Package 'specProc'

November 20, 2026

Type Package

Title Analysis of Laser-Induced Breakdown Spectroscopy Data

Version 0.9.0

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>), wavelength selection (variable importance in projection, selectivity ratio, interval partial least squares), 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>), robust partial least squares regression (Hubert and Vanden Branden (2003) <doi:10.1002/cem.822>) and principal component regression (Hubert and Verboven (2003) <doi:10.1002/cem.783>) with their robust cross-validation (Engelen and Hubert (2005) <doi:10.1016/j.aca.2005.01.015>), robust classification by SIMCA (Vanden Branden and Hubert (2005) <doi:10.1016/j.chemolab.2005.03.002>) and discriminant analysis (Hubert and Van Driessen (2004) <doi:10.1016/S0167-9473(02)00299-2>), 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, and the robust models as 'parsnip' models and engines, 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, generics, ggplot2 (>= 3.5.0), ggrepel, lifecycle, magrittr, MASS, minpack.lm, moments, pls, Rcpp, rlang, robustbase, scales (>= 1.3.0), tibble, tidyr

Suggests bslib, cellWise, cli, ConfidenceEllipse, dials, embed, future, future.apply, HotellingEllipse (>= 1.3.0), jsonlite, knitr, mixOmics, parsnip (>= 1.0.0), patchwork, plotly, plsmod, prospectr, purrr, ragg, recipes (>= 1.1.0), rmarkdown, ropls, rospca, rsample, 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 CRAN

Date/Publication 2026-11-20 10:00:03 UTC

Contents

adjusted_boxplot 7
aki_uncertainty 12
applicability_domain 13
apply_calibration 14
average 15
baseline_als 16
baseline_arpls 18
baseline_lambda 19
baseline_lsp 20
bin_spectra 22

biweight_location 23
biweight_midcorrelation 24
biweight_midcovariance 25
biweight_midvariance 26
biweight_scale 27
boltzmann 29
calibration_curve 30
cellpca 34
center 37
cf_libs 38
contributions 41
control_chart 43
correct_response 45
correct_self_absorption 47
correlation 49
curve_of_growth 52
curve_of_growth_model 54
directional_outlyingness 55
direct_orthogonal 57
direct_osc 59
dmodx 60
doppler_width 62
download_cache 63
ds 65
electron_density 66
electron_density_uncertainty 68
emsc 70
epo 72
exomol_datasets 74
flagged_regions 75
forageLIBS 76
forageShots 77
gap_derivative 78
gaussian_profile 80
generalized_boxplot 82
glsw 86
glsw_alpha 87
hotelling_t2 88
iqr_outliers 90
libs_lines 92
linear_reg_lts 93
line_finder 95
line_intensities 97
loading_peaks 99
lorentzian_profile 100
lte_criteria 102
macropca 104
mcwhirter_criterion 107

medcouple . . . 108
medcouple_weight . . . 110
minmax . . . 112
molecular_bands . . . 113
molecular_spectrum . . . 115
msc . . . 117
multipeak_fit . . . 119
nas . . . 121
nist_ionization_energy . . . 124
nist_levels . . . 125
nist_lines . . . 126
normalize . . . 127
num_intervals . . . 129
o2pls . . . 130
opls . . . 132
osc . . . 135
pareto_scale . . . 137
pds . . . 138
peak_fit . . . 140
plasma_uncertainty . . . 142
plot_boltzmann . . . 144
plot_calibration . . . 146
plot_cell_map . . . 147
plot_contributions . . . 150
plot_control_chart . . . 152
plot_coomans . . . 153
plot_curve_of_growth . . . 155
plot_embedding . . . 156
plot_fit . . . 161
plot_influence . . . 163
plot_lines . . . 165
plot_loadings . . . 167
plot_medcouple . . . 170
plot_outliers . . . 172
plot_outlier_map . . . 175
plot_response . . . 178
plot_shots . . . 179
plot_som . . . 182
plot_spectra . . . 184
plot_wavelength_calibration . . . 188
plot_wavelength_selection . . . 189
pls_rsimpls . . . 191
poisson_scale . . . 192
predict.specproc_calibration . . . 193
predict.specproc_domain . . . 195
predict.specproc_emsc . . . 196
predict.specproc_filter . . . 197
predict.specproc_macropca . . . 198

predict.specproc_nas . . . 199
predict.specproc_opls . . . 200
predict.specproc_robust_da . . . 201
predict.specproc_rpcr . . . 202
predict.specproc_rsimca . . . 203
predict.specproc_rsimpls . . . 204
predict.specproc_slope_bias . . . 205
predict.specproc_som . . . 206
predict.specproc_transfer . . . 207
predict.specproc_transformation . . . 208
predict.specproc_wavelength_calibration . . . 210
projected_osc . . . 211
pseudo_voigt_profile . . . 213
quantile_weight . . . 215
q_residuals . . . 217
read_starkb . . . 220
read_vald . . . 221
reject_shots . . . 223
response_calibration . . . 225
robpca . . . 227
robust_boxcox . . . 230
robust_da . . . 233
robust_gandh . . . 235
robust_rmsecv . . . 238
robust_yeojohnson . . . 241
rospca . . . 244
rousseeuw_croux . . . 246
rpcr . . . 248
rsimca . . . 250
rsimpls . . . 253
saha_boltzmann . . . 256
saturation_summary . . . 258
savgol_derivative . . . 259
savitzky_golay . . . 259
select_wavelengths . . . 261
self_absorption . . . 264
simca . . . 265
simca_gamma . . . 268
slope_bias . . . 269
snv . . . 270
som . . . 271
som_stability . . . 273
specproc-tidy . . . 274
specproc-tidy-models . . . 277
spectral_norm_method . . . 281
sst . . . 282
starkb_lines . . . 283
stark_table . . . 285

stark_width . . . 287
step_baseline . . . 288
step_bin_spectra . . . 290
step_cellpca . . . 292
step_direct_orthogonal . . . 293
step_direct_osc . . . 295
step_emsc . . . 296
step_epo . . . 298
step_gap_derivative . . . 300
step_glsw . . . 301
step_line_intensities . . . 303
step_line_ratio . . . 305
step_macropca . . . 307
step_msc . . . 308
step_o2pls . . . 310
step_opls . . . 313
step_osc . . . 315
step_pareto_scale . . . 317
step_poisson_scale . . . 319
step_projected_osc . . . 320
step_reject_shots . . . 322
step_robpca . . . 323
step_robust_boxcox . . . 325
step_robust_gandh . . . 327
step_robust_yeojohnson . . . 329
step_rospca . . . 330
step_rsimpls . . . 332
step_savgol . . . 334
step_select_wavelengths . . . 335
step_snv . . . 338
step_spectral_norm . . . 339
step_wavelet . . . 341
step_wavelet_denoise . . . 342
step_y_gradient_glsw . . . 344
summary_stats . . . 346
tukey_gh . . . 347
umad . . . 349
voigt_fwhm . . . 350
voigt_profile . . . 352
wavelength_calibration . . . 353
wavelet_denoise . . . 356
wavelet_features . . . 358
wavelet_level . . . 360
y_gradient_glsw . . . 361
zscore . . . 362

Index **365**

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,
  id = NULL,
  group = NULL,
  scales = c("free_y", "fixed"),
  points = c("outliers", "all", "none"),
  label_outliers = FALSE,
  show_n = TRUE,
  show_mean = FALSE,
  annotate = NULL,
  horizontal = FALSE,
  log = FALSE,
  fill = "grey85",
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  caption = TRUE,
  base_size = 11,
  point_size = NULL,
  point_alpha = 0.75,
  x_labels_angle = 0,
  box_width = 0.5,
  notch = FALSE,
  notch_width = 0.5,
  staple_width = 0.5,
  xlabels.angle = deprecated(),
  xlabels.vjust = deprecated(),
  xlabels.hjust = deprecated(),
  box.width = deprecated(),
  notchwidth = deprecated(),
  staplewidth = deprecated()
)
```

Arguments

x	A data frame (or matrix) of numeric variables, with optionally the columns id and group.
plot	A logical value indicating whether to plot the boxplot (default is TRUE) or to return its statistics.
id	Optional column of x (unquoted or as a string) identifying the observations, such as sample names: it names the outlying observations in the outliers table and with label_outliers.
group	Optional column of x (unquoted or as a string) of groups, such as sites or treatments: the boxes of each group are drawn side by side, with their own statistics. Rows with a missing group are left out.
scales	"free_y" (default), one panel per variable with its own value axis, or "fixed", a common value axis.
points	The observations drawn: "outliers" (default), the outlying ones; "all"; or "none".
label_outliers	A logical: label the outlying observations with their id, else their row number (FALSE, default). The labels are moved apart so that they do not overlap; where many outliers crowd together, some are left unlabeled.
show_n	A logical: write the number of values below each box (TRUE, default).
show_mean	A logical: mark the mean of each box with a diamond (FALSE, default).
annotate	Statistics added to each box (none by default): some of "shape", "outliers", "fences", "median", "spread", "location", "test" and "missing", or "all" (see Details).
horizontal	A logical: horizontal boxes (FALSE, default), suited to many variables or long names.
log	A logical: a logarithmic value axis (FALSE, default), for variables spanning orders of magnitude. The statistics are those of the data, not of their logarithm.
fill	The fill color of the boxes (default "grey85"), or one color per group (or per variable without group).
xlab, ylab	The titles of the axis of the variables (or groups) and of the value axis. Default is none.
title	The plot title. A long title is split into a title and a subtitle.
caption	TRUE (default), a caption naming the boxplot and its whiskers; FALSE, no caption; or a caption of your own.
base_size	The size of the text, in points. Default is 11.
point_size	The size of the points of the observations. Default is 1.6 at a base_size of 11, scaled with base_size.
point_alpha	The opacity of the outlying observations, between 0 and 1 (default 0.75); the others, drawn with points = "all", are two thirds as opaque.
x_labels_angle	The angle (in degrees) of the labels below the boxes. Default is 0 (horizontal).
box_width	The width of the boxes (default 0.5).

`notch` A logical value indicating whether to draw notches (default is `FALSE`): the notch spans the median $\pm 1.58\,\mathrm{IQR}/\sqrt{n}$ (McGill et al., 1978); two medians whose notches do not overlap differ roughly at the 5% level.

`notch_width` The width of the notch relative to the box (default 0.5).

`staple_width` The width of the staples at the ends of the whiskers, relative to the box (default 0.5).

`xlabels.angle`, `xlabels.vjust`, `xlabels.hjust`, `box.width`, `notchwidth`, `staplewidth`

[Deprecated] Use `x_labels_angle`, `box_width`, `notch_width` and `staple_width`; the justification of the labels now follows their angle.

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. The fences are, for $\mathrm{MC} \geq 0$,

$$[Q_1 - 1.5e^{-4\,\mathrm{MC}}\,\mathrm{IQR};\ Q_3 + 1.5e^{3\,\mathrm{MC}}\,\mathrm{IQR}]$$

and, for $\mathrm{MC} < 0$,

$$[Q_1 - 1.5e^{-3\,\mathrm{MC}}\,\mathrm{IQR};\ Q_3 + 1.5e^{4\,\mathrm{MC}}\,\mathrm{IQR}],$$

calibrated by Hubert and Vandervieren (2008) so that about 0.7% of the observations of a clean, moderately skewed distribution are flagged, as with Tukey's boxplot for normal data. The whiskers end at the most extreme observations within the fences, and the observations outside are drawn as points. The statistics are computed by `robustbase::adjboxStats()`, whose quartiles are Tukey's hinges, as in `boxplot()`. The method suits distributions that are not excessively skewed, i.e. with $|\mathrm{MC}| \leq 0.6$.

Layout. Each variable is drawn in its own panel, with its own value axis (`scales = "free_y"`), so that variables of different scales or units stay readable; `scales = "fixed"` draws them in one panel, on a common axis. With `group`, the boxes of the groups are side by side in the panel of each variable. Below each box, `n` is the number of non-missing values.

Points. By default, the outlying observations are drawn, slightly spread sideways so that equal values do not hide each other. `points = "all"` draws every observation, the outlying ones in black, as some journals require; `label_outliers = TRUE` names the outlying observations by `id` (else by row number), and the `outliers` table of `plot = FALSE` gives them all.

Annotations. `annotate` adds, for each box:

- `"shape"`: the medcouple (`MC`, marked `(> 0.6)` beyond the range of the method), or the `g` and `h` of the generalized boxplot, which explain the asymmetry of the whiskers;
- `"outliers"`: the number of flagged values, with the number expected in clean data (about 0.7% for the adjusted boxplot, which Hubert and Vandervieren calibrated as Tukey's boxplot for normal data, and `alpha` for the generalized one): many more flagged values than expected point to real outliers;
- `"fences"`: the fences beyond which values are flagged, as dashed marks (on the sides with outlying values), the whiskers ending at the last values inside;

- `"median"`: the median with its distribution-free 95% confidence interval, from the order statistics (`Md`); the notches are made to compare two medians, and are not a confidence interval of one;
- `"spread"`: the interquartile range and the robust coefficient of variation rCV $=$ IQR$/(1.349\,\text{median})$, in percent, read as a relative standard deviation;
- `"location"`: the biweight location (`biweight_location()`), a robust mean, marked by a cross (the mean, with `show_mean`, is pulled by the outlying values);
- `"test"`: with `group`, the p-value of the Kruskal-Wallis test between the groups, below the variable name. With several variables, adjust the p-values for multiple testing (see `p.adjust()` and the `tests` table);
- `"missing"`: the number of missing values, next to `n`.

`annotate = "all"` adds them all (the test only with `group`). The caption gives their key. With several boxes in a panel, each statistic takes two short lines: for many groups, make the figure wider or choose fewer annotations. All the values are in the `stats` and `tests` tables of `plot = FALSE`.

Publication figures. Give the units of the variables in their names (such as `"Ca (%)"`, with `check.names = FALSE` in `data.frame()`) or in `ylab`. Make the plot at its printed size, with `base_size` set to the text size of the journal (often 7 to 9 points), and save it with `ggplot2::ggsave()`, for example `ggsave("figure.pdf", p, width = 85, height = 70, units = "mm")` for a single column. The caption says which boxplot is drawn, because readers assume Tukey's whiskers (1.5 IQR) otherwise; use `caption = FALSE` and state it in the figure legend instead, for example: "Adjusted boxplots (Hubert and Vandervieren, 2008): the boxes show the quartiles and the median, the whiskers extend to the most extreme values within the medcouple-adjusted fences, and the points are the values outside."

Value

- If `plot = TRUE`, a `ggplot2` object.
- If `plot = FALSE`, a list of tibbles: `stats`, with one row per variable (and group): the numbers of values `n` and of missing values `n_missing`, the whisker ends `lower` and `upper` (the most extreme values within the fences), the quartiles `q1` and `q3`, the `median`, the fences, the notch limits, the `medcouple`, the 95% confidence interval of the median (`median_lower`, `median_upper`), the `iqr`, the robust coefficient of variation `rcv` (%), the `biweight` location, the `mean`, and the numbers of outlying values `n_outliers` and of expected ones in clean data `expected_outliers`; `outliers`, with the outlying values, their `row` in `x`, their `id` and their tail `out` (`"lower"` or `"upper"`); and, with `group`, `tests`, the Kruskal-Wallis test between the groups of each variable (`statistic`, `df`, `p_value`).

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
- McGill, R., Tukey, J. W., Larsen, W. A. (1978). Variations of box plots. The American Statistician, 32(1):12-16 (notches)

See Also

`generalized_boxplot()` for skewed and heavy-tailed distributions.

Examples

```
data(forageLIBS)
minerals <- forageLIBS[c("Measurement", "Ca", "Mg", "P", "K", "S")]

# mineral contents, each on its own axis and with its own fill color, the
# outlying samples named: a figure of a journal page width
p <- adjusted_boxplot(minerals, id = Measurement, label_outliers = TRUE,
                      fill = hcl.colors(5, "Pastel 1"), ylab = "Content (%)",
                      base_size = 8)
p
# saved at that size (here in the temporary directory)
ggplot2::ggsave(file.path(tempdir(), "minerals.pdf"), p, width = 175, height = 70,
                units = "mm")

# the statistics and the outlying samples
res <- adjusted_boxplot(minerals, id = Measurement, plot = FALSE)
res$stats
res$outliers

# potassium and phosphorus by calcium level, with notches, means and a
# darker blue for a higher level: a figure of a journal column (85 x 75 mm)
minerals$Ca_level <- cut(minerals$Ca, quantile(minerals$Ca, 0:3 / 3),
                         labels = c("Low Ca", "Mid Ca", "High Ca"),
                         include.lowest = TRUE)
p <- adjusted_boxplot(minerals[c("K", "P", "Ca_level")], group = Ca_level,
                      notch = TRUE, show_mean = TRUE, ylab = "Content (%)",
                      fill = c("#deebf7", "#9ecae1", "#4292c6"),
                      title = "Potassium and phosphorus by calcium level",
                      base_size = 8)
p

# trace elements on a logarithmic axis, horizontal, with all the samples
# and one fill color for all the boxes
traces <- forageLIBS[c("Fe", "Mn", "Zn")]
adjusted_boxplot(traces, log = TRUE, horizontal = TRUE, points = "all",
                 fill = "#fdd49e", ylab = "Content (mg/kg)")
```

`aki_uncertainty` *Uncertainty of Transition Probabilities from NIST Accuracy Grades*

Description

Converts the accuracy grades of the transition probabilities of the NIST Atomic Spectra Database into relative standard uncertainties.

Usage

```
aki_uncertainty(accuracy)
```

Arguments

`accuracy`	A character vector of accuracy grades, such as the column `accuracy` returned by `nist_lines()`.

Details

The grades bound the uncertainty of the transition probability (and of the oscillator strength and line strength) at the level of one standard deviation: AAA $\leq$ 0.3%, AA $\leq$ 1%, A+ $\leq$ 2%, A $\leq$ 3%, B+ $\leq$ 7%, B $\leq$ 10%, C+ $\leq$ 18%, C $\leq$ 25%, D+ $\leq$ 40%, D $\leq$ 50%, and E > 50%. The bound of each grade is returned, so the uncertainties are upper bounds. Grade E, which has no bound, and unknown grades give `NA`: such lines are best left out of plasma diagnostics.

Value

A numeric vector of relative standard uncertainties (0.25 for 25%).

References

- 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.

See Also

`nist_lines()`, `plasma_uncertainty()`

Examples

```
aki_uncertainty(c("AA", "B+", "C", "D", "E"))
```

`applicability_domain` *Applicability Domain of a Calibration*

Description

Describes the space of the calibration spectra by a principal component analysis (PCA), so that the spectra of new samples can be flagged when they lie outside it: their predictions, by any model calibrated on these spectra, are then extrapolations.

Usage

```
applicability_domain(
  x,
  ncomp = NULL,
  var_explained = 0.99,
  conf_level = 0.99,
  center = TRUE,
  scale = FALSE
)
```

Arguments

`x`	The calibration spectra: a numeric matrix or data frame, one spectrum per row.
`ncomp`	The number of principal components. By default, chosen from `var_explained`.
`var_explained`	The fraction of the variance explained by the components, when `ncomp` is not given. Default is 0.99.
`conf_level`	The confidence level of the limits. Default is 0.99.
`center`, `scale`	Passed to `stats::prcomp()`. Default is `TRUE` and `FALSE`.

Details

A new spectrum is outside the applicability domain when its Hotelling T^2, the distance within the plane of the first `ncomp` components, or its Q residual, the squared distance to that plane, exceeds its limit at `conf_level` (see `q_residuals()`). A high T^2 is a sample more extreme than the calibration samples (for example, a higher concentration); a high Q, a sample of another kind (another matrix, a contamination, an instrumental problem). The limit of T^2 is that of a new observation, and that of Q the Jackson-Mudholkar limit.

By default, `ncomp` is the smallest number of components that explain the fraction `var_explained` of the variance of the calibration spectra. Prepare the spectra as for the model (baseline correction, normalization), since the domain is that of the variables the model uses. The PCA is classical: remove the outlying calibration spectra first (see `robpca()`).

Value

An object of class `specproc_domain`, a list with the `pca` (`stats::prcomp()` fit), the number of components `ncomp`, the fraction of the variance they `explained`, the `conf_level` and the `limits` of T^2 and Q. predict() flags new spectra.

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.
- Nomikos, P., MacGregor, J.F. (1995). Multivariate SPC charts for monitoring batch processes. Technometrics, 37(1):41-59.

See Also

`predict.specproc_domain()`, `q_residuals()`, `plot_influence()`

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- as.matrix(forageLIBS[which(wl > 380 & wl < 430)])
domain <- applicability_domain(spectra[1:300, ])
domain
flags <- predict(domain, spectra[301:368, ])
table(flags$inside)
```

---

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, .group_by = deprecated())
```

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.
.group_by	**[Deprecated]** Use group_by.

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,
  max.iter = deprecated()
)
```

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.
`max.iter`	**[Deprecated]** Use `max_iter`.

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,
  max.iter = deprecated()
)
```

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.
`max.iter`	**[Deprecated]** Use `max_iter`.

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"),
  max.iter = deprecated()
)
```

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.
`max.iter`	**[Deprecated]** Use `max_iter`.

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)
```

`bin_spectra` *Binning of Adjacent Channels*

Description

Reduces the number of channels of spectra by replacing each group of `width` adjacent channels by their mean.

Usage

```
bin_spectra(x, width = 3, 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.
`width`	The number of adjacent channels averaged, a positive integer. Default is 3; `width = 1` returns the spectra unchanged.
`segments`	A logical: bin the segments between gaps of the wavelength axis separately (`TRUE`, default). Needs wavelengths as names.

Details

Averaging keeps the intensities on their scale and divides the noise of independent channels by about `sqrt(width)`, at the cost of spectral resolution: keep `width` smaller than the width of the emission lines (a few channels in LIBS spectra) to keep them resolved. Fewer channels also make the models faster to fit.

The channels are grouped from the first one; when their number is not a multiple of `width`, the last group is smaller. 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 detector is binned on its own, so that no group straddles two detectors. The binned channels are named by their mean wavelength (to 10 significant digits), or, when the names are not wavelengths, by the name of their first channel; a channel left alone in its group keeps its name.

Value

The binned spectra, of the same type as `x` (a tibble for a data frame), with `ceiling(n / width)` channels for each detector segment of `n` channels.

Author(s)

Christian L. Goueguel

References

- Vrábel, J., Képeš, E., Duponchel, L., et al. (2020). Classification of challenging laser-induced breakdown spectroscopy soil sample data - EMSLIBS contest. Spectrochimica Acta Part B, 169:105872.

See Also

step_bin_spectra(), gap_derivative(), average()

Examples

```
data(forageLIBS)
spectra <- forageLIBS[1:3, -(1:14)]
binned <- bin_spectra(spectra, width = 3)
ncol(spectra)
ncol(binned)
names(binned)[1:3]
```

---

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,
  drop.na = deprecated()
)
```

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.

drop.na — **[Deprecated]** Use drop_na.

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, drop.na = deprecated())
```

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.
drop.na	**[Deprecated]** Use drop_na.

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,
  drop.na = deprecated()
)
```

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 ($|u_i| < 1$). 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.
drop.na	**[Deprecated]** Use drop_na.

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, $T = -1/(k_B \,\text{slope})$, and its standard error from that of the slope.

The intensities must be corrected for the spectral response of the instrument (see `response_calibration()`), 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`;
- `lines`: the lines of the plot.

Use `plot_boltzmann()` to draw the plot, and `plasma_uncertainty()` to propagate the uncertainties of the intensities and of the atomic data to the temperature.

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,
  regression = "ols",
  concentration_se = NULL,
  signal_se = NULL,
  variance_ratio = NULL,
  lod_convention = "ich",
  alpha = 0.05,
  beta = 0.05,
  replicates = 1
)
```

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.
`regression`	`"ols"` (default, least squares), `"deming"` or `"york"` (see details).
`concentration_se`, `signal_se`	For `regression = "york"` or `"deming"`: the columns of `data` with the standard uncertainties of the concentration and of the signal of each standard, unquoted or as strings.
`variance_ratio`	For `regression = "deming"`: the ratio of the error variance of the signal to that of the concentration. By default, it is computed from `signal_se` and `concentration_se`.
`lod_convention`	The convention of the limits of detection and quantification: `"ich"` (default, $3.3\,\sigma/S$ and $10\,\sigma/S$) or `"iso11843"` (ISO 11843-2 and IUPAC; see details).
`alpha`, `beta`	For `lod_convention = "iso11843"`: the probabilities of false positives and of false negatives. Default is 0.05.
`replicates`	For `lod_convention = "iso11843"`: the number of replicate measurements averaged for a sample. Default is 1.

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"`.

With `lod_convention = "iso11843"`, the limits follow ISO 11843-2 and IUPAC (Currie, 1999), and account for the uncertainty of the curve and for the false positive and false negative risks α and β. The standard deviation of a net signal near zero is $\sigma_0 = \sqrt{\sigma^2/K + s_a^2}$, where σ is the standard deviation of the signal (`"residual"` or `"blank"`), K the number of `replicates` averaged for a sample and s_a the standard error of the intercept. The critical value (decision limit), above which an element is detected, is $x_C = t_{1-\alpha,\nu}\,\sigma_0/S$; the limit of detection, the content detected with probability $1-\beta$, is $x_D = \delta_{\nu,\alpha,\beta}\,\sigma_0/S$, where δ is the noncentrality parameter of the noncentral t distribution (about $2t_{1-\alpha,\nu}$ for $\alpha = \beta$) and ν the degrees of freedom of σ; and the limit of quantification (IUPAC) is $x_Q = 10\,\sigma_0/S$. For a weighted curve, the standard deviation must come from `blank` signals.

Errors in the concentrations. Least squares assume exact concentrations. When those of the standards are uncertain, such as the certified values of reference materials, the slope is biased toward zero and the intervals too narrow. `regression = "york"` fits the line by the method of York et al. (2004), with the standard uncertainties of the concentration (`concentration_se`) and of the signal (`signal_se`) of each standard; its standard errors are multiplied by the square root of the reduced chi-squared when it exceeds 1 (the points scatter more than their uncertainties explain). `regression = "deming"` (Deming regression) assumes constant uncertainties, whose ratio `variance_ratio` (variance of the signal over variance of the concentration, in their units) is given, or computed from the mean squared `signal_se` and `concentration_se`; its standard errors come from the scatter of the points. Both fit straight lines without weights; the tests of linearity are those of the least-squares fit.

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`), the `critical_value` (ISO 11843 only, `NA` otherwise), `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 least-squares `lm` fit, `eiv`, the Deming or York fit (`NULL` for least squares), with its `chi_squared`, and `data`, the calibration data.

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

Author(s)

Christian L. Goueguel

References

- Currie, L.A. (1999). Detection and quantification limits: origins and historical overview. Analytica Chimica Acta, 391(2):127-134.
- ICH (2005). Validation of analytical procedures: text and methodology Q2(R1). International Conference on Harmonisation.
- ISO 11843-2 (2000). Capability of detection, Part 2: Methodology in the linear calibration case. International Organization for Standardization, Geneva.
- 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.
- York, D., Evensen, N.M., López Martínez, M., De Basabe Delgado, J. (2004). Unified equations for the slope, intercept, and standard errors of the best straight line. American Journal of Physics, 72(3):367-375.

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]))

# limits of detection by ISO 11843-2
calibration_curve(standards[1:300, ], signal, K, lod_convention = "iso11843")

# reference materials with certified values known within 3%, and
# signals within 2% (simulated)
set.seed(1)
crm <- data.frame(certified = c(0.5, 1, 2, 4, 6, 8))
crm$signal <- 0.05 + 0.2 * crm$certified * (1 + rnorm(6, sd = 0.02)) +
  0.2 * rnorm(6, sd = 0.03 * crm$certified)
crm$certified_se <- 0.03 * crm$certified
crm$signal_se <- 0.02 * crm$signal
calibration_curve(crm, signal, certified, regression = "york",
                  concentration_se = certified_se, signal_se = signal_se)
```

`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,
  ncomp = NULL,
  alpha = 0.5,
  ncomp_max = 10,
  var_explained = 0.8,
  scale = FALSE,
  ndir = 250,
  maxiter = 1000,
  tol = 1e-06,
  max_col_frac = 0.5,
```

```
  od_cutoff = c("simulated", "chisq"),
  k = deprecated(),
  kmax = deprecated()
)
```

Arguments

x	A numeric matrix or data frame, with one observation per row. Missing values are allowed.
ncomp	The number of principal components. If NULL (default), it is chosen from var_explained and ncomp_max.
alpha	The robustness parameter, between 0.5 and 1: the fraction of observations used in the fit. Default is 0.5.
ncomp_max	The maximum number of components. Default is 10.
var_explained	The fraction of variance used to choose ncomp 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.
k	**[Deprecated]** Use ncomp.
kmax	**[Deprecated]** Use ncomp_max.

Details

cellPCA approximates the data by a fit $\hat{X} = 1_n \mu^T + UV^T$ of rank k (ncomp) 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`, `ncomp_max`, `var_explained`, `scale` and `ndir`), which also chooses `ncomp` 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 `ncomp = 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, ncomp = 2, od_cutoff = "chisq")
fit
plot_outlier_map(fit, shade = TRUE, relative = TRUE, log = TRUE, color_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, drop.na = deprecated())
```

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.
`drop.na`	**[Deprecated]** Use `drop_na`.

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 (see `response_calibration()`), 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;
- `lines`, the lines of the fit, and `settings`, its other arguments.

Use `plot_boltzmann()` to draw the Boltzmann plots, and `plasma_uncertainty()` to propagate the uncertainties of the intensities, of the atomic data and of the electron density to the temperature and the composition.

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,
  ncomp = NULL,
  statistic = c("q", "t2"),
  data = NULL,
  samples = NULL,
  reference = NULL,
  k = deprecated()
)
```

Arguments

model	A stats::prcomp() fit, or an object returned by robpca(), rospca() or macropca().
ncomp	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.
k	**[Deprecated]** Use ncomp.

Details

For a sample x (centered and scaled as in the model), with scores $t = xP$ on the first k components (ncomp; 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, ncomp = 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]
```

---

control_chart *Control Chart for Instrument Drift*

Description

Builds a control chart of a quantity that should stay constant over time, such as the intensity of an internal standard line, a line ratio of a check sample or a plasma temperature, to detect drifts and shifts of the instrument: a Shewhart chart of individual values, an exponentially weighted moving average (EWMA) or a cumulative sum (CUSUM).

Usage

```
control_chart(
  x,
  order = NULL,
  method = "shewhart",
  baseline = NULL,
  center = NULL,
  sigma = NULL,
  width = 3,
  rules = TRUE,
  lambda = 0.2,
  allowance = 0.5,
  decision = 5
)
```

Arguments

x	A numeric vector of the monitored quantity.
order	The time or acquisition order of the values (numbers, dates or date-times), by which they are sorted. By default, their position.
method	"shewhart" (default), "ewma" or "cusum".
baseline	The measurements that set the center and the standard deviation: the number of first values, or a logical or index vector. By default, all values.
center, sigma	The center and standard deviation of the process, if known.
width	The width of the limits, in standard deviations, for the Shewhart and EWMA charts. Default is 3.
rules	A logical: apply the Western Electric rules to the Shewhart chart (TRUE, default).
lambda	The smoothing constant of the EWMA chart. Default is 0.2.
allowance, decision	The allowance k and decision interval h of the CUSUM chart, in standard deviations. Default is 0.5 and 5.

Details

The center line and the standard deviation σ are estimated from the baseline measurements (by default, all of them), unless given: the mean, and the mean moving range of consecutive values divided by 1.128, which, unlike the standard deviation, is little inflated by a drift.

- "shewhart": the values themselves, with limits at width σ from the center. With rules = TRUE, the Western Electric rules also flag (1) a value beyond the limits, (2) two of three consecutive values beyond 2σ on the same side, (3) four of five beyond 1σ on the same side and (4) eight consecutive values on the same side of the center. It detects large shifts quickly.
- "ewma": $z_t = \lambda x_t + (1-\lambda) z_{t-1}$, starting at the center, with limits at width $\sigma \sqrt{\lambda/(2-\lambda)(1-(1-\lambda)^{2t})}$. Small values of λ detect small, slow drifts.
- "cusum": the tabular CUSUM, the sums of the deviations beyond allowance σ above and below the center, $C_t^+ = \max(0, x_t - \mu - k\sigma + C_{t-1}^+)$ and likewise below, with a signal when a sum exceeds decision σ. It detects small persistent shifts.

Value

A tibble of class `specproc_control_chart`, one row per value in time order: `order`, `value`, the plotted `statistic` (the value, the EWMA, or for the CUSUM the upper sum, with the lower sum in `lower_sum`), the `center`, the `lower` and `upper` limits, whether the point is a `signal`, and, for the Shewhart chart, the first Western Electric `rule` broken. The method, center, standard deviation and baseline are in the attributes. Draw it with `plot_control_chart()`.

Author(s)

Christian L. Goueguel

References

- Montgomery, D.C. (2019). Introduction to Statistical Quality Control, 8th ed. Wiley, Hoboken.
- Western Electric Company (1956). Statistical Quality Control Handbook. Western Electric, Indianapolis.

See Also

`plot_control_chart()`, `line_intensities()`

Examples

```
# a check-sample ratio that drifts after the 60th measurement
set.seed(1)
ratio <- 1 + c(rep(0, 60), seq(0, 0.06, length.out = 40)) + rnorm(100, sd = 0.02)
shewhart <- control_chart(ratio, baseline = 40)
shewhart[shewhart$signal, ]
plot_control_chart(shewhart)

# the EWMA detects the slow drift earlier
plot_control_chart(control_chart(ratio, method = "ewma", baseline = 40))
```

---

correct_response *Correct Spectra or Line Intensities for the Spectral Response*

Description

Divides spectra, or the intensities of emission lines, by the relative spectral response of the instrument measured by `response_calibration()`.

Usage

```
correct_response(x, response)
```

Arguments

x	Spectra (a numeric vector named by wavelength, or a matrix or data frame with one spectrum per row and the wavelengths as column names), or a table of lines with `wavelength` and `intensity` columns.
response	The result of `response_calibration()`.

Details

Spectra must have the wavelength axis of the response (the same channels, in the same order): each channel is divided by its response.

A table of lines, such as returned by `line_intensities()`, is recognized by its `wavelength` and `intensity` columns. The `intensity` (and `height`) of each line is divided by the response at the channel of its peak (`peak_wavelength`, or else the channel nearest to its `wavelength`), and its standard error `intensity_se` (created if absent) combines in quadrature its relative standard error and that of the response. Correcting the line intensities rather than the spectra keeps the noise estimates of `line_intensities()` valid. The errors of the response of nearby lines are correlated, which `plasma_uncertainty()` does not account for.

Correct the intensities for the response before `correct_self_absorption()`, `boltzmann()` or `cf_libs()`.

Value

`x`, corrected. For a table of lines, with the corrected `intensity` (and `height`), its standard error `intensity_se`, and the `response` used. Channels or lines without response become `NA`.

See Also

`response_calibration()`, `line_intensities()`

Examples

```
planck <- function(wl, t) 1 / (wl^5 * (exp(1.4388e7 / (wl * t)) - 1))
reference <- data.frame(wavelength = seq(250, 850, by = 10))
reference$value <- planck(reference$wavelength, 3100)
reference$rsd <- 0.03
wl <- seq(260, 840, by = 0.1)
true_response <- exp(-((wl - 500) / 250)^2)
set.seed(1)
lamp <- 2e4 * true_response * planck(wl, 3100) / planck(500, 3100) + rnorm(length(wl), sd = 20)
names(lamp) <- wl
response <- response_calibration(lamp, reference)

# a spectrum with two lines of equal emission, at 300 and 650 nm
emission <- 1000 * exp(-(wl - 300)^2 / 0.01) + 1000 * exp(-(wl - 650)^2 / 0.01)
spectrum <- stats::setNames(true_response * emission + rnorm(length(wl)), wl)
lines <- line_intensities(spectrum, c(a = 300, b = 650))
lines$intensity   # unequal: the response differs
corrected <- correct_response(lines, response)
corrected[c("line", "intensity", "intensity_se", "response")]
```

```
# or the whole spectrum
corrected_spectrum <- correct_response(spectrum, response)
```

`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`. A column `intensity_se` (as returned by `line_intensities()`) becomes the standard error of the corrected intensity, which is computed from the reference line alone: its relative standard error is that of the reference line.

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
}
```

`curve_of_growth` *Self-Absorption by the Curve of Growth*

Description

Fits the curve of growth of a homogeneous plasma in local thermodynamic equilibrium (LTE) to the intensities of emission lines, and returns the optical depth and the self-absorption coefficient of each line, with the intensity it would have if the plasma were optically thin.

Usage

```
curve_of_growth(lines, temperature, units = "energy")
```

Arguments

`lines`	A data frame with one row per line and columns `species` (such as `"K I"`), `intensity` (integrated intensity, corrected for the spectral response), `wavelength` (nm), `Aki` (s^{-1}), gk, `Ek` and `Ei` (upper and lower level energies, eV) and `stark_width` (the Stark full width at half maximum, nm).
`temperature`	The plasma temperature, in K.
`units`	The units of `intensity`: `"energy"` (default) or `"photons"`, as in `boltzmann()`.

Details

In a homogeneous plasma of length l in LTE, the integrated intensity of a line is

$$I = F\, B_\lambda(T) \int \left(1 - e^{-\tau_0 V(\lambda)/V(\lambda_0)}\right) d\lambda$$

where $B_\lambda(T)$ is the Planck function, F a factor of the instrument and of the geometry, common to all lines, V the line profile and τ_0 the optical depth at the line center,

$$\tau_0 = S\, \lambda^4 g_k A_{ki}\, e^{-E_i/k_B T} \left(1 - e^{-hc/\lambda k_B T}\right) V(\lambda_0)$$

The column density S, proportional to the number density of the species times l over its partition function, is the same for all lines of a species. When the plasma is optically thin, the intensity grows as τ_0 (it is the intensity of a Boltzmann plot); as τ_0 grows, the line center saturates and the intensity grows only as $\sqrt{\tau_0}$ (Lorentzian wings) or slower. This is the curve of growth of Gornushkin et al. (1999), and the model of the C-sigma graphs of Aragón and Aguilera (2014), with one column density per species, since their concentrations are unknown.

The profile is a Voigt profile of the Stark (Lorentzian) width of each line, `stark_width`, and of its Doppler (Gaussian) width at the temperature (see `doppler_width()`). The instrumental broadening, which does not change the optical depth, must not be included: take the Lorentzian width

of a Voigt fit with the instrumental width held fixed (see `peak_fit()`), or the Stark width at the electron density of the plasma (see `stark_width()`). Self-absorption broadens lines, so the widths of strongly absorbed lines should come from the Stark parameters.

The factor F and the column densities are fitted by least squares on the logarithms of the intensities (weighted by their relative standard errors when the lines have a column `intensity_se`). The optical depths are determined by the curvature of the intensities: lines of different optical depths, such as resonance and non-resonance lines, are needed, and a species whose lines are all thin keeps a column density near the lower bound, with coefficients close to 1. The self-absorption coefficient is $SA = I/I_{thin}$, the ratio of the intensity to that of an optically thin plasma (as in `self_absorption()`), and the corrected intensities, I/SA, go to `boltzmann()` or `cf_libs()`.

Value

`lines`, as a tibble of class `specproc_curve_of_growth`, with the `measured_intensity`, the Doppler width (`doppler_width`, nm), the `optical_depth` at the line center, the self-absorption coefficient `SA`, the corrected `intensity` (I/SA) and the `residual` of the fit (on the logarithm of the intensity). The attributes hold the `temperature`, the `units`, the fitted `scale` (F), the `column_density` of each species (a tibble), and the root mean square residual `rmse`. Draw it with `plot_curve_of_growth()`.

