calibration(first_detector, reference)
cal
plot_wavelength_calibration(cal)

corrected <- apply_calibration(first_detector, cal)
head(names(corrected))
```

wavelet_features *Wavelet Coefficients of Spectra*

Description

Computes the discrete wavelet transform (DWT) of spectra and returns its coefficients as features: a compressed representation of the spectra for modeling, with far fewer variables than channels.

Usage

```
wavelet_features(x, wavelet = "d4", level = 3, coefficients = "approximation")
```

Arguments

`x`	A numeric matrix or data frame of spectra, one per row, or a numeric vector (one spectrum).
`wavelet`	The wavelet: `"haar"`, `"d4"` (default), `"d6"`, `"d8"` or `"la8"`.
`level`	The number of levels of the decomposition. Default is 3.
`coefficients`	The coefficients returned: `"approximation"` (default) or `"all"`.

Details

The DWT decomposes a spectrum, level by level, into approximation coefficients (the smooth part, at half the resolution of the previous level) and detail coefficients (the fine structure removed at that level), with the pyramid algorithm of Mallat (1989) and an orthonormal wavelet:

- `"haar"`: the Haar wavelet (2 coefficients);
- `"d4"`, `"d6"`, `"d8"`: Daubechies' extremal phase wavelets with 4, 6 and 8 coefficients;
- `"la8"`: Daubechies' least asymmetric wavelet with 8 coefficients (symlet 4), whose nearly symmetric shape suits spectral lines.

The coefficients kept are

- `"approximation"` (default): the approximation at `level`, a smoothed version of the spectrum with about 2^{-level} times as many values;
- `"all"`: the approximation at `level` and the details of every level, the complete transform (as many values as channels, after padding).

The spectrum is extended by reflection at its end to a length divisible by 2^{level}, and the transform is periodic. The transform is orthonormal, so the sum of squares of all the coefficients equals that of the (padded) spectrum. As features for a model, the approximation keeps the broad shape of the spectrum and drops the noise; to keep only the most informative coefficients, `step_wavelet()` can select those with the largest variance in the training data (Trygg and Wold, 1998).

Value

A matrix with one row per spectrum and one column per coefficient, named `A<level>_<i>` for the approximation and `D<j>_<i>` for the details of level `j`.

Author(s)

Christian L. Goueguel

References

- Mallat, S.G. (1989). A theory for multiresolution signal decomposition: the wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(7):674-693.
- Daubechies, I. (1992). Ten Lectures on Wavelets. SIAM, Philadelphia.
- Trygg, J., Wold, S. (1998). PLS regression on wavelet compressed NIR spectra. Chemometrics and Intelligent Laboratory Systems, 42(1-2):209-220.

See Also

`step_wavelet()`, `savitzky_golay()`

Examples

```
data(forageLIBS)
spectra <- forageLIBS[1:4, -(1:14)]
approx <- wavelet_features(spectra, wavelet = "la8", level = 4)
dim(approx)   # 7152 channels -> 447 coefficients
```

---

wavelet_level *Number of Levels of a Wavelet Decomposition*

Description

A dials parameter for the `level` of `step_wavelet()`.

Usage

```
wavelet_level(range = c(1L, 6L), trans = NULL)
```

Arguments

`range`	The range of levels. Default is 1 to 6.
`trans`	Not used.

Value

A dials `quant_param` object.

See Also

`step_wavelet()`

Examples

```
wavelet_level()
dials::value_seq(wavelet_level(), 3)
```

`y_gradient_glsw` *y-Gradient Generalized Least Squares Weighting*

Description

The y-gradient generalized least squares weighting algorithm (GLSW) removes variance from the data (spectra), which is orthogonal to the response.

Usage

