Package 'OpenSpecy'

October 7, 2026

Type Package

Title Analyze, Process, Identify, and Share Raman and (FT)IR Spectra

Version 2.0.3

Date 2026-10-06

Description Raman and (FT)IR spectral analysis tool for plastic particles and other environmental samples (Cowger et al. 2025, <doi:10.1021/acs.analchem.5c00962>). With read_any(), Open Specy provides a single function for reading individual, batch, or map spectral data files like .asp, .csv, .jdx, .spc, .spa, .0, and .zip. process_spec() simplifies processing spectra, including smoothing, baseline correction, range restriction and flattening, intensity conversions, wavenumber alignment, and min-max normalization. Spectra can be identified in batch using an onboard reference library using match_spec(). A bundled Shiny app is available via run_app() or online at <https://www.openanalysis.org/OpenSpecyV2/>.

URL https://github.com/wincowgerDEV/OpenSpecy-package/, https://www.openanalysis.org/OpenSpecyV2/, https://www.openanalysis.org/OpenSpecyV2/pkgdown/

BugReports https://github.com/wincowgerDEV/OpenSpecy-package/issues/

License CC BY 4.0

Encoding UTF-8

LazyLoad true

LazyData true

VignetteBuilder knitr

Depends R (>= 4.3.0)

Imports methods, data.table, jsonlite, caTools, hyperSpec, mmand, plotly, digest, zip, glmnet, cluster, jpeg, png, shiny, hdf5r, matrixStats

Suggests knitr, rmarkdown, testthat (>= 3.1.9), doFuture, doRNG, foreach, future, ranger, sgolay, curl, shinyjs, shinyWidgets, shinyFiles, bs4Dash, dplyr, DT, reshape2, ggplot2, htmltools, scales

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Win Cowger [cre, aut, dtc] (ORCID:
<https://orcid.org/0000-0001-9226-3104>),
Zacharias Steinmetz [aut] (ORCID:
<https://orcid.org/0000-0001-6675-5033>),
Hazel Vaquero [aut] (ORCID: <https://orcid.org/0009-0001-5468-2049>),
Nick Leong [aut] (ORCID: <https://orcid.org/0009-0008-3313-4132>),
Andrea Faltynkova [aut, dtc] (ORCID:
<https://orcid.org/0000-0003-2523-3137>),
Hannah Sherrod [aut] (ORCID: <https://orcid.org/0009-0001-0497-8693>),
Andrew B Gray [ctb] (ORCID: <https://orcid.org/0000-0003-2252-7367>),
Hannah Hapich [ctb] (ORCID: <https://orcid.org/0000-0003-0000-6632>),
Jennifer Lynch [ctb, dtc] (ORCID:
<https://orcid.org/0000-0003-3572-8782>),
Hannah De Frond [ctb, dtc] (ORCID:
<https://orcid.org/0000-0003-1199-0727>),
Garth Covernton [ctb, dtc] (ORCID:
<https://orcid.org/0000-0003-3814-4918>),
Keenan Munno [ctb, dtc] (ORCID:
<https://orcid.org/0000-0003-2916-5944>),
Chelsea Rochman [ctb, dtc] (ORCID:
<https://orcid.org/0000-0002-7624-711X>),
Sebastian Primpke [ctb, dtc] (ORCID:
<https://orcid.org/0000-0001-7633-8524>),
Orestis Herodotou [ctb],
Mary C Norris [ctb],
Christine M Knauss [ctb] (ORCID:
<https://orcid.org/0000-0003-4404-8922>),
Aleksandra Karapetrova [ctb, dtc, rev] (ORCID:
<https://orcid.org/0000-0002-9856-1644>),
Vesna Teofilovic [ctb] (ORCID: <https://orcid.org/0000-0002-3557-1482>),
Laura A. T. Markley [ctb] (ORCID:
<https://orcid.org/0000-0003-0620-8366>),
Nicolas Coca Lopez [ctb],
Shreyas Patankar [ctb, dtc],
Rachel Kozloski [ctb, dtc] (ORCID:
<https://orcid.org/0000-0003-1211-9351>),
Samiksha Singh [ctb],
Katherine Lasdin [ctb],
Cristiane Vidal [ctb] (ORCID: <https://orcid.org/0000-0001-6363-9475>),
Clare Murphy-Hagan [ctb] (ORCID:
<https://orcid.org/0009-0009-9629-2856>),
Philipp Baumann [ctb] (ORCID: <https://orcid.org/0000-0002-3194-8975>),
Pierre Roudier [ctb],
Adam Hauser [ctb],
National Renewable Energy Laboratory [fnd],

Possibility Lab [fnd]

Maintainer Win Cowger <wincowger@gmail.com>

Repository CRAN

Date/Publication 2026-10-07 12:50:35 UTC

Contents

add_visual_image . . . 4
adj_intens . . . 5
adj_res . . . 7
adj_wave . . . 8
area_under_band . . . 9
assess_spec . . . 11
as_OpenSpecy . . . 13
automate_particle_analysis . . . 17
build_lib . . . 21
calculate_emissivity . . . 32
check_lib . . . 33
collapse_spec . . . 36
conform_spec . . . 38
correct_spike . . . 40
cor_spec . . . 42
crowd_lookup . . . 47
c_spec . . . 49
estimate_temperature . . . 50
head.OpenSpecy . . . 53
human_ts . . . 55
io_spec . . . 56
lib_metadata_name_lookup . . . 57
make_rel . . . 59
manage_na . . . 60
material_percentage_uncertainty . . . 62
open_specs . . . 63
particle_image . . . 64
peak_ratio . . . 66
plot.OpenSpecyParticleAnalysis . . . 67
plotly_spec . . . 67
point_intensity . . . 70
predict_class_reference . . . 72
process_spec . . . 72
raman_hdpe . . . 75
read_any . . . 75
read_envi . . . 77
read_opus . . . 78
read_opus_raw . . . 80
read_text . . . 82
restrict_range . . . 85

run_app 88
sig_noise 89
smooth_intens 91
spatial_smooth 93
Specs 94
spec_res 99
split_h5 100
split_spec 101
subtr_baseline 102
test_lib 105

Index **107**

`add_visual_image` *Attach visual images to spectral map objects*

Description

`add_visual_image()` stores a visual image and map-to-image alignment metadata on an `OpenSpecy` or `Specs` object. `visual_image()` retrieves that attribute. `detect_image_origin()` detects the dominant separated red Thermo Fisher iN10 map-box lines and returns the image coordinates needed for overlays and feature color extraction without treating shorter red annotations as bounds.