Author(s)

Christian L. Goueguel

References

- Aragón, C., Aguilera, J.A. (2014). CSigma graphs: a new approach for plasma characterization in laser-induced breakdown spectroscopy. Journal of Quantitative Spectroscopy and Radiative Transfer, 149:90-102.
- Gornushkin, I.B., Anzano, J.M., King, L.A., Smith, B.W., Omenetto, N., Winefordner, J.D. (1999). Curve of growth methodology applied to laser-induced plasma emission spectroscopy. Spectrochimica Acta Part B, 54(3-4):491-503.

See Also

`plot_curve_of_growth()`, `correct_self_absorption()`, `self_absorption()`, `stark_width()`, `cf_libs()`

Examples

```
# simulated lines of a species: two resonance lines (Ei = 0), strongly
# absorbed, and four lines from excited levels
lines <- data.frame(
  species = "Ca I", wavelength = c(400, 405, 520, 560, 610, 650),
  Aki = c(1.5e8, 1.2e8, 3e7, 5e7, 4e7, 2e7), gk = c(4, 2, 6, 4, 3, 5),
  Ei = c(0, 0, 2.0, 2.1, 2.2, 2.4), stark_width = 0.02
)
lines$Ek <- lines$Ei + 1239.842 / lines$wavelength
# their intensities in a plasma where the strongest line has an optical
# depth of 5 at its center
truth <- curve_of_growth_model(lines, temperature = 9000, scale = 1e20, optical_depth = 5)
```

```
truth
lines$intensity <- truth$intensity

cog <- curve_of_growth(lines, temperature = 9000)
cog[c("wavelength", "optical_depth", "SA")]
plot_curve_of_growth(cog)
# the corrected intensities lie on the Boltzmann plot
boltzmann(cog)
```

`curve_of_growth_model` *Intensities of Emission Lines on a Curve of Growth*

Description

Computes the integrated intensities and optical depths of emission lines of a homogeneous plasma in LTE, given the column density of their species, by the model of `curve_of_growth()`, for simulations and for checking a fit.

Usage

```
curve_of_growth_model(
  lines,
  temperature,
  scale,
  optical_depth,
  units = "energy"
)
```

Arguments

`lines`	A data frame with one row per line and columns `species` (such as `"K I"`), `intensity` (integrated intensity, corrected for the spectral response), `wavelength` (nm), `Aki` (s^{-1}), `gk`, `Ek` and `Ei` (upper and lower level energies, eV) and `stark_width` (the Stark full width at half maximum, nm).
`temperature`	The plasma temperature, in K.
`scale`	The factor F of the instrument and geometry.
`optical_depth`	The largest optical depth at the line center of the lines of each species, which sets its column density: one value, or one per species named by species.
`units`	The units of `intensity`: `"energy"` (default) or `"photons"`, as in `boltzmann()`.

Value

A tibble with, for each line, its `wavelength`, `optical_depth`, self-absorption coefficient `SA`, `intensity` and the intensity of an optically thin plasma (`thin_intensity`).

See Also

`curve_of_growth()`

Examples

```
lines <- data.frame(species = "Ca I", wavelength = c(400, 520), Aki = c(1.5e8, 3e7),
                    gk = c(4, 6), Ei = c(0, 2), stark_width = 0.02)
lines$Ek <- lines$Ei + 1239.842 / lines$wavelength
curve_of_growth_model(lines, temperature = 9000, scale = 1, optical_depth = 5)
```

---

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,
  rm_zeroes = FALSE,
  max_ratio = NULL,
  prec_scale = 1e-10,
  cutoff.quantile = deprecated(),
  rmZeroes = deprecated(),
  maxRatio = deprecated(),
  precScale = deprecated()
)
```

Arguments

`x`	A numeric vector
`cutoff_quantile`	A numeric value in (0, 1): the proportion of the observations of a normal distribution that are not flagged. The cutoff of the scores is the `(1 + cutoff_quantile) / 2` quantile of the standard normal distribution. Default is 0.995 (a cutoff of 2.81).
`rm_zeroes`	A logical value. If `TRUE`, the zero distances (the observations equal to the median, within `prec_scale`) are left out of the half samples before their scales are computed, which matters when many observations are tied at the median. Default is `FALSE`, as in the paper.
`max_ratio`	`NULL` (default), or a number of at least 2: each scale is then kept between the scale of all the distances to the median divided and multiplied by `max_ratio`, so that a very short or very long half cannot give extreme scores. Not in the paper.
`prec_scale`	A non-negative number: an initial scale below it is taken as zero (default: 1e-10).
`cutoff.quantile`, `rmZeroes`, `maxRatio`, `precScale`	**[Deprecated]** Use `cutoff_quantile`, `rm_zeroes`, `max_ratio` and `prec_scale`.

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, each with $h = \lfloor (n+1)/2 \rfloor$ observations (they share the median when n is odd), and scaling each half separately (Rousseeuw *et al.*, 2018, Section 2.1). For the distances z_i of a half sample to the median, the scale is the one-step M-estimator

$$s = s_0 \sqrt{\frac{1}{2\alpha h} \sum_i \rho_c \left(\frac{z_i}{s_0} \right)},$$

from the initial scale $s_0 = \mathrm{med}_i(z_i)/\Phi^{-1}(0.75)$, with Huber's ρ-function for scale, $\rho_c(t) = \min\{(t/c)^2, 1\}$, $c = 2.1$ (85% efficiency at the normal distribution), and $\alpha = \int_0^\infty \rho_c(x)\, d\Phi(x)$, which makes the scale consistent at the normal distribution.

The score of an observation is its distance to the median divided by the scale of its own half: on the short side of a skewed distribution, an observation gets a higher score than one as far from the median on the long side. For a normal distribution, both scales estimate the standard deviation, and the score is close to the absolute robust z-score of `zscore()`.

An observation is flagged when its score exceeds the $(1+q)/2$ quantile of the standard normal distribution, where q is `cutoff_quantile` (2.81 for the default 0.995): about $1-q$ of the observations of a normal distribution (0.5%) are flagged.

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.

See Also

`zscore()`, `iqr_outliers()`

Examples

```
# 200 right-skewed values (a gamma distribution shifted to start at 100),
# 3 low outliers below them and 3 high outliers
set.seed(1)
x <- c(100 + rgamma(200, shape = 2, scale = 10), 85, 87, 89, 220, 230, 240)
group <- rep(c("regular", "outlier"), c(200, 6))
```

```
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))

# the robust z-score scales both sides alike: regular values of the long
# right tail are flagged, but not the low outliers (the last three rows)
rob <- zscore(x, robust = TRUE)
subset(rob, flag)
tail(rob, 3)
# directional outlyingness scales each side separately: the low outliers
# score above the regular values of the long tail, and the 6 outliers are
# flagged
head(directional_outlyingness(x), 8)

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))
```

`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 $\mathbf{X}_{DO} = \mathbf{X} - \mathbf{X}\mathbf{P}\mathbf{P}^T$. Direct Orthogonalization is typically simpler and faster than OSC.

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

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, **X**, that is orthogonal to the response variable(s), **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 **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. **Y** is projected onto the column space of **X**: $\hat{\mathbf{Y}} = \mathbf{X}\mathbf{X}^{+}\mathbf{Y}$.
2. **X** is orthogonalized with respect to $\hat{\mathbf{Y}}$: $\mathbf{Z} = \mathbf{X} - \hat{\mathbf{Y}}\hat{\mathbf{Y}}^{+}\mathbf{X}$.
3. PCA of **Z** gives the orthogonal scores **T**.
4. The weights $\mathbf{W} = \mathbf{X}^{+}\mathbf{T}$ express the scores as a linear combination of **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 **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)
```

---

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,
  ncomp,
  newdata = NULL,
  conf_level = 0.975,
  normalized = TRUE,
  df = "effective",
```

```
  t2_method = "f",
  center = TRUE,
  scale = FALSE,
  k = deprecated()
)
```

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.
ncomp	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.
k	**[Deprecated]** Use ncomp.

Details

With the residuals e_{ij} of the K variables after k components (ncomp), 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, ncomp = 3)
  d[d$outlier != "regular", ]
  plot_influence(d, label = forageLIBS$Measurement)
}
```

---

doppler_width *Doppler Width of an Emission Line*

Description

Computes the full width at half maximum of the Doppler (thermal) broadening of a line emitted by atoms or ions of a given mass at a given temperature.

Usage

```
doppler_width(wavelength, temperature, mass)
```

Arguments

wavelength	The wavelength of the line, in nm.
temperature	The temperature of the emitters, in K.
mass	The mass of the emitter: its atomic mass, in u, or its element symbol (such as "Ca"), for the standard atomic weight.

Details

The thermal motion of the emitters gives a Gaussian profile of full width at half maximum

$$w_D = \lambda \sqrt{\frac{8 k_B T \ln 2}{m c^2}}$$

where λ is the wavelength, T the temperature and m the mass of the emitter. In LIBS plasmas, it is usually much smaller than the instrumental width, except for light elements such as hydrogen. Both are Gaussian and add in quadrature, $w_G = \sqrt{w_{inst}^2 + w_D^2}$, the Gaussian width to hold fixed in a Voigt fit (see peak_fit()).

Value

A numeric vector of Doppler widths (FWHM), in nm. The arguments are recycled to a common length.

Author(s)

Christian L. Goueguel

See Also

peak_fit(), voigt_fwhm()

Examples

```
# Ca II 393.37 nm and H-alpha at 10000 K
doppler_width(c(393.37, 656.28), temperature = 10000, mass = c("Ca", "H"))

# with an instrumental width of 0.05 nm: the Gaussian width of a Voigt fit
sqrt(0.05^2 + doppler_width(393.37, 10000, "Ca")^2)
```

---

download_cache *Persistent Cache of Downloaded Data*

Description

Lists or clears the data downloaded from the NIST Atomic Spectra Database, STARK-B and ExoMol and kept on disk, so that they are reused in later sessions and without an internet connection.

Usage

```
download_cache()

clear_download_cache(source = c("nist", "starkb", "exomol"), older_than = NULL)
```

Arguments

source	The sources to clear: "nist", "starkb" and/or "exomol". Default is all of them.
older_than	Clear only the entries older than this number of days, or NULL (default) for all of them.

Details

Downloaded data are always kept for the R session. To keep them between sessions, set the option specProc.cache, for example in your .Rprofile:

- options(specProc.cache = TRUE) keeps them in the user cache directory of specProc, given by tools::R_user_dir();
- options(specProc.cache = "path/to/dir") keeps them in that directory;
- NULL or FALSE (default) keeps them for the session only.

The results of nist_lines(), nist_ionization_energy(), nist_levels() (and so libs_lines() and line_finder()), starkb_lines() and molecular_spectrum() are then read from the cache when they are there. Entries older than the option specProc.cache_max_age (in days, default 180) are downloaded again, as the databases are updated; when the download fails, the older copy is used with a warning. The NIST and STARK-B entries are small (a few kilobytes to a megabyte); ExoMol line lists can be large, see exomol_datasets().

The cache can also be cleared by deleting its directory.

Value

download_cache(): a tibble with one row per file of the persistent cache, with its source, file name, size (kB), modified time and age (days); its directory is the attribute dir (NULL when the cache is not enabled). clear_download_cache(): invisibly, the paths of the deleted files. It also clears the session cache of the sources.

Author(s)

Christian L. Goueguel

See Also

nist_lines(), starkb_lines(), molecular_spectrum()

Examples

```
# keep the downloaded data in a directory (here a temporary one)
old <- options(specProc.cache = file.path(tempdir(), "specProc-cache"))
download_cache()
clear_download_cache()
options(old)
```

`ds` *Direct Standardization*

Description

Transfers spectra from a secondary instrument to the primary instrument, on which the calibration model was built, by direct standardization (DS; Wang et al., 1991): a linear map of the whole secondary spectrum onto the whole primary spectrum, estimated on transfer samples measured on both instruments.

Usage

```
ds(x1, x2, ncomp = NULL, tol = 1e-08)
```

Arguments

`x1`	The spectra of the transfer samples on the primary instrument: a numeric matrix or data frame, one spectrum per row.
`x2`	The spectra of the same samples, in the same order, on the secondary instrument.
`ncomp`	The number of singular vectors of the secondary spectra kept. By default, all those with a singular value above `tol` times the largest.
`tol`	The relative tolerance on the singular values. Default is 1e-8.

Details

With the transfer spectra centered by the mean of each instrument, $\mathbf{X}_1 \approx \mathbf{X}_2\mathbf{F}$, and the transfer matrix is $\mathbf{F} = \mathbf{X}_2^{+}\mathbf{X}_1$, where $\mathbf{X}_2^{+}$ is the pseudo-inverse of the secondary spectra, truncated to their first `ncomp` singular vectors. A new secondary spectrum is transferred as $\bar{x}_1 + (x_2 - \bar{x}_2)\mathbf{F}$.

With as many components as the transfer samples allow (`ncomp = NULL`), the transfer spectra are reproduced exactly, and the noise of the secondary spectra is amplified: fewer components give a smoother, more robust transfer. Unlike `pds()`, DS mixes all wavelengths, so it corrects global differences of the response but needs more transfer samples. The transfer matrix ($p \times p$) is kept in factored form, as `ncomp` columns of each factor.

Value

An object of class `specproc_transfer`. Use predict() to transfer secondary spectra.

Author(s)

Christian L. Goueguel

References

- Wang, Y., Veltkamp, D.J., Kowalski, B.R. (1991). Multivariate instrument standardization. Analytical Chemistry, 63(23):2750-2756.

See Also

pds(), sst(), slope_bias(), predict.specproc_transfer()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- as.matrix(forageLIBS[which(wl > 760 & wl < 780)])  # the K I lines
# a second instrument, with a shifted and less sensitive response
secondary <- 0.8 * cbind(spectra[, -1], spectra[, ncol(spectra)]) + 50
transfer <- ds(spectra[1:30, ], secondary[1:30, ], ncomp = 10)
transfer
# the other samples, measured on the secondary, transferred to the primary
transferred <- predict(transfer, secondary[31:368, ])
mean(abs(transferred - spectra[31:368, ])) / mean(spectra)
```

---

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,
  species = NULL
)
```

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.
species	For method = "stark" with stark: the species of the line, such as "Ca II", required when stark has several species (see stark_width()).

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. Holding the Gaussian width at the measured instrumental width combined with the Doppler width (doppler_width()), with fixed = "wG", determines the Lorentzian width much better.
- 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() and from the VALD3 and Kurucz line lists with read_vald() and read_kurucz(). 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,

$$w = 1.098\,(N_e/10^{17})^{0.67823}\ \text{nm},$$

  which includes the ion dynamics and depends only weakly on temperature.

Value

A numeric vector of electron densities, in cm^{-3}. See `electron_density_uncertainty()` for their uncertainty.

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)
```

---

electron_density_uncertainty

Uncertainty of the Electron Density from Stark Broadening

Description

Computes the electron density from the Stark width of a line, as `electron_density()`, with its standard uncertainty and confidence interval, from the standard error of the width and the uncertainty of the Stark parameter of the line.

Usage

```
electron_density_uncertainty(
  width,
  width_se = 0,
  method = "stark",
  stark = NULL,
  wavelength = NULL,
  temperature = NULL,
  reference_width = NULL,
  reference_density = 1e+17,
  reference_rsd = 0,
  tolerance = 0.5,
```

```
  level = 0.95,
  species = NULL
)
```

Arguments

`width`	The measured Stark full width at half maximum, in nm (see Details). A vector is allowed.
`width_se`	The standard error of `width`, in nm: one value, or one per width. Default is 0.
`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}.
`reference_rsd`	The relative standard uncertainty of the reference width: of the Stark parameter of the line (`method = "stark"`), or of the width law of Hα (`method = "halpha"`). Default is 0.
`tolerance`	For `method = "stark"` with `stark`: the largest difference between `wavelength` and the tabulated wavelength, in nm (see `stark_width()`). Default is 0.5.
`level`	The confidence level of the interval. Default is 0.95.
`species`	For `method = "stark"` with `stark`: the species of the line, such as `"Ca II"`, required when `stark` has several species (see `stark_width()`).

Details

The electron density is proportional to $(w/w_{ref})^p$, where w is the measured width, w_{ref} the reference width of the line (its Stark parameter, or the width law of Hα), $p = 1$ for `method = "stark"` and $p = 1/0.67823$ for `method = "halpha"`. With lognormal errors of w and w_{ref}, the density is lognormal, with

$$\sigma_{\ln N_e} = p\sqrt{\ln(1+u_w^2) + \ln(1+u_{ref}^2)}$$

where u_w and u_{ref} are the relative standard uncertainties of the width and of the reference width. The interval is $N_e e^{\pm z\sigma_{\ln N_e}}$, and the relative standard uncertainty is $\sqrt{e^{\sigma^2_{\ln N_e}} - 1}$.

The width of Hα grows less than linearly with the density, so its relative uncertainty is amplified about 1.5 times in the density. The standard error of the width is that of the fit of the line (the `std.error` of the width in `peak_fit()`), or that of replicate measurements. The uncertainty of the reference width depends on its source: give it with `reference_rsd`. The weak dependence of the Stark width on the temperature is not propagated.

Value

A tibble with one row per width: the `width` and `width_se`, the `electron_density` (cm^{-3}), its standard uncertainty `se`, its relative standard uncertainty `rsd`, and the `lower` and `upper` bounds of the interval. Pass `se` to `plasma_uncertainty()` as `electron_density_se`.

See Also

`electron_density()`, `plasma_uncertainty()`, `peak_fit()`

Examples

```
# H-alpha width of 0.50 +/- 0.02 nm, with a 10% uncertainty of the width law
electron_density_uncertainty(0.5, width_se = 0.02, method = "halpha", reference_rsd = 0.1)

# a line with a Stark width of 0.012 nm at 1e17 cm-3, known within 20%
electron_density_uncertainty(0.024, width_se = 0.002, reference_width = 0.012,
                             reference_rsd = 0.2)
```

---

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 **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 **X** be the spectral matrix. The EPO algorithm aims to split **X** into:

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

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

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`, $\mathbf{X}\mathbf{V}\mathbf{V}^T$.
- `loadings`: The singular vectors **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)
```

`exomol_datasets` *Line Lists of a Molecule in the ExoMol Database*

Description

Lists the line lists of a molecule in the ExoMol database, for each isotopologue, with the sizes of their files, to choose the `dataset` of `molecular_spectrum()`.

Usage

```
exomol_datasets(molecule, timeout = 60)
```

Arguments

`molecule`	The molecule, as named by ExoMol, such as `"CN"`, `"C2"`, `"OH"` or `"AlO"`.
`timeout`	The download timeout, in seconds. Default is 60.

Details

The list is read from the ExoMol API, and needs the jsonlite package and an internet connection.

Value

A tibble with one row per line list: the `molecule`, the `isotopologue` (as in the file names), the `dataset`, whether ExoMol `recommended` it, and the sizes of its transition and states files (`trans_mb`, `states_mb`, in MB; `NA` when not listed).

See Also

`molecular_spectrum()`

Examples

```
# needs an internet connection; gives up after 20 s if ExoMol does not answer
if (rlang::is_installed("jsonlite")) {
  try(exomol_datasets("CN", timeout = 20))
}
```

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

```
data(forageLIBS)
# the 380-430 nm window (Ca II H and K lines), faster than all the channels
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
fit <- macropca(center(spectra), ncomp = 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)
```

---

forageShots *LIBS Laser Shots of Forage Samples*

Description

The single laser shots of 20 of the measurements of forageLIBS: 8 shots per measurement, in two wavelength windows, to study the shot-to-shot variability and the rejection of outlying shots.

Usage

```
forageShots
```

Format

A tibble with 160 rows (shots) and 844 columns:

Measurement Measurement identifier, as in forageLIBS.

Sample Sample identifier, as in forageLIBS.

shot Shot number in the measurement (1 to 8), in firing order.

Ca, K Calcium and potassium contents, in percent.

380.011254, ..., 779.977295 Emission intensity (counts) at each wavelength, in nm.

Details

forageLIBS holds the mean of the 8 shots of each measurement; these are the shots themselves, raw detector counts, in the order in which they were fired. To keep the package small, only two windows are kept: 380 to 430 nm (Ca II 393.37 and 396.85 nm, Ca I 422.67 nm, Al I 394.40 and 396.15 nm, the CN band at 388 nm) and 760 to 780 nm (K I 766.49 and 769.90 nm, O I 777 nm). The strongest lines reach the saturation of the detector (65535 counts) in most shots (132 of the 160); the `wavelength` argument of `reject_shots()` can leave them out.

The 20 measurements are those of the 368 with a shot rejected by `reject_shots()` in these windows (7), and 13 others drawn at random: their outlying shots are weak or strong plasmas (total intensity 0.7 or 1.3 times that of the other shots) with spectra of a different shape.

Source

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

See Also

`reject_shots()`, `plot_shots()`, forageLIBS

Examples

```
data(forageShots)
dim(forageShots)
# the shot-to-shot variability of the total intensity, per measurement
total <- rowSums(forageShots[-(1:5)])
round(tapply(total, forageShots$Measurement, function(v) 100 * sd(v) / mean(v)), 1)
```

`gap_derivative` *Gap-Segment Derivatives*

Description

Computes the first or second derivative of spectra by the gap-segment (Norris-Williams) method: the spectrum is averaged over segments of `segment` channels, and the derivative is the difference between segments separated by `gap` channels.

Usage

```
gap_derivative(x, derivative = 1, gap = 5, segment = 3, 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.
derivative	The derivative: 1 (default) or 2.
gap	The number of channels between two segments, an odd number. Default is 5.
segment	The number of channels averaged in each segment, an odd number. Default is 3.
segments	A logical: filter the segments between gaps of the wavelength axis separately (TRUE, default). Needs wavelengths as names.

Details

The first derivative at a channel is the mean of a segment after it minus the mean of a segment before it, the gap channels between the two segments being centered on the channel, divided by the distance between the centers of the segments (segment + gap channels). The second derivative combines three segments separated by gap channels, the middle one centered on the channel: the mean of the first, minus twice the mean of the middle one, plus the mean of the last, divided by the squared distance between the centers. The filter thus spans 2 * segment + gap channels (first derivative) or 3 * segment + 2 * gap channels (second derivative).

Averaging over a segment smooths the spectrum, and the gap sets the distance over which the difference is taken: unlike the window of savitzky_golay(), which sets both, the two can be chosen separately. With segment = 1, there is no smoothing (Norris gap derivative). A first derivative removes a constant offset, and a second derivative a straight baseline; a curved background, such as the continuum of a LIBS plasma, is reduced but not removed (see baseline_als()). Emission lines are only a few channels wide in LIBS spectra: a gap or segment wider than a line merges it with its neighbors, which matters for measuring lines more than for multivariate models of whole spectra.

The derivatives are exact for straight lines (first derivative) and parabolas (second derivative), and per channel: for derivatives per nm, divide the first derivative by the channel spacing (and the second by its square). Programs count the gap in different ways: here it is the number of channels between two segments, as the argument w of prospectr::gapDer(). Norris and Williams (1984) gave the second derivative the opposite sign.

Every channel is kept: the first and last channels, where the filter does not fit, take the derivative of the nearest channel where it fits. 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 the filter are returned as NA, with a warning.

Value

The derivatives, with the same shape and names as x (a tibble for a data frame).

Author(s)

Christian L. Goueguel

References

- Norris, K.H., Williams, P.C. (1984). Optimization of mathematical treatments of raw near-infrared signal in the measurement of protein in hard red spring wheat. I. Influence of particle size. Cereal Chemistry, 61(2):158-165.
- 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.
- Vrábel, J., Képeš, E., Duponchel, L., et al. (2020). Classification of challenging laser-induced breakdown spectroscopy soil sample data - EMSLIBS contest. Spectrochimica Acta Part B, 169:105872.

See Also

`step_gap_derivative()`, `savitzky_golay()`, `bin_spectra()`

Examples

```
data(forageLIBS)
spectrum <- unlist(forageLIBS[1, -(1:14)])
wl <- as.numeric(names(spectrum))
keep <- wl > 400 & wl < 410
norris <- gap_derivative(spectrum[keep], gap = 5, segment = 3)
savgol <- savitzky_golay(spectrum[keep], window = 11, derivative = 1)
plot(wl[keep], norris, type = "l", xlab = "Wavelength (nm)",
     ylab = "First derivative")
lines(wl[keep], savgol, col = "blue")
```

---

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 = 2 * stats::pnorm(-4 * stats::qnorm(0.75)),
  p = 0.9,
  plot = TRUE,
  id = NULL,
  group = NULL,
  scales = c("free_y", "fixed"),
  points = c("outliers", "all", "none"),
  label_outliers = FALSE,
  show_n = TRUE,
  show_mean = FALSE,
  annotate = NULL,
  horizontal = FALSE,
  log = FALSE,
  fill = "grey85",
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  caption = TRUE,
  base_size = 11,
  point_size = NULL,
  point_alpha = 0.75,
  x_labels_angle = 0,
  box_width = 0.5,
  notch = FALSE,
  notch_width = 0.5,
  staple_width = 0.5,
  xlabels.angle = deprecated(),
  xlabels.vjust = deprecated(),
  xlabels.hjust = deprecated(),
  box.width = deprecated(),
  notchwidth = deprecated(),
  staplewidth = deprecated()
)
```

Arguments

x	A data frame (or matrix) of numeric variables, with optionally the columns id and group.
alpha	The expected proportion of observations outside the fences in clean data, between 0 and 1. Default is $2\,\Phi(-4z_{0.75}) \approx 0.007$, as Tukey's boxplot for normal data.
p	The quantile order, between 0.5 and 1, of the estimation of g and h. Default is 0.9 (a breakdown point of 10%).
plot	A logical value indicating whether to plot the boxplot (default is TRUE) or to return its statistics.
id	Optional column of x (unquoted or as a string) identifying the observations, such as sample names: it names the outlying observations in the outliers table and with label_outliers.
group	Optional column of x (unquoted or as a string) of groups, such as sites or treatments: the boxes of each group are drawn side by side, with their own statistics. Rows with a missing group are left out.
scales	"free_y" (default), one panel per variable with its own value axis, or "fixed", a common value axis.
points	The observations drawn: "outliers" (default), the outlying ones; "all"; or "none".
label_outliers	A logical: label the outlying observations with their id, else their row number (FALSE, default). The labels are moved apart so that they do not overlap; where many outliers crowd together, some are left unlabeled.
show_n	A logical: write the number of values below each box (TRUE, default).
show_mean	A logical: mark the mean of each box with a diamond (FALSE, default).
annotate	Statistics added to each box (none by default): some of "shape", "outliers", "fences", "median", "spread", "location", "test" and "missing", or "all" (see Details).
horizontal	A logical: horizontal boxes (FALSE, default), suited to many variables or long names.
log	A logical: a logarithmic value axis (FALSE, default), for variables spanning orders of magnitude. The statistics are those of the data, not of their logarithm.
fill	The fill color of the boxes (default "grey85"), or one color per group (or per variable without group).
xlab, ylab	The titles of the axis of the variables (or groups) and of the value axis. Default is none.
title	The plot title. A long title is split into a title and a subtitle.
caption	TRUE (default), a caption naming the boxplot and its whiskers; FALSE, no caption; or a caption of your own.
base_size	The size of the text, in points. Default is 11.
point_size	The size of the points of the observations. Default is 1.6 at a base_size of 11, scaled with base_size.

`point_alpha` The opacity of the outlying observations, between 0 and 1 (default 0.75); the others, drawn with `points = "all"`, are two thirds as opaque.

`x_labels_angle` The angle (in degrees) of the labels below the boxes. Default is 0 (horizontal).

`box_width` The width of the boxes (default 0.5).

`notch` A logical value indicating whether to draw notches (default is `FALSE`): the notch spans the median $\pm 1.58\,\mathrm{IQR}/\sqrt{n}$ (McGill et al., 1978); two medians whose notches do not overlap differ roughly at the 5% level.

`notch_width` The width of the notch relative to the box (default 0.5).

`staple_width` The width of the staples at the ends of the whiskers, relative to the box (default 0.5).

`xlabels.angle`, `xlabels.vjust`, `xlabels.hjust`, `box.width`, `notchwidth`, `staplewidth`

[Deprecated] Use `x_labels_angle`, `box_width`, `notch_width` and `staple_width`; the justification of the labels now follows their angle.

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.

The data x_i are first mapped, preserving their ranks, into (0, 1) by $r_i = (\tilde{x}_i - \min_j \tilde{x}_j + 0.1)/(\max_j \tilde{x}_j - \min_j \tilde{x}_j + 0.2)$, with $\tilde{x}_i = (x_i - \text{median})/\text{IQR}$, and then onto the real line by $w_i = \Phi^{-1}(r_i)$. The w_i, standardized by their median and their interquartile range divided by $z_{0.75} - z_{0.25} = 1.349$, are fitted by a Tukey g-and-h distribution, whose skewness g and tail heaviness h are estimated from the quantiles of orders p and $1 - p$. The fences are the quantiles of orders $\alpha/2$ and $1 - \alpha/2$ of the fitted distribution, mapped back to the scale of the data: a proportion α of the observations of a clean distribution, of whatever skewness and tails, is expected outside. The default $\alpha = 2\,\Phi(-4z_{0.75}) \approx 0.7\%$, the rate of Tukey's boxplot for normal data, is that of the authors' implementation (the Stata command `robbox` of Jann, Verardi and Vermandele). The fences are not inside the box, and the whiskers end at the most extreme observations within the fences.

The outlying observations are therefore *atypical at the rate* α: about αn of them are expected in clean data, so that with a large α (such as 5%) many flagged observations are not errors. As h may be negative (tails lighter than normal), where the g-and-h transform turns back before the $\alpha/2$ quantile the fence is its extreme value.

The layout, the points, the annotations and the options for publication figures are those of `adjusted_boxplot()`: see its details.

Value

- If `plot = TRUE`, a `ggplot2` object.
- If `plot = FALSE`, a list of tibbles: `stats`, `outliers` and, with `group`, `tests`, as in `adjusted_boxplot()`, with the estimated `g` and `h` instead of the medcouple, and `alpha * n` expected outlying values.

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
- Verardi, V., Vermandele, C. (2016). Outlier identification for skewed and/or heavy-tailed unimodal multivariate distributions. Journal de la Société Française de Statistique, 157(2):90–114

See Also

adjusted_boxplot()

Examples

```
data(forageLIBS)
minerals <- forageLIBS[c("Measurement", "Ca", "Mg", "P", "K", "S")]

# mineral contents, each on its own axis and with its own fill color, the
# outlying samples named: a figure of a journal page width
p <- generalized_boxplot(minerals, id = Measurement, label_outliers = TRUE,
                         fill = hcl.colors(5, "Pastel 1"), ylab = "Content (%)",
                         base_size = 8)
p
# saved at that size (here in the temporary directory)
ggplot2::ggsave(file.path(tempdir(), "minerals.pdf"), p, width = 175, height = 70,
                units = "mm")

# the statistics and the outlying samples
res <- generalized_boxplot(minerals, id = Measurement, plot = FALSE)
res$stats
res$outliers

# with a detection rate of 5%, about 18 of the 368 samples would be
# flagged per mineral even in clean data
generalized_boxplot(minerals, id = Measurement, alpha = 0.05, plot = FALSE)$stats

# potassium and phosphorus by calcium level, with notches, means and a
# darker blue for a higher level: a figure of a journal column (85 x 75 mm)
minerals$Ca_level <- cut(minerals$Ca, quantile(minerals$Ca, 0:3 / 3),
                         labels = c("Low Ca", "Mid Ca", "High Ca"),
                         include.lowest = TRUE)
p <- generalized_boxplot(minerals[c("K", "P", "Ca_level")], group = Ca_level,
                         notch = TRUE, show_mean = TRUE, ylab = "Content (%)",
                         fill = c("#deebf7", "#9ecae1", "#4292c6"),
                         title = "Potassium and phosphorus by calcium level",
                         base_size = 8)
p

# trace elements on a logarithmic axis, horizontal, with all the samples
# and one fill color for all the boxes
traces <- forageLIBS[c("Fe", "Mn", "Zn")]
generalized_boxplot(traces, log = TRUE, horizontal = TRUE, points = "all",
                    fill = "#fdd49e", ylab = "Content (mg/kg)")
```

`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 `ncomp` 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,
  ncomp = 2,
  group = NULL,
  conf_level = 0.975,
  method = "f",
  k = deprecated()
)
```

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 `ncomp` embedding coordinates (columns named like `PC1`, `UMAP1`, ...), or else the first `ncomp` numeric columns.
`ncomp`	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), `"beta"` or `"new"`.
`k`	**[Deprecated]** Use `ncomp`.

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 and k components, is

- `method = "f"` (default): $k(n-1)/(n-k)\, F_{1-\alpha}(k, n-k)$, the conventional limit of score plots (Jackson, 1991);
- `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;
- `method = "new"`: $k(n+1)(n-1)/(n(n-k))\, F_{1-\alpha}(k, n-k)$, the exact limit for a new sample, independent of the estimates (larger than the F limit by $(n+1)/n$).

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, ncomp = 3)
  t2[t2$outlier_97.5, ]
  # two limits, with the exact distribution of the calibration samples
  hotelling_t2(pca, ncomp = 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, drop.na = deprecated())
```

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.
`drop.na`	**[Deprecated]** Use `drop_na`.

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

See Also

`medcouple()`, `adjusted_boxplot()`, `directional_outlyingness()`

Examples

```
# 200 right-skewed values (a gamma distribution shifted to start at 100),
# 3 low outliers below them and 3 high outliers
set.seed(1)
x <- c(100 + rgamma(200, shape = 2, scale = 10), 85, 87, 89, 220, 230, 240)
group <- rep(c("regular", "outlier"), c(200, 6))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))
medcouple(x)   # positive: skewed to the right

# symmetric fences: the upper fence flags regular values of the long tail,
# and the lower fence lies below the low outliers
```

```
subset(iqr_outliers(x), flag)
# fences adjusted for the skewness: the 6 outliers, and no regular value
subset(iqr_outliers(x, skew = TRUE), flag)

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; gives up after 20 s if NIST does not answer
lines <- try(libs_lines(c("Ca I", "Ca II"), wavelength = c(390, 450), top = 5,
                        timeout = 20))
if (!inherits(lines, "try-error")) lines
```

`linear_reg_lts` *Robust Linear Regression in tidymodels (LTS parsnip Engine)*

Description

specProc adds the `"lts"` engine to `parsnip::linear_reg()`: the reweighted least trimmed squares (LTS) regression of `robustbase::ltsReg()`, which resists vertical outliers and bad leverage points. `lts_fit()` fits it outside of tidymodels.

Usage

```
lts_fit(x, y, alpha = 0.75, nsamp = 500)
```

Arguments

x	A numeric matrix or data frame of the predictors.
y	A numeric vector of the response.
alpha	The fraction of observations whose squared residuals are minimized, between 0.5 and 1. Default is 0.75.
nsamp	The number of random subsets. Default is 500.

Details

The engine is available once parsnip and specProc are both loaded, in either order. It is for regression with one outcome.

Combined with `step_robpca()` in a workflow, it gives the robust principal component regression of Hubert and Verboven (2003), whose robust PCA has `num_comp` components (see also `rpcr()`): the robust scores of the spectra, then the LTS regression on them. It also gives robust calibration curves from a few line intensities.

Value

`lts_fit()` returns an object of class `specproc_lts`, a list with the `coefficients` (intercept first), their standard errors `std_error` and the residual degrees of freedom `df_residual` of the reweighted least-squares fit, the robust `r_squared`, the robust `scale` of the residuals, the `fitted` values, `residuals`, standardized residuals `std_resid` and the `weights` of the reweighted fit (1 for the observations whose standardized residual is within the cut-off). Its `predict()` method takes `newdata`, and `tidy()` and `glance()` summarize it (see specproc-tidy-models).

Engine arguments

- `alpha`: the fraction of observations whose squared residuals are minimized, between 0.5 and 1 (default 0.75, as in `rpcr()`).
- `nsamp`: the number of random subsets (default 500).

The main arguments of `parsnip::linear_reg()`, `penalty` and `mixture`, are not used. LTS uses random subsets, so use `set.seed()` before fitting for reproducible results.

Author(s)

Christian L. Goueguel

References

- Rousseeuw, P.J., Van Driessen, K. (2006). Computing LTS regression for large data sets. Data Mining and Knowledge Discovery, 12(1):29-45.
- Hubert, M., Verboven, S. (2003). A robust PCR method for high-dimensional regressors. Journal of Chemometrics, 17(8-9):438-452.

See Also

`rpcr()`, `step_robpca()`, `parsnip::linear_reg()`

Examples

```
library(parsnip)
data(forageLIBS)
lines <- forageLIBS[c("Ca", "393.3599236", "396.8602175")]  # Ca II K and H
names(lines) <- c("Ca", "CaII_393", "CaII_397")
set.seed(1)
fit <- linear_reg() |>
  set_engine("lts") |>
  fit(Ca ~ ., data = lines[1:300, ])
predict(fit, lines[301:305, ])
extract_fit_engine(fit)$coefficients
```

---

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. The heads of molecular bands can be overlaid too. Lines of a species picked on the spectrum build a Boltzmann plot, whose temperature is fitted again as lines are added or left out.

Usage

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

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() as launch.browser. 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.
launch.browser	**[Deprecated]** Use launch_browser.

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()`);
- molecules whose band heads are marked with dashed lines, at a vibrational temperature (see `molecular_bands()`; no download needed);
- 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()`), or once for all sessions with the persistent cache (see `download_cache()`); later changes of the settings are computed locally. The zoom of the plot is kept when the lines change.

Boltzmann plot. Clicking the dot at the top of a line marker adds the line to the Boltzmann plot of its species, and clicking it again removes it; **Add shown lines** adds all the lines shown for the species chosen in the Boltzmann plot panel. The intensities are measured on the displayed spectrum with `line_intensities()`, around the tabulated wavelengths plus the wavelength shift, with the settings of the panel, and the temperature is fitted with `boltzmann()`. Clicking a point of the plot leaves its line out of the fit or puts it back; lines without a positive intensity are left out. The table under the plot lists the upper-level energy, the signal-to-noise ratio and the residual of each line, **Use T** sets the temperature of the candidate lines to the fitted one, and **Done** closes the app and returns the fit to R. The spectra should be corrected for the spectral response of the instrument first (see `correct_response()`), and the lines should be optically thin (see `self_absorption()`).

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

Value

Invisibly, when the app is closed with **Done**, the `boltzmann()` fit of the Boltzmann plot, whose `lines` element holds the measured lines; `NULL` if the plot has fewer than 3 usable lines. Use `libs_lines()`, `plot_lines()`, `line_intensities()` and `boltzmann()` for the same results in scripts.

Author(s)

Christian L. Goueguel

See Also

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

Examples

```
if (interactive() && rlang::is_installed(c("shiny", "plotly", "bslib"))) {
  data(forageLIBS)
  fit <- line_finder(forageLIBS[-(1:14)])
  # after a Boltzmann plot closed with Done
  if (!is.null(fit)) plot_boltzmann(fit)
}
```

`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()).

intensity_se is the standard error of the intensity due to the noise of the spectrum: the noise (as for snr) propagated through the integration and the baseline subtraction, or, for the Voigt method, the standard error of the fitted area. The noise is taken as white and the same in every channel, so this is a lower bound: it leaves out the shot-to-shot fluctuations of the plasma, which usually dominate in LIBS. These are better estimated from replicate spectra, as the standard deviation of the intensities of the replicates divided by the square root of their number. plasma_uncertainty() propagates intensity_se to the temperature and the composition.