```
y_gradient_glsw(x, y, alpha = 0.01, window = 5)
```

Arguments

`x`	A numeric matrix, data frame or tibble, representing the predictors data.
`y`	A numeric vector representing the response vector.
`alpha`	A positive numeric value specifying the weighting parameter. Typical values range from 1 to 0.0001. Default is 0.01.
`window`	An odd integer giving the width of the Savitzky-Golay window used to compute the gradients. Default is 5.

Details

The y-Gradient GLSW is an alternative method to GLSW, where a continuous **y**-variable is used to develop pseudo-groupings of samples in **X** by comparing the differences in the corresponding **y** values. This is referred to as the *"gradient method"* because it utilizes a gradient of the sorted **X**- and **y**-blocks to calculate a covariance matrix.

The samples are sorted by increasing **y**, and the first derivatives of **X** and **y** along the sample axis are computed with a Savitzky-Golay filter. Samples whose neighbors have similar **y** values receive large weights $w_i = 2^{-\Delta y_i / s_{\Delta y}}$, so the differences between their spectra, $\Delta\mathbf{X}$, describe variation unrelated to **y**. The filter is then built as in `glsw()` from $\mathbf{C} = \Delta\mathbf{X}^T\mathbf{W}^2\Delta\mathbf{X}$.

Value

A tibble containing the $p \times p$ filtering matrix.

Author(s)

Christian L. Goueguel

References

- Zorzetti, B.M., Shaver, J.M., Harynuk, J.J., (2011). Estimation of the age of a weathered mixture of volatile organic compounds. Analytica Chimica Acta, 694(1-2):31–37.

See Also

`step_y_gradient_glsw()` to use the filter in a tidymodels recipe.

Examples

```
data(forageLIBS)
spectra <- forageLIBS[-(1:14)]
wl <- as.numeric(names(spectra))
x <- spectra[wl > 760 & wl < 780]  # the K I resonance lines
G <- y_gradient_glsw(x, forageLIBS$K, alpha = 0.01)
filtered <- as.matrix(x) %*% as.matrix(G)
dim(filtered)
```

`zscore` *Classical or Robust Z-Score*

Description

This function calculates classical or robust z-score (standardization) for a numeric vector.

Usage

```
zscore(x, cutoff = 3, robust = FALSE, drop.na = FALSE)
```

Arguments

`x`	A numeric vector.
`cutoff`	A numeric value indicating the threshold above which data points are identified and flagged as potential outliers. By default, `cutoff = 3`.
`robust`	A logical value indicating whether to calculate classical or robust z-score. If `FALSE` (the default), uses the classical approach. If `TRUE`, computes the robust method, i.e. the so-called Stahel-Donoho outlyingness.
`drop.na`	A logical value indicating whether to remove missing values (`NA`) from the calculations. If `TRUE`, missing values will be removed. If `FALSE` (the default), missing values are kept in the output (with a missing score) and ignored when computing the location and scale.

Details

Z-scores are useful for comparing data points from different distributions because they are dimensionless and standardized. A positive z-score indicates that the data point is above the mean (or the median in the robust approach), while a negative z-score indicates that the data point is below the mean (or the median). One common rule to detect outliers using z-scores is the "three-sigma rule", in which data points with an absolute z-score greater than 3 ($|z| > 3$) can be considered potential outliers (default), as they fall outside the range that covers 99.7% of the data points in a normal distribution. (Note that a cutoff of $|z| > 2.5$ is also often used).

Value

A tibble with two columns:

- `data`: The original numeric values.
- `score`: The calculated z-scores.
- `flag`: `TRUE` if the corresponding data point is flagged as a potential outlier, and `FALSE` otherwise.

Author(s)

Christian L. Goueguel

References

- Rousseeuw, P. J., and Croux, C. (1993). Alternatives to the median absolute deviation. Journal of the American Statistical Association, 88(424), 1273-1283.
- Rousseeuw, P. J., and Hubert, M. (2011). Robust statistics for outlier detection. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 1(1), 73-79.
- Donoho, D., (1982). Breakdown properties of multivariate location estimators. Ph.D. Qualifying paper, Dept. Statistics, Harvard University, Boston.
- Stahel, W., (1981). Robuste Schätzungen: infinitesimale Optimalität und Schätzungen von Kovarianzmatrizen. PhD thesis, ETH Zürich.

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
head(zscore(fe))
head(zscore(fe, robust = TRUE))
```