Usage

```
add_visual_image(
  x,
  image,
  bottom_left = NULL,
  top_right = NULL,
  source = NULL,
  detection_method = NULL,
  diagnostics = NULL,
  transform = NULL,
  ...
)

visual_image(x, require = FALSE, ...)

detect_image_origin(
  image,
  red_threshold = 50,
  red_ratio = 2,
  min_pixels = 1,
  diagnostics = TRUE,
  ...
)
```

Arguments

x	an OpenSpecy or Specs object.
image	image path, raster, matrix, array, raw BMP bytes, or an existing visual-image list.
bottom_left, top_right	numeric length-2 vectors giving the lower-left and upper-right corners of the spectral map in image pixel coordinates. Image x values are left-to-right and image y values are top-to-bottom.
source	optional image source label or file path.
detection_method	optional description of how the origin was detected.
diagnostics	optional diagnostic data to store with the image.
transform	optional list describing a custom transform.
require	logical; should visual_image() error when no image is attached?
red_threshold	minimum red-channel value on the 0-255 scale.
red_ratio	minimum red-to-green and red-to-blue ratio for red-box detection.
min_pixels	minimum number of red pixels required for origin detection.
...	reserved for future methods.

Value

add_visual_image() returns x with a visual_image attribute. visual_image() and detect_image_origin() return lists.

Examples

```
data("raman_hdpe")
img <- array(1, dim = c(10, 10, 3))
with_image <- add_visual_image(raman_hdpe, img,
                               bottom_left = c(1, 10),
                               top_right = c(10, 1))
visual_image(with_image)$bottom_left
```

---

adj_intens *Adjust spectral intensities to standard absorbance units.*

Description

Converts reflectance or transmittance intensity units to absorbance units and adjust log or exp transformed units.

Usage

```
adj_intens(x, ...)

## Default S3 method:
adj_intens(x, type = "none", make_rel = TRUE, log_exp = "none", ...)

## S3 method for class 'OpenSpecy'
adj_intens(x, type = "none", make_rel = TRUE, log_exp = "none", ...)
```

Arguments

x	a list object of class OpenSpecy.
type	a character string specifying whether the input spectrum is in absorbance units ("none", default) or needs additional conversion from "reflectance" or "transmittance" data.
make_rel	logical; if TRUE spectra are automatically normalized with make_rel().
log_exp	a character string specifying whether the input needs to be log transformed "log", exp transformed "exp", or not ("none", default).
...	further arguments passed to submethods; this is to adj_neg() for adj_intens() and to conform_res() for conform_intens().

Details

Many of the Open Specy functions will assume that the spectrum is in absorbance units. For example, see subtr_baseline(). To run those functions properly, you will need to first convert any spectra from transmittance or reflectance to absorbance using this function. The transmittance adjustment uses the $log(1/T)$ calculation which does not correct for system and particle characteristics. The reflectance adjustment uses the Kubelka-Munk equation $(1 - R)^2/2R$. We assume that the reflectance intensity is a percent from 1-100 and first correct the intensity by dividing by 100 so that it fits the form expected by the equation.

Value

adj_intens() returns a data frame containing two columns named "wavenumber" and "intensity".

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

subtr_baseline() for spectral background correction.

Examples

```
data("raman_hdpe")

adj_intens(raman_hdpe)
```

adj_res *Normalization and conversion of spectral data*

Description

adj_res() and conform_res() are helper functions to align wavenumbers in terms of their spectral resolution. adj_neg() converts numeric intensities y < 1 into values >= 1, keeping absolute differences between intensity values by shifting each value by the minimum intensity. make_rel() converts intensities y into relative values between 0 and 1 using the standard normalization equation. mean_replace() replaces missing values by the finite mean. For a matrix, each column is treated as one spectrum and receives its own mean. If na.rm is TRUE, missing values are removed before the computation proceeds.

Usage

```
adj_res(x, res = 1, fun = round)

conform_res(x, res = 5)

adj_neg(y, na.rm = FALSE)

mean_replace(y, na.rm = TRUE)

is_empty_vector(x)
```

Arguments

x	a numeric vector or an R object which is coercible to one by as.vector(x, "numeric"); x should contain the spectral wavenumbers.
res	spectral resolution supplied to fun.
fun	the function to be applied to each element of x; defaults to round() to round to a specific resolution res.
y	a numeric vector or matrix containing spectral intensities. Matrix columns are spectra for mean_replace().
na.rm	logical. Should missing values be removed?

Details

adj_res() and conform_res() are used in Open Specy to facilitate comparisons of spectra with different resolutions. adj_neg() is used to avoid errors that could arise from log transforming spectra when using adj_intens() and other functions. make_rel() is used to retain the relative height proportions between spectra while avoiding the large numbers that can result from some spectral instruments.

Value

`adj_res()` and `conform_res()` return a numeric vector with resolution-conformed wavenumbers. `adj_neg()` and `make_rel()` return normalized intensity data. `mean_replace()` preserves the vector or matrix shape of `y`.

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

`min()` and `round()`; `adj_intens()` for log transformation functions; `conform_spec()` for conforming wavenumbers of an `OpenSpecy` object to be matched with a reference library

Examples

```
adj_res(seq(500, 4000, 4), 5)
conform_res(seq(500, 4000, 4))
adj_neg(c(-1000, -1, 0, 1, 10))
make_rel(c(-1000, -1, 0, 1, 10))
mean_replace(matrix(c(1, NA, 3, 10, 20, NA), nrow = 3))
```

---

adj_wave *Adjust wavelength to wavenumbers for Raman*

Description

Functions for converting between wave* units.

Usage

```
adj_wave(x, ...)

## Default S3 method:
adj_wave(x, laser, ...)

## S3 method for class 'OpenSpecy'
adj_wave(x, laser, ...)
```

Arguments

`x`	an `OpenSpecy` object with wavenumber units specified as wavelength in nm or a wavelength vector.
`laser`	the wavelength in nm of the Raman laser.
`...`	additional arguments passed to submethods.

Value

An OpenSpecy object with new units converted from wavelength to wavenumbers or a vector with the same conversion.

Author(s)

Win Cowger, Zacharias Steinmetz

Examples

```
data("raman_hdpe")
raman_wavelength <- raman_hdpe
raman_wavelength$wavenumber <- (-1*(raman_wavelength$wavenumber/10^7-1/530))^(-1)
adj_wave(raman_wavelength, laser = 530)
adj_wave(raman_wavelength$wavenumber, laser = 530)
```

---

area_under_band *Measure the area under band of spectra*

Description

Area under the band calculations are useful for quantifying spectral features. Specialized fields in spectroscopy have different area under the band regions of interest and their ratios that help with understanding differences in materials. Additional processing is typically required prior to calculating these values for accuracy and reproducibility.

Usage

```
area_under_band(x, ...)

## Default S3 method:
area_under_band(x, ...)

## S3 method for class 'OpenSpecy'
area_under_band(x, min = NULL, max = NULL, na.rm = FALSE, index = NULL, ...)
```

Arguments

x	an OpenSpecy object.
min	a numeric value of the smallest wavenumber to begin calculation.
max	a numeric value of the largest wavenumber to end calculation.
na.rm	a logical value for whether to ignore NA values.
index	an optional character scalar naming a predefined area-under-band ratio. When supplied, min and max must both be omitted. Supported values are "carbonyl_index_saub", "carbonyl_index_pe", "hydroxyl_index_pe", "hydroxyl_index_pp", "carbon_oxygen_index_pe" and "carbon_oxygen_index_pp".
...	additional arguments passed to methods.