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, intensity_se, 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")
```

`lte_criteria` *Criteria of Local Thermodynamic Equilibrium in Transient Plasmas*

Description

Checks the local thermodynamic equilibrium (LTE) of a laser-induced plasma by the McWhirter criterion and by the temporal and spatial criteria of Cristoforetti et al. (2010) for transient and inhomogeneous plasmas.

Usage

```
lte_criteria(
  temperature,
  electron_density,
  delta_e,
  oscillator_strength,
  mass,
  gaunt = 0.2,
  time = NULL,
  length_scale = NULL,
  tolerance = 0.1
)
```

Arguments

`temperature`	The plasma temperature, in K.
`electron_density`	The electron density, in cm^{-3}.
`delta_e`	The energy gap of the transition, in eV.
`oscillator_strength`	The absorption oscillator strength of the transition.
`mass`	The atomic mass of the species, in u, or its element symbol (such as `"Ca"`).
`gaunt`	The effective Gaunt factor. Default is 0.2.
`time`	Optional times of the measurements (one per temperature), in s, for the temporal criterion.
`length_scale`	Optional length over which the temperature and density change, in mm, for the spatial criterion.
`tolerance`	The largest relative change (or ratio of lengths) for a criterion to be satisfied. Default is 0.1.

Details

The McWhirter criterion (see `mcwhirter_criterion()`) gives the minimum electron density for collisions to dominate the population of the levels in a stationary, homogeneous plasma. It is necessary but not sufficient in a laser-induced plasma, whose temperature and density change in time and in space (Cristoforetti et al., 2010):

- Temporal criterion: the plasma must change little during the time τ_{rel} it takes to relax to equilibrium, $|T(t+\tau_{rel}) - T(t)|/T(t) \ll 1$, and likewise for N_e, with

$$\tau_{rel} \approx \frac{6.3 \times 10^4}{N_e f \langle g \rangle} \Delta E \, (k_B T)^{1/2} \exp\left(\frac{\Delta E}{k_B T}\right) \text{ s}$$

- Spatial criterion: the temperature and density must change little over the distance λ an atom diffuses during τ_{rel},

$$\lambda \approx \frac{1.4 \times 10^{12}}{N_e} (k_B T)^{3/4} \left(\frac{\Delta E}{M f \langle g \rangle}\right)^{1/2} \exp\left(\frac{\Delta E}{2 k_B T}\right) \text{ cm}$$

  which must be much shorter than the length over which they change, such as the size of the plasma.

Here N_e is in cm^{-3}, ΔE and $k_B T$ in eV, f is the absorption oscillator strength of the transition across the gap ΔE, M the atomic mass of the species (u) and $\langle g \rangle$ the effective Gaunt factor (Van Regemorter, 1962). The slowest transition to equilibrate is usually the resonance transition between the ground state and the first excited level of the species, such as Ca II 393.37 nm (ΔE = 3.15 eV, f = 0.68); `nist_lines()` gives the oscillator strengths (`fik`). The effective Gaunt factor depends on the transition and on $\Delta E / k_B T$; the default, 0.2, errs toward long relaxation times. The criteria are order-of-magnitude checks.

The temporal criterion needs the temperature and density at several times, `time` (such as acquisition delays): their relative changes during τ_{rel} are computed from their rates of change between the measurements (finite differences of their logarithms). The spatial criterion needs the length over which they change, `length_scale`. A criterion is satisfied when the relative change, or the ratio of the diffusion length to `length_scale`, is at most `tolerance`.

Value

A tibble with one row per measurement (the arguments are recycled to a common length): the `temperature`, `electron_density` and `delta_e`; the McWhirter `minimum_density` and whether it is reached (`mcwhirter`); the `relaxation_time` (s) and `diffusion_length` (mm); with `time`, the `time`, the relative changes of the temperature and of the density during the relaxation time (`temperature_change`, `density_change`) and whether the `temporal` criterion is satisfied; with `length_scale`, the `length_ratio` of the diffusion length to it and whether the `spatial` criterion is satisfied.

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.
- Van Regemorter, H. (1962). Rate of collisional excitation in stellar atmospheres. The Astrophysical Journal, 136:906-915.

See Also

mcwhirter_criterion(), boltzmann(), electron_density()

Examples

```
# Ca II resonance transition (393.37 nm), in a plasma of 1 mm
lte_criteria(temperature = 9000, electron_density = 1e17, delta_e = 3.15,
             oscillator_strength = 0.68, mass = "Ca", length_scale = 1)

# a plasma cooling from 1 to 5 microseconds
lte_criteria(temperature = c(11000, 9500, 8500, 7800),
             electron_density = c(3e17, 1.5e17, 8e16, 5e16), delta_e = 3.15,
             oscillator_strength = 0.68, mass = "Ca", time = c(1, 2, 3, 5) * 1e-6)
```

---

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

Arguments

x	A numeric matrix or data frame, with one observation per row. Missing values are allowed.
ncomp	The number of principal components. If NULL (default), it is chosen from var_explained and ncomp_max.

`alpha`	The robustness parameter, between 0.5 and 1: the fraction of observations used in the fit. Default is 0.5.
`ncomp_max`	The maximum number of components. Default is 10.
`var_explained`	The fraction of variance used to choose `ncomp` 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.
`k`	**[Deprecated]** Use `ncomp`.
`kmax`	**[Deprecated]** Use `ncomp_max`.

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 `ncomp` is `NULL`, it is the smallest number of components that explain `var_explained` of their variance (at most `ncomp_max`).
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

```
data(forageLIBS)
# the 380-430 nm window (Ca II H and K lines), faster than all the channels
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
spectra |>
  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

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

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.

See Also

`lte_criteria()` for the temporal and spatial criteria of Cristoforetti et al. (2010)

Examples

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

---

medcouple *Medcouple*

Description

This function calculates the medcouple (MC), a robust measure of the skewness of a distribution (Brys *et al.* 2004), for a numeric vector.

Usage

```
medcouple(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 medcouple is the median of the kernel

$$h(x_i, x_j) = \frac{(x_j - m) - (m - x_i)}{x_j - x_i}$$

over all the pairs of observations $x_i \leq m \leq x_j$ with $x_i \neq x_j$, where m is the median of the data (with a special definition of h for the ties at the median). It compares the distances to the median of the observations above and below it, scaled by their distance from each other, so it is unchanged by a shift or a rescaling of the data.

The interpretation of the medcouple is as follows:

- It lies between -1 and 1, and is 0 for a symmetric distribution.

- Positive values indicate a right-skewed distribution (a longer upper tail), and negative values a left-skewed one.

Unlike the classical skewness, which is based on the third moment and can be driven by a single extreme value, the medcouple has a breakdown value of 25%. It is computed by `robustbase::mc()`, in $O(n \log n)$ time. The medcouple is used by the adjusted boxplot (`adjusted_boxplot()`) and the skewness-adjusted fences of `iqr_outliers()`; `medcouple_weight()` applies it to each half of the data to measure the weight of the tails.

For spectra, one medcouple per wavelength (`sapply(spectra, medcouple)` for a data frame, `apply(spectra, 2, medcouple)` for a matrix) describes the shape of the distribution of the intensities at each channel, and `plot_medcouple()` draws it along the wavelength axis. Channels that saturate the detector in many spectra have a strongly negative medcouple, since their intensities pile up at the top of the range, while the lines of an element whose contents are skewed, with a few samples far above the others, have a positive one.

Value

The medcouple of `x`, or `NA` when `x` has no values.

Author(s)

Christian L. Goueguel

References

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

See Also

`plot_medcouple()`, `medcouple_weight()`, `adjusted_boxplot()`, `iqr_outliers()`, `summary_stats()`

Examples

```
data(forageLIBS)
# iron contents (mg/kg): a few samples are far above the others
fe <- forageLIBS$Fe[!is.na(forageLIBS$Fe)]
moments::skewness(fe)   # classical, driven by the highest contents
medcouple(fe)           # robust

# one medcouple per column; Fe and Zn have a few missing values
contents <- forageLIBS[c("Ca", "K", "Mg", "P", "Fe", "Mn", "Zn")]
round(sapply(contents, medcouple, drop_na = TRUE), 2)

# one medcouple per wavelength, from 380 to 430 nm
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
mc <- sapply(forageLIBS[which(wl > 380 & wl < 430)], medcouple)
summary(mc)
# the Ca II line at 393.37 nm saturates the detector (65535 counts) in
```

```
# most spectra: a negative medcouple; the Fe I line at 404.58 nm follows
# the skewed iron contents: a positive one
wl <- as.numeric(names(mc))
round(mc[c(which.min(abs(wl - 393.37)), which.min(abs(wl - 404.58)))], 2)
```

`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, drop.na = deprecated())
```

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.
`drop.na`	**[Deprecated]** Use `drop_na`.

Details

The left and right medcouples are robust measures of tail weight (Brys *et al.* 2006), based on the medcouple, a robust measure of skewness (Brys *et al.* 2004; see `medcouple()`). The left medcouple (LMC) is minus the medcouple of the observations below the median, and the right medcouple (RMC) is the medcouple of the observations above the median. A heavier tail skews the half of the distribution on its side toward that tail, which gives a larger value.

The medcouple resists up to 25% of outliers, but each half holds only half of the observations: LMC and RMC have a breakdown value of 12.5%, and resist up to 12.5% of outliers in a tail. This is twice the breakdown value of `quantile_weight()` with its defaults (6.25%).

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

See Also

quantile_weight(), medcouple()

Examples

```
# 10000 values from a normal distribution and from a right-skewed
# lognormal one, with a light left tail and a heavy right tail
set.seed(2)
n <- 10000
tails <- data.frame(
  distribution = rep(c("normal", "lognormal"), each = n),
  value = c(rnorm(n, mean = 1, sd = 0.5), rlnorm(n, meanlog = 0, sdlog = 0.5))
)
ggplot2::ggplot(tails, ggplot2::aes(value, fill = distribution)) +
  ggplot2::geom_density(alpha = 0.4) +
  ggplot2::scale_fill_manual(values = c(lognormal = "#b2182b", normal = "#2166ac"))

# both tails of the normal distribution weigh about 0.2; the left tail of
# the lognormal distribution is lighter and its right tail heavier. The
# kurtosis does not tell which tail is heavy
weights <- function(v) c(kurtosis = moments::kurtosis(v), unlist(medcouple_weight(v)))
round(t(sapply(split(tails$value, tails$distribution), weights)), 3)

# 7% of outliers: more than the quantile weights resist (6.25%), but not
# the medcouple weights (12.5%). The right quantile weight is set by the
# outliers, near its maximum of 1 (0.25 at the normal distribution), while
# the right medcouple weight is pulled up but stays far from it
set.seed(1)
x <- c(rnorm(9300), rnorm(700, mean = 10, sd = 0.5))
group <- rep(c("regular", "outlier"), c(9300, 700))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))
round(c(unlist(quantile_weight(x)), unlist(medcouple_weight(x))), 3)

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, drop.na = deprecated())
```

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.
`drop.na`	**[Deprecated]** Use `drop_na`.

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))
```

`molecular_bands` *Band Heads of Diatomic Molecules*

Description

Lists the band heads of the electronic systems of diatomic molecules that emit in laser-induced plasmas, with their Franck-Condon factors and relative intensities at a vibrational temperature, to identify the molecular bands of a spectrum: CN, C2, CH, OH, NH, N2, N2+, NO, CO, AlO, CaF, CaCl, SrF, SrCl, MgO, TiO, BO and SiO. The bands are computed from spectroscopic constants included in specProc, without a download.

Usage

```
molecular_bands(
  molecule = NULL,
  wavelength = c(200, 900),
  temperature = 5000,
  min_relative = 0.01
)
```

Arguments

`molecule`	A character vector of molecules, such as `c("CN", "C2")`, or `NULL` (default) for all of them.
`wavelength`	A numeric vector of length 2: the wavelength range, in nm. Default is 200 to 900 nm.
`temperature`	The vibrational temperature, in K, for the relative intensities and the formation of the heads. Default is 5000.
`min_relative`	The minimum relative intensity of the bands kept, between 0 and 1. Default is 0.01.

Details

The origin of the band from the vibrational level v' of the upper state to v'' of the lower state is

$$\nu_0 = \nu_{00} + G'(v') - G'(0) - G''(v'') + G''(0)$$

with the measured origin ν_{00} of the 0-0 band and the vibrational terms $G(v) = \omega_e(v + 1/2) - \omega_e x_e(v + 1/2)^2 + \omega_e y_e(v + 1/2)^3$. The head is where the P or R branch turns back,

$$\nu_{head} = \nu_0 - \frac{(B_{v'} + B_{v''})^2}{4(B_{v'} - B_{v''})}$$

with the rotational constants $B_v = B_e - \alpha_e(v + 1/2)$. A band is degraded to the red when $B_{v'} < B_{v''}$ (its head is on the violet side) and to the violet otherwise. When the head would form at a rotational level above twice the most populated one at `temperature`, as in the bands of hydrides such as CH and NH, the band has no head: its strongest feature is the Q branch near the origin, and

`wavelength` is the origin. The spin-orbit components of Π, Δ and Φ states are separate sub-bands when their origins are known; other fine structure is ignored.

The Franck-Condon factor $q_{v'v''}$ is the squared overlap of the vibrational wavefunctions of Morse potentials (harmonic when the anharmonicity is not known). The relative intensity of a band is

$$I \propto q_{v'v''}\, \nu^4 \exp\left(-\frac{hc\,[G'(v') - G'(0)]}{k_B T}\right)$$

(in energy units, with a constant electronic transition moment), normalized to the strongest band of each system. Intensities of different systems or molecules are not comparable: that would need their transition moments and the populations of their upper states. When the rotational constants of a state are not known, only the bands with $v' = v''$ are listed, without Franck-Condon factor or head.

The heads agree with measured heads to about 0.05 nm for well-studied systems, such as the CN violet, C2 Swan, N2 second positive, N2+ first negative and AlO B-X systems. They are less accurate for bands of high v, perturbed states and hydrides, whose rotational structure is open. `molecular_spectrum()` simulates the rotational structure of the bands from a line list.

The constants are those compiled by Huber and Herzberg (1979), from the NIST Chemistry WebBook. Wavelengths are in air between 200 and 2000 nm and in vacuum outside this range, as in `nist_lines()`.

Value

A tibble with one row per band, sorted by wavelength: `molecule`, `system` (such as `"B-X"`), its `name`, the `upper` and `lower` electronic states, the vibrational quantum numbers `v_upper` and `v_lower`, the `wavelength` of the head (nm), the `origin` of the band (nm), the direction in which the band is `degraded` (`"red"` or `"violet"`), whether it has a `head`, the Franck-Condon factor `fcf` and the `relative_intensity`.

Author(s)

Christian L. Goueguel

References

- Huber, K.P., Herzberg, G. (1979). Molecular Spectra and Molecular Structure IV. Constants of Diatomic Molecules. Van Nostrand Reinhold, New York. Data from the NIST Chemistry WebBook, NIST Standard Reference Database 69, https://webbook.nist.gov/chemistry/.
- Herzberg, G. (1950). Molecular Spectra and Molecular Structure I. Spectra of Diatomic Molecules, 2nd edition. Van Nostrand, Princeton.

See Also

`plot_lines()` and `line_finder()` to mark the bands on a spectrum, `molecular_spectrum()`, `libs_lines()`

Examples

```
# the CN violet and C2 Swan bands
molecular_bands(c("CN", "C2"), wavelength = c(380, 520), min_relative = 0.05)

# the CN bands in the mean spectrum of the forage samples
data(forageLIBS)
spectrum <- colMeans(forageLIBS[-(1:14)])
wl <- as.numeric(names(spectrum))
plot_lines(spectrum[wl > 383 & wl < 390], bands = molecular_bands("CN"),
           interactive = FALSE)
```

`molecular_spectrum` *Simulated Emission Spectra of Diatomic Molecules*

Description

Simulates the emission spectrum of a diatomic molecule in local thermodynamic equilibrium, from a line list of the ExoMol database, convolved with the instrument profile, to compare with the molecular bands of a measured spectrum.

Usage

```
molecular_spectrum(
  molecule,
  temperature = 5000,
  wavelength,
  resolution = 0.05,
  isotopologue = NULL,
  dataset = NULL,
  units = "energy",
  lines = FALSE,
  cache = NULL,
  timeout = 300
)
```

Arguments

`molecule`	The molecule, as named by ExoMol, such as `"CN"`, `"C2"`, `"OH"` or `"AlO"`.
`temperature`	The temperature, in K. Default is 5000.
`wavelength`	A numeric vector of length 2: the wavelength range, in nm.
`resolution`	The full width at half maximum of the instrument profile, in nm. Default is 0.05.
`isotopologue`	The isotopologue, as in the file names of ExoMol, such as `"12C-14N"` or `"13C-14N"`. Default is the first one listed by ExoMol, the most abundant.
`dataset`	The line list, such as `"MoLLIST-CN"`. See Details for the default.
`units`	The units of the intensities: `"energy"` (default) or `"photons"`.

`lines`	A logical: return the lines instead of the convolved spectrum (`FALSE`, default).
`cache`	A directory where the files are kept between sessions. By default (`NULL`), the persistent cache of specProc when it is enabled (see `download_cache()`), or a temporary directory for the session.
`timeout`	The download timeout, in seconds. Default is 300.

Details

The line list (energy levels and Einstein coefficients of the rovibronic transitions) is downloaded from ExoMol. In a plasma in LTE at temperature T, optically thin, the intensity of the line from the upper level u (energy E_u, degeneracy g_u) to the lower level l is

$$I \propto g_u A_{ul}\, \nu \exp\left(-\frac{hcE_u}{k_B T}\right)$$

in energy units (without ν for photon counts), with the same temperature for rotation and vibration. The lines are convolved with a Gaussian profile of full width at half maximum `resolution`, and the spectrum is normalized to its maximum in the range.

By default, the empirical MoLLIST line list of Bernath (2020) is used when ExoMol has one for the most abundant isotopologue (CN, C2, CH, OH, NH, CaF, MgH, CaH), as they are small (about 1 MB) and their line positions accurate, and the recommended line list otherwise (`AlO`: 33 MB). Other line lists are much larger: hundreds of megabytes for TiO, MgO or CaO. `exomol_datasets()` lists the line lists of a molecule and their sizes.

The files are kept for the R session, or on disk to be reused in later sessions: in the directory `cache`, or in the persistent cache of specProc when it is enabled (see `download_cache()`), where line lists are kept until the cache is cleared. A molecule whose files are on disk needs no internet connection when `isotopologue` and `dataset` are given. Partition functions are not needed, as the spectrum is normalized. Wavelengths are in air between 200 and 2000 nm.

ExoMol data are released under the Creative Commons Attribution-ShareAlike 4.0 license; cite ExoMol and the article of the line list (given on the ExoMol page of the data set) when you use them.

Value

A tibble with the `wavelength` (nm) and the relative `intensity` of the spectrum, on a grid of step `resolution / 10`; with `lines = TRUE`, a tibble with one row per line: `wavelength` (nm), `wavenumber` (cm^{-1}), relative `intensity`, Einstein coefficient `A` (s^{-1}), energies `E_upper` and `E_lower` (cm^{-1}) and total angular momenta `J_upper` and `J_lower`. The molecule, isotopologue, line list, temperature and resolution are attributes.

Author(s)

Christian L. Goueguel

References

- Tennyson, J., Yurchenko, S.N., et al. (2020). The 2020 release of the ExoMol database: molecular line lists for exoplanet and other hot atmospheres. Journal of Quantitative Spectroscopy and Radiative Transfer, 255:107228. https://exomol.com

- Bernath, P.F. (2020). MoLLIST: molecular line lists, intensities and spectra. Journal of Quantitative Spectroscopy and Radiative Transfer, 240:106687.

See Also

molecular_bands(), exomol_datasets()

Examples

```
# needs an internet connection (about 1 MB); gives up after 20 s if ExoMol
# does not answer
cn <- try(molecular_spectrum("CN", temperature = 6000, wavelength = c(383, 389),
                             timeout = 20))
if (!inherits(cn, "try-error")) {
  plot(cn$wavelength, cn$intensity, type = "l", xlab = "Wavelength (nm)",
       ylab = "Relative intensity")
}
```

---

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,
  drop.offset = deprecated(),
  drop.na = deprecated()
)
```

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.

`drop.offset`, `drop.na`
[Deprecated] Use `drop_offset` and `drop_na`.

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,
  wavelength = NULL,
  id = NULL,
  max_iter = 200,
  fixed = NULL,
  lock_centers = FALSE,
  share_widths = FALSE,
  area_ratios = NULL,
  max.iter = deprecated(),
  wlgth.min = deprecated(),
  wlgth.max = deprecated()
)
```

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)
`wavelength`	The wavelength range of the fit, `c(min, max)` in nm, or `NULL` (default) for all the channels.
`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)
fixed	The widths held at their given values (wG, wL, one value or one per peak) instead of being fitted: "wG", "wL" or both, for the peaks whose profile has that width. Default is NULL (all widths fitted). See peak_fit() for holding the Gaussian width at the instrumental and Doppler width.
lock_centers	Keep the separations of peaks (see details): FALSE (default), TRUE, a logical vector or group labels, one per peak.
share_widths	Share the widths of the lines (see details): FALSE (default), TRUE, a logical vector or group labels, one per peak. The lines of a group must have the same profile.
area_ratios	The fixed relative areas of the lines, one per peak (NA for a free area), or NULL (default) for free areas.
max.iter, wlgth.min, wlgth.max	**[Deprecated]** Use max_iter, and wavelength = c(wlgth.min, wlgth.max).

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.

The lines of a multiplet, or of a fine-structure doublet, are related, and constraints make the fit of overlapping lines much more stable:

- lock_centers: the lines keep the separations of their peaks (accurate wavelengths, such as those of the NIST database) and move together by one fitted shift, which also absorbs an offset of the wavelength axis. The shift is listed in tidied as shift (or shift_1_2, ..., named by its lines, when there are several groups).
- share_widths: the lines share their Gaussian and Lorentzian widths, as lines of the same multiplet have nearly the same Stark width.
- area_ratios: the areas of the lines are in fixed ratios, with one fitted area. In an optically thin plasma in LTE, the ratio of two lines is $g_k A_{ki} \lambda^{-1} \exp(-E_k/k_B T)$ (energy units), independent of the temperature for lines from the same upper level: the relative_intensity of libs_lines(). Self-absorbed lines, such as resonance lines, break these ratios.

lock_centers and share_widths take TRUE (all lines), a logical vector (the lines constrained together) or one group label per line (NA for a line left free), so that, for example, a multiplet can be constrained while an interfering line of another element is not. area_ratios gives a ratio per line, NA for a line with a free area.

tidied lists the center, widths and area of every line (named with the suffix _i), with standard errors propagated from the fitted parameters: lines that share a parameter have the same width, or centers that differ by a constant.

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]]

# the same doublet with the NIST separation of its lines and a common width
res <- multipeak_fit(doublet, peaks = c(404.414, 404.721), profiles = "gaussian",
                     lock_centers = TRUE, share_widths = TRUE)
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 $\mathbf{b}$ is orthogonal to this space, so the NAS space within the model is one-dimensional and spanned by $\mathbf{b}$ (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 $\mathbf{b}$.
- `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; gives up after 20 s if NIST does not answer
try(nist_ionization_energy(c("Ca I", "Mg I"), timeout = 20))
```

`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, and between sessions with the persistent cache (see `download_cache()`). 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; gives up after 20 s if NIST does not answer
ca <- try(nist_levels("Ca I", timeout = 20))
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 vector of emitters in spectroscopic notation, such as `"Ca II"` or `c("Ca I", "Ca II")`. The lines of several species are returned in one table, species by species.
`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, and between sessions with the persistent cache (see `download_cache()`). 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; gives up after 20 s if NIST does not answer
ca <- try(nist_lines(c("Ca I", "Ca II"), wavelength = c(390, 400), timeout = 20))
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,
  drop.na = deprecated()
)
```

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.
drop.na	**[Deprecated]** Use drop_na.

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]
```

---

num_intervals *Number of Intervals*

Description

The number of spectral intervals selected by interval PLS, a tuning parameter of `step_select_wavelengths()` with `method = "ipls"`.

Usage

```
num_intervals(range = c(1L, 10L), trans = NULL)
```

Arguments

range	A two-element vector with the smallest and largest numbers of intervals. Default is 1 to 10.
trans	A transformation object from the scales package, or NULL.

Value

A dials quantitative parameter.

Examples

```
num_intervals()
```

---

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` (**T**) and `y` (**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` (**W**) and `y` (**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 $\mathbf{t}$ (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,
  max.iter = deprecated(),
  pls.ncomp = deprecated()
)
```

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.

`max.iter`, `pls.ncomp`
[Deprecated] Use `max_iter` and `pls_ncomp`.

Details

The OSC algorithm identifies and removes the orthogonal variation in the input spectral matrix, **X**, by iteratively deflating **X** with respect to the response vector **y**. The resulting **X**-matrix contains only the variation that is relevant to the **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 **y** and the weights are obtained by a PLS regression of the orthogonalized score on **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 **X** subject to the scores being exactly orthogonal to **y**.

For the iterative methods, **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, drop.na = deprecated())
```

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`.
`drop.na`	**[Deprecated]** Use `drop_na`.

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 primary (reference) instrument.
x2	A matrix or data frame containing spectra of the same samples acquired with the secondary instrument, to be standardized.
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 primary (reference) instrument, a PLSR model is fitted between the response of the primary instrument at j and the responses of the secondary 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 secondary 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 secondary spectrum to the whole primary spectrum is also fitted, and the transfer matrix and intercept are $(1 - \alpha)$ times the local ones plus α times the global ones.

Value

An object of class `specproc_transfer`, a list with

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

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

and the settings of the transfer. Use predict() to transfer secondary spectra.

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.

See Also

`ds()`, `sst()`, `slope_bias()`, `predict.specproc_transfer()`

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 <- predict(res, x2)
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,
  wavelength = NULL,
  id = NULL,
  max_iter = 200,
  fixed = NULL,
  max.iter = deprecated(),
  wlgth.min = deprecated(),
  wlgth.max = deprecated()
)
```

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)
wavelength	The wavelength range of the fit, c(min, max) in nm, or NULL (default) for all the channels.
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)
fixed	The widths held at their given values (wG, wL) instead of being fitted: "wG", "wL" or both. Default is NULL (all widths fitted).
max.iter, wlgth.min, wlgth.max	**[Deprecated]** Use max_iter, and wavelength = c(wlgth.min, wlgth.max).

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`.

The Gaussian and Lorentzian widths of a Voigt profile are strongly correlated, so a free fit determines the Lorentzian (Stark) width poorly when the line is narrow. In a LIBS plasma, the Gaussian width is the instrumental width, combined with the Doppler width in quadrature, $w_G = \sqrt{w_{inst}^2 + w_D^2}$, and can be held fixed with `fixed = "wG"` and the value of `wG`. The instrumental width is the width of a line of a low-pressure lamp (mercury, argon), whose Stark and Doppler widths are negligible, fitted with a Gaussian profile at a nearby wavelength; the Doppler width is given by `doppler_width()`. A fixed width is listed in `tidied` without standard error.

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, of class `specproc_peak_fit` (see tidy() and glance()),
- `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]]
```

```
# a Voigt line whose Gaussian width (instrument and Doppler) is known
x <- seq(395, 397, by = 0.01)
set.seed(1)
y <- voigt_profile(x, y0 = 10, xc = 396, wG = 0.08, wL = 0.05, A = 50) +
  rnorm(length(x), sd = 2)
line <- tibble::as_tibble(as.list(stats::setNames(y, x)))
peak_fit(line, profile = "voigt", wG = 0.08, fixed = "wG")$tidied[[1]]
```

`plasma_uncertainty` *Monte Carlo Uncertainty of Plasma Diagnostics*

Description

Propagates the uncertainties of the line intensities, of the transition probabilities and of the electron density to the temperature of a Boltzmann or Saha-Boltzmann plot, or to the temperature and the composition of a calibration-free (CF-LIBS) analysis, by Monte Carlo simulation.

Usage

```
plasma_uncertainty(
  object,
  n = 1000,
  level = 0.95,
  aki_rsd = NULL,
  electron_density_se = 0,
  temperature_se = 0
)
```

Arguments

`object`	A fit returned by `boltzmann()`, `saha_boltzmann()` or `cf_libs()`.
`n`	The number of draws. Default is 1000.
`level`	The confidence level of the intervals. Default is 0.95.
`aki_rsd`	The relative standard uncertainties of the transition probabilities, such as 0.1 for 10%: one value, or one per line. By default, they are taken from the column `accuracy` of the lines.
`electron_density_se`	The standard uncertainty of the electron density of the fit, in cm^{-3}. Default is 0.
`temperature_se`	For a `cf_libs()` fit given the temperature: the standard uncertainty of the temperature, in K. Default is 0.

Details

Each of the `n` draws perturbs the inputs of the fit within their uncertainties and repeats the fit. The standard deviation of the results is their standard uncertainty, and their quantiles give the interval at `level`. The perturbations are lognormal (multiplicative), with the relative standard uncertainties of the inputs, so that intensities, transition probabilities and densities stay positive. The sources are:

- the intensities: the column `intensity_se` of the lines (standard errors, in the units of `intensity`), as returned by `line_intensities()`;
- the transition probabilities: `aki_rsd`, or else the accuracy grades of the NIST Atomic Spectra Database in the column `accuracy` (see `aki_uncertainty()`);
- the electron density, for Saha-Boltzmann plots and CF-LIBS: `electron_density_se`, for example from `electron_density_uncertainty()`;
- the temperature, for a `cf_libs()` fit given the temperature: `temperature_se`.

A source that is not given is taken as exact. The errors of different lines are taken as independent, and the partition functions and the ionization energies as exact. Lines corrected by `correct_self_absorption()` are not independent: their corrected intensities are computed from their reference line.

The interval reflects the stated uncertainties of the inputs, whereas the standard error of the fit (`temperature_se` of the fit) reflects the scatter of the points about the fitted line. The reduced chi-squared compares the two: $\chi^2_\nu = \sum_i (r_i/\sigma_i)^2/\nu$, where r_i are the residuals of the fit, σ_i the standard uncertainties of the ordinates of the points (from those of the intensities and of the transition probabilities) and ν the residual degrees of freedom. Near 1, the stated uncertainties explain the scatter. Well above 1, other errors scatter the points (self-absorption, spectral response, departures from LTE, or understated uncertainties) and the interval is too narrow. Well below 1, the uncertainties are overstated; the NIST grades are upper bounds.

A draw whose Boltzmann plot has a non-negative slope has no finite temperature: it counts as an infinite temperature in the interval, whose upper bound can then be infinite, and is left out of the standard deviation. A CF-LIBS draw that fails is left out, with a warning.

Use `set.seed()` for reproducible results.

Value

An object of class `specproc_uncertainty`, a list with

- `estimates`: a tibble with one row per quantity: the `quantity` (`"temperature"`, `"atomic_fraction"` or `"mass_fraction"`), the `element` (CF-LIBS), the `estimate` of the fit, the standard uncertainty `se` (the standard deviation of the draws), and the `lower` and `upper` bounds of the interval;
- `draws`: a tibble of the draws, with the `draw` number, the `temperature` and, for CF-LIBS, the `element`, `atomic_fraction` and `mass_fraction`;
- `inputs`: a tibble with the relative standard uncertainties of the intensity (`intensity_rsd`) and of the transition probability (`aki_rsd`) of each line;
- `electron_density_rsd` and `temperature_rsd`: the relative standard uncertainties of the electron density and of a given temperature (`NULL` when not used by the fit);
- `chi_squared`: the reduced chi-squared of the fit (`NA` when the temperature is given, or when a point has no stated uncertainty);
- `temperature_se`: the standard error of the temperature of the fit;
- `n`, `level`, the number of `failed` draws and the `method`.

Author(s)

Christian L. Goueguel

References

- JCGM 101:2008. Evaluation of measurement data - Supplement 1 to the "Guide to the expression of uncertainty in measurement" - Propagation of distributions using a Monte Carlo method. Joint Committee for Guides in Metrology.

See Also

aki_uncertainty(), electron_density_uncertainty(), boltzmann(), saha_boltzmann(), cf_libs(), line_intensities()

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
# calcium lines, with their atomic data and accuracy grades 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),
  accuracy = c("C+", "C+", "C+", "C", "C", "C", "C", "C", "C", "C"),
  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 <- saha_boltzmann(lines, ionization_energy = 6.113, electron_density = 1.9e17)

# with a 15% standard uncertainty of the electron density
set.seed(1)
u <- plasma_uncertainty(fit, electron_density_se = 0.15 * 1.9e17)
u
u$estimates
```

---

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,
  xlab = NULL,
```

```
  ylab = NULL,
  base_size = 11,
  point_size = 2.5,
  point_alpha = 1
)
```

Arguments

object	An object returned by boltzmann(), saha_boltzmann() or cf_libs().
title	The plot title. By default, the method and temperature.
xlab, ylab	The axis titles. Default are the upper-level energy and the Boltzmann ordinate.
base_size	The size of the text, in points. Default is 11.
point_size	The size of the points of the lines. Default is 2.5.
point_alpha	The opacity of the points, between 0 and 1. Default is 1.

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,
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  point_size = 2,
  point_alpha = 1
)
```

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.
`xlab`, `ylab`	The axis titles. Default are the names of the concentration and signal columns.
`base_size`	The size of the text, in points. Default is 11.
`point_size`	The size of the points of the standards. Default is 2.
`point_alpha`	The opacity of the points of the standards, between 0 and 1. Default is 1; lower it when replicates overlap.

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, point_alpha = 0.4, xlab = "K (%)",
                 ylab = "K I 769.90 nm / C I 247.86 nm")
```

---

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,
  wavelength = NULL,
  xlab = NULL,
  ylab = NULL,
  base_size = 11
)
```

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".
wavelength	Optional (NULL by default). The wavelength range c(min, max) to show, in nm: the columns within it (of columns, if given). Needs variables named by wavelength.
xlab, ylab	The axis titles of the map. Default are "Wavelength (nm)" (or "Variable") and "Observation", with the order of the rows.
base_size	The size of the text, in points. Default is 11.

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

```
data(forageLIBS)
# the 380-430 nm window (Ca II H and K lines), faster than all the channels
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
fit <- macropca(center(spectra), ncomp = 3)

if (requireNamespace("patchwork", quietly = TRUE)) {
  plot_cell_map(fit, order = "cluster")
  # the Ca II H and K lines
  plot_cell_map(fit, wavelength = c(390, 400), order = "od")
}
```

`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,
  wavelength = NULL,
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  linewidth = 0.3
)
```

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.
wavelength	Optional (NULL by default). The wavelength range c(min, max) to draw, in nm; the top largest peaks within it are labeled. Needs variables named by wavelength.
xlab, ylab	The axis titles. Default are "Wavelength (nm)" (or "Variable") and the name of the values drawn. In interactive plots, each panel has its own vertical axis title and ylab is ignored.
base_size	The size of the text, in points. Default is 11; the peak labels scale with it. Ignored for interactive plots.
linewidth	The width of the bars (or, for interactive plots, of the line) drawn at each variable. Default is 0.3.

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, ncomp = 3)
q <- contributions(fit, data = spectra, samples = c(49, 127), reference = "regular")
plot_contributions(q, spectra = spectra)
# the 390-400 nm window: the Ca II H and K lines
plot_contributions(q, spectra = spectra, wavelength = c(390, 400), top = 4)
```

`plot_control_chart` *Plot a Control Chart*

Description

Draws a control chart returned by `control_chart()`: the monitored statistic in time order, the center line, the control limits and the signals, with the end of the baseline marked.

Usage

```
plot_control_chart(
  object,
  title = NULL,
  xlab = "Order",
  ylab = NULL,
  base_size = 11,
  point_size = 1.3,
  point_alpha = 1
)
```

Arguments

`object`	The result of `control_chart()`.
`title`	The plot title. By default, the method and the number of signals.
`xlab, ylab`	The axis titles. Default are `"Order"` and the name of the statistic.
`base_size`	The size of the text, in points. Default is 11.
`point_size`	The size of the points. Default is 1.3.
`point_alpha`	The opacity of the points, between 0 and 1. Default is 1.

Value

A ggplot object.

See Also

`control_chart()`

Examples

```
set.seed(1)
ratio <- 1 + c(rep(0, 60), seq(0, 0.06, length.out = 40)) + rnorm(100, sd = 0.02)
plot_control_chart(control_chart(ratio, method = "cusum", baseline = 40))
```

plot_coomans *Coomans Plot of a Robust SIMCA Model*

Description

Plots the distance of each observation to one class of a model fitted by `rsimca()` against its distance to another class (Coomans plot), with the class boundaries and the classification boundary. It shows which observations belong to one class, to both, or to neither, and which are misclassified.

Usage

```
plot_coomans(
  object,
  newdata = NULL,
  group = NULL,
  classes = NULL,
  labels = 3,
  log = FALSE,
  shade = FALSE,
  title = NULL,
  ...,
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  point_size = 2.2,
  point_alpha = 0.85
)
```

Arguments

`object`	An object returned by `rsimca()`.
`newdata`	Optional new observations to add to the plot (a numeric matrix or data frame with the training variables).
`group`	Optional classes of `newdata`, to fill their points.
`classes`	The two classes to plot, on the horizontal and vertical axes. Default is the first two classes of the model.
`labels`	The number of observations farthest from both classes to label, among those beyond 1 for both (by their row names, or row numbers). Default is 3; use 0 for no labels.
`log`	If `TRUE`, use logarithmic axes, which spread out the observations close to the classes when others are far away. Default is `FALSE`.
`shade`	If `TRUE`, shade the regions of the observations close to both classes and to neither, and name the four regions. Default is `FALSE`.
`title`	The plot title.

`...`	Further arguments passed to `ggplot2::geom_point()` to style the points, such as `stroke` (default 0.4) or `colour` (the outline, `"black"`).
`xlab`, `ylab`	The axis titles. Default are the distances to the two classes.
`base_size`	The size of the text, in points. Default is 11.
`point_size`	The size of the points (default 2.2), or a numeric vector with one value per observation (the training observations, then those of `newdata`), to vary the size of the points, with a legend.
`point_alpha`	The opacity of the points, between 0 and 1 (default 0.85).

Details

The distances are the combined distances D_j of the classification rule of `rsimca()`, built from the score and orthogonal distances to the robust PCA model of each class, divided by their cut-offs. The dashed lines are at 1: an observation beyond 1 for a class is beyond at least one of the cut-offs of this class, and an observation within both cut-offs of a class is within 1. The diagonal is the classification boundary: the observations above it are assigned to the class of the horizontal axis, those below it to the class of the vertical axis.

The four corners of the plot hold the observations close to the class of the horizontal axis only (top left), to the class of the vertical axis only (bottom right), to both (bottom left, where the classes overlap), and to neither (top right, outliers for both classes, or, with more than two classes, members of another class).

The points are filled by their class: the training class for the training observations, and for new observations (`newdata`, triangles) the classes given in `group`, or `"unknown"`. Observations of a class on the wrong side of the diagonal are misclassified.

Value

A ggplot object.

Author(s)

Christian L. Goueguel

References

- Vanden Branden, K., Hubert, M. (2005). Robust classification in high dimensions based on the SIMCA method. Chemometrics and Intelligent Laboratory Systems, 79(1-2):10-21.

See Also

`rsimca()`, `plot_outlier_map()`

Examples

```
# iris: train on 100 flowers, predict the 50 others
set.seed(1)
train <- sample(nrow(iris), 100)
fit <- rsimca(iris[train, 1:4], iris$Species[train], ncomp = 2)
```

```
# the first two classes, setosa and versicolor
plot_coomans(fit)
# versicolor and virginica, with the 50 new flowers (triangles)
plot_coomans(fit, classes = c("versicolor", "virginica"),
             newdata = iris[-train, 1:4], group = iris$Species[-train],
             log = TRUE, shade = TRUE)
```

`plot_curve_of_growth` *Plot a Curve of Growth*

Description

Draws the curve of growth fitted by `curve_of_growth()`: the reduced intensity of each line against its optical depth at the line center, on logarithmic axes, with the optically thin limit and the curve of growth of the median line profile.

Usage

```
plot_curve_of_growth(
  object,
  title = NULL,
  xlab = "Optical depth at the line center",
  ylab = "Reduced intensity",
  base_size = 11,
  point_size = 2.5,
  point_alpha = 1
)
```

Arguments

`object`	The result of `curve_of_growth()`.
`title`	The plot title. By default, the temperature and the largest optical depth.
`xlab, ylab`	The axis titles. Default are `"Optical depth at the line center"` and `"Reduced intensity"`.
`base_size`	The size of the text, in points. Default is 11.
`point_size`	The size of the points of the lines. Default is 2.5.
`point_alpha`	The opacity of the points, between 0 and 1. Default is 1.

Details

The reduced intensity, $IV(\lambda_0)/(FB_\lambda(T))$, equals the optical depth for an optically thin line, and bends below it as the line saturates. Points of all species lie on the same curve when the model holds (up to the differences of their line profiles), as in the C-sigma graphs of Aragón and Aguilera (2014).

Value

A ggplot object.

See Also

curve_of_growth()

Examples

```
lines <- data.frame(
  species = "Ca I", wavelength = c(400, 405, 520, 560, 610, 650),
  Aki = c(1.5e8, 1.2e8, 3e7, 5e7, 4e7, 2e7), gk = c(4, 2, 6, 4, 3, 5),
  Ei = c(0, 0, 2.0, 2.1, 2.2, 2.4), stark_width = 0.02
)
lines$Ek <- lines$Ei + 1239.842 / lines$wavelength
lines$intensity <- curve_of_growth_model(lines, 9000, scale = 1e20, optical_depth = 5)$intensity
plot_curve_of_growth(curve_of_growth(lines, temperature = 9000))
```

---

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,
  color_by = NULL,
  point_size = NULL,
  point_alpha = 0.8,
  ellipse = FALSE,
  conf_level = 0.975,
  robust = FALSE,
  distribution = "normal",
  hotelling = "none",
  ncomp = 2,
  t2_method = "f",
  flag = FALSE,
  label = NULL,
  biplot = FALSE,
  biplot_top = 10,
```

```
  aspect_ratio = 0.7,
  title = NULL,
  legend_title = NULL,
  palette = NULL,
  shapes = TRUE,
  legend = c("right", "bottom", "top", "inside", "none"),
  panel = c("shaded", "white"),
  caption = TRUE,
  base_size = 11,
  xlab = NULL,
  ylab = NULL,
  colour = deprecated(),
  size = deprecated(),
  alpha = deprecated(),
  k = deprecated()
)
```

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).
`color_by`	The variable coloring the points: a column of `data` (unquoted or as a string), or a vector with one value per sample.
`point_size`	The size of the points: a number, or a numeric column of `data` or vector with one value per sample to vary it. Default is 2 at a `base_size` of 11, scaled with `base_size`.
`point_alpha`	The opacity of the points, between 0 and 1. Default is 0.8.
`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 `color_by`). Needs the HotellingEllipse package (1.3.0 or later).
`ncomp`	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), `"beta"` or `"new"` (see `hotelling_t2()`). Ignored for a robust PCA, whose limit is the chi-square quantile of ROBPCA.
`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; "equal" gives the two axes the same scale, so that the distances between the samples are not distorted (for scores in the same units).
title	The plot title.
legend_title	The title of the legend of color_by. Default is its name; give it with units, such as "K (%)".
palette	The colors of color_by: NULL (default: Dark2 for up to 8 groups, viridis for a numeric variable), the name of a viridis palette ("viridis", "magma", "cividis", ...) or a vector of colors.
shapes	A logical: give each group of a discrete color_by its own point shape (up to 6 groups), so that the groups stay apart in grayscale (TRUE, default).
legend	The position of the legend: "right" (default), "bottom", "top", "inside" (the top right corner of the panel) or "none".
panel	"shaded" (default), the panel grey outside the outermost ellipse and white inside, or "white".
caption	TRUE (default), a caption saying what the ellipses and limits are; FALSE, no caption; or a caption of your own.
base_size	The size of the text, in points. Default is 11; use the text size of the journal (often 7 to 9 points) for a figure saved at its printed size. The points (unless point_size is a number), lines and labels scale with it.
xlab, ylab	The axis titles. Default are the names of the axes, with the variance they explain for a PCA.
colour, size, alpha	**[Deprecated]** Use color_by, point_size and point_alpha.
k	**[Deprecated]** Use ncomp.