Index

∗ **datasets**
forageLIBS, 51

adjusted_boxplot, 5
apply_calibration, 7
apply_calibration(), *129*, *215*, *217*
average, 8
average(), *140*, *195*

baseline_als, 9
baseline_als(), *163–165*
baseline_arpls, 10
baseline_arpls(), *163–165*
baseline_lambda, 11
baseline_lambda(), *164*
baseline_lsp, 12
baseline_lsp(), *163–165*
biweight_location, 14
biweight_midcorrelation, 15
biweight_midcovariance, 16
biweight_midvariance, 17
biweight_scale, 18
boltzmann, 19
boltzmann(), *28*, *33, 34*, *64*, *82, 83*, *97*, *148, 149*

calibration_curve, 21
calibration_curve(), *64, 65*, *98*, *121, 122*
cellpca, 23
cellpca(), *49, 50*, *68*, *71*, *99, 100*, *113, 114*, *165–167*
center, 26
center(), *31*
cf_libs, 27
cf_libs(), *33, 34*, *64, 65*, *81, 82*, *97*
ConfidenceEllipse::confidence_ellipse(), *104*
contributions, 30
contributions(), *101, 102*
correct_self_absorption, 32
correlation, 34

dials, *11*, *57*
dials::degree_int(), *164*, *171*
dials::num_comp(), *184*, *187*, *189*, *197*
dials::penalty(), *200*
direct_orthogonal, 37
direct_orthogonal(), *79, 80*, *124, 125*, *167, 168*
direct_osc, 39
direct_osc(), *124, 125*, *169, 170*
directional_outlyingness, 40
dmodx, 42
dmodx(), *108, 109*, *135*
dplyr::bind_rows(), *160*

electron_density, 44
electron_density(), *28*, *30*, *148, 149*, *153*, *158*, *160, 161*, *163*, *213*
emsc, 46
emsc(), *123*, *170*, *172*
epo, 48
epo(), *47*, *124, 125*, *172, 173*

flagged_regions, 49
flagged_regions(), *100, 101*
forageLIBS, 51

gaussian_profile, 52
gaussian_profile(), *96*, *215*
generalized_boxplot, 53
ggplot2::geom_point(), *108*, *114*
glsw, 55
glsw(), *174, 175*, *220*
glsw_alpha, 57
glsw_alpha(), *175*, *208*

hotelling_t2, 57
hotelling_t2(), *42, 43*, *104, 105*, *134, 135*
HotellingEllipse::ellipseCoord(), *105*
HotellingEllipse::ellipseParam(), *57*

iqr_outliers, 59

libs_lines, 61
libs_lines(), *50*, *63*, *66, 67*, *100*, *102*, *110–113*
line_finder, 62
line_finder(), *61, 62*, *111*, *217*
line_intensities, 64
line_intensities(), *22, 23*, *34*, *175*, *177*, *217*
loading_peaks, 66
loading_peaks(), *112, 113*
lorentzian_profile, 67
lorentzian_profile(), *96*, *215*

macropca, 68
macropca(), *23*, *25, 26*, *31*, *49, 50*, *58*, *66*, *99–101*, *103–105*, *111–115*, *125, 126*, *142*, *179, 180*
mcwhirter_criterion, 71
mcwhirter_criterion(), *20*
medcouple_weight, 72
minmax, 74
msc, 75
msc(), *46, 47*, *181, 182*
multipeak_fit, 76
multipeak_fit(), *45*, *106*

nas, 78
nas(), *23*, *38*, *126, 127*
nist_ionization_energy, 80
nist_ionization_energy(), *28*, *33*, *82, 83*
nist_levels, 81
nist_levels(), *28*, *30*
nist_lines, 82
nist_lines(), *28–30*, *33*, *61–65*, *81, 82*, *216, 217*
normalize, 84
normalize(), *179*, *204, 205*

o2pls, 85
o2pls(), *88*, *124, 125*, *182, 183*, *185*, *192*
opls, 87
opls(), *127, 128*, *131*, *186, 187*
osc, 90
osc(), *124, 125*, *187–189*