Details

Named indices define band ratios only; they do not apply baseline correction, normalization, or other preprocessing. Compose those steps explicitly before calling `area_under_band()`. The preset definitions and intended materials are:

- `carbonyl_index_saub`: 1650–1850 / 1420–1500 cm^-1; specified area-under-band carbonyl index used for polyethylene (PE) and polypropylene (PP).
- `carbonyl_index_pe`: 1700–1770 / 1423–1495 cm^-1; PE.
- `hydroxyl_index_pe`: 3021–3353 / 1467–1504 cm^-1; PE.
- `hydroxyl_index_pp`: 3300–3400 / 952–986 cm^-1; PP.
- `carbon_oxygen_index_pe`: 924–1197 / 2866–2987 cm^-1; PE.
- `carbon_oxygen_index_pp`: 1000–1200 / 2885–2940 cm^-1; PP.

All bounds are inclusive. The function preserves signed intensities and does not clip negative values.

Value

A named numeric vector with one value per spectrum. Custom `min`/`max` calculations return the inclusive sum of intensities in that band. Named indices return the numerator-band sum divided by the denominator-band sum. An index is returned as `NA` when the shared wavenumber axis does not fully cover its required bands or its ratio is not finite.

Author(s)

Win Cowger

References

Almond J, Sugumaar P, Wenzel MN, Hill G, Wallis C (2020). Determination of the carbonyl index of polyethylene and polypropylene using specified area under band methodology with ATR-FTIR spectroscopy. *e-Polymers*, 20, 369–381. doi:10.1515/epoly20200041.

Campanale C, Savino I, Massarelli C, Uricchio VF (2023). Fourier transform infrared spectroscopy to assess the degree of alteration of artificially aged and environmentally weathered microplastics. *Polymers*, 15, 911. doi:10.3390/polym15040911.

Examples

```
data("raman_hdpe")
#Single area calculation
area_under_band(raman_hdpe, min = 1000,max = 2000)
#Ratio of two areas.
area_under_band(raman_hdpe, min = 1000,max = 2000)/area_under_band(raman_hdpe, min = 500,max = 700)
#A predefined ratio (requires spectra covering both bands)
area_under_band(raman_hdpe, index = "carbonyl_index_saub")
```

assess_spec *Assess common spectral quality issues*

Description

assess_spec() scans spectra for common quality-control issues and returns one row for each issue found.

Usage

```
assess_spec(x, ...)

## Default S3 method:
assess_spec(x, ...)

## S3 method for class 'OpenSpecy'
assess_spec(
  x,
  checks = c("high_tail", "silent_region", "co2_region", "missing_values",
    "flat_spectrum", "negative_intensity", "low_snr"),
  high_prob = 0.9,
  artifact_ratio = 2,
  tail_n = 5L,
  silent_region = c(2420, 2550),
  co2_region = c(2200, 2420),
  snr_threshold = 4,
  flat_tol = sqrt(.Machine$double.eps),
  negative_tol = 0,
  na.rm = TRUE,
  report = c("issues", "all"),
  snr_metric = "run_sig_over_noise",
  spike_args = list(),
  saturation = "auto",
  saturation_min_run = NULL,
  saturation_tolerance = sqrt(.Machine$double.eps),
  ...
)
```

Arguments

x	an OpenSpecy object.
checks	character; checks to run. Options include "high_tail", "silent_region", "co2_region", "missing_values", "flat_spectrum", "negative_intensity", "low_snr", "spike", and "saturation". Spike and saturation checks are opt-in.
high_prob	numeric; spectrum-wide quantile used as the high intensity threshold for the silent-region check.

artifact_ratio	numeric; minimum ratio between the normalized maximum in a tail or carbon dioxide region and the normalized maximum outside both artifact regions required to flag an issue. The default 2 flags a candidate at or above twice the control-region maximum.
tail_n	integer; number of points to check at each end of the spectrum.
silent_region	numeric length two; wavenumber range expected to be mostly silent. The default is c(2420, 2550) cm^-1.
co2_region	numeric length two; carbon dioxide wavenumber range.
snr_threshold	numeric; spectra with run signal-to-noise below this value are flagged.
flat_tol	numeric; maximum finite intensity range considered flat.
negative_tol	numeric; minimum allowed intensity before a spectrum is flagged as negative.
na.rm	logical; indicating whether missing values should be removed when calculating thresholds and metrics.
report	character; "issues" preserves the issue-only return contract, while "all" returns an explicit pass, warning, or error row for every requested check and spectrum.
snr_metric	character; signal-to-noise metric passed to sig_noise() for the "low_snr" check.
spike_args	named list of arguments passed to the shared internal spike detector when "spike" is requested.
saturation	"auto" or one finite numeric detector ceiling used by the "saturation" check.
saturation_min_run	integer or NULL; minimum saturated run used by the shared detector.
saturation_tolerance	numeric; relative equality tolerance for automatic detector plateaus.
...	further arguments passed to sig_noise() for the "low_snr" check.

Value

With report = "issues", a data.table-class() with one row per issue found and columns describing the spectrum, check, issue, likely cause, potential fix, metric value, threshold, and region. If no issues are found, an empty table with the same columns is returned. With report = "all", one status row is returned for every requested spectrum/check pair, plus any applicable batch-level error, with stable IDs and correction diagnostics.

Author(s)

Win Cowger

Examples

```
data("raman_hdpe")
assess_spec(raman_hdpe)
```

as_OpenSpecy *Create* OpenSpecy *objects*

Description

Functions to check if an object is an OpenSpecy, or coerce it if possible.

Usage

```
as_OpenSpecy(x, ...)

## S3 method for class 'OpenSpecy'
as_OpenSpecy(x, session_id = FALSE, compute_file_id = TRUE, ...)

## S3 method for class 'list'
as_OpenSpecy(x, ...)

## S3 method for class 'hyperSpec'
as_OpenSpecy(x, ...)

## S3 method for class 'data.frame'
as_OpenSpecy(x, colnames = list(wavenumber = NULL, spectra = NULL), ...)

## Default S3 method:
as_OpenSpecy(
  x,
  spectra,
 metadata = list(file_name = NULL, user_name = NULL, contact_info = NULL, organization =
  NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL, material_form
  = NULL, material_phase = NULL, material_producer = NULL, material_purity = NULL,
  material_quality = NULL, material_color = NULL, material_other = NULL, cas_number =
  NULL, instrument_used = NULL, instrument_accessories = NULL, instrument_mode = NULL,
   intensity_units = NULL, spectral_resolution = NULL, laser_light_used = NULL,
   number_of_accumulations = NULL,
    total_acquisition_time_s = NULL,
  data_processing_procedure = NULL, level_of_confidence_in_identification = NULL,
   other_info = NULL, license = "CC BY-NC"),
 attributes = list(intensity_unit = NULL, derivative_order = NULL, baseline = NULL,
   spectra_type = NULL, visual_image = NULL),
  coords = "gen_grid",
  session_id = FALSE,
  compute_file_id = TRUE,
  comma_decimal = FALSE,
  ...
)

is_OpenSpecy(x)
```

```
check_OpenSpecy(x)

OpenSpecy(x, ...)

gen_grid(n)
```