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 color_by uses a continuous viridis scale, other types a discrete palette.

With ellipse = TRUE, a confidence ellipse is drawn for each group of a discrete color_by (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 `ncomp` 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 `ncomp` 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 `color_by` 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 `ncomp` components, so with `ncomp > 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 (`panel = "white"` for a white panel), with no grid and a fixed `aspect_ratio` (0.7 by default; `NULL` lets the plot fill the space, `"equal"` gives both axes the same scale), 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. The groups of a discrete `color_by` also get their own point shapes (`shapes`), and the axes plain numbers. A caption says what the ellipses and limits are (their level, distribution and estimates), which readers cannot tell from the plot.

Publication figures. Make the plot at its printed size, with `base_size` set to the text size of the journal, for example `ggsave("scores.pdf", p, width = 85, height = 75, units = "mm")` with `base_size = 8` for a single column.

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 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. `point_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(ncomp = 2) |>
    plot_embedding(hotelling = "all", flag = FALSE, label = TRUE)
}

# PCA of the K I lines, the samples grouped by potassium level, for a
# journal column
if (rlang::is_installed("ConfidenceEllipse")) {
  data(forageLIBS)
  wl <- suppressWarnings(as.numeric(names(forageLIBS)))
  pca <- stats::prcomp(forageLIBS[which(wl > 760 & wl < 772)])
  level <- cut(forageLIBS$K, 3, labels = c("Low K", "Mid K", "High K"))
  p <- plot_embedding(pca, color_by = level, ellipse = TRUE, legend_title = "Potassium",
                      legend = "bottom", base_size = 8)
  p
  # saved at that size (here in the temporary directory)
  ggplot2::ggsave(file.path(tempdir(), "scores.pdf"), p, width = 85, height = 80,
                  units = "mm")
}


# 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, color_by = 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, and the fitted parameters of each peak.

Usage

```
plot_fit(
  data,
  title = NULL,
  annotate = TRUE,
  show_fwhm = FALSE,
  ncol = NULL,
  xlab = "Wavelength (nm)",
  ylab = "Intensity (arb. units)",
  caption = TRUE,
  base_size = 11,
  point_size = 1.5,
  point_alpha = 1,
  point_color = "grey20",
  linewidth = 0.8,
  fit_color = "#b2182b",
  pt.size = deprecated(),
  pt.colour = deprecated(),
  pt.shape = deprecated(),
  pt.fill = deprecated(),
  line.size = deprecated(),
  line.colour = deprecated(),
  linetype = deprecated(),
  resid.shape = deprecated(),
  resid.size = deprecated(),
  resid.fill = deprecated(),
  resid.colour = deprecated()
)
```

Arguments

data	The output of peak_fit() or multipeak_fit().
title	The plot title. A long title is split into a title and a subtitle.
annotate	A logical: write the fitted parameters and the quality of the fit above each panel (TRUE, default).
show_fwhm	A logical: draw the FWHM of each peak as a bar at its half maximum (FALSE, default).
ncol	The number of spectra per row. Default is up to 2.
xlab, ylab	The axis titles. Default are "Wavelength (nm)" and "Intensity (arb. units)". The unit in the parentheses of xlab is that of the center and FWHM in the annotations.
caption	TRUE (default), a caption naming the profiles and the lines drawn; FALSE, no caption; or a caption of your own.
base_size	The size of the text, in points. Default is 11; use the text size of the journal (often 7 to 9 points) for a figure saved at its printed size.
point_size	The size of the data points (default 1.5).
point_alpha	The opacity of the data points, between 0 and 1 (default 1).
point_color	The color of the data points (default "grey20").
linewidth	The width of the fitted curve (default 0.8).
fit_color	The color of the fitted curve (default "#b2182b").
pt.size, pt.colour, pt.shape, pt.fill, line.size, line.colour, linetype, resid.shape, resid.size, resid.fill, resid.colour	**[Deprecated]** Use point_size, point_color, linewidth and fit_color; the other styles are fixed.

Details

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, numbered at their maximum. The dotted line is the fitted baseline y_0.

Parameters. With annotate = TRUE, the top of each panel gives, for each peak, its center, full width at half maximum (FWHM) and area, with their standard errors, and the quality of the fit: R^2 and the root mean square error (RMSE, the residual standard error). The model is $y = y_0 + \sum_j A_j f_j(x)$, with f_j of unit area, so that A_j is the area of peak j. The FWHM is wG (Gaussian), wL (Lorentzian), or for a Voigt or pseudo-Voigt profile the FWHM of the profile from both (Olivero and Longbothum, 1977, see voigt_fwhm()), its standard error propagated from the covariance of the estimates. With show_fwhm = TRUE, the FWHM of each peak is drawn as a bar at its half maximum. The estimates are rounded to two significant digits of their standard errors; all of them are in the tidied column of the fit.

Residuals. The lower panel gives the residuals, with dashed lines at plus and minus twice the RMSE: a structure in the residuals, beyond these lines or along the wavelength, shows a profile that does not fit the line (such as an asymmetric or self-absorbed line).

When several spectra were fitted, each gets its own fit and residual panels, ncol per row.

Value

A patchwork (ggplot2) object.

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

peak_fit(), multipeak_fit(), voigt_fwhm()

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
wl <- as.numeric(names(mean_spectrum))
# the Stark-broadened H-alpha line, with its FWHM
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",
         show_fwhm = TRUE)

# the K I doublet: two Gaussian peaks, for a journal column
doublet <- tibble::as_tibble(as.list(mean_spectrum[wl > 404.1 & wl < 405.0]))
plot_fit(multipeak_fit(doublet, peaks = c(404.414, 404.721), profiles = "gaussian"),
         title = "K I 404.41 and 404.72 nm", base_size = 8)
```

---

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,
  title = NULL,
  ...,
  color_by = c("type", "distance"),
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  point_size = 2.2,
  point_alpha = 0.85,
  colour_by = deprecated()
)
```

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`.
title	The plot title.
...	Further arguments passed to `ggplot2::geom_point()` to style the points, such as `stroke` (default 0.4) or `colour` (the outline, `"black"`).
color_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()`.
xlab, ylab	The axis titles. Default are the names of the distances.
base_size	The size of the text, in points. Default is 11.
point_size	The size of the points (default 2.2), or 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.
point_alpha	The opacity of the points, between 0 and 1 (default 0.85).
colour_by	**[Deprecated]** Use `color_by`.

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, ncomp = 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, color_by = "distance")

# a single limit at 97.5%, with the outlying regions shaded
influence_975 <- q_residuals(pca, ncomp = 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, and at the heads of molecular bands, to identify the lines and bands of the spectrum.

Usage

```
plot_lines(
  spectrum,
  lines = NULL,
  shift = 0,
  scale_markers = TRUE,
  interactive = TRUE,
  title = NULL,
  bands = NULL,
  wavelength = NULL,
  xlab = "Wavelength (nm)",
  ylab = "Intensity (arb. units)",
  base_size = 11,
  linewidth = 0.3,
  spectrum_color = "grey35",
  marker_linewidth = 0.5,
  marker_alpha = 0.7
)
```

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(), or NULL to draw only molecular bands.
shift	A wavelength shift, in nm, added to the line and band 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.
bands	An optional tibble of molecular band heads returned by molecular_bands().
wavelength	Optional (NULL by default). The wavelength range c(min, max) to draw, in nm. The markers are scaled to the maximum of the spectrum within the range, and the lines and bands outside it are left out.
xlab, ylab	The axis titles. Default are "Wavelength (nm)" and "Intensity (arb. units)".
base_size	The size of the text, in points. Default is 11. Ignored for interactive plots.
linewidth	The width of the spectrum line. Default is 0.3.
spectrum_color	The color of the spectrum. Default is "grey35".
marker_linewidth	The width of the line and band markers. Default is 0.5.
marker_alpha	The opacity of the line and band markers, between 0 and 1. Default is 0.7.

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.

The heads of molecular bands (`bands`, see `molecular_bands()`) are drawn as dashed markers, one color per molecule, scaled by their relative intensity within their system.

Value

A plotly or ggplot object.

See Also

`libs_lines()`, `molecular_bands()`, `line_finder()`

Examples

```
data(forageLIBS)
spectrum <- colMeans(forageLIBS[-(1:14)])
# 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, wavelength = c(403, 406), 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,
  wavelength = NULL,
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  linewidth = 0.3
)
```

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.
wavelength	Optional (NULL by default). The wavelength range c(min, max) to draw, in nm; the top largest peaks within it are labeled. Needs variables named by wavelength.
xlab, ylab	The axis titles. Default are "Wavelength (nm)" (or "Variable") and the name of the values drawn. In interactive plots, each panel has its own vertical axis title and ylab is ignored.
base_size	The size of the text, in points. Default is 11; the peak labels scale with it. Ignored for interactive plots.

`linewidth` The width of the bars (or, for interactive plots, of the line) drawn at each variable. Default is 0.3.

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 their `ncomp` components, they are shares of the variance of these 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)
# the Ca II H and K lines
plot_loadings(fit, spectra = spectra, wavelength = c(390, 400), top = 4)

# 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"),
                        timeout = 20))
if (!inherits(lines, "try-error")) plot_loadings(fit, lines = lines, spectra = spectra)
```

`plot_medcouple` *Plot of the Medcouple by Wavelength*

Description

Draws the medcouple (`medcouple()`), a robust measure of skewness, of the intensities at each wavelength of a set of spectra: as bars along the wavelength axis, or with `group` as a heatmap with one row per group of spectra, such as the shots of each sample.

Usage

```
plot_medcouple(
  x,
  group = NULL,
  drop_na = FALSE,
  color = c("#2166ac", "#b2182b"),
  interactive = FALSE,
  title = NULL,
  wavelength = NULL,
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  linewidth = 0.3
)
```

Arguments

`x`	A data frame, tibble or matrix with one spectrum per row and the spectral columns named by their wavelengths; the other columns are ignored.
`group`	An optional column of `x` (unquoted or as a string) that splits the spectra into groups, such as the sample of each shot: one row of the heatmap per group. Default is `NULL`, a single set of spectra.

drop_na	Logical value indicating whether to remove missing values (NA) from the intensities of each channel. If FALSE (default), a channel with missing values has no medcouple (a gap in the plot).
color	The colors of the scale from -1 to 1: two colors, for the negative and positive medcouples (through white at 0), or three or more spread evenly. The bars use the first and the last. Default is c("#2166ac", "#b2182b").
interactive	If TRUE, the plot is made with plotly, with the wavelength and the medcouple (and the group) on hover. Default is FALSE.
title	The plot title. Default is "Medcouple by wavelength". A long title is split into a title and a subtitle.
wavelength	Optional (NULL by default). The wavelength range c(min, max) to draw, in nm.
xlab, ylab	The axis titles. Default are "Wavelength (nm)" and "Medcouple", or the name of group for the heatmap.
base_size	The size of the text, in points. Default is 11. Ignored for interactive plots.
linewidth	The width of the bars (without group), or of the line of an interactive plot. Default is 0.3.

Details

The medcouple of a channel describes the shape of the distribution of its intensities over the spectra: 0 when it is symmetric, positive when a few spectra are far above the others (a longer upper tail), negative when they are far below. Channels that saturate the detector in many spectra are strongly negative, since their intensities pile up at the top of the range; the lines of an element whose contents are skewed, or that appears in a few spectra only, are positive.

Without group, the plot has one bar per channel, from 0 to its medcouple, in the first color of the scale if it is negative and in the last if it is positive. With group, the medcouple is computed within each group and the plot is a heatmap: one row per group (in the order of the factor levels, or of their first appearance in x), and one column per wavelength, each tile as wide as the spacing of the channels, with gaps between detectors (grey where there are no channels). The medcouple of a few spectra takes few distinct values and varies much from one group to another: groups of 20 spectra or more give a clearer picture.

The color scale is fixed from -1 to 1, so that a color means the same skewness in every row and in every plot. With two colors (default, blue and red), the scale is diverging, from the first color at -1 to the second at 1 through white at 0, so that the symmetric channels fade out and the skewed ones stand out. With three colors or more, they are spread evenly from -1 to 1, such as grDevices::hcl.colors(9, "Spectral") or grDevices::rainbow(7); use an odd number of colors to place the middle one at 0. A diverging palette shows the sign of the medcouple, its main information, better than a rainbow, whose bright yellow and cyan bands draw the eye to values that are not special.

Value

A ggplot object, or a plotly object if interactive = TRUE. The data of the ggplot holds the medcouple of each wavelength (and group).

Author(s)

Christian L. Goueguel

See Also

medcouple(), plot_spectra(), reject_shots()

Examples

```
data(forageLIBS)
# the saturated Ca II lines (393.37 and 396.85 nm) are strongly negative
plot_medcouple(forageLIBS, wavelength = c(380, 430))

# one row per third of the calcium contents: the Ca II lines saturate at
# every level
forage <- forageLIBS
forage$Calcium <- cut(forage$Ca, stats::quantile(forage$Ca, 0:3 / 3),
                      include.lowest = TRUE, labels = c("low", "medium", "high"))
plot_medcouple(forage, group = Calcium, wavelength = c(380, 430))

# any gradient from -1 to 1
plot_medcouple(forage, group = Calcium, wavelength = c(380, 430),
               color = grDevices::hcl.colors(9, "Spectral"))
```

---

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 shows them on each variable, or by their robust distance.

Usage

```
plot_outliers(
  x,
  quan = 1/2,
  alpha = 0.025,
  plot = TRUE,
  id = NULL,
  cutoff = c("adaptive", "quantile"),
  type = c("scores", "distance"),
  color_by = c("outlier", "distance", "both"),
  label_outliers = FALSE,
  ylab = NULL,
  title = NULL,
```

```
  caption = TRUE,
  base_size = 11,
  xlab = NULL,
  point_size = NULL,
  point_alpha = 1,
  show.outlier = deprecated(),
  show.mahal = deprecated()
)
```

Arguments

x	A matrix or data frame of numeric variables, with optionally the column id.
quan	A numeric value, between 0.5 and 1, the proportion of the observations used for the MCD estimates. Default is 0.5 (the largest breakdown point).
alpha	A numeric value between 0 and 1: the quantile $\chi^2_{p,1-\alpha}$ from which the adaptive cutoff is searched (or the cutoff itself with cutoff = "quantile"). Default is 0.025.
plot	A logical: plot (TRUE, default) or return the table of the samples.
id	Optional column of x (unquoted or as a string) identifying the samples, such as sample names: it names the outliers in the table and with label_outliers.
cutoff	"adaptive" (default), the adaptive cutoff of Filzmoser et al. (2005), or "quantile", the fixed $\chi^2_{p,1-\alpha}$ quantile.
type	"scores" (default), the robust z-scores of each variable, or "distance", the robust distance of each sample.
color_by	The colors of the points: "outlier" (default), the outliers in red and the others in grey; "distance", the robust distance; or "both", the distance, with the outliers as triangles.
label_outliers	A logical: label the outliers with their id, else their row number (FALSE, default). In the panels of robust z-scores, only the outliers beyond the univariate limits of the panel are labeled; the distance plot names them all.
title	The plot title. A long title is split into a title and a subtitle.
caption	TRUE (default), a caption saying how the outliers are flagged and how to read the plot; FALSE, no caption; or a caption of your own.
base_size	The size of the text, in points. Default is 11; use the text size of the journal (often 7 to 9 points) for a figure saved at its printed size.
xlab, ylab	The axis titles. Default are none (the samples are spread along the horizontal axis) and "Robust z-score", or "Sample (row)" and "Robust distance" with type = "distance".
point_size	The size of the points. Default is 1.4 at a base_size of 11, scaled with base_size.
point_alpha	The opacity of the points, between 0 and 1. Default is 1.
show.outlier, show.mahal	**[Deprecated]** Use color_by ("outlier", "distance" or "both"), and plot = FALSE for the table.

Details

The robust location and scatter of the data are estimated by the Minimum Covariance Determinant (MCD, Rousseeuw and Van Driessen, 1999), from which the robust Mahalanobis distance of each sample is computed. The outliers are the samples whose squared distance exceeds the **adaptive cutoff** of Filzmoser, Garrett and Reimann (2005): the tail of the distances is compared with the χ^2_p distribution beyond its $1 - \alpha$ quantile, and the cutoff is moved out to where they depart from it. When they do not depart from it, no sample is an outlier: unlike a fixed quantile, which flags about α of clean data, the adaptive cutoff flags (nearly) none. `cutoff = "quantile"` uses the fixed $\chi^2_{p,1-\alpha}$ quantile instead. The method follows the functions `arw()` and `aq.plot()` of the mvoutlier package (its `uni.plot()`, which this plot follows, flags the samples beyond the fixed quantile).

The adaptive cutoff exists only when more distances exceed the quantile than clean data would give (a proportion of about $0.24/\sqrt{n}$): a single outlier in a small sample, however far, is then not flagged. It stands out above the dashed quantile in the distance plot; use `cutoff = "quantile"` to flag it.

Robust z-scores (`type = "scores"`, the default). Each variable is drawn in its own panel, standardized by the adaptive reweighted location and scale (the mean and standard deviation of the samples that are not outliers). Each point is a sample, at the same horizontal position in every panel, so that a sample can be followed from one variable to the next. The outliers are flagged on all the variables jointly: they need not be extreme in any single variable, and a sample beyond the dashed univariate limits (± 2.5) need not be a multivariate outlier.

Robust distances (`type = "distance"`). The robust distance of each sample, in the order of the rows, with the cutoff (solid) and the $\chi^2_{p,1-\alpha}$ quantile (dashed).

Rows with missing values are left out, with a message.

Value

A `ggplot` object, or with `plot = FALSE` a tibble with one row per sample (without missing values): its `row` in `x`, its `id`, its robust Mahalanobis distance `mahalanobis`, the `cutoff` (on the same scale), `outlier`, its `weight` in the adaptive reweighted estimates (0 for the samples at or beyond the adaptive cutoff, 1 for the others), and its robust z-score on each variable.

Author(s)

Christian L. Goueguel

References

- Filzmoser, P., Garrett, R. G., Reimann, C. (2005). Multivariate outlier detection in exploration geochemistry. Computers & Geosciences, 31(5):579-587
- Rousseeuw, P. J., Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator. Technometrics, 41(3):212-223

See Also

`adjusted_boxplot()` and `generalized_boxplot()` for univariate outliers.

Examples

```
data(forageLIBS)
# mineral contents (%) of the forage samples
contents <- forageLIBS[c("Measurement", "Ca", "Mg", "P", "K")]
plot_outliers(contents, id = Measurement)

# the robust distance of each sample, the outliers named
plot_outliers(contents, id = Measurement, type = "distance", label_outliers = TRUE)

# colored by robust distance, the outliers as triangles
plot_outliers(contents, id = Measurement, color_by = "both")

# the table of the samples
res <- plot_outliers(contents, id = Measurement, plot = FALSE)
res[res$outlier, ]
```

---

plot_outlier_map *Outlier Map of a Robust PCA or Robust PLS Model*

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. For a robust calibration model fitted by `rsimpls()` or `rpcr()`, it is by default the regression outlier map of Hubert and Vanden Branden (2003): the residual distance (with one response, the absolute standardized residual) against the score distance. With `map = "score"`, it is the score outlier map of the predictors: the orthogonal distance of each spectrum to the model against its score distance, as for a robust PCA.

Usage

```
plot_outlier_map(
  object,
  newdata = NULL,
  labels = 3,
  relative = FALSE,
  shade = FALSE,
  log = FALSE,
  title = NULL,
  map = c("regression", "score"),
  ...,
  color_by = c("type", "distance"),
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  point_size = 2.2,
  point_alpha = 0.85,
```

```
  colour_by = deprecated()
)
```

Arguments

object	An object returned by robpca(), rospca(), macropca(), cellpca(), rsimpls() or rpcr().
newdata	Optional new observations to add to the map (a numeric matrix or data frame with the calibration variables). Not available for the regression outlier map of rsimpls() and rpcr() fits.
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.
title	The plot title.
map	For rsimpls() and rpcr() fits, the map to draw: "regression" (default), the regression outlier map, or "score", the score outlier map of the predictors. Not used for robust PCA fits.
...	Further arguments passed to ggplot2::geom_point() to style the points, such as stroke (default 0.4) or colour (the outline, "black").
color_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.
xlab, ylab	The axis titles. Default are the names of the distances.
base_size	The size of the text, in points. Default is 11.
point_size	The size of the points (default 2.2), or 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.
point_alpha	The opacity of the points, between 0 and 1 (default 0.85).
colour_by	**[Deprecated]** Use color_by.

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.

In the regression outlier map of `rsimpls()` or `rpcr()`, the top left region holds the **vertical outliers**: observations with a typical spectrum but a large residual, such as a wrong reference value. Bad leverage points have both an outlying spectrum and a large residual; good leverage points have an outlying spectrum that the model still fits.

New observations (`newdata`) are projected onto the model with predict() (or predict() for the score outlier map of an `rsimpls()` or `rpcr()` fit) 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()`, `rsimpls()`, `rpcr()`, `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, color_by = "distance", point_alpha = 0.6)
```

plot_response *Plot a Spectral Response*

Description

Plots the relative spectral response measured by response_calibration(), with a band of plus or minus one standard uncertainty.

Usage

```
plot_response(
  object,
  title = NULL,
  wavelength = NULL,
  xlab = "Wavelength (nm)",
  ylab = "Relative response",
  base_size = 11,
  linewidth = 0.6
)
```

Arguments

object	The result of response_calibration().
title	The plot title. By default, the number of lamp spectra and the median uncertainty.
wavelength	Optional (NULL by default). The wavelength range c(min, max) to draw, in nm.
xlab, ylab	The axis titles. Default are "Wavelength (nm)" and "Relative response".
base_size	The size of the text, in points. Default is 11.
linewidth	The width of the response line. Default is 0.6.

Value

A ggplot object.

See Also

response_calibration()

Examples

```
planck <- function(wl, t) 1 / (wl^5 * (exp(1.4388e7 / (wl * t)) - 1))
reference <- data.frame(wavelength = seq(250, 850, by = 10))
reference$value <- planck(reference$wavelength, 3100)
reference$rsd <- 0.03
wl <- seq(260, 840, by = 0.1)
set.seed(1)
lamp <- 2e4 * exp(-((wl - 500) / 250)^2) * planck(wl, 3100) / planck(500, 3100) +
```

```
  rnorm(length(wl), sd = 20)
names(lamp) <- wl
plot_response(response_calibration(lamp, reference))
```

---

plot_shots *Plots of the Laser Shots and of their Rejection*

Description

Shows the shots of a LIBS data set and the shots rejected by `reject_shots()`: their criteria, their spectra, the trends along the shots, and the samples, to judge the rejection before averaging the shots.

Usage

```
plot_shots(
  x,
  type = c("heatmap", "criteria", "spectra", "order", "samples"),
  sample = NULL,
  wavelength = NULL,
  label = 10,
  acquisition = NULL,
  arrange = c("data", "extreme"),
  smooth = NULL,
  plot = TRUE,
  title = NULL,
  caption = TRUE,
  base_size = 11,
  xlab = NULL,
  ylab = NULL,
  point_size = NULL,
  point_alpha = 0.85,
  linewidth = 0.3
)
```

Arguments

x	The result of `reject_shots()` (with `drop = FALSE`).
type	The view: `"heatmap"` (default), `"criteria"`, `"spectra"`, `"order"` or `"samples"` (see Details).
sample	For `type = "spectra"`, the sample to draw: a value of the sample column. Default is the sample with the most rejected shots.
wavelength	For `type = "spectra"`, an optional wavelength range `c(min, max)` (nm) to zoom on.
label	For `type = "criteria"`, the number of rejected shots labeled, the most extreme first. Default is 10; 0 for none.

acquisition	For type = "heatmap", an optional column of x (unquoted or as a string) giving the order of the measurements, such as the measurement number or the acquisition time: the rows follow it.
arrange	For type = "heatmap" without acquisition, the order of the rows: "data" (default), that of the samples in x, or "extreme", the samples with the most extreme shots first.
smooth	For type = "heatmap", an odd number k of consecutive measurements: each cell is the running median of the z-scores of that shot number over them. Needs acquisition. Default is NULL, no smoothing.
plot	A logical: plot (TRUE, default) or return the table of the view (the shots with their z-scores, smoothed or not, the shots of the sample, the shot numbers or the samples).
title	The plot title. A long title is split into a title and a subtitle.
caption	TRUE (default), a caption saying how to read the plot, with the number of rejected shots; FALSE, no caption; or a caption of your own.
base_size	The size of the text, in points. Default is 11.
xlab, ylab	For type = "criteria" and "spectra", the axis titles. Default are the names of the criteria, or the wavelength and intensity.
point_size	For type = "criteria" and "samples", the size of the points. Default is 1.6 at a base_size of 11, scaled with base_size.
point_alpha	For type = "criteria" and "samples", the opacity of the points, between 0 and 1. Default is 0.85.
linewidth	For type = "spectra", the width of the kept shots; the median and the rejected shots are drawn 5/3 as wide. Default is 0.3.

Details

Five views of the result of reject_shots():

- "heatmap" (default): a map of the samples (rows) by the shot number (columns), one panel per criterion, colored by the robust z-score of each shot: white below 1 (the ordinary variability of the shots), then in steps up to the cutoff and beyond it, the rejected shots outlined. It shows at a glance the rejected shots, the samples whose shots vary (rows), consecutive rejected shots (a problem of focus or surface rather than a misfire), and the trends along the shot number (columns). With acquisition, the rows follow the order of the measurements; arrange = "extreme" puts the samples with the most extreme shots first. With smooth = k (and acquisition), each cell is the running median of the z-scores of that shot number over k consecutive measurements, the rejected shots as points: the drift of the shots along the acquisition (laser energy, fouling of the optics), when the trend along the shot number changes from one period to another. Without the acquisition order, neighboring rows are unrelated samples, and smoothing them would invent patterns.
- "criteria": the robust z-scores of the shots (total intensity against the dissimilarity of shape, or the criteria used), with the cutoff: the rejected shots are colored by their criteria, and the most extreme labeled (sample and shot). Shots just beyond the cutoff are borderline; a gap between the rejected shots and the others shows clear outliers.

- `"spectra"`: the shots of one sample (`sample`, by default the sample with the most rejected shots): the kept shots in grey, the median spectrum in black, and each rejected shot in color, with its criteria, its total intensity relative to the median shot and its correlation with the median spectrum. `wavelength` zooms on a range.
- `"order"`: along the shot number, the total intensity of the shots relative to their sample, their dissimilarity of shape to the median spectrum of their sample ($1 - r$, on a log scale) and the share of rejected shots. A trend shows the first shots on the surface of the sample (cleaning shots) or the drift of the ablation in its crater, which the rejection, made within each sample, does not correct: discard the first shots, or check that the trend is the same for all samples.
- `"samples"`: the number of samples by their number of rejected shots, and the relative standard deviation (RSD) of the total intensity of the shots of each sample, with all its shots and with the kept ones: the gain of precision of the rejection, and the samples whose shots vary most (heterogeneous samples, or a poor laser focus).

The shot numbers are the `.shot` column of `reject_shots()` (its `shot` argument, or the order of the rows in each sample).

Value

A ggplot (or patchwork) object, or with `plot = FALSE` a tibble.

Author(s)

Christian L. Goueguel

See Also

`reject_shots()`, forageShots

Examples

```
data(forageShots)
res <- reject_shots(forageShots, Measurement, shot = shot)
if (rlang::is_installed("patchwork")) {
  # the samples by the shot number, in the order of the measurements
  plot_shots(res, acquisition = Measurement)
  # the drift of the shots along the acquisition: running medians over 5
  # consecutive measurements
  plot_shots(res, acquisition = Measurement, smooth = 5)
}
# the criteria of the shots, the rejected ones labeled
plot_shots(res, type = "criteria")
# the shots of the sample with the most rejected shots
plot_shots(res, type = "spectra")
# a zoom on the K I lines
plot_shots(res, type = "spectra", sample = 121306, wavelength = c(764, 772))

if (rlang::is_installed("patchwork")) {
  # the trends along the shots, and the precision of the samples
  plot_shots(res, type = "order")
  plot_shots(res, type = "samples")
```

```
}

# the table of the samples
plot_shots(res, type = "samples", plot = 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,
  color_by = NULL,
  newdata = NULL,
  units = NULL,
  title = NULL,
  wavelength = NULL,
  xlab = NULL,
  ylab = NULL,
  base_size = 11,
  point_size = 1.6,
  point_alpha = 0.8,
  linewidth = 0.4,
  colour = deprecated()
)
```

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.
`color_by`	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.

wavelength	For "prototypes": the wavelength range c(min, max) to draw, in nm. Default is all the variables.
xlab, ylab	For "prototypes": the axis titles. Default are "Wavelength (nm)" (or "Variable") and "Prototype". The maps have no axes.
base_size	The size of the text, in points. Default is 11.
point_size, point_alpha	For "mapping": the size (default 1.6) and opacity (default 0.8) of the points of the training spectra.
linewidth	For "prototypes": the width of the prototype spectra. Default is 0.4.
colour	**[Deprecated]** Use color_by.

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 color_by, 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", color_by = forageLIBS$K)
plot_som(fit, type = "prototypes", wavelength = c(760, 780))
```

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). The spectra can be split into panels by a grouping variable, summarized by their mean or median, and labeled with the emission lines they show.

Usage

```
plot_spectra(
  x,
  id = NULL,
  colvar = NULL,
  interactive = FALSE,
  drop_na = FALSE,
  offset = NULL,
  wavelength = NULL,
  transform = NULL,
  panel = NULL,
  layout = c("vertical", "horizontal"),
  grid = FALSE,
  color_as = c("line", "fill"),
  legend = c("none", "right", "top", "bottom", "inside"),
  legend_title = NULL,
  xlab = "Wavelength (nm)",
  ylab = "Intensity (arb. units)",
  title = NULL,
  lines = NULL,
  palette = "viridis",
  base_size = 11,
  linewidth = 0.5,
  scales = c("fixed", "free_y"),
  panel_tags = FALSE,
  label_spectra = FALSE,
  summary = c("none", "mean", "median"),
  rasterize = FALSE,
  .interactive = deprecated()
)
```

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` A logical: an interactive plotly plot (`TRUE`) or a ggplot (`FALSE`, default).

`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. With `panel`, the rows are counted in each panel, so that the first spectrum of every panel stays in place. With `summary`, the summary spectra are offset.

`wavelength` Optional (`NULL` by default). The wavelength range `c(min, max)` to draw, in nm, or a list of ranges, drawn side by side, such as `list("Ca II H and K" = c(392, 398), "K I" = c(765, 771))` (see Details).

`transform` Optional (`NULL` by default). A function applied to the spectra before they are drawn, such as `snv` or `\(s) baseline_als(s, lambda = 1e5)`, a formula such as `~ savitzky_golay(.x, derivative = 1)`, or a list of them, applied in order (see Details).

`panel` Optional (`NULL` by default). Column name of a grouping variable: the spectra of each group are drawn in a separate panel.

`layout` The arrangement of the panels: `"vertical"` (default), one above the other, or `"horizontal"`, side by side. Ignored without `panel`, and with several wavelength ranges.

`grid` A logical: draw grid lines at the major breaks of the axes (`FALSE`, default).

`color_as` How the colors of `colvar` (or of `id`, without `colvar`) are shown: as the color of the lines (`"line"`, default), or as the fill of the area under each spectrum (`"fill"`).

`legend` The position of the legend of `colvar` (a color bar) or `id`: `"none"` (default), `"right"`, `"top"`, `"bottom"`, or `"inside"` (the top right corner of the plot).

`legend_title` The title of the legend. Default is the name of `colvar` (or `id`); give it with units, such as `"K (%)"`.

`xlab, ylab` The axis titles. Default are `"Wavelength (nm)"` and `"Intensity (arb. units)"`.

`title` The plot title. A long title is split into a title and a subtitle.

`lines` Optional table of emission lines, such as returned by `libs_lines()`: a data frame with the columns `species` (such as `"K I"`) and `wavelength` (nm). Each line within the wavelength range is marked and labeled (see Details).

`palette` The colors of `colvar` or `id`: the name of a viridis palette (`"viridis"`, default, `"magma"`, `"inferno"`, `"plasma"`, `"cividis"`, `"rocket"`, `"mako"` or `"turbo"`), all readable by colorblind readers and in grayscale except `"turbo"`, or a vector of at least two colors, interpolated for `colvar` and for the levels of `id` when there are fewer colors than levels.

base_size	The size of the text, in points. Default is 11; use the text size of the journal (often 7 to 9 points) for a figure saved at its printed size. The axis lines and labels scale with it.
linewidth	The width of the spectra lines. Default is 0.5; about 0.3 suits small figures.
scales	"fixed" (default), the same intensity axis for all panels, or "free_y", an intensity axis fitted to each panel.
panel_tags	A logical: tag the panels (a), (b), (c), ... before their labels (FALSE, default): the groups of panel, else the wavelength ranges.
label_spectra	A logical: label each spectrum at its right end (FALSE, default), with its id, else its value of colvar, else its row number (or "Mean"/"Median" for summaries). Most useful for spectra stacked by offset.
summary	"none" (default), each spectrum drawn; "mean", the mean spectrum of each group with a band of plus or minus one standard deviation; or "median", the median spectrum of each group with a band of the first to third quartiles (see Details).
rasterize	FALSE (default), TRUE, or a resolution in dpi: draw the spectra as an image, of 300 dpi with TRUE, within an otherwise vector plot (see Details). Ignored for interactive plots.
.interactive	**[Deprecated]** Use interactive.

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, colvar and panel must be named by their wavelength.

Filled spectra. With color_as = "fill", the area between each spectrum and its baseline (zero intensity, shifted by offset) is filled, with the spectrum drawn as a thin dark line. Overlaid spectra are filled with translucent colors. Stacked spectra (with a vertical offset) are filled with opaque colors and drawn from the back to the front (from the top to the bottom of the stack), so that the front spectra hide the back ones, as in a waterfall plot.

Detectors. The spectra are not drawn across the gaps between detectors (wavelength steps of more than 5 times the channel spacing), and the channels of overlapping detectors (where the wavelengths step back in column order) are drawn as separate lines, not joined in a zigzag.

Summaries. With summary = "mean", each group of spectra is drawn as its mean spectrum, in a band of plus or minus one standard deviation; with summary = "median", as its median spectrum, in a band from the first to the third quartile. The groups are the panels and, within them, the values of id; the color of colvar is then the mean of colvar in each group. The band is not drawn with color_as = "fill".

Transformations. transform receives the spectral columns of x (a data frame of the columns named by their wavelength, without id, colvar and panel) and returns the transformed spectra, one row per spectrum: a data frame or a matrix with columns named by their wavelength, or a list holding them as correction, as returned by snv(), msc(), baseline_als() and the other baseline corrections. The number of channels may change, as with bin_spectra(). The whole spectra are transformed, before wavelength selects the channels to draw, so that a normalization or a baseline correction uses all the channels; the summaries, offsets and labels are those of the transformed spectra. Give ylab for the units of the transformed intensities.

Wavelength windows. With a list of ranges in `wavelength`, each range is drawn in its own panel, side by side and of the same width, with its own wavelength axis: a zoom on several groups of lines at once. The ranges must not overlap; they are drawn in increasing wavelength order, under their names (if any). With `scales = "free_y"`, each range has its own intensity axis, so that weak lines stay readable next to strong ones. With `panel`, the groups are drawn in rows and the ranges in columns (`layout` is ignored), and `scales = "free_y"` fits the intensity axis to each row.

Emission lines. Each line of `lines` within the wavelength range is marked by a dashed vertical line, labeled with its species and wavelength above the plot (above the top panels, or above each row of panels with several wavelength ranges). Lines closer than 2% of the wavelength range (of their window) share a label, such as "Ca II 393.37, 396.85". The markers are at the tabulated wavelengths: with a horizontal `offset`, they match the first spectrum of each panel. Over a wide range, keep the strongest lines, for example with the `top` or `min_relative` arguments of `libs_lines()`, or labels become long.

Publication figures. Journals print figures about 85 mm wide for a single column and 175 mm for a double column, with text of 7 to 9 points. Make the plot at its printed size, with `base_size` set to the text size, and save it with `ggplot2::ggsave()`, in a vector format (PDF or EPS) or as a TIFF of 600 dpi, for example `ggsave("figure.pdf", p, width = 85, height = 65, units = "mm")`. Many long spectra make large and slow vector files: `rasterize = TRUE` draws the spectra as an image within the plot, keeping the text, axes and markers as vectors (this needs the ragg package).

Value

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

Author(s)

Christian L. Goueguel

See Also

`libs_lines()` for the emission lines.

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, legend = "right",
             legend_title = "K (%)")

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

# one panel per potassium level, side by side, with the area under the
# spectra filled by potassium content and grid lines
spectra <- forageLIBS[1:30, c("K", k_lines)]
```

```
spectra$level <- cut(spectra$K, 3, labels = c("Low K", "Medium K", "High K"))
plot_spectra(spectra, colvar = K, panel = level, layout = "horizontal",
             offset = c(0.2, 8000), color_as = "fill", grid = TRUE)

# a figure for a journal column: the mean spectrum of each potassium level,
# with the K I lines labeled (from libs_lines() in practice)
lines <- data.frame(species = "K I", wavelength = c(766.49, 769.90))
p <- plot_spectra(spectra, colvar = K, panel = level, summary = "mean",
                  lines = lines, panel_tags = TRUE, legend = "right",
                  legend_title = "K (%)", ylab = "Intensity (counts)",
                  base_size = 8, linewidth = 0.3, wavelength = c(765, 771))
p
# saved at that size (here in the temporary directory)
ggplot2::ggsave(file.path(tempdir(), "figure.pdf"), p, width = 85, height = 90,
                units = "mm")

# the spectra normalized by SNV over all the channels, then drawn in two
# windows side by side, each with its own intensity axis
channels <- names(forageLIBS)[-(1:14)]
plot_spectra(forageLIBS[1:20, c("K", channels)], colvar = K, transform = snv,
             wavelength = list("Ca II H and K" = c(392, 398), "K I" = c(765, 771)),
             scales = "free_y", ylab = "SNV intensity")

# several transformations, in order: a baseline correction, then the first
# derivative
plot_spectra(forageLIBS[1:20, c("K", k_lines)], colvar = K,
             transform = list(\(s) baseline_als(s, lambda = 1e4),
                              ~ savitzky_golay(.x, window = 7, derivative = 1)),
             ylab = "First derivative")
```

`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,
  xlab = "Measured wavelength (nm)",
  ylab = "Reference - measured (nm)",
  base_size = 11,
  point_size = 2.5,
```

```
  point_alpha = 1
)
```

Arguments

object	The result of wavelength_calibration().
title	The plot title.
xlab, ylab	The axis titles. Default are "Measured wavelength (nm)" and "Reference - measured (nm)".
base_size	The size of the text, in points. Default is 11; the line labels scale with it.
point_size	The size of the points of the lines. Default is 2.5.
point_alpha	The opacity of the points, between 0 and 1. Default is 1.

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)
```

---

plot_wavelength_selection

Plot of a Wavelength Selection

Description

Shows the variables selected by select_wavelengths(): the mean spectrum, with the selected regions shaded, above the importance of each variable (VIP or selectivity ratio, with the threshold when one was used) or, for iPLS, the RMSECV of each interval alone, with the selected intervals colored.