pareto_scale, 92
pareto_scale(), *88*, *190, 191*
parsnip::pls(), *184*, *189*
partition_function *(*nist_levels*)*, 81
partition_function(), *29*
pds, 93
peak_fit, 95
peak_fit(), *45*, *65*, *77*, *106*, *213*, *215*
plot_boltzmann, 97
plot_boltzmann(), *20*, *29, 30*, *149*
plot_calibration, 98
plot_calibration(), *22, 23*, *122*
plot_cell_map, 99
plot_cell_map(), *23*, *26*, *49, 50*, *68*, *71*, *115*
plot_contributions, 101
plot_contributions(), *31, 32*
plot_embedding, 103
plot_embedding(), *59*
plot_fit, 106
plot_influence, 108
plot_influence(), *43*, *135*
plot_lines, 110
plot_lines(), *61–63*
plot_loadings, 111
plot_loadings(), *66, 67*, *102*, *105*
plot_outlier_map, 113
plot_outlier_map(), *23*, *26*, *32*, *68*, *71*, *101*, *104, 105*, *108, 109*, *113*, *126*, *135*, *142*, *146*
plot_outliers, 115
plot_som, 117
plot_som(), *129*, *156*
plot_spectra, 118
plot_wavelength_calibration, 119
plot_wavelength_calibration(), *217*
poisson_scale, 120
poisson_scale(), *191, 192*
predict(), *22, 23*, *38, 39*, *47, 48*, *68*, *79*, *89*, *91*, *98*, *114*, *131*, *142*, *154*, *156*, *217*
predict.o2pls *(*predict.specproc_filter*)*, 124
predict.o2pls(), *185*
predict.specproc_calibration, 121
predict.specproc_calibration(), *23*, *98*
predict.specproc_cellpca *(*predict.specproc_macropca*)*, 125
predict.specproc_direct_orthogonal *(*predict.specproc_filter*)*, 124
predict.specproc_direct_osc

(predict.specproc_filter), 124
predict.specproc_emsc, 123
predict.specproc_epo
(predict.specproc_filter), 124
predict.specproc_filter, 124
predict.specproc_filter(), *38*, *40*, *49*, *87*, *92*, *123*, *131*, *168*, *170*, *173*, *189*, *194*
predict.specproc_macropca, 125
predict.specproc_nas, 126
predict.specproc_opls, 127
predict.specproc_opls(), *90*, *187*
predict.specproc_osc
(predict.specproc_filter), 124
predict.specproc_projected_osc
(predict.specproc_filter), 124
predict.specproc_robpca
(predict.specproc_macropca), 125
predict.specproc_som, 128
predict.specproc_som(), *118*, *156*
predict.specproc_wavelength_calibration, 129
projected_osc, 130
projected_osc(), *88*, *124*, *125*, *192–194*
pseudo_voigt_profile, 132
pseudo_voigt_profile(), *96*, *214*, *215*

q_residuals, 134
q_residuals(), *31*, *32*, *43*, *59*, *108*, *109*
quantile_weight, 136

read_starkb, 138
read_starkb(), *45*, *160*, *163*
recipes::bake(), *164*, *166*, *168*, *169*, *171*, *173*, *174*, *176*, *178*, *180*, *182*, *183*, *186*, *188*, *190*, *192*, *193*, *196*, *198*, *200*, *201*, *203*, *204*, *206*, *208*
recipes::selections(), *164*, *166*, *168*, *169*, *171*, *172*, *174*, *176*, *178*, *180*, *181*, *183*, *186*, *188*, *190*, *191*, *193*, *196*, *198*, *200*, *201*, *203*, *204*, *206*, *207*
recipes::step_BoxCox(), *197*
recipes::step_center(), *190*
recipes::step_normalize(), *204*
recipes::step_pca(), *196*
recipes::step_YeoJohnson(), *197*
recipes::update_role(), *195*
reject_shots, 139
reject_shots(), *194*, *195*
robpca, 141
robpca(), *23*, *25*, *26*, *31*, *58*, *59*, *66*, *68–71*, *103–105*, *111–115*, *125*, *126*, *135*, *145*, *146*, *156*, *196*, *197*
robust_bcyj, 143
robust_bcyj(), *197–199*
rospca, 145
rospca(), *31*, *58*, *66*, *103–105*, *111–115*, *125*, *126*, *142*, *199*, *200*
rousseeuw_croux, 147
rsample::group_vfold_cv(), *195*