Arguments

x	depending on the method, a list with all OpenSpecy parameters, a vector with the wavenumbers for all spectra, or a data.frame with a full spectrum in the classic Open Specy format.
session_id	logical. Whether to add a session ID to the metadata. The session ID is based on current session info so metadata of the same spectra will not return equal if session info changes. Sometimes that is desirable.
compute_file_id	logical. Whether to add a file ID hash to metadata when one is not already present.
colnames	names of the wavenumber column and spectra column, makes assumptions based on column names or placement if NULL.
spectra	spectral intensities formatted as a matrix with one column per spectrum.
metadata	metadata for each spectrum with one row per spectrum, see details.
attributes	a list of attributes describing critical aspects for interpreting the spectra. see details.
coords	spatial coordinates for the spectra.
comma_decimal	logical(1) whether commas may represent decimals.
n	number of spectra to generate the spatial coordinate grid with.
...	additional arguments passed to submethods.

Details

as_OpenSpecy() converts spectral datasets to a three part list; the first with a vector of the wavenumbers of the spectra, the second with a matrix of all spectral intensities ordered as columns, the third item is another data.table with any metadata the user provides or is harvested from the files themselves.

The metadata argument may contain a named list with the following details (* = minimum recommended).

file_name* The file name, defaults to basename() if not specified

user_name* User name, e.g. "Win Cowger"

contact_info Contact information, e.g. "1-513-673-8956, wincowger@gmail.com"

organization Affiliation, e.g. "University of California, Riverside"

citation Data citation, e.g. "Primpke, S., Wirth, M., Lorenz, C., & Gerdts, G. (2018). Reference database design for the automated analysis of microplastic samples based on Fourier transform infrared (FTIR) spectroscopy. *Analytical and Bioanalytical Chemistry*. doi:10.1007/s00216-0181156x"

`spectrum_type*` Raman or FTIR

`spectrum_identity*` Material/polymer analyzed, e.g. "Polystyrene"

`material_form` Form of the material analyzed, e.g. textile fiber, rubber band, sphere, granule

`common_use` Curated predominant global end-market: consumer, industrial, mixed, or missing. Review evidence may be quantitative mass shares or a cited qualitative application source

`material_phase` Phase of the material analyzed (liquid, gas, solid)

`material_producer` Producer of the material analyzed, e.g. Dow

`material_purity` Purity of the material analyzed, e.g. 99.98%

`material_quality` Quality of the material analyzed, e.g. consumer product, manufacturer material, analytical standard, environmental sample

`material_color` Color of the material analyzed, e.g. blue, #0000ff, (0, 0, 255)

material_other Other material description, e.g. 5 µm diameter fibers, 1 mm spherical particles

`cas_number` CAS number, e.g. 9003-53-6

`instrument_used` Instrument used, e.g. Horiba LabRam

instrument_accessories Instrument accessories, e.g. Focal Plane Array, CCD

`instrument_mode` Instrument modes/settings, e.g. transmission, reflectance

`intensity_units*` Units of the intensity values for the spectrum, e.g. transmittance, reflectance, absorbance, `W m^-2 sr^-1 (cm^-1)^-1` for calibrated spectral radiance, or emissivity

`radiance_level` Radiometric measurement level when applicable, e.g. `surface_leaving` for calibrated thermal-emission spectra

`radiometric_calibration` Description or identifier for the absolute radiometric calibration when applicable

`spectral_resolution` Spectral resolution, e.g. 4/cm

`laser_light_used` Wavelength of the laser/light used, e.g. 785 nm

`number_of_accumulations` Number of accumulations, e.g 5

`total_acquisition_time_s` Total acquisition time (s), e.g. 10 s

`data_processing_procedure` Data processing procedure, e.g. spikefilter, baseline correction, none

`level_of_confidence_in_identification` Level of confidence in identification, e.g. 99%

`other_info` Other information

`license` The license of the shared spectrum; defaults to `"CC BY-NC"` (see https://creativecommons.org/licenses/by-nc/4.0/ for details). Any other creative commons license is allowed, for example, CC0 or CC BY

`session_id` A unique user and session identifier; populated automatically with `paste(digest(Sys.info()), digest(sessionInfo()), sep = "/")`

`file_id` A unique file identifier; populated automatically with `digest(object[c("wavenumber", "spectra")])`

The `attributes` argument may contain a named list with the following details, when set, they will be used to automate transformations and warning messages:

`intensity_unit` supported options include `"absorbance"`, `"transmittance"`, `"reflectance"`, `"W m^-2 sr^-1 (cm^-1)^-1"`, or `"emissivity"`

`derivative_order` supported options include `"0"`, `"1"`, or `"2"`

`baseline` supported options include `"raw"` or `"nobaseline"`

`spectra_type` supported options include `"ftir"` or `"raman"`

`visual_image` optional visual-image metadata created by `add_visual_image()`

Value

`as_OpenSpecy()` and `OpenSpecy()` returns three part lists described in details. `is_OpenSpecy()` returns `TRUE` if the object is an OpenSpecy and `FALSE` if not. `gen_grid()` returns a `data.table` with `x` and `y` coordinates to use for generating a spatial grid for the spectra if one is not specified in the data.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

`read_spec()` for reading `OpenSpecy` objects.

Examples

```
data("raman_hdpe")

# Inspect the spectra
raman_hdpe # see how OpenSpecy objects print.
raman_hdpe$wavenumber # look at just the wavenumbers of the spectra.
raman_hdpe$spectra # look at just the spectral intensities matrix.
raman_hdpe$metadata # look at just the metadata of the spectra.

# Creating a list and transforming to OpenSpecy
as_OpenSpecy(list(wavenumber = raman_hdpe$wavenumber,
                  spectra = raman_hdpe$spectra,
                  metadata = raman_hdpe$metadata[,-c("x", "y")]))

# If you try to produce an OpenSpecy using an OpenSpecy it will just return
# the same object.
as_OpenSpecy(raman_hdpe)

# Creating an OpenSpecy from a data.frame
as_OpenSpecy(x = data.frame(wavenumber = raman_hdpe$wavenumber,
                            spectra = raman_hdpe$spectra[, "intensity"]))

# Test that the spectrum is formatted as an OpenSpecy object.
is_OpenSpecy(raman_hdpe)
is_OpenSpecy(raman_hdpe$spectra)
```

automate_particle_analysis *Automate particle analysis for spectral maps*

Description

automate_particle_analysis() generalizes the batch map workflow used for particle detection, spectral matching, particle details, summaries, and optional base-graphics particle images. Visual images attached to map objects or read from supported H5 mosaics are used for particle color extraction when feature definition is requested. It keeps file output optional and returns all results as R objects. S/N thresholds that remove every pixel return an empty analysis without library matching. Thresholds that retain every pixel continue normally; a connected collapse treats the full extent of each source map as one particle. Both threshold extremes emit an informational message.