Usage

```
plot_wavelength_selection(
  object,
  title = NULL,
  wavelength = NULL,
  xlab = NULL,
  base_size = 11,
  linewidth = 0.3
)
```

Arguments

object	An object returned by select_wavelengths().
title	The plot title.
wavelength	Optional (NULL by default). The wavelength range c(min, max) to draw, in nm. Needs variables named by wavelength.
xlab	The title of the wavelength axis. Default is "Wavelength (nm)" (or "Variable"). The panels are named on their left.
base_size	The size of the text, in points. Default is 11.
linewidth	The width of the mean spectrum and importance lines. Default is 0.3.

Value

A ggplot object.

Author(s)

Christian L. Goueguel

See Also

select_wavelengths()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
sel <- select_wavelengths(spectra, forageLIBS$Ca, method = "sr", num_terms = 50)
plot_wavelength_selection(sel)
# the Ca II H and K lines
plot_wavelength_selection(sel, wavelength = c(390, 400))
```

`pls_rsimpls` *Robust PLS Regression in tidymodels (parsnip Engine)*

Description

specProc adds the `"rsimpls"` engine to `parsnip::pls()`, which fits the robust PLS regression of `rsimpls()`: outlying spectra and wrong reference values have little influence on the model. It can be tuned, resampled and combined with recipes like any parsnip model.

Details

The engine is available once parsnip and specProc are both loaded, in either order. It is for regression only, with one or several outcomes (for example `cbind(y1, y2) ~ .`).

Tuning parameters

- `num_comp` (`ncomp` of `rsimpls()`): the number of PLS components. It has no default and must be given (or tuned). It can be tuned with `tune::tune()`, using `dials::num_comp()`.
- `predictor_prop`, the proportion of predictors of sparse PLS, is not used by this engine.

The models with 1 to `ncomp_max` components all come from the same ROBPCA fit (see `rsimpls()`), so `parsnip::multi_predict()` gives the predictions of all of them from one fit, and `tune::tune_grid()` fits a single model per resample for a grid of `num_comp`. The predictions of a model with fewer components are those of a separate fit with the same `ncomp_max`, as long as `ncomp_max` is at least the largest `num_comp`.

Engine arguments

The other arguments of `rsimpls()` are set with `parsnip::set_engine()`: `ncomp_max`, the largest number of components (default 10), `alpha`, the fraction of observations assumed to be regular (default 0.75), and `ndir` and `nsamp` of the ROBPCA step.

Preprocessing and randomness

The predictors are centered robustly by the model, and are not scaled: scale them beforehand if their units differ. Factor predictors are converted to indicator variables. `rsimpls()` uses random directions and subsets, so use `set.seed()` before fitting for reproducible results.

Outlier diagnostics

The fitted `rsimpls()` model is returned by `parsnip::extract_fit_engine()`, for its outlier maps (`plot_outlier_map()`) and distances.

Author(s)

Christian L. Goueguel

See Also

rsimpls(), step_rsimpls(), parsnip::pls()

Examples

```
library(parsnip)
data(forageLIBS)
# calcium and the Ca II and Ca I lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "Ca"), which(wl > 380 & wl < 430))]
spec <- pls(num_comp = 4) |>
  set_engine("rsimpls") |>
  set_mode("regression")
set.seed(1)
fit <- fit(spec, Ca ~ ., data = dat[1:300, ])
predict(fit, dat[301:368, ])
# predictions with 1 to 4 components
multi_predict(fit, dat[301:303, ], num_comp = 1:4)$.pred[[1]]
plot_outlier_map(extract_fit_engine(fit))
```

---

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(),
  drop.na = deprecated()
)
```

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.
drop.na	**[Deprecated]** Use drop_na.

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), whether it is below the limit of detection (below_lod) or of quantification (below_loq), whether it lies outside the range of the standards (outside_range, an extrapolation), and, with the ISO 11843 limits, whether the element is detected (above the critical value). 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_domain

Flag Spectra Outside the Applicability Domain

Description

Computes Hotelling's T^2 and the Q residual of new spectra in the PCA model of applicability_domain(), and flags those outside the domain.

Usage

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

Arguments

object	The result of applicability_domain().
newdata	The new spectra: a numeric matrix or data frame with the variables of the calibration spectra.
...	Not used.

Value

A tibble of class specproc_influence, as returned by q_residuals(), with one row per spectrum: sample, t2 and its limit, q and its limit, outlier ("regular", "extreme" for a high T^2, "residual" for a high Q, or "both"), and inside, whether the spectrum lies inside the domain. Draw it with plot_influence().

See Also

applicability_domain()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- as.matrix(forageLIBS[which(wl > 380 & wl < 430)])
domain <- applicability_domain(spectra[1:300, ])
flags <- predict(domain, spectra[301:368, ])
flags[!flags$inside, ]
```

---

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_robust_da

Predictions of a Robust Discriminant Analysis

Description

Predicts the classes, or the posterior probabilities of the classes, of new observations with a model fitted by `robust_da()`.

Usage

```
## S3 method for class 'specproc_robust_da'
predict(object, newdata, type = c("class", "prob"), ...)
```

Arguments

`object`	An object returned by `robust_da()`.
`newdata`	A numeric matrix or data frame with the same variables as the training data.
`type`	`"class"` (default) for the predicted classes, or `"prob"` for the posterior probabilities.
`...`	Not used.

Value

With `type = "class"`, a factor of the predicted classes. With `type = "prob"`, a tibble with one column of posterior probabilities per class.

Author(s)

Christian L. Goueguel

See Also

robust_da()

Examples

```
# iris: train on 100 flowers, predict the 50 others
set.seed(1)
train <- sample(nrow(iris), 100)
fit <- robust_da(iris[train, 1:4], iris$Species[train], method = "quadratic")
head(predict(fit, iris[-train, 1:4]))
head(predict(fit, iris[-train, 1:4], type = "prob"))
```

---

predict.specproc_rpcr *Predictions of a Robust PCR Model*

Description

Predicts the responses of new observations with a model fitted by rpcr(), or computes their scores.

Usage

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

Arguments

object	An object returned by rpcr().
newdata	A numeric matrix or data frame with the same variables as the calibration data.
type	"response" (default) for the predicted responses, or "scores" for the scores and their score and orthogonal distances.
ncomp	With type = "response", the number of components of the predictions, from 1 to the ncomp_max of the model. Default is the ncomp of the model. The models with fewer or more components use the same robust PCA (see rpcr()).
...	Not used.

Value

With type = "response", a numeric vector of predictions (one response) or a matrix with one column per response. With type = "scores", a tibble with the scores (Comp1, ...), the score distance sd and the orthogonal distance od of each observation.

Author(s)

Christian L. Goueguel

See Also

rpcr()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
fit <- rpcr(spectra[1:300, ], forageLIBS$Ca[1:300], ncomp = 5)
head(predict(fit, spectra[301:368, ]))
head(predict(fit, spectra[301:368, ], type = "scores"))
```

---

predict.specproc_rsimca

Predictions of a Robust SIMCA Model

Description

Predicts the classes of new observations with a model fitted by rsimca(), or their distances to the classes.

Usage

```
## S3 method for class 'specproc_rsimca'
predict(object, newdata, type = c("class", "distances"), ...)
```

Arguments

object	An object returned by rsimca().
newdata	A numeric matrix or data frame with the same variables as the training data.
type	"class" (default) for the predicted classes, or "distances" for the combined distances to the classes.
...	Not used.

Value

With type = "class", a factor of the predicted classes. With type = "distances", a tibble with the combined distance of each observation to each class (one column per class), and outlying, TRUE for the observations outlying for every class.

Author(s)

Christian L. Goueguel

See Also

rsimca()

Examples

```
# iris: train on 100 flowers, predict the 50 others
set.seed(1)
train <- sample(nrow(iris), 100)
fit <- rsimca(iris[train, 1:4], iris$Species[train], ncomp = 2)
head(predict(fit, iris[-train, 1:4]))
head(predict(fit, iris[-train, 1:4], type = "distances"))
```

---

predict.specproc_rsimpls

Predictions of a Robust PLS Model

Description

Predicts the responses of new observations with a model fitted by rsimpls(), or computes their scores.

Usage

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

Arguments

object	An object returned by rsimpls().
newdata	A numeric matrix or data frame with the same variables as the calibration data.
type	"response" (default) for the predicted responses, or "scores" for the scores and their score and orthogonal distances.
ncomp	With type = "response", the number of components of the predictions, from 1 to the ncomp_max of the model. Default is the ncomp of the model. The models with fewer or more components come from the same ROBPCA fit (see rsimpls()).
...	Not used.

Value

With type = "response", a numeric vector of predictions (one response) or a matrix with one column per response. With type = "scores", a tibble with the scores (Comp1, ...), the score distance sd and the orthogonal distance od of each observation.

Author(s)

Christian L. Goueguel

See Also

rsimpls()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
fit <- rsimpls(spectra[1:300, ], forageLIBS$Ca[1:300], ncomp = 4)
head(predict(fit, spectra[301:368, ]))
head(predict(fit, spectra[301:368, ], ncomp = 2))
head(predict(fit, spectra[301:368, ], type = "scores"))
```

---

predict.specproc_slope_bias

Correct Predictions for Slope and Bias

Description

Applies the correction fitted by slope_bias() to new predictions.

Usage

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

Arguments

object	The result of slope_bias().
newdata	Predictions of the model for new secondary spectra.
...	Not used.

Value

The corrected predictions.

See Also

slope_bias()

Examples

```
set.seed(1)
reference <- runif(20, 0.5, 4)
predicted <- 0.2 + 1.1 * reference + rnorm(20, sd = 0.05)
predict(slope_bias(predicted, reference), c(1.3, 2.4, 3.5))
```

`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_transfer` *Transfer Spectra to Another Instrument*

Description

Transfers spectra measured on the secondary instrument to the primary instrument, with a transfer fitted by `ds()`, `sst()` or `pds()`.

Usage

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

Arguments

`object`	A transfer returned by `ds()`, `sst()` or `pds()`.
`newdata`	Spectra of the secondary instrument: a numeric matrix or data frame with the wavelengths of the transfer spectra.
`...`	Not used.

Value

The transferred spectra, a matrix (or a tibble for a data frame) with the wavelengths of the primary spectra.

See Also

`ds()`, `sst()`, `pds()`

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- as.matrix(forageLIBS[which(wl > 760 & wl < 780)])
secondary <- 0.8 * cbind(spectra[, -1], spectra[, ncol(spectra)]) + 50
transfer <- pds(spectra[1:30, ], secondary[1:30, ], win = 3)
transferred <- predict(transfer, secondary[31:40, ])
```

`predict.specproc_transformation` *Apply a Robust Transformation to New Data, or Back-Transform*

Description

Transforms new data with the parameters (and, with `standardize = TRUE`, the mean and standard deviation of the inliers) estimated by `robust_boxcox()`, `robust_yeojohnson()` or `robust_gandh()`. With `inverse = TRUE`, it transforms values back to the original scale, such as the predictions of a model fitted to the transformed variables.

Usage

```
## S3 method for class 'specproc_transformation'
predict(
  object,
  newdata,
  inverse = FALSE,
  estimate = c("median", "mean"),
  residuals = NULL,
  ...
)
```

Arguments

`object`	An object returned by `robust_boxcox()` or `robust_yeojohnson()` or `robust_gandh()`.
`newdata`	A numeric vector, matrix or data frame with some or all of the variables of the fit: new data, or with `inverse = TRUE`, values on the transformed scale.
`inverse`	A logical value: transform the values back to the original scale (`TRUE`), or transform new data (`FALSE`, default).
`estimate`	With `inverse = TRUE`, the estimate on the original scale: `"median"` (default) or `"mean"`, which needs `residuals`.
`residuals`	With `estimate = "mean"`, the residuals of the model on the transformed scale: a numeric vector, used for every variable of `newdata`, or a matrix or data frame with a column for each of them, matched by name (without column names, in the order of the columns of `newdata`). Missing values are ignored.
`...`	Not used.

Details

`newdata` can hold any of the variables of the fit, matched by name. A matrix or vector without column names must have all of them, in their order.

The back-transformation undoes, in reverse order, the standardization, the transformation and the pre-standardization of each variable. When a model predicts $\hat{z}$ on the transformed scale with symmetric errors, the back-transformed value $g^{-1}(\hat{z})$ estimates the median of the variable on the original scale (`estimate = "median"`, the default). It needs no other assumption, and suits the prediction of one sample.

For the power transformations, on the original scale the mean is higher than the median when $\lambda < 1$ (the transformation compresses the upper tail), and lower when $\lambda > 1$. `estimate = "mean"` gives the smearing estimate of Duan (1983),

$$\hat{x} = \frac{1}{n} \sum_{k=1}^{n} g^{-1}(\hat{z} + e_k),$$

where $e_1, \ldots, e_n$ are the `residuals` of the model on the transformed scale (preferably cross-validated, since the residuals of the fit underestimate the errors of prediction). It assumes no distribution of the errors, only that they do not depend on $\hat{z}$. The mean matters when the predictions are averaged or summed, such as for the mean content of a lot or a mass balance. It is not robust: an outlying residual pulls it, so pass the residuals of the inliers of a robust model (for example, of the spectra that `rsimpls()` does not flag). Smearing is also available for `robust_gandh()`; the direction of the mean-median difference depends on its fitted shape and residuals.

The back-transformation is not defined everywhere, because some transformations are bounded: the Box-Cox transformation above for $\lambda < 0$, and the Yeo-Johnson transformation above for $\lambda < 0$ and below for $\lambda > 2$. Values beyond the bound (with `estimate = "mean"`, values that pass it with one of the residuals) become `NA`, with a warning.

Value

A tibble with the variables of `newdata`, in the order of the fit. Missing values stay missing, and with the Box-Cox transformation, new values that are zero or negative become `NA`.

References

- Duan, N. (1983). Smearing estimate: a nonparametric retransformation method. Journal of the American Statistical Association, 78(383):605-610.

See Also

`robust_boxcox()`, `robust_yeojohnson()`, `robust_gandh()`

Examples

```
data(forageLIBS)
contents <- forageLIBS[c("Ca", "K", "P", "Mn")]
fit <- robust_boxcox(contents[1:300, ])
new <- predict(fit, contents[301:368, ])
# back to mg/kg
head(predict(fit, new, inverse = TRUE))
# Box-Cox is not defined for zero or negative values
x <- contents[301:302, ]
x$Ca[2] <- 0
predict(fit, x)

# a calibration of the transformed Mn contents on the Mn II line at
# 257.61 nm, back to mg/kg
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
line <- forageLIBS[[which.min(abs(wl - 257.61))]]
```

```
fit <- robust_boxcox(forageLIBS[1:300, "Mn", drop = FALSE])
model <- lm(Mn ~ log(line), data = data.frame(Mn = fit$transformation$Mn,
                                              line = line[1:300]))
pred <- data.frame(Mn = predict(model, data.frame(line = line[301:368])))
med <- predict(fit, pred, inverse = TRUE)
avg <- predict(fit, pred, inverse = TRUE, estimate = "mean",
               residuals = residuals(model))
summary(avg$Mn / med$Mn)
# the mean content of the 68 samples
c(median = mean(med$Mn), mean = mean(avg$Mn),
  reference = mean(forageLIBS$Mn[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
```

---

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,
  drop.na = deprecated()
)
```

Arguments

x	A numeric vector.
p	A probability between 0 and 0.5 that chooses the quantiles of the left half compared by LQW: $Q(p/2)$ and $Q((1-p)/2)$ of the data. A smaller value looks further into the left tail. Default is 0.125.
q	A probability between 0.5 and 1 that chooses the quantiles of the right half compared by RQW: $Q(1-q/2)$ and $Q((1+q)/2)$ of the data. A larger value looks further into the right tail. Default is 0.875.
drop_na	Logical value indicating whether to remove missing values (NA). If FALSE (default) and x contains missing values, NA is returned.
drop.na	**[Deprecated]** Use drop_na.

Details

The left quantile weight (LQW) and the right quantile weight (RQW) are robust measures of tail weight (Brys *et al.* 2006). They apply the quantile skewness of Hinkley (1975),

$$SK(p) = \frac{Q(1-p) + Q(p) - 2Q(0.5)}{Q(1-p) - Q(p)},$$

to each half of the distribution, divided at the median: the left half, whose median is the first quartile $Q(0.25)$, and the right half, whose median is the third quartile $Q(0.75)$:

$$LQW(p) = -\frac{Q((1-p)/2) + Q(p/2) - 2Q(0.25)}{Q((1-p)/2) - Q(p/2)}$$

$$RQW(q) = \frac{Q((1+q)/2) + Q(1-q/2) - 2Q(0.75)}{Q((1+q)/2) - Q(1-q/2)}$$

where Q is the quantile function of the data. `p` and `q` are probabilities that choose the quantiles compared: LQW compares $Q(p/2)$ and $Q((1-p)/2)$, the p and $1-p$ quantiles of the left half, around $Q(0.25)$; RQW compares $Q(1-q/2)$ and $Q((1+q)/2)$, the $1-q$ and q quantiles of the right half, around $Q(0.75)$. A heavier tail stretches the outer quantile further than the inner one, which gives a larger weight. With $q = 1-p$, both weights look equally far into their tails. The defaults, `p = 0.125` and `q = 0.875`, compare $Q(0.0625)$ and $Q(0.4375)$ for LQW, and $Q(0.5625)$ and $Q(0.9375)$ for RQW.

The breakdown value of LQW is $p/2$ and that of RQW is $(1-q)/2$: 6.25% for the defaults, which resist up to 6.25% of outliers in a tail. A smaller `p` (or a `q` closer to 1) looks further into the tails, which makes the weights more sensitive to the tail weight but less robust; `p = 0.25` and `q = 0.75` give a breakdown value of 12.5%, as `medcouple_weight()`.

Interpretation of the quantile weights:

- At the normal distribution, LQW = RQW = 0.249 with the defaults (0.144 with `p = 0.25` and `q = 0.75`).
- 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.

See Also

medcouple_weight(), medcouple()

Examples

```
# 10000 values from three symmetric distributions: uniform (light tails),
# normal, and Laplace (double exponential: heavy tails)
set.seed(1)
n <- 10000
tails <- data.frame(
  distribution = factor(rep(c("uniform", "normal", "Laplace"), each = n),
                        levels = c("uniform", "normal", "Laplace")),
  value = c(runif(n, -2, 2), rnorm(n),
            rexp(n) * sample(c(-1, 1), n, replace = TRUE))
)
ggplot2::ggplot(tails, ggplot2::aes(value, fill = distribution)) +
  ggplot2::geom_density(alpha = 0.4) +
  ggplot2::coord_cartesian(xlim = c(-5, 5)) +
  ggplot2::scale_fill_manual(values = c("#fdb863", "#2166ac", "#b2182b"))

# the weights grow with the tail weight, in both tails: about 0 for the
# uniform distribution, 0.25 for the normal one and 0.43 for the Laplace one
weights <- function(v) c(kurtosis = moments::kurtosis(v), unlist(quantile_weight(v)))
round(t(sapply(split(tails$value, tails$distribution), weights)), 3)

# 1% of outliers: 9900 normal values and 100 near 10. The kurtosis jumps
# from 3 to 25, while the right quantile weight hardly moves and the left
# one does not see the outliers
set.seed(2)
x <- c(rnorm(9900), rnorm(100, mean = 10, sd = 0.5))
group <- rep(c("regular", "outlier"), c(9900, 100))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))
round(weights(x), 3)

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
```

---

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 `ncomp` 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,
  ncomp,
  newdata = NULL,
  conf_level = 0.975,
  method = "jackson",
  t2_method = "f",
  center = TRUE,
  scale = FALSE,
  k = deprecated()
)
```

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`.
`ncomp`	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`.
`k`	**[Deprecated]** Use `ncomp`.

Details

With scores t_{ia} and eigenvalues λ_a, and k = `ncomp` components,

$$T_i^2 = \sum_{a=1}^{k} \frac{t_{ia}^2}{\lambda_a}, \qquad Q_i = \|x_i - \hat{x}_i\|^2$$

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, ncomp = 3)
  influence[influence$outlier != "regular", ]
  plot_influence(influence, label = forageLIBS$Measurement)
  # several limits
  plot_influence(q_residuals(pca, ncomp = 3, conf_level = c(0.95, 0.99)))
}
```

`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()`, `read_vald()`, `read_kurucz()`, `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)
```

`read_vald` *Stark Widths from VALD3 and Kurucz Line Lists*

Description

Reads the quadratic Stark damping constants of a line list saved from the Vienna Atomic Line Database (VALD3) or from the line lists of R. L. Kurucz, as a table of Stark widths in the format of `starkb_lines()`, for `stark_width()` and `electron_density()`. These line lists give a width for many lines that STARK-B does not have, but a cruder one.

Usage

```
read_vald(
  file,
  species = NULL,
  wavelength = NULL,
  temperature = c(2500, 5000, 10000, 20000, 40000)
)

read_kurucz(
  file,
  species = NULL,
  wavelength = NULL,
  temperature = c(2500, 5000, 10000, 20000, 40000)
)
```

Arguments

`file`	The path to the line list: an extract saved from VALD3 for `read_vald()`, a Kurucz line list for `read_kurucz()`.
`species`	An optional character vector of the species to keep, in spectroscopic notation, such as `c("Ca II", "Mg I")`. By default all the species of the file are kept.
`wavelength`	An optional numeric vector of length 2: the wavelength range to keep, in nm.
`temperature`	The temperatures (K) at which the widths are tabulated. Default is 2500 to 40 000 K; `stark_width()` interpolates between them.

Details

Files. The data of VALD3 are free, but need a registration: request an "Extract All", "Extract Element" or "Extract Stellar" list from the VALD website or its email service, in the short or the long format, with any wavelength unit (Å, nm or cm^{-1}), medium and energy unit, and give the saved file to `read_vald()`. The long format also gives the designations of the levels (`lower` and `upper`). The line lists of Kurucz are free to download from `kurucz.harvard.edu`: the whole list (`gfall*.dat`, in air above 200 nm, or `gfallvac*.dat`, in vacuum) or the list of one ion (such as `atoms/2001/gf2001.pos` for Ca II), in the 160-column format of these files. The wavenumber files (`gfallwn*.dat`) are not read. The wavelengths are kept in the medium of the file: in air, they match those of `nist_lines()` above 200 nm.

Widths. Both line lists give, for each line, the decimal logarithm of the quadratic Stark damping constant γ at 10 000 K per electron: the full width at half maximum of the Lorentzian profile in angular frequency (rad s^{-1}), divided by the electron density (cm^{-3}). The full width at half maximum in wavelength is

$$w = \frac{\lambda^2}{2\pi c}\gamma N_e$$

It is tabulated at the density 10^{17} cm^{-3} (the width is proportional to it) and at each of `temperature`, scaled as $(T/10^4\,\mathrm{K})^{1/6}$, the temperature dependence of the quadratic Stark width in the impact approximation (Gray, 2005). The shift is not given: it is set to zero. Lines without a Stark constant (zero in the file) are left out, and a line listed several times (as its isotopic or hyperfine components) is kept once.

Accuracy. Most of these constants come from approximate formulas rather than from calculations such as those of STARK-B, and the temperature law is approximate too: prefer STARK-B (`starkb_lines()`) or measured widths (`stark_table()`) when they exist, and use these for the other lines. A table of several species needs `species` in `stark_width()` and `electron_density()`.

When you use these data, cite the line list and the references of the constants, given in the `source` column (the reference codes of VALD are described in the BibTeX file that comes with each extract).

Value

A tibble in the format of `starkb_lines()`, with one row per line and temperature: `species`, `wavelength` (nm), `upper` and `lower` (the level designations, empty when the file does not give them), `perturber` (`"electron"`), `temperature`, `density` (10^{17} cm^{-3}), `width` (nm), `shift` (0) and `source` (the line list, the logarithm of the constant and its references).

Author(s)

Christian L. Goueguel

References

- Ryabchikova, T., Piskunov, N., Kurucz, R.L., Stempels, H.C., Heiter, U., Pakhomov, Yu., Barklem, P.S. (2015). A major upgrade of the VALD database. Physica Scripta, 90(5):054005. doi:10.1088/00318949/90/5/054005
- Kurucz, R.L. Atomic line lists, `kurucz.harvard.edu/linelists.html`.
- Gray, D.F. (2005). The Observation and Analysis of Stellar Photospheres, 3rd edition. Cambridge University Press.

See Also

starkb_lines(), stark_table(), stark_width(), electron_density()

Examples

```
# two lines of a VALD3 "Extract Element" list, short format (made-up
# references)
file <- tempfile(fileext = ".txt")
writeLines(c(
  "                                          Damping parameters    Lande",
  "Elm Ion       WL_air(A) Excit(eV) log(gf)   Rad.  Stark    Waals  factor References",
  "'Ca 2',     3933.6630,  0.0000,  0.135, 8.170,-6.270,  -7.760, 1.170,'  1 K10  Ca+  '",
  "'Ca 2',     3968.4690,  0.0000, -0.180, 8.160,-6.270,  -7.760, 1.330,'  1 K10  Ca+  '"
), file)
ca <- read_vald(file)
ca
stark_width(ca, wavelength = 393.366, temperature = 9000, density = 5e16)

# the same line in the 160-column format of the Kurucz line lists
kurucz <- tempfile(fileext = ".pos")
writeLines(paste0(
  "   393.3663  0.111 20.01       0.000  0.5 4s 2S        25414.400  1.5 4p 2P",
  "       8.17 -6.27 -7.76K10 "
), kurucz)
read_kurucz(kurucz, temperature = c(5000, 20000))
```

---

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,
  shot = NULL,
  scale = c("floor", "sample", "pooled")
)
```

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).
`shot`	An optional column (unquoted or as a string) giving the order of the shots in each sample, such as the shot number. Default is the order of the rows.
`scale`	The robust scale of the z-scores: `"floor"` (default), `"sample"` or `"pooled"` (see Details).

Details

Each shot is compared with the other shots of its sample, by robust z-scores of one or more criteria:

- `"intensity"`: the total intensity of the spectrum, on a log scale (its deviation is relative to the median shot of the sample). Both weak and unusually strong shots are flagged.
- `"correlation"`: the Pearson correlation r 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. The z-scores are those of Fisher's $z = \tanh^{-1}(r)$: correlations near 1 are bounded and skewed, so that small differences such as 0.998 and 0.9995 would otherwise look extreme. Only shots with a low correlation are flagged.
- `"distance"`: the Euclidean distance of the spectrum to the median spectrum of the sample, relative to the norm of the median spectrum, on a log scale, which combines both. Only far shots are flagged.

Scale. A z-score is the deviation of a shot from the median of its sample divided by a robust scale (the MAD). With the few shots of a sample (often 5 to 10), the MAD of the sample is unstable: when its shots happen to be very similar, a shot that differs by little gets a large z-score. With `scale = "floor"` (default), the scale of a sample is its own MAD, but not less than the pooled MAD of the deviations of all the samples (the typical shot-to-shot variability of the data set); `"sample"` uses the MAD of each sample alone, and `"pooled"` the pooled MAD for all samples, which suits samples of similar variability. In simulations of 8 shots per sample with noise varying between samples, `"floor"` with Fisher's z rejected 0.1% to 0.5% of the regular shots (against 2.5% to 5% with the MAD of each sample and the raw correlations), and still detected the shots of a plasma weakened by a quarter or more, or with an extra line of 10% of the strongest line. Lower `cutoff` to detect milder deviations.

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 a MAD is zero, the mean absolute deviation is used.

The result keeps the raw criteria (`.intensity`, the total intensity relative to the median shot of the sample; `.correlation`; `.distance`), easier to read than the z-scores, and the settings, which `plot_shots()` uses to show the shots and the rejections. 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 the shot number in its sample `.shot`, a logical column `.rejected`, the criteria that rejected each shot in `.reason` (`NA` for kept shots), the raw criteria `.intensity`, `.correlation` and `.distance`, and the robust z-score of each criterion used (`.intensity_z`, `.correlation_z`, `.distance_z`). Its attribute `"reject_shots"` holds the settings. 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.
- Fisher, R.A. (1915). Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. Biometrika, 10(4):507-521.

See Also

`plot_shots()`, `step_reject_shots()`, `average()`, `saturation_summary()`, forageShots

Examples

```
data(forageShots)
# 8 shots of each of 20 forage samples
res <- reject_shots(forageShots, Measurement, shot = shot)
res[res$.rejected, c("Measurement", ".shot", ".reason", ".intensity", ".correlation")]
# the kept shots only
nrow(reject_shots(forageShots, Measurement, drop = TRUE))
# the former behavior, with the MAD of each sample alone
sum(reject_shots(forageShots, Measurement, scale = "sample")$.rejected)
```

---

response_calibration *Spectral Response from a Calibration Lamp*

Description

Measures the relative spectral response of an instrument (the efficiency of the collection optics, fiber, grating and detector at each wavelength) from spectra of a calibration lamp of known spectral irradiance or radiance, such as a deuterium-halogen lamp, with its uncertainty. `correct_response()` then corrects spectra or line intensities.

Usage

```
response_calibration(
  lamp,
  reference,
  dark = NULL,
  smooth = 1,
  exclude = NULL,
  segments = TRUE
)
```

Arguments

`lamp`	The spectra of the calibration lamp: 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). Several spectra are averaged.
`reference`	The certified spectral irradiance (or radiance) of the lamp: a data frame with columns `wavelength` (nm) and `value`, and optionally `rsd`, the relative standard uncertainties of the values (0.03 for 3%).
`dark`	An optional dark spectrum (or several, averaged) on the same axis as `lamp`, subtracted from it.
`smooth`	The width of the smoothing window, in nm. Default is 1; 0 turns the smoothing off.
`exclude`	Wavelength ranges left out of the response and interpolated: a numeric vector of length 2, a list of them, or a two-column matrix. Default is `NULL`.
`segments`	A logical: smooth each detector segment separately (`TRUE`, default).

Details

The response at each channel is the ratio of the measured lamp spectrum (after subtraction of `dark`) to the certified irradiance of the lamp, interpolated at the channel. The certified values, often tabulated every few nanometers, are interpolated by a monotone cubic spline of their logarithm; channels outside their range get no response. The ratio is smoothed by a local linear fit over `smooth` nm, within each detector segment when `segments = TRUE` (the axis is split where the wavelengths step back or jump, as in `wavelength_calibration()`), so that the noise of the lamp spectrum is averaged without biasing the steep edges of the response. The channels in the `exclude` ranges, such as the Balmer lines of a deuterium lamp (486.0 and 656.1 nm) that the certificate leaves out, are interpolated from their neighbors. The response is scaled to a maximum of 1.

The relative standard uncertainty of the response (`rsd`) combines in quadrature that of the certified values (column `rsd` of `reference`, interpolated linearly) and that of the measurement: with several lamp spectra, the standard error of their mean response (each spectrum scaled to a common level first, so that drifts of the lamp output do not count); with one, the noise of the spectrum (estimated from its channel-to-channel differences) averaged over the smoothing window. Certificates often give expanded uncertainties (coverage factor 2): divide them by 2.

The lamp must be measured through the same optical path as the plasma (collection optics, fiber, spectrometer settings), and its spectrum must not be saturated. With a certificate in energy units (W per nm), the corrected intensities are proportional to the spectral radiance (energy units, as `units = "energy"` of `boltzmann()`).

Value

An object of class `specproc_response`, a list with

- `response`: a tibble with, for each channel in column order, its `wavelength`, detector `segment`, relative `response` (`NA` without response), and its relative standard uncertainty `rsd`, with the parts from the reference (`reference_rsd`) and from the measurement (`measurement_rsd`);
- `n_spectra`, `smooth`, `exclude` and the wavelength `range` of the reference.

Use `correct_response()` to correct spectra or line intensities, and `plot_response()` to draw the response.

Author(s)

Christian L. Goueguel

See Also

`correct_response()`, `plot_response()`, `line_intensities()`

Examples

```
# the certified spectral irradiance of a lamp, every 10 nm, known within
# 3% (simulated here as a blackbody at 3100 K)
planck <- function(wl, t) 1 / (wl^5 * (exp(1.4388e7 / (wl * t)) - 1))
reference <- data.frame(wavelength = seq(250, 850, by = 10))
reference$value <- planck(reference$wavelength, 3100)
reference$rsd <- 0.03

# five spectra of the lamp, through an instrument of unknown response
wl <- seq(260, 840, by = 0.1)
true_response <- exp(-((wl - 500) / 250)^2)
set.seed(1)
lamp <- t(replicate(5, 2e4 * true_response * planck(wl, 3100) / planck(500, 3100) +
                      rnorm(length(wl), sd = 20)))
colnames(lamp) <- wl

response <- response_calibration(lamp, reference)
response
plot_response(response)
```

---

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,
  ncomp = NULL,
  ncomp_max = 10,
  alpha = 0.75,
  ndir = 250,
  var_explained = 0.8,
  nsamp = 500,
  k = deprecated(),
  kmax = deprecated()
)
```

Arguments

x	A numeric matrix or data frame, with one observation per row.
ncomp	The number of principal components. If NULL (default), it is chosen from var_explained and ncomp_max.
ncomp_max	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 $h = \max(\lfloor \alpha n \rfloor, \lfloor (n + k_{max} + 1)/2 \rfloor)$, with k_{max} = ncomp_max.
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 ncomp when it is not given. Default is 0.8.
nsamp	The number of random subsets of FAST-MCD. Default is 500.
k	**[Deprecated]** Use ncomp.
kmax	**[Deprecated]** Use ncomp_max.

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 (ncomp) is chosen (if not given) as the smallest number that explains var_explained of the variance of H_0, with at most ncomp_max. 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. When ncomp equals the rank of the data, this subspace is the whole space and $H_1 = H_0$.

4. All observations are projected onto this subspace, and the reweighted FAST-MCD estimator of the scores gives the final center, loadings and eigenvalues. When `ncomp` equals the rank, this is the reweighted MCD of the data.

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 $OD^{2/3}$. When `ncomp` equals the rank, the ODs and their cut-off are zero: there are no orthogonal outliers. `plot_outlier_map()` plots both distances.

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, the largest element of each loading vector positive.
- `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).
- `ncomp`, `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_boxcox` *Robust Box-Cox Transformation*

Description

Transforms each variable toward central normality with the Box-Cox transformation, whose parameter λ is fitted robustly (Raymaekers and Rousseeuw, 2021), so that outlying observations do not drive it. `robust = FALSE` gives the classical fit, by maximum likelihood, for comparison.

Usage

```
robust_boxcox(
  x,
  quantile = 0.99,
  nbsteps = 2,
  standardize = TRUE,
  robust = TRUE
)
```

Arguments

`x`	A numeric vector, matrix, data frame or tibble, with one variable per column (such as line intensities, contents or spectral channels).
`quantile`	The quantile of the absolute value of a standard normal variable beyond which the standardized transformed values are given weight zero in the re-weighting steps, strictly between 0 and 1. Default is 0.99.
`nbsteps`	The number of re-weighting steps, a positive integer. Default is 2.
`standardize`	A logical value: standardize the transformed variables by the mean and standard deviation of the inliers (`TRUE`, default).
`robust`	A logical value: fit λ robustly (`TRUE`, default), or by classical maximum likelihood on all the values (`FALSE`), to compare (see Robust fit).

Details

The Box-Cox transformation (Box and Cox, 1964) of a strictly positive value x is

$$x^{(\lambda)} = \frac{x^\lambda - 1}{\lambda}$$

for $\lambda \neq 0$, and $x^{(\lambda)} = \log x$ for $\lambda = 0$. A λ below 1 compresses the upper tail, for right-skewed variables ($\lambda = 0$ for log-normal ones), and a λ above 1 compresses the lower tail, for left-skewed ones.

Each variable is first divided by its median, so that its transformed median is 0 and λ does not depend on its units. Variables with zero or negative values are left unchanged, with a warning: use `robust_yeojohnson()` for them, as for baseline-corrected or SNV-scaled spectra. When predict() is applied to new data, values that are zero or negative become `NA`.

Value

An object of class `specproc_boxcox` (a list), which predict() applies to new data, or with `inverse = TRUE` uses to transform values back to the original scale (as a median or a mean), with the following components:

- `transformation`: A tibble of the transformed variables.
- `summary`: A tibble with one row per variable: `variable`, `lambda` (`NA` for the variables left unchanged) and `objective` (the robust distance between the transformed values and the normal quantiles of step 2; lower is closer to normality).
- `fits`: The fitted parameters of each variable, used by `predict()`.
- `standardize`: The `standardize` argument.
- `robust`: The `robust` argument.

Robust fit

The parameter λ of each variable is fitted as follows:

1. The variable is pre-standardized (see Details).
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 transformed values are standardized with Huber's M-estimates of location and scale, and those farther from the center than the `quantile` of the absolute value of a standard normal variable (2.58 for the default 0.99) are given weight zero (first with the rectified transformation of step 2). λ is then re-estimated by maximum likelihood on the others. This is repeated `nbsteps` times.
4. With `standardize = TRUE`, the transformed variable is standardized by the mean and standard deviation of these inliers.