saha_boltzmann, 148
saha_boltzmann(), *20*, *29*, *30*, *33*, *64*, *80–83*, *97*
saturation_summary, 149
saturation_summary(), *65*, *140*, *178*
savgol_derivative, 150
savitzky_golay, 151
savitzky_golay(), *201*, *202*, *219*
self_absorption, 152
self_absorption(), *20*, *29*, *34*, *45*, *178*
set.seed(), *88*, *142*, *155*
shiny::runApp(), *62*
snv, 153
snv(), *202*, *203*
som, 154
som(), *117*, *118*, *128*, *129*, *157*
som_stability, 157
som_stability(), *156*
spectral_norm_method, 158
stark_table, 158
stark_table(), *45*, *138*, *163*
stark_width, 160
stark_width(), *44*, *45*, *153*, *158*, *160*, *163*
starkb_lines, 162
starkb_lines(), *44*, *45*, *83*, *138*, *158*, *160*, *161*
stats::hclust(), *36*
stats::prcomp(), *31*, *42*, *58*, *66*, *103–105*, *111*, *112*, *134*
step_baseline, 163
step_baseline(), *11*
step_cellpca, 165
step_cellpca(), *26*
step_direct_orthogonal, 167
step_direct_orthogonal(), *38*, *168*, *170*, *188*, *193*
step_direct_osc, 169

step_direct_osc(), *40*, *168*, *170*, *188*, *193*
step_emsc, 170
step_emsc(), *47*, *182*
step_epo, 172
step_epo(), *49*, *168*, *170*, *174, 175*, *189*, *193*
step_glsw, 174
step_glsw(), *56, 57*, *168*, *170*, *189*, *193*, *208*
step_line_intensities, 175
step_line_intensities(), *65*
step_line_ratio, 177
step_line_ratio(), *177*, *195*, *205*
step_macropca, 179
step_macropca(), *71*, *167*, *197*
step_msc, 181
step_msc(), *172*
step_o2pls, 182
step_o2pls(), *87*, *168*, *170*, *188*, *193*
step_opls, 186
step_opls(), *90*, *128*, *168*, *170*, *184, 185*, *188*, *193*
step_osc, 187
step_osc(), *92*, *125*, *187*
step_pareto_scale, 190
step_pareto_scale(), *88*, *187*, *192*
step_poisson_scale, 191
step_poisson_scale(), *191*
step_projected_osc, 192
step_projected_osc(), *131*, *168*, *170*, *184*, *187, 188*, *193*
step_reject_shots, 194
step_reject_shots(), *140*, *179*
step_robpca, 196
step_robpca(), *142*, *166, 167*, *180*, *200*
step_robust_bcyj, 197
step_rospca, 199
step_rospca(), *146*, *197*
step_savgol, 201
step_savgol(), *150–152*, *207*
step_snv, 202
step_snv(), *202*, *205*
step_spectral_norm, 204
step_spectral_norm(), *158*
step_wavelet, 205
step_wavelet(), *218, 219*
step_y_gradient_glsw, 207
step_y_gradient_glsw(), *57*, *168*, *170*, *175*, *189*, *193*, *221*
summary_stats, 209
tidy(), *165*, *171*, *173*, *175*, *177, 178*, *182*, *185*, *187*, *189, 190*, *192*, *195*, *197, 198*, *202, 203*, *205, 206*, *208*
tukey_gh, 210
tune::tune(), *164*, *171*, *173*, *175*, *184*, *187*, *189*, *197*, *200*, *208*
tune::tune_grid(), *168*, *170*, *184*, *188*, *193*

umad, 212

voigt_fwhm, 213
voigt_fwhm(), *45*
voigt_profile, 214
voigt_profile(), *96*, *133*, *213*

wavelength_calibration, 215
wavelength_calibration(), *7*, *112*, *120*, *129*
wavelet_features, 218
wavelet_features(), *205*, *207*
wavelet_level, 219
workflows::workflow(), *168*, *170*, *188*, *193*

y_gradient_glsw, 220
y_gradient_glsw(), *207, 208*

zscore, 221