Usage

```
automate_particle_analysis(
  x,
  library,
  output_dir = NULL,
  images = NULL,
  bottom_left = NULL,
  top_right = NULL,
  origins = NULL,
  material_col = "material_class",
  library_id_col = "sample_name",
 particle_id_strategy = c("collapse", "partial_collapse", "nonspatial_collapse",
    "all_cell_id", "raw"),
  spectral_smooth = FALSE,
  sigma1 = c(1, 1, 1),
  sigma2 = c(3, 3),
  close = FALSE,
  close_kernel = c(4, 4),
  sn_threshold_min = 0.04,
  sn_threshold_max = Inf,
  cor_threshold = 0.7,
  area_threshold = 1,
  label_unknown = FALSE,
  remove_materials = NULL,
  remove_unknown = FALSE,
  pixel_length = 25,
  metric = "sig_times_noise",
  abs = FALSE,
  collapse_function = stats::median,
  outputs = c("details", "summary"),
  process_args = list(),
  specs_steps = c("pca", "kmeans"),
```

```
  specs_centers = NULL,
  file_processing = c("stream", "memory"),
  ...
)

## Default S3 method:
automate_particle_analysis(
  x,
  library,
  output_dir = NULL,
  images = NULL,
  bottom_left = NULL,
  top_right = NULL,
  origins = NULL,
  material_col = "material_class",
  library_id_col = "sample_name",
 particle_id_strategy = c("collapse", "partial_collapse", "nonspatial_collapse",
    "all_cell_id", "raw"),
  spectral_smooth = FALSE,
  sigma1 = c(1, 1, 1),
  sigma2 = c(3, 3),
  close = FALSE,
  close_kernel = c(4, 4),
  sn_threshold_min = 0.04,
  sn_threshold_max = Inf,
  cor_threshold = 0.7,
  area_threshold = 1,
  label_unknown = FALSE,
  remove_materials = NULL,
  remove_unknown = FALSE,
  pixel_length = 25,
  metric = "sig_times_noise",
  abs = FALSE,
  collapse_function = stats::median,
  outputs = c("details", "summary"),
  process_args = list(),
  specs_steps = c("pca", "kmeans"),
  specs_centers = NULL,
  file_processing = c("stream", "memory"),
  ...
)

## S3 method for class 'FileSpecs'
automate_particle_analysis(
  x,
  library,
  output_dir = NULL,
  images = NULL,
```

```
  bottom_left = NULL,
  top_right = NULL,
  origins = NULL,
  material_col = "material_class",
  library_id_col = "sample_name",
 particle_id_strategy = c("collapse", "partial_collapse", "nonspatial_collapse",
    "all_cell_id", "raw"),
  spectral_smooth = FALSE,
  sigma1 = c(1, 1, 1),
  sigma2 = c(3, 3),
  close = FALSE,
  close_kernel = c(4, 4),
  sn_threshold_min = 0.04,
  sn_threshold_max = Inf,
  cor_threshold = 0.7,
  area_threshold = 1,
  label_unknown = FALSE,
  remove_materials = NULL,
  remove_unknown = FALSE,
  pixel_length = 25,
  metric = "sig_times_noise",
  abs = FALSE,
  collapse_function = stats::median,
  outputs = c("details", "summary"),
  process_args = list(),
  specs_steps = c("pca", "kmeans"),
  specs_centers = NULL,
  file_processing = c("stream", "memory"),
  ...
)
```

Arguments

x	character vector of files, an OpenSpecy/Specs object, or a list of objects/files. H5/HDF5 paths can be analyzed as bounded file-backed FileSpecs sources or materialized in memory before analysis.
library	reference OpenSpecy object or trained model library passed to match_spec().
output_dir	optional directory for CSV/RDS/PNG outputs. Per-source filenames retain the complete input basename (without its extension); multi-region sources append the region after that basename.
images	optional image path(s) or image objects aligned with x. When omitted for an ENVI DAT/IMG path, an unambiguous same-basename JPG, JPEG, or PNG in the same directory is discovered automatically.
bottom_left, top_right	optional lists of image corners; if missing and an image is supplied, detect_image_origin() is attempted.
origins	optional list with x and y origin offsets for map-unit outputs.

`material_col` material/class column in matched library metadata.

`library_id_col` library metadata column used to join match metadata.

`particle_id_strategy`
one of `"collapse"`, `"partial_collapse"`, `"nonspatial_collapse"`, `"all_cell_id"`, or `"raw"`.

`spectral_smooth`, `sigma1`
apply 3D Gaussian smoothing to spectral maps; file readers apply this while reading and in-memory maps are smoothed after coercion.

`sigma2` shape kernel passed to `def_features()`.

`close`, `close_kernel`
passed to `def_features()`.

`sn_threshold_min`, `sn_threshold_max`
signal/noise thresholds.

`cor_threshold` minimum match value for confident particle labels.

`area_threshold` minimum feature area in pixels (inclusive).

`label_unknown` logical; label low-correlation matches as `"unknown"`.

`remove_materials`
optional material labels to remove after matching.

`remove_unknown` logical; remove `"unknown"` after matching.

`pixel_length` map pixel length used for output dimensions.

`metric`, `abs` signal/noise arguments passed to `sig_noise()`.

`collapse_function`
function used by `collapse_spec()`.

`outputs` character vector containing any of `"details"`, `"summary"`, `"particle_image"`, `"particle_heatmap"`, `"particle_heatmap_thresholded"`, `"cor_heatmap"`, `"sn_histogram"`, `"cor_histogram"`, `"raw"`, `"processed"`, or `"time"`. Short aliases `"heatmap"`, `"thresholded"`, and `"correlation"` are also accepted.

`process_args` optional named list overriding `process_spec()` arguments for spectra before matching.

`specs_steps` retained for signature compatibility; clustering strategies require the concrete `c("pca", "kmeans")` workflow.

`specs_centers` requested K-means cluster count for clustering strategies; the effective count is clamped to the eligible data.

`file_processing`
file-backed execution policy. `"stream"` (the default) reads bounded chunks and supports the file-backed strategies; `"memory"` materializes every region before using the full in-memory workflow. The memory mode can be faster for smaller files but requires enough RAM for all spectra and their analysis intermediates.

`...` catches removed legacy arguments and otherwise is reserved.

Value

A list with samples, particle_details_all_csv, and particle_summary_all_csv. Each per-sample entry has particle_details_csv, particle_summary_csv, particles_raw_rds, particles_rds, and time_rds, and uses the same filename-based sample_id stem as its per-source output files (including an appended region for multi-region sources), with summary rows reporting full map area, particle count, observed percentage, 95% percentage confidence-interval half-width and bounds, total concentration RSD, and total, mean, and median particle area in square micrometres for each material class, plus one plot-data list for each requested plot output: particle_image, particle_heatmap, particle_heatmap_thresholded, cor_heatmap, sn_histogram, and cor_histogram. Each plot-data list carries the grid or histogram values needed to build a custom plot()/plotly/ggplot2 view (a type field plus x/y/z, values, thresholds, or levels as appropriate), or type = "empty" with a reason string when nothing passed filtering. output_dir still writes the matching static PNG/JPG for each requested plot. The result has class OpenSpecyParticleAnalysis; use its plot() method to draw one of these plots with base graphics.