With `robust = FALSE`, λ is the classical maximum likelihood estimate (Box and Cox, 1964; Yeo and Johnson, 2000): it maximizes the normal likelihood of all the transformed values, and the transformed variable is standardized by the mean and standard deviation of all the values (`quantile` and `nbsteps` are not used). For the Box-Cox transformation, this is the λ of `MASS::boxcox()`. Outliers then pull λ toward the value that brings them closer to the other values: the transformed regular values are skewed, and the outliers can be hidden among them (see the examples).

Variables with fewer than 5 values or no spread (zero MAD) are left unchanged. Missing values are ignored by the fit, and stay missing.

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(2):211-252.

See Also

`robust_yeojohnson()` for variables with zero or negative values, `step_robust_boxcox()` to use it in a tidymodels recipe, `medcouple()` to measure the skewness before and after.

Examples

```
# 300 log-normal values, which the logarithm (lambda = 0) makes normal,
# and 10 high outliers, 20 times their median
set.seed(6)
x <- c(rlnorm(300, meanlog = log(100), sdlog = 0.6),
       rlnorm(10, meanlog = log(2000), sdlog = 0.1))
group <- rep(c("regular", "outlier"), c(300, 10))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))

# the classical fit (that of MASS::boxcox()) lowers lambda to pull the
# outliers in; the robust fit stays close to 0
classical <- robust_boxcox(x, robust = FALSE)
robust <- robust_boxcox(x)
round(c(classical = classical$summary$lambda, robust = robust$summary$lambda), 2)

# the classical transformation skews the regular values to the left and
# leaves the outliers within 3 standard deviations; the robust one makes
# the regular values symmetric and leaves the outliers far out
transformed <- data.frame(
  fit = rep(c("classical", "robust"), each = length(x)),
  value = c(classical$transformation[[1]], robust$transformation[[1]]),
  group = rep(group, 2)
```

```
)
ggplot2::ggplot(transformed, ggplot2::aes(value)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac")) +
  ggplot2::facet_wrap(~fit, ncol = 1)
regular <- transformed$group == "regular"
# the skewness of the transformed regular values, and the outlier closest
# to them
round(tapply(transformed$value[regular], transformed$fit[regular], medcouple), 2)
round(tapply(transformed$value[!regular], transformed$fit[!regular], min), 2)

data(forageLIBS)
contents <- forageLIBS[c("Ca", "K", "P", "Mn")]  # mg/kg
fit <- robust_boxcox(contents[1:300, ])
fit$summary
# the skewness of the contents, before and after
round(sapply(contents[1:300, ], medcouple), 2)
round(sapply(fit$transformation, medcouple), 2)
# new samples, with the transformation of the first 300
head(predict(fit, contents[301:368, ]))
```

`robust_da` *Robust Linear and Quadratic Discriminant Analysis*

Description

Robust linear (LDA) and quadratic (QDA) discriminant analysis of Hubert and Van Driessen (2004): the class centers and covariance matrices are estimated by the minimum covariance determinant (MCD) estimator, so that outlying observations of the training data have little influence on the discriminant rules.

Usage

```
robust_da(
  x,
  group,
  method = c("linear", "quadratic"),
  alpha = 0.75,
  prior = NULL,
  nsamp = 500
)
```

Arguments

`x`	A numeric matrix or data frame of the predictors, one observation per row.
`group`	The classes of the observations: a factor, or a vector converted to one.
`method`	The discriminant rule: `"linear"` (default) or `"quadratic"`.

alpha	The fraction of observations of each class assumed to be regular (the MCD subset size), between 0.5 and 1. Default is 0.75.
prior	The membership (prior) probabilities of the classes, in the order of the levels of group (or named by them). Default is the proportions of the regular training observations in each class.
nsamp	The number of random subsets of the MCD. Default is 500.

Details

Quadratic rule. The center $\hat{\mu}_j$ and covariance $\hat{\Sigma}_j$ of each class are its reweighted MCD estimates. An observation x is assigned to the class with the largest discriminant score

$$d_j(x) = -\frac{1}{2}\log|\hat{\Sigma}_j| - \frac{1}{2}(x-\hat{\mu}_j)^T\hat{\Sigma}_j^{-1}(x-\hat{\mu}_j) + \log p_j.$$

Linear rule. The classes share one covariance matrix: the reweighted MCD of the observations centered by the MCD center of their class, whose own center shifts the class centers. The scores are those of the quadratic rule with this common covariance.

The membership (prior) probabilities p_j are, by default, the proportions of the regular observations of the training data in each class: those within the cut-off $\sqrt{\chi^2_{p,0.975}}$ of the robust distance to the center of their class. The posterior probabilities are proportional to p_j times the normal density of each class.

The MCD needs more observations than variables in each class, so robust discriminant analysis applies to low-dimensional data: line intensities or ratios, or the robust principal component scores of spectra (robpca(), step_robpca()). For whole spectra, see rsimca().

The misclassification rates are estimated on the regular observations of the training data (misclassification), which tends to underestimate them: estimate them on test data or by cross-validation (for example with the "mcd" engine of parsnip::discrim_linear() and parsnip::discrim_quad() in tidymodels).

This is an independent implementation of the published description. The MCD uses random subsets, so use set.seed() for reproducible results.

Value

An object of class specproc_robust_da, a list with:

- center: the robust centers of the classes (one row per class).
- cov: the common covariance matrix (linear rule), or a list of the covariance matrices of the classes (quadratic rule).
- prior: the membership probabilities.
- rd: the robust distance of each training observation to the center of its class, and cutoff_rd its cut-off.
- weights: 1 for the regular training observations, 0 for the outliers.
- fitted: the classes assigned to the training observations.
- misclassification: a tibble with the misclassification rate of each class on its regular training observations, and the overall rate weighted by the membership probabilities.

- `method`, `levels`, `alpha`.

Use predict() for the classes and posterior probabilities of new observations.

tidymodels

specProc adds the `"mcd"` engine to `parsnip::discrim_linear()` (linear rule) and `parsnip::discrim_quad()` (quadratic rule); `alpha`, `prior` and `nsamp` are engine arguments. The engines are available once parsnip and specProc are both loaded, in either order.

Author(s)

Christian L. Goueguel

References

- Hubert, M., Van Driessen, K. (2004). Fast and robust discriminant analysis. Computational Statistics and Data Analysis, 45(2):301-320.
- Rousseeuw, P.J., Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator. Technometrics, 41(3):212-223.

See Also

`predict.specproc_robust_da()`, `rsimca()`

Examples

```
# iris: train on 100 flowers, predict the 50 others
set.seed(1)
train <- sample(nrow(iris), 100)
fit <- robust_da(iris[train, 1:4], iris$Species[train])
fit
table(predicted = predict(fit, iris[-train, 1:4]), true = iris$Species[-train])
head(predict(fit, iris[-train, 1:4], type = "prob"))
```

---

robust_gandh *Robust Tukey g-and-h Transformation*

Description

Transforms each variable toward central normality by inverting a Tukey g-and-h distribution fitted by robust quantile least squares (rQLS; Xu, Iglewicz and Chervoneva, 2014). Accepts positive, zero and negative values, and fits skewness and tail heaviness separately. `robust = FALSE` gives the classical quantile least squares fit (QLS), for comparison.

Usage

```
robust_gandh(
  x,
  n_quantiles = 10,
  cutoff = NULL,
  max_iter = 50,
  tol = 1e-06,
  standardize = TRUE,
  robust = TRUE
)
```

Arguments

x	A numeric vector, matrix, data frame or tibble, one variable per column.
n_quantiles	Number of fitting quantiles, an integer from 4 to 20. Default is 10, as recommended for rQLS in the paper.
cutoff	Positive, finite Tukey biweight cutoff on the median/MAD standardized input scale. NULL (default) selects it adaptively for each variable. It is an order-statistic fitting tolerance, not a normal score cutoff or a contamination percentage.
max_iter	Maximum number of trimming/refitting iterations per cutoff. Default is 50.
tol	Positive, finite relative tolerance for convergence. Default is 1e-6.
standardize	Logical: additionally standardize latent scores using the fitting inliers (TRUE, default).
robust	Logical: fit by robust quantile least squares (TRUE, default), or by classical quantile least squares on all the observations (FALSE), to compare (see Details).

Details

The fitted distribution maps a standard normal value z to

$$x = A + B\frac{\exp(gz) - 1}{g}\exp(hz^2/2),$$

with $B > 0$ and $h \geq 0$. At $g = 0$, the fraction is z. The transformation solves this equation for z directly, without rounding extreme cumulative probabilities to zero or one. The inverse transformation evaluates the equation.

Each variable is first centered by its median and divided by its MAD. QLS fits model quantiles to sample order statistics at probabilities $p_j = (j - 1/3)/(m + 1/3)$. Initialization compares this QLS fit with a second QLS fit using equally spaced probabilities from 0.15 to 0.85, selecting the smaller median absolute order-statistic discrepancy. This additional start reduces masking when contamination reaches an outer fitting quantile. The rQLS iterations compare all order statistics with their fitted quantiles, calculate Tukey biweight weights, and refit QLS to quantiles of the observations with nonzero weights. The positive weights are used for selection, not as quantile weights. All original observations, including the trimmed ones, are transformed.

With cutoff = NULL, a bounded numerical version of the paper's adaptive search is used: starting at least at 10 on the standardized input scale, a geometric pilot search brackets the cutoffs retaining more than half and all observations, followed by a descending grid with the paper's grading interval

and tail-weight ordering checks. At most 25 pilot and 51 grading fits are made. If no candidate meets the ordering checks, the largest converged candidate retaining more than half the observations is used, with a warning and `tuning_met = FALSE`. A supplied cutoff bypasses this search. This numerical search need not give exactly the same cutoff as the authors' implementation.

With `robust = FALSE`, the distribution is fitted by QLS to all the observations, without trimming (the non-robust fit of the paper), and the latent scores are standardized by the mean and standard deviation of all the observations; `cutoff`, `max_iter` and `tol` are not used. Outliers in a tail then move the outer fitting quantiles, which inflates g and h: the transformation compresses that tail too much, and pulls the outliers in among the other values (see the examples).

With `standardize = TRUE`, latent normal scores are additionally centered and scaled by the mean and standard deviation of the fitting inliers. With `FALSE`, the fitted latent scores are returned directly; they already account for the distribution's location and scale.

Variables with fewer than `max(10, n_quantiles + 1)` observed values or zero MAD are left unchanged. Failed fits are left unchanged with a warning. Missing values are ignored by fitting and stay missing. Infinite input values are rejected. This is a continuous-distribution model; substantial ties, mixtures and censoring need separate consideration. For $h = 0, g \neq 0$, new values outside the fitted distribution's support become `NA` with a warning. Inverse values exceeding floating point range also become `NA` with a warning.

Value

An object of class `specproc_gandh`, inheriting from `specproc_transformation`, with the same components as `robust_boxcox()`:

- `transformation`: A tibble of transformed values.
- `summary`: A tibble with `variable`, `location`, `scale` (both on the original input scale), `g`, `h`, `objective` (quantile mean squared error on the median/MAD input scale), `n_trimmed`, `cutoff` (on the standardized input scale), `converged` and `tuning_met`. Parameters and diagnostics are `NA` for variables left unchanged. `tuning_met` is `NA` when a cutoff was supplied. With `robust = FALSE`, `n_trimmed` is 0, and `cutoff` and `tuning_met` are `NA`.
- `fits`: Parameters and fitting diagnostics used by `predict()`.
- `standardize`: The `standardize` argument.
- `robust`: The `robust` argument.

Author(s)

Christian L. Goueguel

References

Xu, Y., Iglewicz, B., Chervoneva, I. (2014). Robust estimation of the parameters of g-and-h distributions, with applications to outlier detection. Computational Statistics & Data Analysis, 75:66-80. doi:10.1016/j.csda.2014.01.003.

See Also

`robust_boxcox()`, `robust_yeojohnson()`, `tukey_gh()`, `predict.specproc_transformation()`, `step_robust_gandh()`

Examples

```
# 300 values from a g-and-h distribution, skewed to the right (g = 0.5)
# with heavier tails than the normal distribution (h = 0.1), and 15 high
# outliers near 400
set.seed(6)
x <- c(tukey_gh(type = "r", n = 300, location = 100, scale = 10, g = 0.5, h = 0.1),
       rnorm(15, mean = 400, sd = 2))
group <- rep(c("regular", "outlier"), c(300, 15))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))

# the classical fit (QLS on all the values) overestimates the skewness g
# and the tail weight h; the robust fit (rQLS) trims the 15 outliers and
# stays close to g = 0.5 and h = 0.1
classical <- robust_gandh(x, robust = FALSE)
robust <- robust_gandh(x)
params <- c("g", "h", "n_trimmed")
round(rbind(classical = unlist(classical$summary[params]),
            robust = unlist(robust$summary[params])), 2)

# the classical transformation compresses the right tail too much, and
# pulls the outliers within 3 standard deviations; the robust one leaves
# them far out
transformed <- data.frame(
  fit = rep(c("classical", "robust"), each = length(x)),
  value = c(classical$transformation[[1]], robust$transformation[[1]]),
  group = rep(group, 2)
)
ggplot2::ggplot(transformed, ggplot2::aes(value)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac")) +
  ggplot2::facet_wrap(~fit, ncol = 1)
regular <- transformed$group == "regular"
# the skewness of the transformed regular values, and the outlier closest
# to them
round(tapply(transformed$value[regular], transformed$fit[regular], medcouple), 2)
round(tapply(transformed$value[!regular], transformed$fit[!regular], min), 2)

data(forageLIBS)
contents <- forageLIBS[c("Ca", "K", "P", "Mn")]
fit <- robust_gandh(contents[1:300, ])
fit$summary
new <- predict(fit, contents[301:368, ])
head(predict(fit, new, inverse = TRUE))
```

`robust_rmsecv` *Robust Cross-Validation of a Robust Calibration Model*

Description

Robust root mean squared error of cross-validation (R-RMSECV) of the robust PLS (`rsimpls()`) or robust PCR (`rpcr()`) models with 1 to `ncomp_max` components, and the robust component selection (RCS) criterion of Engelen and Hubert (2005), to choose the number of components when the calibration data contain outliers.

Usage

```
robust_rmsecv(
  x,
  y,
  method = c("rsimpls", "rpcr"),
  ncomp_max = 10,
  folds = 10,
  gamma = 0.5,
  alpha = 0.75,
  ndir = 250,
  nsamp = 500,
  ...
)
```

Arguments

`x`	A numeric matrix or data frame of the predictors (spectra), one observation per row.
`y`	A numeric vector, matrix or data frame of the responses, with one row per observation.
`method`	The model: `"rsimpls"` (default), robust PLS, or `"rpcr"`, robust PCR.
`ncomp_max`	The largest number of components. Default is 10.
`folds`	The number of folds of the cross-validation, from 2 to the number of observations (leave-one-out). Default is 10.
`gamma`	The weight of the cross-validated error in the RCS criterion, between 0 and 1. Default is 0.5.
`alpha`	The robustness parameter of `robpca()`: the fraction of observations assumed to be regular, between 0.5 and 1. Default is 0.75.
`ndir`	The number of random directions of the outlyingness in `robpca()`. Default is 250.
`nsamp`	The number of random subsets of FAST-MCD (and, for `"rpcr"`, of the LTS or MCD regression). Default is 500.
`...`	Not used.

Details

The squared prediction errors of outliers would dominate the usual RMSECV. The robust RMSECV leaves them out: the model is first fitted to all the observations, and only those regular in every model

with 1 to ncomp_max components (residual distance below its cut-off) enter the error,

$$R\text{-}RMSECV_k = \sqrt{\frac{1}{q \sum_i w_i} \sum_i w_i \|y_i - \hat{y}_{-i,k}\|^2},$$

where w_i is 1 for these observations and 0 for the others, and $\hat{y}_{-i,k}$ is the prediction of observation i by the model with k components fitted without it.

The robust root mean squared error of the fit to the calibration data, RMSE (the square root of the robust residual sum of squares, R-RSS), decreases with the number of components, while the cross-validated error eventually increases. The RCS criterion combines them,

$$RCS_k = \sqrt{\gamma\, R\text{-}RMSECV_k^2 + (1-\gamma)\, R\text{-}RSS_k},$$

and its first local minimum, or the point after which it decreases little, suggests the number of components. gamma = 1 gives the robust RMSECV, and gamma = 0 the R-RSS.

Differences from the paper. The paper computes the leave-one-out errors with a fast approximation that updates the robust fits instead of refitting them. Here, each fold is refitted. Leave-one-out (folds = nrow(x)) is then slow, and K-fold cross-validation (default 10 folds) is used instead.

The folds are fitted in parallel when a parallel plan is set with future::plan() (and the future.apply package is installed). The folds and the fits are random: use set.seed() for reproducible results.

Value

A tibble with one row per number of components: ncomp, the robust R2 and RMSE of the fit to all the observations (as the components of rsimpls() and rpcr()), the robust RMSECV and the RCS criterion. The weights w_i are its "weights" attribute.

Author(s)

Christian L. Goueguel

References

- Engelen, S., Hubert, M. (2005). Fast model selection for robust calibration methods. Analytica Chimica Acta, 544(1-2):219-228.

See Also

rsimpls(), rpcr()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
robust_rmsecv(spectra[1:300, ], forageLIBS$Ca[1:300], ncomp_max = 8, folds = 5)
```

`robust_yeojohnson` *Robust Yeo-Johnson Transformation*

Description

Transforms each variable toward central normality with the Yeo-Johnson transformation, whose parameter λ is fitted robustly (Raymaekers and Rousseeuw, 2021), so that outlying observations do not drive it. Unlike the Box-Cox transformation (`robust_boxcox()`), it accepts zero and negative values. `robust = FALSE` gives the classical fit, by maximum likelihood, for comparison.

Usage

```
robust_yeojohnson(
  x,
  quantile = 0.99,
  nbsteps = 2,
  standardize = TRUE,
  robust = TRUE
)
```

Arguments

`x`	A numeric vector, matrix, data frame or tibble, with one variable per column (such as line intensities, contents or spectral channels).
`quantile`	The quantile of the absolute value of a standard normal variable beyond which the standardized transformed values are given weight zero in the re-weighting steps, strictly between 0 and 1. Default is 0.99.
`nbsteps`	The number of re-weighting steps, a positive integer. Default is 2.
`standardize`	A logical value: standardize the transformed variables by the mean and standard deviation of the inliers (`TRUE`, default).
`robust`	A logical value: fit λ robustly (`TRUE`, default), or by classical maximum likelihood on all the values (`FALSE`), to compare (see Robust fit).

Details

The Yeo-Johnson transformation (Yeo and Johnson, 2000) of a value $x \geq 0$ is

$$\psi_\lambda(x) = \frac{(1+x)^\lambda - 1}{\lambda}$$

($\log(1+x)$ for $\lambda = 0$), and of a value $x < 0$

$$\psi_\lambda(x) = -\frac{(1-x)^{2-\lambda} - 1}{2-\lambda}$$

($-\log(1-x)$ for $\lambda = 2$). $\lambda = 1$ leaves the values unchanged; a λ below 1 compresses the upper tail, for right-skewed variables, and a λ above 1 compresses the lower tail, for left-skewed ones.

Each variable is first centered by its median and divided by its MAD, so that λ does not depend on its location or units. It describes the shape of the distribution on this robust standard scale, and cannot be compared with the λ of `robust_boxcox()`: on the same data, it is often much closer to 1.

It suits the variables with zero or negative values, such as baseline-corrected, derivative or SNV-scaled spectra. Strictly positive variables, such as raw intensities or contents, can be transformed with either transformation.

Value

An object of class `specproc_yeojohnson` (a list), which predict() applies to new data, or with `inverse = TRUE` uses to transform values back to the original scale (as a median or a mean), with the following components:

- `transformation`: A tibble of the transformed variables.
- `summary`: A tibble with one row per variable: `variable`, `lambda` (`NA` for the variables left unchanged) and `objective` (the robust distance between the transformed values and the normal quantiles of step 2; lower is closer to normality).
- `fits`: The fitted parameters of each variable, used by `predict()`.
- `standardize`: The `standardize` argument.
- `robust`: The `robust` argument.

Robust fit

The parameter λ of each variable is fitted as follows:

1. The variable is pre-standardized (see Details).
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 transformed values are standardized with Huber's M-estimates of location and scale, and those farther from the center than the `quantile` of the absolute value of a standard normal variable (2.58 for the default 0.99) are given weight zero (first with the rectified transformation of step 2). λ is then re-estimated by maximum likelihood on the others. This is repeated `nbsteps` times.
4. With `standardize = TRUE`, the transformed variable is standardized by the mean and standard deviation of these inliers.

With `robust = FALSE`, λ is the classical maximum likelihood estimate (Box and Cox, 1964; Yeo and Johnson, 2000): it maximizes the normal likelihood of all the transformed values, and the transformed variable is standardized by the mean and standard deviation of all the values (`quantile` and `nbsteps` are not used). For the Box-Cox transformation, this is the λ of `MASS::boxcox()`. Outliers then pull λ toward the value that brings them closer to the other values: the transformed regular values are skewed, and the outliers can be hidden among them (see the examples).

Variables with fewer than 5 values or no spread (zero MAD) are left unchanged. Missing values are ignored by the fit, and stay missing.

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.
- Yeo, I.-K., Johnson, R.A. (2000). A new family of power transformations to improve normality or symmetry. Biometrika, 87(4):954-959.

See Also

`robust_boxcox()`, `step_robust_yeojohnson()` to use it in a tidymodels recipe, `medcouple()` to measure the skewness before and after.

Examples

```
# 300 right-skewed values on both sides of 0 (a gamma distribution shifted
# by -2) and 10 low outliers near -6
set.seed(6)
x <- c(rgamma(300, shape = 2) - 2, rnorm(10, mean = -6, sd = 0.3))
group <- rep(c("regular", "outlier"), c(300, 10))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))

# the low outliers balance the long right tail: the classical fit finds
# lambda close to 1, no transformation; the robust fit compresses the
# right tail (lambda below 1)
classical <- robust_yeojohnson(x, robust = FALSE)
robust <- robust_yeojohnson(x)
round(c(classical = classical$summary$lambda, robust = robust$summary$lambda), 2)

# the classical transformation leaves the regular values skewed to the
# right and the outliers near 3 standard deviations; the robust one makes
# the regular values symmetric and leaves the outliers far out
transformed <- data.frame(
  fit = rep(c("classical", "robust"), each = length(x)),
  value = c(classical$transformation[[1]], robust$transformation[[1]]),
  group = rep(group, 2)
)
ggplot2::ggplot(transformed, ggplot2::aes(value)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac")) +
  ggplot2::facet_wrap(~fit, ncol = 1)
regular <- transformed$group == "regular"
# the skewness of the transformed regular values, and the outlier closest
# to them
round(tapply(transformed$value[regular], transformed$fit[regular], medcouple), 2)
```

```
round(tapply(transformed$value[!regular], transformed$fit[!regular], max), 2)

data(forageLIBS)
spectra <- snv(forageLIBS[-(1:14)])$correction
wl <- as.numeric(names(spectra))
# the Mg II lines at 279.55 and 280.27 nm: after SNV, most channels have
# negative values in some spectra, which Box-Cox cannot transform
mg <- spectra[which(wl > 279 & wl < 281)]
sum(vapply(mg, min, numeric(1)) < 0)
fit <- robust_yeojohnson(mg[1:300, ])
summary(fit$summary$lambda)
# the skewness of the channels, before and after
summary(sapply(mg[1:300, ], medcouple))
summary(sapply(fit$transformation, medcouple))
# new spectra, with the transformation of the first 300
dim(predict(fit, mg[301:368, ]))
```

`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,
  ncomp = NULL,
  lambda = 1,
  alpha = 0.75,
  ndir = 250,
  stand = FALSE,
  ngrid = 10,
  maxiter = 10,
  ncomp_max = 10,
  var_explained = 0.8,
  k = deprecated()
)
```

Arguments

`x`	A numeric matrix or data frame, with one observation per row.
`ncomp`	The number of principal components. If `NULL` (default), it is chosen from `var_explained` and `ncomp_max`.

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 $h = \max(\lfloor \alpha n \rfloor, \lfloor (n + k_{max} + 1)/2 \rfloor)$, with k_{max} = ncomp_max.
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.
ncomp_max	The maximum number of components when ncomp is chosen. Default is 10.
var_explained	The fraction of variance used to choose ncomp when it is not given. Default is 0.8.
k	**[Deprecated]** Use ncomp.

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, the number of components k (ncomp) is chosen (if not given) as the smallest number that explains var_explained of the variance of H_0, with at most ncomp_max, and the observations whose orthogonal distance to the k-dimensional PCA subspace of H_0 is below the cut-off form H_1 ($H_1 = H_0$ when ncomp equals the rank of the data, as the subspace is then the whole space).
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 ($H_2 = H_1$ when the sparse subspace contains all the observations), 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,
  drop.na = deprecated()
)
```

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.
drop.na	**[Deprecated]** Use drop_na.

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"))
```

`rpcr` *Robust Principal Component Regression (RPCR)*

Description

Robust principal component regression by the RPCR method of Hubert and Verboven (2003): the robust principal components of the predictors (`robpca()`), followed by a robust regression of the responses on the scores. Outlying spectra and wrong reference values have little influence on the model, and the regression outlier map (`plot_outlier_map()`) classifies them.

Usage

```
rpcr(x, y, ncomp, ncomp_max = 10, alpha = 0.75, ndir = 250, nsamp = 500)
```

Arguments

`x`	A numeric matrix or data frame of the predictors (spectra), one observation per row.
`y`	A numeric vector, matrix or data frame of the responses, with one row per observation.
`ncomp`	The number of components of the model.
`ncomp_max`	The largest number of components considered: the number of components of the robust PCA, so the model also depends on `ncomp_max` (see Details). Default is 10; it is raised to `ncomp` if needed, and lowered when there are too few observations or variables.
`alpha`	The fraction of observations assumed to be regular, between 0.5 and 1, for `robpca()` and the robust regression. Default is 0.75.
`ndir`	The number of random directions of the outlyingness in `robpca()`. Default is 250.
`nsamp`	The number of random subsets of FAST-MCD in `robpca()`, and of the LTS or MCD regression. Default is 500.

Details

The method has two steps:

1. **Robust PCA.** `robpca()` is applied to the predictors with `ncomp_max` components. It gives a robust center $\hat{\mu}_x$, the loadings P and the eigenvalues l_a, and the scores $t_i = P^T(x_i - \hat{\mu}_x)$.
2. **Robust regression.** The responses are regressed on the first `ncomp` scores. With one response, this is the reweighted least trimmed squares (LTS) regression of `robustbase::ltsReg()`. With several responses, it is the MCD regression of Rousseeuw, Van Aelst, Van Driessen and Agullo (2004): least squares from the reweighted MCD of the scores and responses, then least squares on the observations whose residual distance is below $\sqrt{\chi^2_{q,0.99}}$.

The regression coefficients are $B = P_k A_k$, from the loadings P_k and the slopes A_k of the regression on the first ncomp scores.

Diagnostics. As for rsimpls(): the score distance $SD_i = \sqrt{\sum_a t_{ia}^2/l_a}$ has the cut-off $\sqrt{\chi^2_{k,0.975}}$, the residual distance (with one response, the absolute standardized residual) the cut-off $\sqrt{\chi^2_{q,0.975}}$, and the orthogonal distance $OD_i = \|x_i - \hat{\mu}_x - Pt_i\|$ the cut-off of robpca(). Observations beyond the score cut-off only are **good leverage** points, beyond the residual cut-off only **vertical outliers**, and beyond both **bad leverage** points. The robust R^2 of the model (R2) is computed on the observations whose orthogonal and residual distances are both below their cut-offs.

Number of components. components gives, for 1 to ncomp_max components, the robust R^2 and root mean squared error on the observations that are regular in every one of these models; use robust_rmsecv() for the robust cross-validated error. These models all come from the same robust PCA, with ncomp_max components, of which they use the first ncomp, so the model also depends on ncomp_max. With ncomp_max = ncomp, the robust PCA has ncomp components, as in the paper.

Differences from the paper.

- The models with fewer components than ncomp_max use the first components of the robust PCA with ncomp_max components (see above).
- The residual covariance of the MCD regression is multiplied by the consistency factor $0.99/P(\chi^2_{q+2} \leq \chi^2_{q,0.99})$, as the observations beyond the cut-off are left out.

This is an independent implementation of the published description. robpca(), the LTS and the MCD use random subsets, so use set.seed() for reproducible results.

Value

An object of class specproc_rpcr, a list with:

- coefficients, intercept: the regression coefficients (a $p \times q$ matrix) and intercepts.
- x_loadings, x_scores, eigenvalues: the loadings P, scores T and eigenvalues of the ncomp components.
- y_loadings: the slopes A of the regression of the responses on the scores (ncomp rows).
- center: the robust center of the predictors.
- fitted, residuals: the fitted values and residuals.
- sigma: the robust covariance matrix of the residuals.
- sd, rd, od: the score, residual and orthogonal distances of each observation, and cutoff_sd, cutoff_rd, cutoff_od their cut-offs.
- R2: the robust R^2 of the model (see Details).
- outlier_type: a factor classifying each observation as "regular", "good leverage", "vertical outlier" or "bad leverage".
- weights: 1 for the observations of the final (reweighted) regression, 0 for the others.
- components: a tibble with the robust R2 and RMSE of the models with 1 to ncomp_max components.
- models: the coefficients and intercept of the models with 1 to ncomp_max components, for predictions with another number of components (predict(fit, newdata, ncomp =)).
- ncomp, ncomp_max, h, alpha: the settings used, and regression, "LTS" or "MCD".

Use predict() for the predictions of new observations.

Author(s)

Christian L. Goueguel

References

- Hubert, M., Verboven, S. (2003). A robust PCR method for high-dimensional regressors. Journal of Chemometrics, 17(8-9):438-452.
- Rousseeuw, P.J., Van Aelst, S., Van Driessen, K., Agullo, J. (2004). Robust multivariate regression. Technometrics, 46(3):293-305.
- Hubert, M., Rousseeuw, P.J., Vanden Branden, K. (2005). ROBPCA: a new approach to robust principal component analysis. Technometrics, 47(1):64-79.

See Also

predict.specproc_rpcr(), plot_outlier_map(), robust_rmsecv(), rsimpls(), robpca()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]  # Ca II and Ca I lines
cal <- 1:300
set.seed(1)
fit <- rpcr(spectra[cal, ], forageLIBS$Ca[cal], ncomp = 5)
fit
head(predict(fit, spectra[-cal, ]))
plot_outlier_map(fit)
```

---

rsimca *Robust SIMCA Classification (RSIMCA)*

Description

Robust soft independent modeling of class analogy (RSIMCA) of Vanden Branden and Hubert (2005): a robust PCA model (robpca()) of each class, and the assignment of each observation to the class whose model is the closest, from its score and orthogonal distances. It applies to high-dimensional data such as whole spectra, and outlying spectra of the training data have little influence on the class models.

Usage

```
rsimca(
  x,
  group,
  ncomp = NULL,
  ncomp_max = 10,
  alpha = 0.75,
```

```
  gamma = 0.5,
  squared = TRUE,
  var_explained = 0.8,
  prior = NULL,
  ndir = 250,
  nsamp = 500
)
```

Arguments

x	A numeric matrix or data frame of the predictors (spectra), one observation per row.
group	The classes of the observations: a factor, or a vector converted to one. Each class needs at least 5 observations.
ncomp	The number of components of the robust PCA of each class: a single number for all the classes, one per class (in the order of the levels of group, or named by them), or NULL (default) to let robpca() choose them from var_explained.
ncomp_max	The largest number of components of the robust PCA. Default is 10.
alpha	The robustness parameter of robpca(): the fraction of observations of each class assumed to be regular, between 0.5 and 1. Default is 0.75.
gamma	The weight of the orthogonal distances in the classification rule, between 0 and 1. Default is 0.5.
squared	If TRUE (default), the rule combines the squared scaled distances (R2 of the paper); otherwise the scaled distances (R1).
var_explained	The proportion of variance explained used by robpca() to choose the number of components when ncomp is NULL. Default is 0.8.
prior	The membership (prior) probabilities of the classes, used to weight the overall misclassification rate, in the order of the levels of group (or named by them). Default is the proportions of the regular training observations in each class.
ndir, nsamp	The number of random directions of the outlyingness and of random subsets of FAST-MCD in robpca(). Defaults are 250 and 500.

Details

Each class is modeled by robpca() with ncomp components (or the number chosen by robpca() from var_explained). For an observation x and the model of class j, the score distance $SD_j(x)$ and the orthogonal distance $OD_j(x)$ are divided by their cut-offs, so that 1 marks the boundary of the class. The observation is assigned to the class with the smallest combined distance

$$D_j(x) = \gamma \left(\frac{OD_j(x)}{c_{OD,j}} \right)^2 + (1-\gamma) \left(\frac{SD_j(x)}{c_{SD,j}} \right)^2$$

(rule R2 of the paper), or with the distances instead of their squares when squared = FALSE (rule R1). gamma weights the orthogonal distances against the score distances.

An observation whose two scaled distances exceed 1 for every class is an outlier for all of them (outlying): it is still assigned to the closest class, but probably belongs to none.

`plot_coomans()` draws the distances to two classes against each other (Coomans plot), and `plot_outlier_map(fit$models` the score and orthogonal distances to the robust PCA model of class `j`.

The misclassification rates are estimated on the regular observations of the training data (those regular in the robust PCA of their class), and the overall rate is weighted by the membership probabilities, by default the proportions of these observations in each class. They tend to underestimate the error: estimate it on test data or by cross-validation.

In tidymodels, `rsimca()` is the `"rsimca"` engine of the `simca()` parsnip model, where `ncomp` (`num_comp`), `gamma` and `squared` can be tuned.

Differences from the paper. The number of components of each class is given (`ncomp`) or chosen by `robpca()` from the proportion of variance explained, instead of by robust cross-validation (PRESS).

This is an independent implementation of the published description. `robpca()` uses random directions and subsets, so use `set.seed()` for reproducible results.

Value

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

- `models`: the `robpca()` model of each class.
- `ncomp`: the number of components of each class model.
- `distances`: a tibble with the combined distance D_j of each training observation to each class.
- `group`: the classes of the training observations, and `fitted`, the classes assigned to them.
- `weights`: 1 for the observations regular in the model of their class, 0 for the others.
- `outlying`: `TRUE` for the observations outlying for every class.
- `misclassification`: a tibble with the misclassification rate of each class on its regular training observations, and the overall rate.
- `prior`, `gamma`, `squared`, `levels`, `alpha`.

Use predict() for the classes and distances of new observations.

Author(s)

Christian L. Goueguel

References

- Vanden Branden, K., Hubert, M. (2005). Robust classification in high dimensions based on the SIMCA method. Chemometrics and Intelligent Laboratory Systems, 79(1-2):10-21.
- Hubert, M., Rousseeuw, P.J., Vanden Branden, K. (2005). ROBPCA: a new approach to robust principal component analysis. Technometrics, 47(1):64-79.

See Also

`predict.specproc_rsimca()`, `plot_coomans()`, `simca()`, `robpca()`, `robust_da()`

Examples

```
# iris: train on 100 flowers, predict the 50 others
set.seed(1)
train <- sample(nrow(iris), 100)
fit <- rsimca(iris[train, 1:4], iris$Species[train], ncomp = 2)
fit
table(predicted = predict(fit, iris[-train, 1:4]), true = iris$Species[-train])
plot_coomans(fit, classes = c("versicolor", "virginica"),
             newdata = iris[-train, 1:4], group = iris$Species[-train])
```

---

rsimpls *Robust Partial Least Squares Regression (RSIMPLS)*

Description

Robust PLS regression by the RSIMPLS algorithm of Hubert and Vanden Branden (2003): the SIMPLS algorithm (de Jong, 1993) computed from a robust covariance matrix of the predictors and the responses, followed by a robust regression of the responses on the scores. Outlying spectra and wrong reference values have little influence on the model, and the regression outlier map (`plot_outlier_map()`) classifies them.

Usage

```
rsimpls(x, y, ncomp, ncomp_max = 10, alpha = 0.75, ndir = 250, nsamp = 500)
```

Arguments

`x`	A numeric matrix or data frame of the predictors (spectra), one observation per row.
`y`	A numeric vector, matrix or data frame of the responses, with one row per observation.
`ncomp`	The number of components of the model.
`ncomp_max`	The largest number of components considered. ROBPCA is applied with `ncomp_max` plus the number of responses components, so the model also depends on `ncomp_max` (see Details). Default is 10, as in the paper; it is raised to `ncomp` if needed, and lowered when there are too few observations or variables (at most one less than the number of variables).
`alpha`	The robustness parameter of `robpca()`: the fraction of observations assumed to be regular, between 0.5 and 1. Default is 0.75.
`ndir`	The number of random directions of the outlyingness in `robpca()`. Default is 250.
`nsamp`	The number of random subsets of FAST-MCD in `robpca()`. Default is 500.

Details

The algorithm follows the published description in three steps:

1. **Robust covariance.** `robpca()` is applied to $Z = (X, Y)$ with $k_0 = k_{max} + q$ components, where q is the number of responses. It gives a robust center $\hat{\mu}_z$ and a robust covariance matrix $\hat{\Sigma}_z = PLP^T$, of rank k_0. The observations that are regular in this fit (both distances below their cut-offs) get the weight $w_i = 1$, the others $w_i = 0$.
2. **Robust scores.** The SIMPLS weight vectors r_a are computed from the blocks $\hat{\Sigma}_{xy}$ and $\hat{\Sigma}_x$ of $\hat{\Sigma}_z$ instead of the empirical covariance matrices, and the scores are $t_{ia} = (x_i - \hat{\mu}_x)^T r_a$.
3. **Robust regression.** The responses are regressed on the scores by least squares on the observations with $w_i = 1$ (the ROBPCA regression of the paper). The observations whose residual distance $RD_i = \sqrt{r_i^T \hat{\Sigma}_e^{-1} r_i}$ exceeds $\sqrt{\chi^2_{q,0.975}}$ are set aside, and the regression is recomputed by least squares on the others (reweighting).

The regression coefficients are $B = R_k A_k$, from the weight vectors R_k and the slopes A_k of the regression on the first `ncomp` scores.

Diagnostics. The score distance SD_i is the Mahalanobis distance of the scores, with the center and covariance of the scores of the observations with $w_i = 1$, and its cut-off is $\sqrt{\chi^2_{k,0.975}}$. The residual distance has the cut-off $\sqrt{\chi^2_{q,0.975}}$; with one response, it is the absolute standardized residual. Observations beyond the score cut-off only are **good leverage** points, beyond the residual cut-off only **vertical outliers**, and beyond both **bad leverage** points.

The orthogonal distance $OD_i = \|x_i - \hat{\mu}_x - Pt_i\|$ is the norm of the residual of the spectrum, the part that the `ncomp` components do not describe, and its cut-off is that of `robpca()`. Observations beyond it are orthogonal outliers in the predictor space, which the regression outlier map does not show; the score outlier map (`plot_outlier_map(fit, map = "score")`) does. The robust R^2 of the model (`R2`) is computed on the observations whose orthogonal and residual distances are both below their cut-offs; with several responses, it is one minus the ratio of the determinants of the residual and total sums of squares and cross-products.

Number of components. `components` gives, for 1 to `ncomp_max` components, the robust R^2 of the paper (Remark 7) and the root mean squared error, on the observations that are regular in every one of these models. They describe the fit to the calibration data; for predictions, choose `ncomp` by cross-validation: with `robust_rmsecv()`, which leaves the outliers out of the error, or for example with the `"rsimpls"` engine of `parsnip::pls()` in tidymodels (pls_rsimpls). These models all come from the same ROBPCA fit, with $k_0 = k_{max} + q$ components, so the model with `ncomp` components also depends on `ncomp_max`. With `ncomp_max = ncomp`, ROBPCA is applied with `ncomp` plus q components, and `components` stops at `ncomp`.

Differences from the paper.

- ROBPCA is not scale equivariant, so the scale of the responses relative to the spectra matters. Spectral intensities are usually much larger than concentrations, and a response on its own scale would play almost no part in the ROBPCA fit, which would then miss wrong reference values. The responses are therefore scaled before ROBPCA so that their total robust variance (the sum of the squared MADs, or standard deviations for variables with a zero MAD) equals that of the predictors (block scaling). The results are returned in the units of the responses.

- The cut-off of the orthogonal distances, in the ROBPCA step and of the model, is that of `robpca()` (the Wilson-Hilferty approximation of Hubert, Rousseeuw and Vanden Branden, 2005), which the paper mentions as an alternative.
- The residual covariance of the reweighted regression is multiplied by the consistency factor of the reweighted MCD, $0.975/P(\chi^2_{q+2} \leq \chi^2_{q,0.975})$, as the observations beyond the cut-off are left out.

This is an independent implementation of the published description. `robpca()` uses random directions and random subsets, so use `set.seed()` for reproducible results.

Value

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

- `coefficients`, `intercept`: the regression coefficients (a $p \times q$ matrix) and intercepts, in the units of the responses.
- `x_weights`, `x_loadings`, `x_scores`: the weight vectors R, the loadings P and the scores T of the `ncomp` components.
- `y_loadings`: the slopes A of the regression of the responses on the scores (`ncomp` rows).
- `center`, `y_center`: the robust centers of the predictors and responses.
- `fitted`, `residuals`: the fitted values and residuals.
- `sigma`: the robust covariance matrix of the residuals.
- `sd`, `rd`, `od`: the score, residual and orthogonal distances of each observation, and `cutoff_sd`, `cutoff_rd`, `cutoff_od` their cut-offs.
- `R2`: the robust R^2 of the model (see Details).
- `outlier_type`: a factor classifying each observation as `"regular"`, `"good leverage"`, `"vertical outlier"` or `"bad leverage"`.
- `weights`: 1 for the observations of the final regression (residual distance below the cut-off of the initial fit), 0 for the others.
- `robpca_weights`: the weights w_i of the ROBPCA step.
- `components`: a tibble with the robust `R2` and `RMSE` of the models with 1 to `ncomp_max` components.
- `models`: the `coefficients` and `intercept` of the models with 1 to `ncomp_max` components, for predictions with another number of components (`predict(fit, newdata, ncomp = )`).
- `ncomp`, `ncomp_max`, `h`, `alpha`: the settings used, and `y_scale`, the factor applied to the responses before ROBPCA.

Use predict() for the predictions of new observations.

Author(s)

Christian L. Goueguel

References

- Hubert, M., Vanden Branden, K. (2003). Robust methods for partial least squares regression. Journal of Chemometrics, 17(10):537-549.
- de Jong, S. (1993). SIMPLS: an alternative approach to partial least squares regression. Chemometrics and Intelligent Laboratory Systems, 18(3):251-263.
- Hubert, M., Rousseeuw, P.J., Vanden Branden, K. (2005). ROBPCA: a new approach to robust principal component analysis. Technometrics, 47(1):64-79.

See Also

predict.specproc_rsimpls(), plot_outlier_map(), step_rsimpls(), robust_rmsecv(), rpcr(), robpca()

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]  # Ca II and Ca I lines
cal <- 1:300
set.seed(1)
fit <- rsimpls(spectra[cal, ], forageLIBS$Ca[cal], ncomp = 4)
fit
head(predict(fit, spectra[-cal, ]))
plot_outlier_map(fit)
plot_outlier_map(fit, map = "score", newdata = spectra[-cal, ])
```

---

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`, the `electron_density`, and the other arguments of the fit in `settings`.