Examples

```
tiny_map <- read_extdata("CA_tiny_map.zip") |> read_any()
data("test_lib")
res <- automate_particle_analysis(tiny_map, test_lib,
                                  outputs = c("details", "summary"),
                                  sn_threshold_min = 0.1)
names(res)
```

---

build_lib *Build spectral libraries*

Description

Create reference libraries from source files and OpenSpecy objects. When output_dir is supplied, build_lib() runs the official end-to-end workflow and returns raw, processed, medoid, model, and assessment artifacts in one object. Supporting functions remain available for advanced composition.

Usage

```
build_lib(
  x,
  recipes = .default_lib_recipes(),
  range = "full",
  res = 6,
  id_col = "sample_name",
  exclude_ids = NULL,
  dedupe = TRUE,
  metadata_lookups = NULL,
  material_hierarchy = NULL,
  metadata_name_lookup = lib_metadata_name_lookup(),
```

```
  clean_metadata_values = NULL,
  convert_intensity = TRUE,
  restrict_range_args = NULL,
  signal_noise = TRUE,
  assess = FALSE,
  prune = NULL,
  progress = TRUE,
  workflow_data = NULL,
  output_dir = NULL,
  previous_library_dir = "system",
  reuse = TRUE,
  remove_other = TRUE,
  seed = 123,
  holdout = 0.1,
  ...
)

rebuild_lib_artifacts(
  x,
  output_dir,
  previous_library_dir = "system",
  reuse = TRUE,
  seed = 123,
  holdout = 0.1,
  progress = TRUE
)

make_lib_lookup_template(x, columns, add = NULL, path = NULL)

join_lib_metadata(
  x,
  lookup,
  by,
  require_complete = FALSE,
  return = c("object", "table", "report"),
  suffixes = c(".x", ".y")
)

join_material_hierarchy(
  x,
  hierarchy,
  key_col = "material",
  levels = c("material", "material_class", "material_type"),
  output_names = levels,
  require_complete = FALSE,
  return = c("object", "table", "report")
)
```

```
dedupe_spec(
  x,
  id_col = "sample_name",
  exclude_ids = NULL,
  duplicate = c("first", "remove_all", "none"),
  scale = 100,
  algo = "md5"
)

prune_lib(
  x,
  class_col = "material_class",
  type_col = "spectrum_type",
  material_type_col = "material_type",
  id_col = "sample_name",
  min_n = 10,
  cross_class = FALSE,
  cross_class_threshold = 0.9,
  exclude = c(2200, 2420),
  return = c("object", "ids", "report"),
  progress = TRUE
)

reduce_lib(
  x,
  group_cols = "material_class",
  id_col = "sample_name",
  k = 50,
  min_n = k,
  return = c("object", "ids"),
  progress = FALSE,
  ...
)

build_model_lib(
  x,
  class_col = "material_class",
  type_col = "spectrum_type",
  min_n = 10,
  alpha = 0.1,
  seed = 123,
  grouped = TRUE,
  weights = TRUE,
  make_relative = TRUE,
  method = c("logistic_regression", "random_forest"),
  ...
)
```

```
train_spec_model(
  x,
  class_col = "material_class",
  type_col = "spectrum_type",
  min_n = 10,
  alpha = 0.1,
  seed = 123,
  grouped = TRUE,
  weights = TRUE,
  make_relative = TRUE,
  method = c("logistic_regression", "random_forest"),
  ...
)

assess_lib(
  x,
  class_col = NULL,
  id_col = "sample_name",
  nearest = !is.null(class_col)
)
```

Arguments

x	an OpenSpecy or Specs object for metadata helpers. For build_lib(), one OpenSpecy, a nonempty list containing only OpenSpecy objects, or a nonempty character vector of file paths. The official workflow requires x; it does not guess source-library locations. Each RDS path may store either one OpenSpecy or a list of them; other paths are read with read_any(). Large same-axis source lists are prepared in bulk to avoid repeated legacy object coercion. For rebuild_lib_artifacts(), a completed four-part build object, its RDS path, or a build/output directory containing completed libraries checkpoints.
recipes	named list of process_spec() argument lists or functions. Names become names of the returned libraries.
range, res	wavenumber range and resolution passed to c_spec() when build_lib() combines multiple sources.
id_col	metadata column used as the spectrum identifier.
exclude_ids	identifiers to remove before returning a library.
dedupe	logical; whether to generate stable IDs and remove duplicated spectra in build_lib().
metadata_lookups	a lookup table, csv path, or list of lookup tables and paths. A lookup may instead be supplied as list(lookup = x, by = key) to use an explicit key (including a named metadata-to-lookup key mapping). fill_only = TRUE preserves existing nonblank metadata values while filling gaps from the lookup. The older fallback_by lookup field is deprecated; canonical metadata keys are now filled from reviewed internal aliases before any external join. If non-NULL, each is joined with join_lib_metadata(). Automatic ordinary lookups use the single shared column that has overlapping values and unique lookup keys. Lookups

with no usable shared key are skipped with a message; lookups with multiple usable shared keys are considered ambiguous and stop. Lookup values that share non-key metadata column names are coalesced back into those columns, with non-missing lookup values taking precedence.

material_hierarchy
: hierarchy table or csv path used when non-NULL. It is joined with join_material_hierarchy() using the default "material" metadata key.

metadata_name_lookup
: a data.frame or data.table with canonical_name, source_name, and optional regex columns. The default is returned by lib_metadata_name_lookup(); use NULL to clean names without coalescing aliases.

clean_metadata_values
: logical or NULL; whether build_lib() should lowercase, trim, ASCII-normalize, and normalize blank/unknown character metadata values before joining. NULL enables cleaning for the official output workflow and disables it for composable in-memory builds. Invalid byte sequences are always converted to UTF-8.

convert_intensity
: logical; whether to infer reflectance, transmittance, or absorbance units from each source and convert known non-absorbance spectra with adj_intens() before merging. Object attribute intensity_unit is authoritative when supplied; otherwise metadata column intensity_units is evaluated per spectrum.

restrict_range_args
: optional named list of arguments passed to restrict_range() after unit conversion and source merging. Supplying the list triggers restriction; make_rel = FALSE is used unless explicitly overridden.

signal_noise
: logical; whether to append the default sig_noise() result as metadata column sn.

assess
: logical; whether to run assess_spec() on each output library and append assessment summaries to its metadata.

prune
: NULL, or a named list mapping recipe names to argument lists for prune_lib(). Selected recipes are pruned independently after processing and assessment. NULL preserves unpruned outputs.