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. `step_gap_derivative()` uses it with the range 1 to 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()`, `step_gap_derivative()`

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")
```

---

select_wavelengths *Wavelength Selection for PLS Regression*

Description

Selects the spectral variables (wavelengths) that are informative for a response, from a PLS regression model:

- `"vip"`: the variable importance in projection (VIP) of each variable;
- `"sr"`: the selectivity ratio (SR) of each variable;
- `"ipls"`: forward interval PLS (iPLS), which selects contiguous spectral intervals.

With `robust = TRUE`, the PLS model is the robust model of `rsimpls()`, so that outlying spectra and wrong reference values do not drive the selection. `step_select_wavelengths()` performs the selection in a tidymodels recipe, where it is repeated on every resample.

Usage

```
select_wavelengths(
  x,
  y,
  method = c("vip", "sr", "ipls"),
  ncomp = 5,
  num_terms = NULL,
  threshold = NULL,
  recursive = FALSE,
  prop_drop = 0.25,
  intervals = 40,
  num_intervals = NULL,
  folds = 5,
  robust = FALSE,
  ...
)
```

Arguments

x	A numeric matrix or data frame of the spectra, one observation per row and the variables in wavelength order.
y	A numeric vector of the response.
method	The selection method: "vip" (default), "sr" or "ipls".
ncomp	The number of PLS components of the model that ranks the variables ("vip", "sr"), or the largest number of components of the interval models ("ipls"). Default is 5.
num_terms	The number of variables to keep ("vip", "sr"). If NULL (default), the variables above threshold are kept.
threshold	The importance above which variables are kept when num_terms = NULL. If NULL (default), 1 for VIP; SR has no default threshold.
recursive	If TRUE, remove the variables by backward elimination ("vip", "sr"; requires num_terms). Default is FALSE.
prop_drop	The fraction of the remaining variables removed at each round of backward elimination. Default is 0.25.
intervals	The number of contiguous intervals ("ipls"). Default is 40.
num_intervals	The number of intervals to select ("ipls"). If NULL (default), intervals are added while the RMSECV decreases.
folds	The number of cross-validation groups ("ipls"). Default is 5.
robust	If TRUE, use the robust PLS model of rsimpls(). Default is FALSE.
...	Further arguments of rsimpls() when robust = TRUE: alpha, ndir and nsamp.

Details

VIP (Wold, Johansson and Cocchi, 1993) sums the squared PLS weights of each variable over the components, weighted by the variance of the response that each component explains:

$$VIP_j = \sqrt{p \sum_a SSY_a w_{ja}^2 / \sum_a SSY_a}.$$

The weights are the orthonormal PLS weights of the NIPALS algorithm (an orthonormal basis of the SIMPLS weights, which span the same nested subspaces). The squared VIP values average 1, so the usual rule $VIP > 1$ keeps the variables of above-average importance. The PLS weights favor variables of large variance, so on unscaled spectra a strong line unrelated to the response can get a large VIP through the later components.

SR (Rajalahti et al., 2009) splits each variable, by target projection on the normalized regression vector $w_{TP} = b/\|b\|$, into a part explained by the predictive component $t_{TP} = Xw_{TP}$ and a residual, and divides their variances. Unlike VIP, it does not depend on the intensity of the variable: a weak emission line that follows the response as closely as a strong one gets the same ratio, which matters for unscaled spectra. The ratios depend on how much of the variance of each variable is unrelated to the response (in LIBS, shot-to-shot fluctuations), so there is no general threshold: in the forageLIBS spectra, the largest ratio for calcium, at the Ca II 317.9 nm line, is about 0.5.

Both keep the num_terms variables with the largest values or, if num_terms = NULL, those above threshold (by default 1 for VIP; SR needs num_terms or threshold). With recursive = TRUE (backward variable elimination), the model is refitted on the remaining variables and the fraction prop_drop of the least important ones is removed, until num_terms variables are left, so that the importance is recomputed without the variables already removed.

iPLS (Nørgaard et al., 2000) splits the variables into intervals contiguous intervals of nearly equal sizes (in the order of the columns, which should be sorted by wavelength). Starting from none, it adds at each step the interval that gives the lowest cross-validated error (RMSECV) of a PLS model on the intervals selected so far, with the best number of components up to ncomp. It stops after num_intervals intervals or, if num_intervals = NULL, when no interval lowers the RMSECV. The observations are split into folds interleaved groups (observation i in group (i - 1) %% folds + 1); average replicate spectra of a sample first, or they will fall in different groups. When a parallel plan is set with future::plan() (and the future.apply package is installed), the candidate intervals of each step are evaluated in parallel; within tune::tune_grid(), which runs the resamples in parallel, they are evaluated sequentially.

Robust selection. With robust = TRUE, VIP and SR are computed from an rsimpls() model, on the observations of its final regression (those that are not regression outliers; good leverage points are kept). For iPLS, an rsimpls() model of all the variables identifies these observations once, and the intervals are selected by classical PLS on them, so that outliers affect neither the models nor the RMSECV. Refitting a robust model for every candidate interval would be about 35 times slower. Use set.seed() for reproducible results.

The selection uses the response, so it must be estimated on training data only: to assess a model built on the selected variables, select within each resample with step_select_wavelengths().

Value

An object of class specproc_wavelength_selection, a list with:

- selected: the names of the selected variables, in column order.
- importance: the VIP or SR of every variable (from the model of all the variables), or for iPLS the RMSECV of the interval of each variable alone.
- intervals: for iPLS, a tibble with one row per interval: its first and last variables, size, RMSECV alone, and step, the step at which it was selected (NA if not selected).
- path: a tibble of the selection path: the number of variables left after each round of backward elimination, or for iPLS the interval added at each step, with the RMSECV and number of components.
- method, ncomp, robust, threshold (when used), observations (the number of observations used; with robust = TRUE, those of the robust regression) and mean_spectrum (for plot_wavelength_selection()).

Author(s)

Christian L. Goueguel

References

- Wold, S., Johansson, E., Cocchi, M. (1993). PLS: partial least squares projections to latent structures. In Kubinyi, H. (ed.), 3D QSAR in Drug Design: Theory, Methods and Applications, 523-550. ESCOM, Leiden.
- Rajalahti, T., Arneberg, R., Berven, F.S., Myhr, K.-M., Ulvik, R.J., Kvalheim, O.M. (2009). Biomarker discovery in mass spectral profiles by means of selectivity ratio plot. Chemometrics and Intelligent Laboratory Systems, 95(1):35-48.
- Nørgaard, L., Saudland, A., Wagner, J., Nielsen, J.P., Munck, L., Engelsen, S.B. (2000). Interval partial least-squares regression (iPLS): a comparative chemometric study with an example from near-infrared spectroscopy. Applied Spectroscopy, 54(3):413-419.
- Mehmood, T., Liland, K.H., Snipen, L., Sæbø, S. (2012). A review of variable selection methods in partial least squares regression. Chemometrics and Intelligent Laboratory Systems, 118:62-69.

See Also

`step_select_wavelengths()`, `plot_wavelength_selection()`, `rsimpls()`

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]  # Ca II and Ca I lines
sel <- select_wavelengths(spectra, forageLIBS$Ca, method = "sr", num_terms = 50)
sel
head(sel$selected)
select_wavelengths(spectra, forageLIBS$Ca, method = "ipls", intervals = 10)
```

---

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)
```

`simca` *SIMCA Classification Model (parsnip)*

Description

`simca()` defines a soft independent modeling of class analogy (SIMCA) classification model for tidymodels: a principal component model of each class, and the assignment of each observation to the closest class. Its engine `"rsimca"` (the default) is the robust SIMCA of `rsimca()`, whose class models resist outlying spectra of the training data.

Usage

```
simca(
  mode = "classification",
  num_comp = NULL,
  gamma = NULL,
  squared = NULL,
  engine = "rsimca"
)

## S3 method for class 'simca'
update(
  object,
  parameters = NULL,
  num_comp = NULL,
  gamma = NULL,
  squared = NULL,
  fresh = FALSE,
  ...
)
```

Arguments

mode	The type of model: "classification", the only one.
num_comp	The number of components of the PCA model of each class, or NULL (default) to let the engine choose them.
gamma	The weight of the orthogonal distances in the classification rule, between 0 and 1, or NULL (default) for 0.5.
squared	A logical: combine the squared scaled distances (TRUE) or the scaled distances (FALSE) in the classification rule, or NULL (default) for TRUE.
engine	The computational engine: "rsimca" (default).
object	A simca model specification.
parameters	A one-row tibble or named list of main parameters to update, such as those returned by tune::select_best().
fresh	If TRUE, the arguments replace those of object; if FALSE (default), they update them.
...	Not used.

Details

simca() is a parsnip model specification: it is fitted with parsnip::fit() or in a workflow, and can be tuned and resampled like any parsnip model. It needs the parsnip package, and the "rsimca" engine is available once parsnip and specProc are both loaded.

Value

A model specification of classes simca and model_spec.

Main arguments

- num_comp is the number of components of the robust PCA of each class (ncomp of rsimca()), the same for all the classes. With NULL (the default), each class gets the number chosen by robpca() from the proportion of variance explained.
- gamma is the weight of the orthogonal distances, against the score distances, in the classification rule (gamma of rsimca()). With NULL (the default), it is 0.5.
- squared chooses the classification rule (squared of rsimca()): the squared scaled distances (TRUE, rule R2 of the paper) or the scaled distances (FALSE, rule R1). With NULL (the default), it is TRUE.

All three can be tuned with tune::tune(), using dials::num_comp(), simca_gamma() and simca_squared().

Engine arguments

The other arguments of rsimca() are set with parsnip::set_engine(): ncomp_max, alpha, var_explained, prior, ndir and nsamp.

Predictions

predict() gives the classes (type = "class"), or with type = "raw" the combined distances to the classes and whether each observation is outlying for all of them (predict.specproc_rsimca()). SIMCA does not give class probabilities: when tuning, use metrics of the predicted classes, such as yardstick::metric_set(yardstick::accuracy, yardstick::kap).

The fitted rsimca() model is returned by parsnip::extract_fit_engine(). rsimca() uses random directions and subsets, so use set.seed() before fitting for reproducible results.

Author(s)

Christian L. Goueguel

See Also

rsimca(), predict.specproc_rsimca(), simca_gamma(), simca_squared()

Examples

```
library(parsnip)
spec <- simca(num_comp = 2, gamma = 0.5) |>
  set_engine("rsimca", alpha = 0.75)
spec
# iris: train on 100 flowers, predict the 50 others
set.seed(1)
train <- sample(nrow(iris), 100)
fit <- fit(spec, Species ~ ., data = iris[train, ])
table(predict(fit, iris[-train, ])$.pred_class, iris$Species[-train])
head(predict(fit, iris[-train, ], type = "raw"))
```

simca_gamma *Tuning Parameters of the SIMCA Model*

Description

dials parameters for the classification rule of the `simca()` parsnip model (and of `rsimca()`):

- `simca_gamma()`, for `gamma`, the weight of the orthogonal distances against the score distances;
- `simca_squared()`, for `squared`, the combination of the squared scaled distances (rule R2) or of the scaled distances (rule R1).

Usage

```
simca_gamma(range = c(0, 1), trans = NULL)

simca_squared(values = c(TRUE, FALSE))
```

Arguments

`range`	A two-element vector with the range of `gamma`. Default is `c(0, 1)`.
`trans`	A transformation object from the scales package, or `NULL` (default) for none.
`values`	The values of `squared`. Default is `c(TRUE, FALSE)`.

Value

A `quant_param` object (`simca_gamma()`) or a `qual_param` object (`simca_squared()`).

See Also

`simca()`, `rsimca()`

Examples

```
simca_gamma()
dials::value_seq(simca_gamma(), 5)
simca_squared()
```

slope_bias *Slope and Bias Correction of Predictions*

Description

Corrects the predictions of a model applied to spectra of another instrument (or measured under other conditions) by a straight line, estimated on transfer samples whose reference values are known.

Usage

```
slope_bias(predicted, reference, slope = TRUE)
```

Arguments

predicted	The predictions of the model for the secondary spectra of the transfer samples.
reference	Their reference values (or the predictions for their primary spectra).
slope	A logical: correct the slope and the bias (TRUE, default), or the bias only.

Details

The model of the primary instrument, applied to the secondary spectra of the transfer samples, gives predictions $\hat{y}$; the reference values y are regressed on them, $y = a + b\,\hat{y}$, and the predictions of new secondary spectra are corrected likewise. With slope = FALSE, only the bias is corrected ($b = 1$). The correction leaves the spectra and the model unchanged, so it suits differences that affect all samples alike; it cannot correct differences of spectral shape that change the ranking of the samples (see ds(), sst() or pds()). The tests of the bias against 0 and of the slope against 1 tell whether the correction is needed.

Value

An object of class specproc_slope_bias, a list with the coefficients (a tibble with the bias and slope, their standard errors, the values they are tested against and the p-values), the number of transfer samples n, and the root mean square errors before and after correction (rmse). Use predict() to correct new predictions.

Author(s)

Christian L. Goueguel

References

- Feudale, R.N., Woody, N.A., Tan, H., Myles, A.J., Brown, S.D., Ferré, J. (2002). Transfer of multivariate calibration models: a review. Chemometrics and Intelligent Laboratory Systems, 64(2):181-192.

See Also

predict.specproc_slope_bias(), ds(), sst(), pds()

Examples

```
# predictions of the potassium content of transfer samples on a second
# instrument, 10% too high and offset by 0.2%
set.seed(1)
reference <- runif(20, 0.5, 4)
predicted <- 0.2 + 1.1 * reference + rnorm(20, sd = 0.05)
correction <- slope_bias(predicted, reference)
correction
predict(correction, c(1.3, 2.4, 3.5))
```

---

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, drop.na = deprecated())
```

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.
drop.na	**[Deprecated]** Use drop_na.

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", color_by = 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)
# the 380-430 nm window (Ca II H and K lines), faster than all the channels
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- forageLIBS[which(wl > 380 & wl < 430)]
set.seed(1)
som_stability(spectra, runs = 5)$stability
```

---

specproc-tidy *Tidy and Glance Methods for Fits, Calibrations and Plasma Diagnostics*

Description

`tidy()` summarizes the estimates of a fit, a calibration or a plasma diagnostic in a tibble, one row per estimate, and `glance()` summarizes the fit in a one-row tibble, following the conventions of the broom package. The tibbles of several spectra or samples can then be bound together.

Usage

```
## S3 method for class 'specproc_peak_fit'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'specproc_peak_fit'
glance(x, ...)

## S3 method for class 'specproc_calibration'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

```
## S3 method for class 'specproc_calibration'
glance(x, ...)

## S3 method for class 'specproc_wavelength_calibration'
tidy(x, ...)

## S3 method for class 'specproc_wavelength_calibration'
glance(x, ...)

## S3 method for class 'specproc_slope_bias'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'specproc_slope_bias'
glance(x, ...)

## S3 method for class 'specproc_boltzmann'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'specproc_boltzmann'
glance(x, ...)

## S3 method for class 'specproc_cflibs'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'specproc_cflibs'
glance(x, ...)

## S3 method for class 'specproc_uncertainty'
tidy(x, ...)

## S3 method for class 'specproc_uncertainty'
glance(x, ...)

## S3 method for class 'specproc_curve_of_growth'
tidy(x, ...)

## S3 method for class 'specproc_curve_of_growth'
glance(x, ...)
```

Arguments

x	A peak fit, or an object returned by calibration_curve(), wavelength_calibration(), slope_bias(), boltzmann(), saha_boltzmann(), cf_libs(), plasma_uncertainty() or curve_of_growth().
conf.int	A logical: add the confidence interval (FALSE, default).
conf.level	The confidence level of the interval. Default is 0.95.
...	Not used.

Details

`tidy()` returns the columns `term`, `estimate` and `std.error`, with the `statistic` and `p.value` of a t test where one applies, and, with `conf.int = TRUE`, the bounds `conf.low` and `conf.high` of the confidence interval at `conf.level`:

- a peak fit (an element of the `fit` column of `peak_fit()` or `multipeak_fit()`): the parameters of each line and the common shifts, as in `tidied`; the test is that of a zero value;
- `calibration_curve()`: the `intercept`, `slope` and `quadratic` term, with the test of a zero value;
- `wavelength_calibration()`: the coefficients of the polynomial correction of each `segment`, in powers of the wavelength minus the `center` of the segment (nm), without standard errors;
- `slope_bias()`: the `bias` and `slope`, with the test of no correction (a bias of 0, a slope of 1);
- `boltzmann()` and `saha_boltzmann()`: the `temperature` (K) and the `intercept` of the plot; the interval of the temperature is that of the slope, transformed, so it is not symmetric (its upper bound is infinite when the interval of the slope reaches zero);
- `cf_libs()`: the `temperature` (with an interval as above), and the `atomic_fraction` and `mass_fraction` of each `element`, without standard errors (see `plasma_uncertainty()`);
- `plasma_uncertainty()`: the same quantities with their Monte Carlo standard uncertainties, and always their intervals, at the `level` of the object;
- `curve_of_growth()`: the column density of each species (`term`), with its number of lines (`n.lines`) and its largest optical depth (`max.optical.depth`).

`glance()` returns:

- a peak fit: `sigma`, the root mean square error `rmse`, `isConv`, `finTol`, `logLik`, `AIC`, `BIC`, `deviance`, `df.residual` and `nobs`, as for `nls` fits in broom (with `rmse`);
- `calibration_curve()`: `nobs`, `r.squared`, `sigma`, `df.residual`, the `sensitivity`, the `critical.value` (ISO 11843 only), `lod`, `loq`, the reduced `chi.squared` of a Deming or York fit, the p-values of Mandel's test (`mandel.p.value`) and of the lack-of-fit test (`lack.of.fit.p.value`), the `regression` and the `model`;
- `wavelength_calibration()`: the number of reference lines `n.lines`, of lines used `nobs`, of `n.segments`, the `degree`, and the `rmse` and largest absolute residual `max.residual` of the lines used (nm);
- `slope_bias()`: `nobs`, the root mean square differences to the reference before (`rmse.before`) and after (`rmse.after`) the correction, and whether the slope was fitted (`slope.fitted`);
- `boltzmann()` and `saha_boltzmann()`: `r.squared`, `sigma`, `nobs`, `df.residual`, the range of the abscissa `energy.range` (eV), the `electron.density` and number of `iterations` (Saha-Boltzmann only), the `method` and the `units`;
- `cf_libs()`: `r.squared` (about the mean), `sigma`, `nobs`, `df.residual` (`NA` when the temperature is given), the numbers of `n.species` and `n.elements`, the `electron.density`, the number of `iterations`, the `method` and the `units`;
- `plasma_uncertainty()`: the number of draws `n.draws`, of `failed` draws, the reduced `chi.squared` of the fit, the `level` and the `method`;
- `curve_of_growth()`: the `temperature`, the `rmse` of the fit (on the log scale), `nobs`, the `scale` and the `units`.

Value

A tibble.

Author(s)

Christian L. Goueguel

See Also

specproc-tidy-models for the chemometric models, `peak_fit()`, `calibration_curve()`, `boltzmann()`, `cf_libs()`, `plasma_uncertainty()`

Examples

```
data(forageLIBS)
mean_spectrum <- colMeans(forageLIBS[-(1:14)])
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 <- saha_boltzmann(lines, ionization_energy = 6.113, electron_density = 1.9e17)
tidy(fit, conf.int = TRUE)
glance(fit)

# the K I 769.90 nm line, normalized to the C I 247.86 nm line of the matrix
k <- line_intensities(forageLIBS[-(1:14)], c(C = 247.856, K = 769.896), baseline = TRUE)
standards <- data.frame(K = forageLIBS$K,
                        signal = k$intensity[k$line == "K"] / k$intensity[k$line == "C"])
cal <- calibration_curve(standards, signal, K)
tidy(cal)
glance(cal)
```

`specproc-tidy-models` *Tidy and Glance Methods for Chemometric Models*

Description

`tidy()` lists the estimates of a chemometric model in a long tibble, and `glance()` summarizes the model in a one-row tibble, following the conventions of the broom package. Through parsnip, they also apply to the fitted models and workflows of the `"rsimpls"`, `"lts"`, `"rsimca"` and `"mcd"` engines.

Usage

```
## S3 method for class 'specproc_robpca'
tidy(x, matrix = "eigenvalues", ...)

## S3 method for class 'specproc_robpca'
glance(x, ...)

## S3 method for class 'specproc_rsimpls'
tidy(x, matrix = "coefficients", ...)

## S3 method for class 'specproc_rsimpls'
glance(x, ...)

## S3 method for class 'specproc_rpcr'
tidy(x, matrix = "coefficients", ...)

## S3 method for class 'specproc_rpcr'
glance(x, ...)

## S3 method for class 'specproc_lts'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

## S3 method for class 'specproc_lts'
glance(x, ...)

## S3 method for class 'specproc_rsimca'
tidy(x, ...)

## S3 method for class 'specproc_rsimca'
glance(x, ...)

## S3 method for class 'specproc_robust_da'
tidy(x, ...)

## S3 method for class 'specproc_robust_da'
glance(x, ...)

## S3 method for class 'specproc_opls'
tidy(x, matrix = "coefficients", ...)

## S3 method for class 'specproc_opls'
glance(x, ...)

## S3 method for class 'specproc_som'
tidy(x, ...)

## S3 method for class 'specproc_som'
glance(x, ...)
```

```
## S3 method for class 'specproc_wavelength_selection'
tidy(x, ...)

## S3 method for class 'specproc_wavelength_selection'
glance(x, ...)

## S3 method for class 'specproc_nas'
tidy(x, ...)

## S3 method for class 'specproc_nas'
glance(x, ...)

## S3 method for class 'specproc_domain'
tidy(x, ...)

## S3 method for class 'specproc_domain'
glance(x, ...)
```

Arguments

x	An object returned by robpca(), rospca(), macropca(), cellpca(), rsimpls(), rpcr(), lts_fit(), rsimca(), robust_da(), opls(), som(), select_wavelengths(), nas() or applicability_domain().
matrix	The table to return (see Details).
...	Not used.
conf.int	A logical: add the confidence interval (FALSE, default).
conf.level	The confidence level of the interval. Default is 0.95.

Details

tidy() returns, choosing the table with matrix when a model has several:

- robust PCA (robpca(), rospca(), macropca(), cellpca()): "eigenvalues" (default), one row per component PC with its eigenvalue and std.dev; "loadings", the value of each variable (column) on each PC; "scores", the value of each observation (row) on each PC, as tidy() of a prcomp object in broom (whose names of the tables are also accepted, such as "rotation" and "x");
- rsimpls() and rpcr(): "coefficients" (default), the estimate of each term (the intercept and the variables) for each response, in the units of the responses; "components", the robust r.squared and the rmse of the models with 1 to ncomp_max components (ncomp); "loadings" and "scores", as for robust PCA;
- lts_fit(): the term, estimate, std.error, statistic and p.value of the reweighted least-squares fit, and, with conf.int = TRUE, its confidence interval (conf.low, conf.high);
- rsimca() and robust_da(): one row per class, with its prior, the number n of its regular observations and their misclassification rate error, and, for RSIMCA, its number of components ncomp and the cutoffs of the score and orthogonal distances (cutoff.sd, cutoff.od);

- opls(): "coefficients" (default), the estimate of each term (variable) with its vip and ortho.vip; "components", the explained variances r2x, r2y and q2 of each component, their cumulative values (r2x.cum, ...) and their significance;
- som(): one row per unit of the map, with its row and col in the grid, its position (x, y), the number n of observations it represents and their mean quantization error qe;
- select_wavelengths(): one row per variable (term), with its importance, whether it is selected, and, for iPLS, its interval;
- nas(): the regression vector, the estimate of each term;
- applicability_domain(): one row per component PC of the PCA, with its std.dev, percent of the variance and cumulative percent (as fractions).

glance() returns the number of observations nobs and of variables nvar, and:

- robust PCA: the number of components ncomp, h, alpha, the cutoffs cutoff.sd and cutoff.od, the numbers of n.good.leverage, n.orthogonal.outlier and n.bad.leverage observations, the number of flagged.cells (MacroPCA and cellPCA) and the method;
- rsimpls() and rpcr(): ncomp, ncomp_max, h, alpha, the robust r.squared, the robust scale of the residuals sigma (one response), the numbers of n.good.leverage, n.vertical.outlier and n.bad.leverage observations, the method and, for RPCR, the regression;
- lts_fit(): the robust r.squared, the robust scale of the residuals sigma, df.residual, alpha and the number of n.outliers (weight 0 in the reweighted fit);
- rsimca() and robust_da(): the number of n.classes, the overall misclassification rate error, alpha, the number of observations left out (n.outliers; for RSIMCA, those outlying for all classes) and the method;
- opls(): the numbers of predictive and orthogonal components (ncomp, ncomp.ortho), r2x.cum, r2y.cum, q2.cum, rmsee and, after a permutation test, its p-values p.r2y and p.q2;
- som(): the number of n.units, the topology, the quantization.error and topographic.error of the map, the number of n.novel observations (beyond the cutoff of the quantization error), the epochs and whether the map is robust;
- select_wavelengths(): the method, ncomp, the number of selected variables n.selected, robust, the threshold and, for iPLS, the rmsecv of the final model;
- nas(): the method, ncomp and the figures of merit (sensitivity, selectivity and, with a noise level, analytical.sensitivity, lod and loq);
- applicability_domain(): the number of components ncomp, the variance explained, the conf.level and the limits t2.limit and q.limit.

Value

A tibble.

Author(s)

Christian L. Goueguel

See Also

specproc-tidy for the fits, calibrations and plasma diagnostics

Examples

```
data(forageLIBS)
wl <- as.numeric(names(forageLIBS[-(1:14)]))
spectra <- forageLIBS[-(1:14)][, wl > 380 & wl < 430]
set.seed(1)
fit <- rsimpls(spectra[1:300, ], forageLIBS$Ca[1:300], ncomp = 4)
tidy(fit, matrix = "components")
glance(fit)
# the regression vector, one row per wavelength
head(tidy(fit))

pca <- robpca(spectra, ncomp = 3)
tidy(pca)
glance(pca)
```

---

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"))
```

`sst` *Spectral Space Transformation*

Description

Transfers spectra from a secondary instrument to the primary instrument, on which the calibration model was built, by spectral space transformation (SST; Du et al., 2011): the secondary spectra are moved, in the space of the principal components shared by both instruments, by the difference between the instruments.

Usage

```
sst(x1, x2, ncomp = NULL, variance = 0.99)
```

Arguments

`x1`	The spectra of the transfer samples on the primary instrument: a numeric matrix or data frame, one spectrum per row.
`x2`	The spectra of the same samples, in the same order, on the secondary instrument.
`ncomp`	The number of principal components. By default, chosen from `variance`.
`variance`	The fraction of the variance explained, when `ncomp` is not given. Default is 0.99.

Details

The transfer spectra of both instruments, centered by the mean of each instrument, are joined side by side, $[\mathbf{X}_1\ \mathbf{X}_2]$, and decomposed by a principal component analysis, $[\mathbf{X}_1\ \mathbf{X}_2] \approx \mathbf{T}[\mathbf{P}_1^T\ \mathbf{P}_2^T]$: the same scores $\mathbf{T}$ describe a sample on both instruments, and the loadings $\mathbf{P}_1$ and $\mathbf{P}_2$ the spectra of each. A new secondary spectrum gets the scores $t = (x_2 - \bar{x}_2)(\mathbf{P}_2^T)^+$ and is transferred by adding the difference of the instruments in that space:

$$x_{2\to1} = \bar{x}_1 + (x_2 - \bar{x}_2) + t\,(\mathbf{P}_1^T - \mathbf{P}_2^T)$$

The part of the spectrum outside the shared components is kept unchanged, so SST corrects only the differences the transfer samples show.

By default, `ncomp` is the smallest number of components that explain the fraction `variance` of the variance of the joined spectra.

Value

An object of class `specproc_transfer`. Use predict() to transfer secondary spectra.

Author(s)

Christian L. Goueguel

References

- Du, W., Chen, Z.-P., Zhong, L.-J., Wang, S.-X., Yu, R.-Q., Nordon, A., Littlejohn, D., Holden, M. (2011). Maintaining the predictive abilities of multivariate calibration models by spectral space transformation. Analytica Chimica Acta, 690(1):64-70.

See Also

`ds()`, `pds()`, `slope_bias()`, `predict.specproc_transfer()`

Examples

```
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
spectra <- as.matrix(forageLIBS[which(wl > 760 & wl < 780)])  # the K I lines
secondary <- 0.8 * cbind(spectra[, -1], spectra[, ncol(spectra)]) + 50
transfer <- sst(spectra[1:30, ], secondary[1:30, ])
transfer
transferred <- predict(transfer, secondary[31:368, ])
mean(abs(transferred - spectra[31:368, ])) / mean(spectra)
```

---

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, and between sessions with the persistent cache (see `download_cache()`).

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, `read_vald()` and `read_kurucz()` for the lines STARK-B does not have, and `stark_table()` for data from other sources.

Examples

```
# needs an internet connection; gives up after 20 s if STARK-B does not answer
ca <- try(starkb_lines("Ca II", wavelength = c(390, 400), perturber = "electron",
                       timeout = 20))
if (!inherits(ca, "try-error")) head(ca)
```

`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(), read_vald()

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,
  species = NULL
)
```

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.
`species`	The species of the line, such as `"Ca II"`. Required when `stark` has several species; by default, its only species.

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.

The line is matched by its wavelength only, so a table of several species, such as those of `read_vald()` and `read_kurucz()`, needs `species`: the line of another species within `tolerance` could be taken otherwise.

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)
```

---

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_bin_spectra *Binning Recipe Step*

Description

`step_bin_spectra()` creates a *specification* of a recipe step that replaces each group of `width` adjacent channels of the spectra by their mean, with `bin_spectra()`.

Usage

```
step_bin_spectra(
  recipe,
  ...,
  width = 3,
  segments = TRUE,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  bins = NULL,
  skip = FALSE,
  id = recipes::rand_id("bin_spectra")
)
```

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().
width	The number of adjacent channels averaged, a positive integer. Default is 3; width = 1 returns the spectra unchanged.
segments	A logical: bin the segments between gaps of the wavelength axis separately (TRUE, default). Needs wavelengths as names.
role	The role of the binned columns. Default is "predictor".
trained	A logical indicating whether the step has been trained.
columns	The names of the selected columns, stored once the step has been trained.
bins	The groups of channels and the names of the binned columns, set 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 are replaced by the fewer binned columns, named by their mean wavelength (see bin_spectra()). The groups of channels are set from the column names when the recipe is prepped; nothing is estimated from the training data. The width can be tuned with tune::tune() (dials parameter window_size()). tidy() returns the terms, width and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

bin_spectra(), step_gap_derivative(), step_spectral_norm()

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  # binning, normalization to the total area and gap-segment derivative
  rec <- recipes::recipe(K ~ ., data = forageLIBS[-c(1:10, 12:14)]) |>
    step_bin_spectra(recipes::all_predictors(), width = 3) |>
    step_spectral_norm(recipes::all_predictors(), method = "area") |>
    step_gap_derivative(recipes::all_predictors(), gap = 3, segment = 1) |>
    recipes::prep()
  dim(recipes::bake(rec, new_data = NULL))
}
```

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_gap_derivative` *Gap-Segment Derivative Recipe Step*

Description

`step_gap_derivative()` creates a *specification* of a recipe step that computes the gap-segment (Norris-Williams) derivative of the spectra with `gap_derivative()`.

Usage

```
step_gap_derivative(
  recipe,
  ...,
  derivative = 1,
  gap = 5,
  segment = 3,
  segments = TRUE,
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("gap_derivative")
)
```

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()`.
`derivative`	The derivative: 1 (default) or 2.
`gap`	The number of channels between two segments, an odd number. Default is 5.
`segment`	The number of channels averaged in each segment, an odd number. Default is 3.
`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 their derivative. Nothing is estimated from the training data. The derivative, gap and segment can be tuned with tune::tune() (dials parameters savgol_derivative(), with values 1 and 2, and window_size()); an even gap or segment, as a tuning grid may propose, is increased to the next odd number. tidy() returns the terms, derivative, gap, segment and id.

Value

An updated version of recipe with the new step added to the sequence of existing steps.

See Also

gap_derivative(), step_savgol(), step_bin_spectra()

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageLIBS)
  rec <- recipes::recipe(K ~ ., data = forageLIBS[-c(1:10, 12:14)]) |>
    step_gap_derivative(recipes::all_predictors(), gap = 5, segment = 3) |>
    recipes::prep()
  recipes::tidy(rec, number = 1)
}
```

---

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"),
  drop.offset = deprecated()
)
```

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.
drop.offset	**[Deprecated]** Use drop_offset.

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,
  scale = c("floor", "sample", "pooled"),
  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.
`scale`	The robust scale of the z-scores: `"floor"` (default), `"sample"` or `"pooled"` (see Details).
`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`, the `scale` 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()`, and the `"lts"` engine of `parsnip::linear_reg()` (linear_reg_lts) for robust principal component regression

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_boxcox *Robust Box-Cox Transformation Recipe Step*

Description

`step_robust_boxcox()` creates a *specification* of a recipe step that transforms the selected variables toward central normality with the robust Box-Cox transformation of `robust_boxcox()`. It is the robust counterpart of `recipes::step_BoxCox()`.

Usage

```
step_robust_boxcox(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  quantile = 0.99,
  nbsteps = 2,
  standardize = TRUE,
```

```
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("robust_boxcox")
)
```

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.
quantile	The quantile of the absolute value of a standard normal variable beyond which the standardized transformed values are given weight zero in the re-weighting steps, strictly between 0 and 1. Default is 0.99.
nbsteps	The number of re-weighting steps, a positive integer. Default is 2.
standardize	A logical value: standardize the transformed variables by the mean and standard deviation of the inliers (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 (Raymaekers and Rousseeuw, 2021), as in robust_boxcox(), 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 (constant ones and, with a warning, those with zero or negative values) are left unchanged. Missing values are ignored by the fit and stay missing; new values that are zero or negative become NA.

tidy() returns a tibble with columns terms, lambda (NA for the variables left unchanged) 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_boxcox(), step_robust_yeojohnson() for variables with zero or negative values.

Examples

```
library(recipes)
data(forageLIBS)
contents <- forageLIBS[c("Ca", "K", "P", "Mn")]
rec <- recipe(~ ., data = contents[1:300, ]) |>
  step_robust_boxcox(all_numeric_predictors())
prepped <- prep(rec)
tidy(prepped, number = 1)
bake(prepped, new_data = contents[301:368, ])
```

---

step_robust_gandh *Robust Tukey g-and-h Transformation Recipe Step*

Description

Creates a recipe step applying robust_gandh() to the selected variables. The rQLS fit and all standardization constants are estimated during prep() on the training data and applied unchanged during bake().

Usage

```
step_robust_gandh(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  n_quantiles = 10,
  cutoff = NULL,
  max_iter = 50,
  tol = 1e-06,
  standardize = TRUE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("robust_gandh")
)
```

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.
n_quantiles	Number of fitting quantiles, an integer from 4 to 20. Default is 10, as recommended for rQLS in the paper.
cutoff	Positive, finite Tukey biweight cutoff on the median/MAD standardized input scale. NULL (default) selects it adaptively for each variable. It is an order-statistic fitting tolerance, not a normal score cutoff or a contamination percentage.
max_iter	Maximum number of trimming/refitting iterations per cutoff. Default is 50.
tol	Positive, finite relative tolerance for convergence. Default is 1e-6.
standardize	A logical value: standardize the transformed variables by the mean and standard deviation of the inliers (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

Selected columns are replaced in place. Missing values stay missing and variables without enough observations or spread are left unchanged, as in robust_gandh(). tidy() returns terms, location, scale, g, h and id; location and scale are on the original input scale.

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_gandh(), step_robust_boxcox(), step_robust_yeojohnson()

Examples

```
data(forageLIBS)
contents <- forageLIBS[c("Ca", "K", "P", "Mn")]
rec <- recipes::recipe(~ ., data = contents[1:300, ]) |>
  step_robust_gandh(recipes::all_numeric_predictors())
trained <- recipes::prep(rec)
recipes::bake(trained, new_data = contents[301:368, ])
```

step_robust_yeojohnson *Robust Yeo-Johnson Transformation Recipe Step*

Description

`step_robust_yeojohnson()` creates a *specification* of a recipe step that transforms the selected variables toward central normality with the robust Yeo-Johnson transformation of `robust_yeojohnson()`. It is the robust counterpart of `recipes::step_YeoJohnson()`.