progress
: logical; whether build_lib() reports named processing stages and elapsed time, or reduce_lib() reports group sizes and correlation/PAM timings.

workflow_data
: optional directory containing the curated reference CSV tables. If NULL, the directory is discovered beside the calling script or under the current working directory.

output_dir
: NULL for the composable in-memory return, or a directory that triggers the complete checkpointed workflow. The official workflow requires an explicit output directory.

previous_library_dir
: directory containing the seven legacy artifacts used for complete old/new assessment, "system", or NULL to skip external comparison. The end-to-end workflow defaults to "system" and retrieves missing artifacts with get_lib().

reuse
: logical; whether manifest-compatible completed checkpoints and versioned release files may be reused.

remove_other	logical; in the official end-to-end workflow, whether spectra with blank spectrum_identity or the unresolved literal "other" class are removed before quality control, medoid selection, and model fitting. Reviewed broad "other plastic" and "other material" categories remain in the reference libraries and are eligible for prune_lib()'s constrained nearest-class reassignment. Removed and reviewed rows remain visible in assessments$cleanup$summary. Source-only composable builds do not apply the official filter.
seed	random seed used before model training.
holdout	fraction of stable spectrum groups reserved for assessment.
columns	metadata columns to deduplicate into a template.
add	blank columns to add to a template.
path	optional csv path. If NULL, template helpers return a data.table.
lookup	a data.frame, data.table, or csv file path used as a metadata lookup table.
by	named character vector mapping metadata columns to lookup columns, or an unnamed character vector when the names are the same in both tables.
require_complete	logical; if TRUE, incomplete joins fail.
return	whether to return an updated OpenSpecy object, joined table, report list, or selected ids depending on the helper.
suffixes	suffixes used when joined metadata and lookup tables share non-key column names.
hierarchy	a data.frame, data.table, or csv file path with hierarchical material metadata.
key_col	metadata column containing material labels to match.
levels	hierarchy columns ordered from most-specific to most-general.
output_names	names to use for hierarchy columns added to metadata.
duplicate	how duplicated generated identifiers should be handled.
scale	numeric multiplier used before hashing intensity values.
algo	hash algorithm passed to digest().
class_col, type_col	metadata columns used for model labels.
material_type_col	metadata column used to require plastic candidates for "other plastic" and to update the type after generic-class reassignment. "other material" candidates are restricted by class_col to "organic matter" or "mineral"; "other" may match any established class in the spectral pool.
min_n	For prune_lib(), the minimum spectra required for a resolved class within one spectrum type across the complete input database, not within each source library. Smaller groups are reassigned as a whole to their most-correlated eligible class, or removed when no valid destination exists; groups exactly at the threshold are retained. For reduce_lib(), groups with min_n or fewer spectra are kept whole. Model trainers fit only classes meeting the threshold.

cross_class	logical; whether prune_lib() should resolve within-library and then independent-library high-correlation matches between different non-generic material classes before generic-class reassignment. This requires complete canonical library_name metadata. The composable default is FALSE; the official derivative and no-baseline workflow enables it.
cross_class_threshold	numeric Pearson-correlation threshold in [0, 1]. Cross-class correlations strictly greater than this value are conflicts when cross_class = TRUE.
exclude	numeric length-two wavenumber interval excluded from pruning correlations.
group_cols	metadata columns defining groups for reduction.
k	maximum representatives to keep for groups larger than min_n.
alpha	alpha value passed to glmnet().
grouped	logical; whether multinomial coefficients use grouped penalties.
weights	logical; whether to use inverse class-frequency weights for logistic regression or inverse-frequency case sampling for random forest.
make_relative	logical; whether to normalize model inputs with make_rel().
method	classifier to train: "logistic_regression" (the backward-compatible default) or "random_forest". The latter requires the suggested **ranger** package.
nearest	logical; if TRUE, assess_lib() compares each spectrum with its highest-correlation neighbor and reports the fraction where that neighbor has the same class_col value.
...	further arguments passed to the underlying operation.

Details

build_lib() combines sources over their full wavenumber range, optionally adds ordinary and hierarchical metadata, removes requested identifiers, optionally generates stable source-stage duplicate IDs, and applies named processing recipes. Source-stage IDs follow the reference library's legacy hash recipe: each source spectrum is trimmed with manage_na(type = "remove"), conformed at resolution 8, smoothed, and hashed from the resulting wavenumber/intensity vectors before later merging and range restriction. The older 100–4000 cm-1 hash is kept in sample_name_old when id_col = "sample_name" so exclude_ids can remove both current and legacy curated bad IDs. Metadata column names are first converted to lowercase underscore names and known aliases are coalesced using metadata_name_lookup; see lib_clean_metadata() for automatic and regular-expression matching. Metadata values can optionally be normalized to lowercase trimmed character values before lookup joins. spectrum_identity is also reduced to a basename when it is a recognizable path, then trailing extensions supported by read_any() are removed. The same normalization is applied to exact lookup keys. Regex class rules belong in a separate table and can be applied afterward with predict_class_reference(). This keeps filenames usable as identities without treating file containers as part of a material name. By default, each source is also converted to absorbance before merging when its intensity units are known. A nonempty intensity_unit object attribute takes precedence over the per-spectrum intensity_units metadata column. Each recipe is either a named list of arguments passed to process_spec() or a function accepting one OpenSpecy object. An empty recipe returns an unprocessed copy. Signal-to-noise is added by default, and optional assess_spec() results are summarized into one metadata row per spectrum.

Progress messages report named stages and elapsed time by default so long-running builds remain observable.