Usage

```
step_robust_yeojohnson(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  quantile = 0.99,
  nbsteps = 2,
  standardize = TRUE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("robust_yeojohnson")
)
```

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.
`quantile`	The quantile of the absolute value of a standard normal variable beyond which the standardized transformed values are given weight zero in the re-weighting steps, strictly between 0 and 1. Default is 0.99.
`nbsteps`	The number of re-weighting steps, a positive integer. Default is 2.
`standardize`	A logical value: standardize the transformed variables by the mean and standard deviation of the inliers (`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 (Raymaekers and Rousseeuw, 2021), as in `robust_yeojohnson()`, 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. Missing values are ignored by the fit and stay missing.

tidy() returns a tibble with columns `terms`, `lambda` (`NA` for the variables left unchanged) 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_yeojohnson()`, `step_robust_boxcox()`

Examples

```
library(recipes)
data(forageLIBS)
spectra <- snv(forageLIBS[-(1:14)])$correction
wl <- as.numeric(names(spectra))
mg <- spectra[which(wl > 279 & wl < 281)]  # the Mg II lines, after SNV
rec <- recipe(~ ., data = mg[1:300, ]) |>
  step_robust_yeojohnson(all_numeric_predictors())
prepped <- prep(rec)
tidy(prepped, number = 1)
```

---

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_rsimpls *Robust PLS (RSIMPLS) Recipe Step*

Description

`step_rsimpls()` creates a *specification* of a recipe step that converts the selected variables into robust partial least squares scores, with `rsimpls()`. It is the robust counterpart of `recipes::step_pls()`: outlying spectra and wrong reference values have little influence on the components.

Usage

```
step_rsimpls(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  outcome = NULL,
  options = list(),
  prefix = "RPLS",
  distances = FALSE,
  keep_original_cols = FALSE,
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("rsimpls")
)
```

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 PLS components. Default is 2.
outcome	The outcome variable(s), as bare names or a selector. If NULL (default), the outcomes of the recipe are used.
options	A list of further arguments passed to rsimpls(), such as ncomp_max, alpha or ndir.
prefix	The prefix of the new column names. Default is "RPLS".
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 RSIMPLS 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 RSIMPLS model is estimated on the training data, with the outcome, when the recipe is prepped, and new data are projected onto it when they are baked; the outcome is not needed then. As the step uses the outcome, it must be estimated on the training data only: in a workflow, it is re-estimated for every resample. Several outcomes give one model of all of them, as with rsimpls().

The new columns are named with prefix followed by the component number, zero-padded as in recipes::step_pls(). With distances = TRUE, two more columns hold the score distance (<prefix>_SD) and the orthogonal distance (<prefix>_OD) of each observation, which can be used to screen new spectra; the residual distance needs the outcome and is not given. 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 weight vectors, which give the scores of the centered predictors, as a tibble with columns terms, value, component and id.

See Also

rsimpls(), step_robpca()

Examples

```
library(recipes)
data(forageLIBS)
# calcium and the Ca II and Ca I lines
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "Ca"), which(wl > 380 & wl < 430))]
set.seed(1)
rec <- recipe(Ca ~ ., data = dat[1:300, ]) |>
  step_rsimpls(all_predictors(), num_comp = 4, distances = TRUE)
prepped <- prep(rec)
bake(prepped, new_data = dat[301:368, ])
tidy(prepped, number = 1)
```

---

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_select_wavelengths

Wavelength Selection Recipe Step

Description

step_select_wavelengths() creates a *specification* of a recipe step that keeps the predictors (wavelengths) selected by select_wavelengths() and removes the others. The selection is made on the training data when the recipe is prepped, so that within tune::tune_grid() it is repeated on the analysis set of every resample, as it must be: a selection made on all the data would make the cross-validated error too optimistic.

Usage

```
step_select_wavelengths(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  outcome = NULL,
  method = "vip",
  num_terms = NULL,
  threshold = NULL,
  num_comp = 5,
  recursive = FALSE,
  intervals = 40,
  num_intervals = NULL,
  robust = FALSE,
  options = list(),
  res = NULL,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("select_wavelengths")
)
```

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_terms	The number of variables to keep ("vip", "sr"). If NULL (default), the variables above threshold are kept.
threshold	The importance above which variables are kept when num_terms = NULL. If NULL (default), 1 for VIP; SR has no default threshold.
num_comp	The number of components of the selection model (ncomp of select_wavelengths()). Default is 5.
recursive	If TRUE, remove the variables by backward elimination ("vip", "sr"; requires num_terms). Default is FALSE.
intervals	The number of contiguous intervals ("ipls"). Default is 40.
num_intervals	The number of intervals to select ("ipls"). If NULL (default), intervals are added while the RMSECV decreases.
robust	If TRUE, use the robust PLS model of rsimpls(). Default is FALSE.

options	A list of further arguments of select_wavelengths(): prop_drop, folds, and alpha, ndir, nsamp with robust = TRUE.
res	The selection, 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

See select_wavelengths() for the methods: "vip" and "sr" keep the num_terms most important predictors (or those above threshold), and "ipls" selects num_intervals contiguous intervals. 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_terms, num_intervals and num_comp (the components of the selection model) can be tuned with tune::tune(). Their default ranges are dials::num_terms() with 20 to 1000 predictors, num_intervals() with 1 to 10 intervals, and dials::num_comp() with 1 to 10 components. When the model also has a num_comp argument, give the step's parameter its own id, such as num_comp = tune("select_comp").

Tidying

tidy() returns a tibble with columns terms (the selected predictors), selected, importance and id.

Author(s)

Christian L. Goueguel

See Also

select_wavelengths(), plot_wavelength_selection()

Examples

```
library(recipes)
data(forageLIBS)
wl <- suppressWarnings(as.numeric(names(forageLIBS)))
dat <- forageLIBS[c(which(names(forageLIBS) == "Ca"), which(wl > 380 & wl < 430))]
rec <- recipe(Ca ~ ., data = dat[1:300, ]) |>
  step_select_wavelengths(all_predictors(), method = "sr", num_terms = 50)
prepped <- prep(rec)
ncol(bake(prepped, new_data = dat[301:368, -1]))
```

```
tidy(prepped, 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), the wavelength at their center (see wavelet_features(); NA when the column names are not wavelengths), 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(), step_wavelet_denoise()

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_wavelet_denoise *Wavelet Denoising Recipe Step*

Description

step_wavelet_denoise() creates a *specification* of a recipe step that denoises the spectra with wavelet_denoise().

Usage

```
step_wavelet_denoise(
  recipe,
  ...,
  wavelet = "la8",
  level = 2,
  threshold = "sure",
  rule = "hard",
  segments = TRUE,
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = recipes::rand_id("wavelet_denoise")
)
```

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: "la8" (default; Daubechies' least asymmetric wavelet with 8 coefficients, whose nearly symmetric shape suits spectral lines), "haar", "d4", "d6" or "d8" (see wavelet_features()).
level	The number of levels thresholded. Default is 2: the scales of 2 to 4 channels.
threshold	The threshold: "sure" (default) or "universal".
rule	The thresholding rule: "hard" (default) or "soft".
segments	A logical: denoise 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 denoised values. Nothing is estimated from the training data: the noise and thresholds are those of each spectrum. The level can be tuned with tune::tune() (dials parameter wavelet_level(), from 1 to 4 by default). tidy() returns the terms, wavelet, level, threshold, rule and id.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps.

See Also

`wavelet_denoise()`, `step_savgol()`, `step_wavelet()`

Examples

```
if (rlang::is_installed("recipes")) {
  data(forageShots)
  rec <- recipes::recipe(K ~ ., data = forageShots[-c(1:4)]) |>
    step_wavelet_denoise(recipes::all_predictors()) |>
    recipes::prep()
  recipes::tidy(rec, number = 1)
}
```

---

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,
  drop.na = deprecated()
)
```

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.
drop.na	**[Deprecated]** Use drop_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,
  log.p = deprecated()
)
```

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.
log.p	**[Deprecated]** Use log_p.

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, drop.na = deprecated())
```

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.
drop.na	**[Deprecated]** Use drop_na.

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

```
# mad() is unbiased for the standard deviation of a normal distribution
# only in large samples. In 2000 samples of 5 replicate values from a
# standard normal distribution (standard deviation 1), mad() is low by
# about 16% on average, while umad() is unbiased
set.seed(1)
samples <- replicate(2000, rnorm(5), simplify = FALSE)
est <- data.frame(estimator = rep(c("mad()", "umad()"), each = 2000),
                  value = c(sapply(samples, mad), sapply(samples, umad)))
means <- tapply(est$value, est$estimator, mean)
round(means, 3)
# the estimates from 5 values are skewed: the peak of umad() lies below 1,
# but its mean (solid line) is at 1 (dashed line)
ggplot2::ggplot(est, ggplot2::aes(value, fill = estimator)) +
  ggplot2::geom_density(alpha = 0.4) +
  ggplot2::geom_vline(xintercept = means, colour = c("#b2182b", "#2166ac")) +
  ggplot2::geom_vline(xintercept = 1, linetype = "dashed") +
  ggplot2::scale_fill_manual(values = c("#b2182b", "#2166ac"))

# the bias of mad() fades as the samples grow: the means of each
# estimator over 1000 samples of n values
set.seed(2)
bias <- sapply(c(3, 5, 10, 30, 100), function(n) {
  s <- replicate(1000, rnorm(n), simplify = FALSE)
  c(n = n, mad = mean(sapply(s, mad)), umad = mean(sapply(s, umad)))
})
round(bias, 3)

# both are robust: two outliers inflate the standard deviation, but
# neither MAD
set.seed(1)
x <- c(rnorm(50), 10, 12)
sd(x)
mad(x)
umad(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_denoise` *Wavelet Denoising of Spectra*

Description

Removes noise from spectra by thresholding their undecimated wavelet coefficients (translation-invariant wavelet shrinkage). Unlike a smoothing filter, it keeps the height and width of narrow lines.

Usage

```
wavelet_denoise(
  x,
  wavelet = "la8",
  level = 2,
  threshold = "sure",
  rule = "hard",
  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.
`wavelet`	The wavelet: `"la8"` (default; Daubechies' least asymmetric wavelet with 8 coefficients, whose nearly symmetric shape suits spectral lines), `"haar"`, `"d4"`, `"d6"` or `"d8"` (see `wavelet_features()`).
`level`	The number of levels thresholded. Default is 2: the scales of 2 to 4 channels.
`threshold`	The threshold: `"sure"` (default) or `"universal"`.
`rule`	The thresholding rule: `"hard"` (default) or `"soft"`.
`segments`	A logical: denoise the segments between gaps of the wavelength axis separately (`TRUE`, default). Needs wavelengths as names.

Details

The spectrum is decomposed by the maximal overlap discrete wavelet transform (MODWT, Percival and Walden, 2000), the undecimated version of the transform of `wavelet_features()`: every level has one detail coefficient per channel. The details of the levels 1 to `level` are thresholded, and the spectrum is rebuilt from them and the smooth part at `level`, which keeps the continuum and the broad features. Thresholding the undecimated transform is the translation-invariant denoising of Coifman and Donoho (1995), the average of the denoised decimated transforms of all the shifts of the spectrum: unlike thresholding the decimated transform, it gives no artifacts that depend on where a line falls on the channel grid.

The noise σ_j of each level j is estimated in each spectrum as the median absolute detail of that level divided by 0.6745, which allows for noise correlated between neighboring channels (Johnstone and

Silverman, 1997). The median is not affected by the lines as long as they cover less than half of the channels. The threshold of a level is

- `threshold = "sure"` (default): the SureShrink threshold, which minimizes Stein's unbiased estimate of the risk of soft thresholding at that level, or the universal threshold when the details of the level are sparse (Donoho and Johnstone, 1995). The undecimated details are correlated, so the risk estimate is approximate;
- `threshold = "universal"`: $\sigma_j \sqrt{2 \log n}$ for n channels (Donoho and Johnstone, 1994), about 4.2 times the noise for 7000 channels. It removes nearly all the noise, but also the weak lines.

With `rule = "hard"` (default), the details above the threshold are kept unchanged, so the lines keep their height. With `rule = "soft"`, they are shrunk by the threshold: a smoother result, but lower lines.

The default settings suit spectra whose lines are a few channels wide. In the single shots of forageShots, whose lines are 2 to 3 channels wide (FWHM), the noise is correlated between neighboring channels: more than half of it is at level 2, the scale of the lines. Compared with the mean of the other shots of each sample, the defaults lowered the error of the continuum by about 10%, while the lines 3 to 15 times higher than the noise lost about 4% of their height and the strong lines 0.2%; the universal threshold lowered the weak lines by a third. A Savitzky-Golay filter of 5 channels (`savitzky_golay()`) lowered the strong lines by 10%. Because the noise has the scale of the lines, no filter separates them well in such spectra, and the PLS calibrations of the averaged spectra of forageLIBS hardly change after denoising. Spectra with wider lines and noise at finer scales gain more, with a higher `level`.

The ends of the spectrum are extended by reflection. With `segments = TRUE`, the spectrum is split between the detectors of a multi-spectrometer system, as in `savitzky_golay()`, and each segment is denoised on its own. A segment shorter than the wavelet filter at `level`, $(2^{level} - 1)(L - 1) + 1$ channels for a filter of L coefficients, is denoised with fewer levels, and a segment shorter than the filter itself is returned unchanged. A segment with missing values is returned unchanged, with a warning.

The threshold follows the typical noise of a spectrum, mostly that of the continuum. In LIBS, the noise grows with the intensity, so the noise on top of strong lines is mostly kept, and weak lines whose details do not exceed the threshold are removed with the noise. Denoising lowers the apparent noise of blank spectra: compute detection limits (as with `calibration_curve()`) from the spectra before denoising.

Value

The denoised spectra, with the same shape and names as `x` (a tibble for a data frame).

Author(s)

Christian L. Goueguel

References

- Donoho, D.L., Johnstone, I.M. (1994). Ideal spatial adaptation by wavelet shrinkage. Biometrika, 81(3):425-455.
- Donoho, D.L., Johnstone, I.M. (1995). Adapting to unknown smoothness via wavelet shrinkage. Journal of the American Statistical Association, 90(432):1200-1224.

- Coifman, R.R., Donoho, D.L. (1995). Translation-invariant de-noising. In: Antoniadis, A., Oppenheim, G. (eds), Wavelets and Statistics. Lecture Notes in Statistics, 103:125-150. Springer, New York.
- Johnstone, I.M., Silverman, B.W. (1997). Wavelet threshold estimators for data with correlated noise. Journal of the Royal Statistical Society: Series B, 59(2):319-351.
- Percival, D.B., Walden, A.T. (2000). Wavelet Methods for Time Series Analysis. Cambridge University Press, Cambridge.

See Also

`step_wavelet_denoise()`, `savitzky_golay()`, `wavelet_features()`

Examples

```
data(forageShots)
# one laser shot, around the Mn I triplet at 403 nm
shot <- unlist(forageShots[1, -(1:5)])
clean <- wavelet_denoise(shot)
wl <- as.numeric(names(shot))
keep <- wl > 400 & wl < 408
plot(wl[keep], shot[keep], type = "l", col = "grey", xlab = "Wavelength (nm)",
     ylab = "Intensity")
lines(wl[keep], clean[keep], col = "blue")
```

---

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). Their names are the wavelengths.
`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).

Each coefficient is a weighted sum of a window of channels, which widens with the level, and is placed at the center of that window: the centroid of the weights for the approximation, their center of energy for the details. The window of a coefficient is not centered on its block of 2^{level} channels (about 37 channels later for `"la8"` at level 4). When the names of the channels are wavelengths, the wavelengths at these centers are returned as the attribute `"wavelength"`, so that the approximation can be plotted against the wavelength and the coefficients kept by `step_wavelet()` traced back to the spectrum. The approximation at `level` is $2^{level/2}$ times the local mean of the spectrum. The last coefficients, whose window extends past the last channel (the padding, then the start of the spectrum), are placed past the end of the wavelength axis, extended with the spacing of its last channels.

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`. When the names of `x` are wavelengths, its attribute `"wavelength"` gives the wavelength at the center of each coefficient.

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(), wavelet_denoise(), 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
head(attr(approx, "wavelength"))   # the wavelength of each coefficient
```

---

wavelet_level *Number of Levels of a Wavelet Decomposition*

Description

A dials parameter for the level of step_wavelet() and step_wavelet_denoise().

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(), step_wavelet_denoise()

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, drop.na = deprecated())
```

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.
drop.na	**[Deprecated]** Use drop_na.

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 three columns, sorted by decreasing score:

- `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.

See Also

`iqr_outliers()`, `directional_outlyingness()` for skewed data.

Examples

```
# 90 regular values and a cluster of 10 outliers near 62
set.seed(1)
x <- c(rnorm(90, mean = 50, sd = 2), rnorm(10, mean = 62, sd = 0.5))
group <- rep(c("regular", "outlier"), c(90, 10))
ggplot2::ggplot(data.frame(x, group), ggplot2::aes(x)) +
  ggplot2::geom_density(fill = "#2166ac", alpha = 0.3) +
  ggplot2::geom_rug(ggplot2::aes(colour = group)) +
  ggplot2::scale_colour_manual(values = c(outlier = "#b2182b", regular = "#2166ac"))

# classical: the cluster inflates the standard deviation, which hides it
# (masking): the outliers score below 3 and none is flagged
head(zscore(x), 11)
# robust: the median and the MAD hardly move, and the 10 outliers are
# flagged
head(zscore(x, robust = TRUE), 11)

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, 76
    forageShots, 77

adjusted_boxplot, 7
adjusted_boxplot(), *84, 85*, *91*, *109*, *174*
aki_uncertainty, 12
aki_uncertainty(), *143, 144*
applicability_domain, 13
applicability_domain(), *195, 196*, *279, 280*
apply_calibration, 14
apply_calibration(), *210*, *353*, *355*
average, 15
average(), *23*, *225*, *323*

baseline_als, 16
baseline_als(), *79*, *186*, *288, 289*
baseline_arpls, 18
baseline_arpls(), *288, 289*
baseline_lambda, 19
baseline_lambda(), *289*
baseline_lsp, 20
baseline_lsp(), *288, 289*
bin_spectra, 22
bin_spectra(), *80*, *186*, *290, 291*
biweight_location, 23
biweight_location(), *10*
biweight_midcorrelation, 24
biweight_midcovariance, 25
biweight_midvariance, 26
biweight_scale, 27
boltzmann, 29
boltzmann(), *39*, *46–48*, *52–54*, *96, 97*, *104*, *126, 127*, *142*, *144, 145*, *226*, *257*, *275–277*
boxplot(), *9*

calibration_curve, 30
calibration_curve(), *97, 98*, *146, 147*, *193–195*, *275–277*, *357*
cellpca, 34
cellpca(), *75*, *104*, *106*, *147, 148*, *175, 176*, *279*, *292, 293*
center, 37
center(), *42*
cf_libs, 38
cf_libs(), *46*, *48*, *53*, *97, 98*, *125, 126*, *142–145*, *275–277*
clear_download_cache *(*download_cache*)*, 63
ConfidenceEllipse::confidence_ellipse(), *158*
contributions, 41
contributions(), *150, 151*
control_chart, 43
control_chart(), *152*
correct_response, 45
correct_response(), *96*, *225*, *227*
correct_self_absorption, 47
correct_self_absorption(), *46*, *53*, *143*
correlation, 49
curve_of_growth, 52
curve_of_growth(), *54*, *155, 156*, *275, 276*
curve_of_growth_model, 54

data.frame(), *10*
dials, *19*, *87*, *268*
dials::degree_int(), *289*, *298*
dials::num_comp(), *191*, *267*, *312*, *314*, *316*, *325*, *333*, *337*
dials::num_terms(), *337*
dials::penalty(), *331*
direct_orthogonal, 57
direct_orthogonal(), *123*, *197, 198*, *293, 294*
direct_osc, 59
direct_osc(), *197, 198*, *295, 296*
directional_outlyingness, 55

directional_outlyingness(), *91*, *363*
dmodx, 60
dmodx(), *163–165*, *219*
doppler_width, 62
doppler_width(), *52*, *67*, *141*
download_cache, 63
download_cache(), *96*, *116*, *125*, *127*, *284*
dplyr::bind_rows(), *286*
ds, 65
ds(), *139*, *207*, *269*, *283*

electron_density, 66
electron_density(), *39*, *41*, *68*, *70*, *104*, *221–223*, *257*, *265*, *284–286*, *288*, *351*
electron_density_uncertainty, 68
electron_density_uncertainty(), *68*, *143*, *144*
emsc, 70
emsc(), *196*, *296*, *298*
epo, 72
epo(), *71*, *197*, *298*, *299*
exomol_datasets, 74
exomol_datasets(), *64*, *116*, *117*

flagged_regions, 75
flagged_regions(), *149*
forageLIBS, 76, *77*, *78*, *357*
forageShots, 77, *181*, *225*, *357*

gap_derivative, 78
gap_derivative(), *23*, *300*, *301*
gaussian_profile, 80
gaussian_profile(), *141*, *353*
generalized_boxplot, 82
generalized_boxplot(), *11*, *174*
ggplot2::geom_point(), *154*, *164*, *176*
ggplot2::ggsave(), *10*, *187*
glance(), *94*, *141*
glance.specproc_boltzmann *(specproc-tidy)*, 274
glance.specproc_calibration *(specproc-tidy)*, 274
glance.specproc_cflibs *(specproc-tidy)*, 274
glance.specproc_curve_of_growth *(specproc-tidy)*, 274
glance.specproc_domain *(specproc-tidy-models)*, 277
glance.specproc_lts *(specproc-tidy-models)*, 277
glance.specproc_nas *(specproc-tidy-models)*, 277
glance.specproc_opls *(specproc-tidy-models)*, 277
glance.specproc_peak_fit *(specproc-tidy)*, 274
glance.specproc_robpca *(specproc-tidy-models)*, 277
glance.specproc_robust_da *(specproc-tidy-models)*, 277
glance.specproc_rpcr *(specproc-tidy-models)*, 277
glance.specproc_rsimca *(specproc-tidy-models)*, 277
glance.specproc_rsimpls *(specproc-tidy-models)*, 277
glance.specproc_slope_bias *(specproc-tidy)*, 274
glance.specproc_som *(specproc-tidy-models)*, 277
glance.specproc_uncertainty *(specproc-tidy)*, 274
glance.specproc_wavelength_calibration *(specproc-tidy)*, 274
glance.specproc_wavelength_selection *(specproc-tidy-models)*, 277
glsw, 86
glsw(), *301*, *302*, *361*
glsw_alpha, 87
glsw_alpha(), *302*, *345*

hotelling_t2, 88
hotelling_t2(), *61*, *62*, *157*, *159*, *160*, *218*, *219*
HotellingEllipse::ellipseCoord(), *159*
HotellingEllipse::ellipseParam(), *88*

iqr_outliers, 90
iqr_outliers(), *56*, *109*, *363*

libs_lines, 92
libs_lines(), *64*, *75*, *76*, *96*, *99*, *100*, *114*, *120*, *148*, *150*, *151*, *166–169*, *185*, *187*
line_finder, 95
line_finder(), *64*, *93*, *114*, *167*, *355*
line_intensities, 97

line_intensities(), *32, 33*, *45, 46*, *48*, *96*, *143, 144*, *227*, *303, 304*, *355*
linear_reg_lts, 93, *325*
loading_peaks, 99
loading_peaks(), *169*
lorentzian_profile, 100
lorentzian_profile(), *141*, *353*
lte_criteria, 102
lte_criteria(), *108*
lts_fit *(*linear_reg_lts*)*, 93
lts_fit(), *279, 280*

macropca, 104
macropca(), *34*, *36, 37*, *42*, *75, 76*, *88*, *99*, *147–149*, *157*, *159*, *168, 169*, *175–177*, *198, 199*, *229*, *279*, *307, 308*
MASS::boxcox(), *232*, *242*
mcwhirter_criterion, 107
mcwhirter_criterion(), *30*, *102*, *104*
medcouple, 108
medcouple(), *91*, *110, 111*, *170*, *172*, *217*, *232*, *243*
medcouple_weight, 110
medcouple_weight(), *109*, *216, 217*
minmax, 112
molecular_bands, 113
molecular_bands(), *96*, *117*, *166, 167*
molecular_spectrum, 115
molecular_spectrum(), *64*, *74*, *114*
msc, 117
msc(), *70–72*, *186*, *308–310*
multipeak_fit, 119
multipeak_fit(), *67*, *161–163*, *276*

nas, 121
nas(), *33*, *58*, *199, 200*, *279, 280*
nist_ionization_energy, 124
nist_ionization_energy(), *39*, *47*, *64*, *126, 127*
nist_levels, 125
nist_levels(), *39*, *41*, *64*
nist_lines, 126
nist_lines(), *12*, *39–41*, *48*, *64*, *93*, *96–98*, *103*, *114*, *124–126*, *222*, *354, 355*
normalize, 127
normalize(), *306*, *339, 340*
num_intervals, 129
num_intervals(), *337*

o2pls, 130
o2pls(), *133*, *197, 198*, *310, 311*, *313*, *320*
opls, 132
opls(), *200, 201*, *212*, *279, 280*, *313–315*
osc, 135
osc(), *197, 198*, *315–317*, *336*

p.adjust(), *10*
pareto_scale, 137
pareto_scale(), *133*, *317, 318*
parsnip::discrim_linear(), *234, 235*
parsnip::discrim_quad(), *234, 235*
parsnip::extract_fit_engine(), *191*, *267*
parsnip::fit(), *266*
parsnip::linear_reg(), *93, 94*, *325*
parsnip::multi_predict(), *191*
parsnip::pls(), *191, 192*, *254*, *311*, *316*
parsnip::set_engine(), *191*, *267*
partition_function *(*nist_levels*)*, 125
partition_function(), *40*
pds, 138
pds(), *65, 66*, *207*, *269*, *283*
peak_fit, 140
peak_fit(), *53*, *63*, *67–70*, *98*, *120, 121*, *161–163*, *276, 277*, *351*, *353*
plasma_uncertainty, 142
plasma_uncertainty(), *12*, *29*, *40*, *46*, *70*, *98*, *275–277*
plot_boltzmann, 144
plot_boltzmann(), *29, 30*, *40, 41*, *257*
plot_calibration, 146
plot_calibration(), *32, 33*, *195*
plot_cell_map, 147
plot_cell_map(), *34*, *37*, *75, 76*, *104*, *106*, *177*
plot_contributions, 150
plot_contributions(), *43*
plot_control_chart, 152
plot_control_chart(), *45*
plot_coomans, 153
plot_coomans(), *252*
plot_curve_of_growth, 155
plot_curve_of_growth(), *53*
plot_embedding, 156
plot_embedding(), *89*
plot_fit, 161
plot_influence, 163
plot_influence(), *14*, *62*, *195*, *219*
plot_lines, 165

plot_lines(), *93*, *96*, *114*
plot_loadings, 167
plot_loadings(), *99*, *100*, *151*, *160*
plot_medcouple, 170
plot_medcouple(), *109*
plot_outlier_map, 175
plot_outlier_map(), *34*, *37*, *43*, *104*, *106*, *149*, *154*, *159*, *160*, *163–165*, *169*, *191*, *199*, *219*, *229*, *246*, *248*, *250*, *253*, *256*
plot_outliers, 172
plot_response, 178
plot_response(), *227*
plot_shots, 179
plot_shots(), *78*, *225*
plot_som, 182
plot_som(), *206*, *273*
plot_spectra, 184
plot_spectra(), *172*
plot_wavelength_calibration, 188
plot_wavelength_calibration(), *355*
plot_wavelength_selection, 189
plot_wavelength_selection(), *263*, *264*, *337*
pls_rsimpls, 191, *254*
poisson_scale, 192
poisson_scale(), *319*, *320*
predict(), *13*, *32–34*, *58*, *59*, *65*, *71*, *73*, *94*, *104*, *123*, *133*, *136*, *139*, *146*, *177*, *212*, *229*, *231*, *235*, *242*, *249*, *252*, *255*, *267*, *269*, *271*, *272*, *282*, *355*
predict.o2pls *(*predict.specproc_filter*)*, 197
predict.o2pls(), *313*
predict.specproc_calibration, 193
predict.specproc_calibration(), *33*, *147*
predict.specproc_cellpca *(*predict.specproc_macropca*)*, 198
predict.specproc_direct_orthogonal *(*predict.specproc_filter*)*, 197
predict.specproc_direct_osc *(*predict.specproc_filter*)*, 197
predict.specproc_domain, 195
predict.specproc_domain(), *14*
predict.specproc_emsc, 196
predict.specproc_epo *(*predict.specproc_filter*)*, 197
predict.specproc_filter, 197
predict.specproc_filter(), *58*, *60*, *73*, *131*, *137*, *196*, *212*, *294*, *296*, *299*, *317*, *321*
predict.specproc_macropca, 198
predict.specproc_nas, 199
predict.specproc_opls, 200
predict.specproc_opls(), *134*, *315*
predict.specproc_osc *(*predict.specproc_filter*)*, 197
predict.specproc_projected_osc *(*predict.specproc_filter*)*, 197
predict.specproc_robpca *(*predict.specproc_macropca*)*, 198
predict.specproc_robust_da, 201
predict.specproc_robust_da(), *235*
predict.specproc_rpcr, 202
predict.specproc_rpcr(), *250*
predict.specproc_rsimca, 203
predict.specproc_rsimca(), *252*, *267*
predict.specproc_rsimpls, 204
predict.specproc_rsimpls(), *256*
predict.specproc_slope_bias, 205
predict.specproc_slope_bias(), *269*
predict.specproc_som, 206
predict.specproc_som(), *183*, *273*
predict.specproc_transfer, 207
predict.specproc_transfer(), *66*, *139*, *283*
predict.specproc_transformation, 208
predict.specproc_transformation(), *237*
predict.specproc_wavelength_calibration, 210
projected_osc, 211
projected_osc(), *133*, *197*, *198*, *320*, *321*
pseudo_voigt_profile, 213
pseudo_voigt_profile(), *141*, *352*, *353*

q_residuals, 217
q_residuals(), *13*, *14*, *43*, *61*, *62*, *89*, *163–165*, *195*
quantile_weight, 215
quantile_weight(), *110*, *111*

read_kurucz *(*read_vald*)*, 221
read_kurucz(), *67*, *220*, *284*, *287*
read_starkb, 220
read_starkb(), *67*, *284*, *286*

read_vald, 221
read_vald(), *67*, *220*, *284*, *286, 287*
recipes::bake(), *289*, *291*, *293, 294*, *296, 297*, *299*, *301, 302*, *304*, *306*, *308, 309*, *311*, *314*, *316*, *318, 319*, *321*, *324*, *326*, *328, 329*, *331*, *333*, *335*, *337, 338*, *340*, *342, 343*, *345*
recipes::selections(), *289*, *291, 292*, *294, 295*, *297*, *299, 300*, *302*, *304, 305*, *307*, *309*, *311*, *314*, *316*, *318, 319*, *321*, *324*, *326, 327*, *329*, *331, 332*, *334*, *336*, *338, 339*, *341*, *343, 344*
recipes::step_BoxCox(), *325*
recipes::step_center(), *318*
recipes::step_normalize(), *340*
recipes::step_pca(), *323*
recipes::step_pls(), *332, 333*
recipes::step_YeoJohnson(), *329*
recipes::update_role(), *322*
reject_shots, 223
reject_shots(), *78*, *172*, *179–181*, *322, 323*
response_calibration, 225
response_calibration(), *29*, *40*, *45, 46*, *178*
robpca, 227
robpca(), *13*, *34*, *36, 37*, *42*, *88, 89*, *99*, *104–106*, *156, 157*, *159, 160*, *168, 169*, *175–177*, *198, 199*, *219*, *234*, *239*, *244–246*, *248–256*, *267*, *272*, *279*, *323–325*
robust_boxcox, 230
robust_boxcox(), *208, 209*, *237*, *241–243*, *325–327*
robust_da, 233
robust_da(), *201, 202*, *252*, *279, 280*
robust_gandh, 235
robust_gandh(), *208, 209*, *327, 328*
robust_rmsecv, 238
robust_rmsecv(), *249, 250*, *254*, *256*
robust_yeojohnson, 241
robust_yeojohnson(), *208, 209*, *231, 232*, *237*, *329, 330*
robustbase::adjboxStats(), *9*
robustbase::ltsReg(), *93*, *248*
robustbase::mc(), *109*
rospca, 244
rospca(), *42*, *88*, *99*, *157*, *159*, *168, 169*, *175–177*, *198, 199*, *229*, *279*, *330, 331*
rousseeuw_croux, 246
rpcr, 248
rpcr(), *94*, *175–177*, *202, 203*, *239, 240*, *256*, *279, 280*
rsample::group_vfold_cv(), *323*
rsimca, 250
rsimca(), *153, 154*, *203, 204*, *234, 235*, *265*, *267, 268*, *279, 280*
rsimpls, 253
rsimpls(), *175–177*, *191, 192*, *204, 205*, *209*, *239, 240*, *249, 250*, *261–264*, *279, 280*, *332, 333*, *336*

saha_boltzmann, 256
saha_boltzmann(), *30*, *40, 41*, *48*, *97*, *124*, *126, 127*, *142*, *144, 145*, *275, 276*
saturation_summary, 258
saturation_summary(), *98*, *225*, *306*
savgol_derivative, 259
savitzky_golay, 259
savitzky_golay(), *79, 80*, *334, 335*, *357, 358*, *360*
select_wavelengths, 261
select_wavelengths(), *189, 190*, *279, 280*, *335–337*
self_absorption, 264
self_absorption(), *29*, *40*, *48*, *53*, *67*, *96*, *306*
set.seed(), *94*, *133*, *143*, *191*, *229*, *234*, *240*, *249*, *252*, *255*, *263*, *267*, *272*
shiny::runApp(), *95*
simca, 265
simca(), *252*, *268*
simca_gamma, 268
simca_gamma(), *267*
simca_squared *(*simca_gamma*)*, 268
simca_squared(), *267*
slope_bias, 269
slope_bias(), *66*, *139*, *205*, *275, 276*, *283*
snv, 270
snv(), *186*, *338, 339*
som, 271
som(), *182, 183*, *206*, *273, 274*, *279, 280*
som_stability, 273
som_stability(), *272, 273*
specproc-tidy, 274, *280*
specproc-tidy-models, *94*, *277*, 277
spectral_norm_method, 281

sst, 282
sst(), *66*, *139*, *207*, *269*
stark_table, 285
stark_table(), *67*, *220*, *222*, *223*, *284*
stark_width, 287
stark_width(), *53*, *67–69*, *221–223*, *265*, *284–286*
starkb_lines, 283
starkb_lines(), *64*, *67–69*, *127*, *220–223*, *285–288*
stats::hclust(), *50*
stats::prcomp(), *13*, *42*, *43*, *61*, *88*, *99*, *156*, *157*, *159*, *168*, *169*, *218*
step_baseline, 288
step_baseline(), *19*
step_bin_spectra, 290
step_bin_spectra(), *23*, *301*
step_cellpca, 292
step_cellpca(), *37*
step_direct_orthogonal, 293
step_direct_orthogonal(), *58*, *294*, *296*, *316*, *321*
step_direct_osc, 295
step_direct_osc(), *60*, *294*, *296*, *316*, *321*
step_emsc, 296
step_emsc(), *72*, *310*
step_epo, 298
step_epo(), *73*, *294*, *296*, *302*, *316*, *321*
step_gap_derivative, 300
step_gap_derivative(), *80*, *259*, *291*
step_glsw, 301
step_glsw(), *87*, *294*, *296*, *316*, *321*, *345*
step_line_intensities, 303
step_line_intensities(), *98*
step_line_ratio, 305
step_line_ratio(), *304*, *323*, *340*
step_macropca, 307
step_macropca(), *106*, *293*, *325*
step_msc, 308
step_msc(), *298*
step_o2pls, 310
step_o2pls(), *131*, *294*, *296*, *316*, *321*
step_opls, 313
step_opls(), *134*, *201*, *294*, *296*, *311*, *313*, *316*, *321*
step_osc, 315
step_osc(), *137*, *198*, *314*
step_pareto_scale, 317
step_pareto_scale(), *133*, *314*, *320*
step_poisson_scale, 319
step_poisson_scale(), *318*
step_projected_osc, 320
step_projected_osc(), *212*, *294*, *296*, *311*, *314*, *316*, *321*
step_reject_shots, 322
step_reject_shots(), *225*, *306*
step_robpca, 323
step_robpca(), *94*, *229*, *234*, *293*, *308*, *331*, *333*
step_robust_boxcox, 325
step_robust_boxcox(), *232*, *328*, *330*
step_robust_gandh, 327
step_robust_gandh(), *237*
step_robust_yeojohnson, 329
step_robust_yeojohnson(), *243*, *327*, *328*
step_rospca, 330
step_rospca(), *245*, *246*, *325*
step_rsimpls, 332
step_rsimpls(), *192*, *256*
step_savgol, 334
step_savgol(), *259*, *260*, *301*, *342*, *344*
step_select_wavelengths, 335
step_select_wavelengths(), *129*, *261*, *263*, *264*
step_snv, 338
step_snv(), *335*, *340*
step_spectral_norm, 339
step_spectral_norm(), *281*, *291*
step_wavelet, 341
step_wavelet(), *344*, *359*, *360*
step_wavelet_denoise, 342
step_wavelet_denoise(), *342*, *358*, *360*
step_y_gradient_glsw, 344
step_y_gradient_glsw(), *87*, *294*, *296*, *302*, *316*, *321*, *362*
summary_stats, 346
summary_stats(), *109*

tidy(), *94*, *141*, *289*, *291*, *298*, *299*, *301*, *302*, *304*, *306*, *309*, *313*, *314*, *317–319*, *323*, *325*, *326*, *330*, *333*, *335*, *337*, *338*, *340*, *342*, *343*, *345*
tidy.specproc_boltzmann *(*specproc-tidy*)*, 274
tidy.specproc_calibration *(*specproc-tidy*)*, 274

tidy.specproc_cflibs *(specproc-tidy)*, 274
tidy.specproc_curve_of_growth *(specproc-tidy)*, 274
tidy.specproc_domain *(specproc-tidy-models)*, 277
tidy.specproc_lts *(specproc-tidy-models)*, 277
tidy.specproc_nas *(specproc-tidy-models)*, 277
tidy.specproc_opls *(specproc-tidy-models)*, 277
tidy.specproc_peak_fit *(specproc-tidy)*, 274
tidy.specproc_robpca *(specproc-tidy-models)*, 277
tidy.specproc_robust_da *(specproc-tidy-models)*, 277
tidy.specproc_rpcr *(specproc-tidy-models)*, 277
tidy.specproc_rsimca *(specproc-tidy-models)*, 277
tidy.specproc_rsimpls *(specproc-tidy-models)*, 277
tidy.specproc_slope_bias *(specproc-tidy)*, 274
tidy.specproc_som *(specproc-tidy-models)*, 277
tidy.specproc_uncertainty *(specproc-tidy)*, 274
tidy.specproc_wavelength_calibration *(specproc-tidy)*, 274
tidy.specproc_wavelength_selection *(specproc-tidy-models)*, 277
tools::R_user_dir(), *64*
tukey_gh, 347
tukey_gh(), *237*
tune::select_best(), *266*
tune::tune(), *191*, *267*, *289*, *298*, *299*, *302*, *312*, *314*, *316*, *325*, *331*, *333*, *337*, *345*
tune::tune_grid(), *191*, *263*, *294*, *296*, *311*, *312*, *316*, *321*, *335*

umad, 349
update.simca *(simca)*, 265

voigt_fwhm, 350
voigt_fwhm(), *63*, *67*, *68*, *162*, *163*
voigt_profile, 352
voigt_profile(), *141*, *214*, *351*

wavelength_calibration, 353
wavelength_calibration(), *14*, *15*, *169*, *189*, *210*, *226*, *275*, *276*
wavelet_denoise, 356
wavelet_denoise(), *342*, *344*, *360*
wavelet_features, 358
wavelet_features(), *341–343*, *356*, *358*
wavelet_level, 360
workflows::workflow(), *294*, *296*, *316*, *321*

y_gradient_glsw, 361
y_gradient_glsw(), *344*, *345*

zscore, 362
zscore(), *56*