The official workflow requires explicit source paths and an output directory. When `workflow_data` is omitted, `build_lib()` looks for the curated helper tables under `data/` beside the calling script, then under `data/` or `workflows/data/` in the current working directory. It writes each completed stage under `output_dir/checkpoints`, and promotes validated legacy-compatible files into a versioned release directory. With `reuse = TRUE`, a checkpoint is reused only when its manifest signature matches the source files, curated tables, relevant arguments, package version, and builder implementation. Full assessments use the complete candidate and legacy artifacts. Seeded ten-percent holdouts are allocated independently within each source across class/type strata after physical identifiers and exact spectral-content duplicates have been joined into stable groups. Candidate artifacts are assessed on candidate data and legacy artifacts on legacy data, so taxonomy changes do not require fuzzy cross-version class matching. Query identifiers are removed from full and references by group before matching to prevent transformed duplicate or physical-replicate self-matches. Each medoid artifact separately identifies its complete corresponding processed library, measuring the deployed medoid search directly without another split. Model assessments do not retrain models: each existing candidate or legacy model identifies its complete corresponding source dataset once. This measures the deployed artifact directly and keeps assessment generation bounded by prediction rather than model fitting. After derivative and baseline-removal processing, FTIR spectra whose 2200–2420 CO2-region maximum exceeds twice the 2420–2550 silent-region maximum are flattened and reassessed; failed postconditions are removed. High-tail checks use each spectrum's finite support; tails are trimmed and reassessed, failed corrections are removed, and spectra with running signal-to-noise below two are removed before pruning. Full artifacts are then partitioned into Raman (200–4000), FTIR (400–4000), and NIR (4000–12000) `OpenSpecy` objects. After pruning and class reassignment, the official workflow derives `material_form` from the curated form-regex CSV and all atomic metadata values, then joins evidence-backed `common_use` by final material class. Existing form values take precedence when uniquely standardized; conflicting form categories remain missing and are audited. Common use may be `"consumer"`, `"industrial"`, `"mixed"`, or missing; quantitative mass shares are retained when available, while sourced qualitative proposals remain explicit in the review table. After full and medoid libraries are complete, metadata columns containing only missing values are removed, except that these two standardized fields are retained, and the remainder are stably ordered from the fewest to the most missing values. Spectra, metadata rows, identifiers, axes, and object attributes are unchanged. Official class completion temporarily assigns unresolved identities to `"other"`. By default, spectra with a blank identity or that unresolved literal class are removed before quality control and retained in the `other_review` assessment table. Reviewed `"other plastic"` and `"other material"` rows stay in the reference library and enter `prune_lib()`'s nearest-class semisupervised pathway. When requested, `prune_lib()` first resolves high-correlation conflicts between different reviewed classes within each source library. It repeatedly removes the spectrum with the most active wrong-class neighbors, recalculates after each round, and removes both endpoints of an adjacent maximum-score tie. It then resolves between-library conflicts by the number of distinct independent libraries supporting the opposing class. Generic and unclassified labels do not participate. Official builds repeat closure on the rounded typed and model-range views before deriving medoids, and retain excluded spectra plus conflict provenance in `quarantined_spectra.rds`. Pruning then reassigns generic classes by nearest same-technique correlation: `"other"` may use any established class, `"other plastic"` requires a plastic candidate, and `"other material"` requires `"organic matter"` or `"mineral"`. The matched material type and a correlation audit are retained. Once those labels are resolved, each class/spectrum-type group with fewer than `min_n` spectra is reassigned as a whole

to its most-correlated established class in the same technique pool and material type. A group is removed only when no eligible correlated destination exists. The report identifies its support, destination, class-level correlation, action, reason, and affected spectrum identifiers.

`make_lib_lookup_template()` creates a deduplicated table of metadata values from an `OpenSpecy` or `Specs` object. Users can fill the added columns in R or write the template to CSV and curate it elsewhere.

`join_lib_metadata()` left-joins lookup columns onto object metadata and reports unmatched metadata keys, duplicate lookup keys, and missing joined values. Joins are exact; clean or harmonize values before calling this helper.

`join_material_hierarchy()` joins user-defined hierarchical material metadata. The supplied `levels` are tried from most-specific to most-general so a material label can match any level in the hierarchy.

`dedupe_spec()` hashes the current spectra and wavenumber axis to create stable IDs and remove duplicated spectra. Process or conform spectra before this step when that should affect duplicate detection.

`reduce_lib()` uses PAM medoids to keep representative spectra within each metadata group. It uses OpenSpecy's optimized correlation routine on relative spectra whose missing values are temporarily replaced by each spectrum's finite mean. Groups of at most 3,000 spectra use exact PAM; oversized groups use five deterministic 1,000-spectrum PAM samples and keep the candidate set with the best full-group correlation-distance objective. Official medoids are then selected from the original object so their genuine missing values are preserved.

`train_spec_model()` trains either OpenSpecy's multinomial logistic regression model (`method = "logistic_regression"`) or an experimental probability random forest (`method = "random_forest"`). Logistic regression uses inverse class weights and stratified cross-validation to select the lambda with the highest out-of-fold macro class accuracy. Random forest uses inverse-frequency balanced case sampling and out-of-bag predictions. This improves minority-class representation in each bootstrap sample without applying a second class-vote correction. Treat the stored out-of-bag metrics as fit diagnostics: balanced resampling can make them optimistic. The library workflow separately reports accuracy from applying each deployed model to its complete corresponding dataset. Missing training values are replaced with the finite mean at each wavenumber. The returned model carries the same training-mean filler so `match_spec()` can identify partially covered spectra. `build_model_lib()` is the backward-compatible wrapper used by older scripts; `build_lib()` calls the dedicated trainer for official models.

`assess_lib()` returns a compact summary of object validity, library size, class balance, and optionally nearest-neighbor class consistency.

`rebuild_lib_artifacts()` starts from completed type-keyed libraries and rebuilds only medoids, models, and assessments. Its input and output locations are explicit, and every downstream component is checkpointed so a compatible interrupted run can resume without repeating completed work. Checkpoint and release payloads carry SHA-256 hashes, and an existing versioned release path is accepted only when its payload is unchanged.

Value

Each library returned by `build_lib()` includes a `spectrum_identity_cleanup_report` attribute listing changed original and normalized identities with their counts. In composable mode, `build_lib()` returns a named list of `OpenSpecy` libraries. Its end-to-end mode returns one list containing `libraries`,

medoids, models, and assessments. Official libraries and medoids are nested by recipe and then ftir, raman, or nir. Models are nested by algorithm, recipe, and spectrum type. FTIR and Raman medoids/models use 800–3200 while the NIR interval is derived from finite coverage within 4000–12000. Assessments use five ordered process lists: cleanup, ref_lib, medoid, model, and functionality, with no more than ten nonempty review tables in total. Every spectrum receives a derived library_name: populated organization first, otherwise user_name. The cleanup summary includes source-library counts at each major stage and identifies the first stage and reason whenever an entire source library is dropped. Accuracy tables contain overall aggregate metrics only in long form, confusion tables retain misidentifications only, and model error-mode tables compare accuracy percentages with and without each automated-test flag. Review tables reject columns with more than 10 percent missing values. Row-level tests, split manifests, model-training diagnostics, and release manifests remain hash-addressed evidence attributes rather than additional review leaves. Each in-memory training model contains one tests data.table and a one-spectrum fill object. Versioned release directories instead store global build and model diagnostics only in assessments.rds; library, medoid, and model files retain only runtime data, scientific attributes, and prediction state. The companion reference_library_build.rds is a lightweight release index. The companion quarantined_spectra.rds stores valid OpenSpecy objects by recipe/type, a long conflict table, and build provenance for spectra excluded by cross-class closure. join_lib_metadata(), join_material_hierarchy(), dedupe_spec(), prune_lib(), and reduce_lib() return an updated spectral object unless return requests a table, report, or ids. make_lib_lookup_template() returns a data.table unless path is supplied, in which case it writes the csv and invisibly returns the table. train_spec_model() and build_model_lib() return a list suitable for AI classification with match_spec() and one tidy tests table instead of separate accuracy/confusion summaries. It also contains typed lambda_metrics and support tables. Random-forest results also contain out-of-bag metrics and feature importance. assess_lib() returns a data.table summary.

Author(s)

Win Cowger

See Also

match_spec() for deploying trained models and plotly_spec() for logistic coefficient overlays.

Examples

```
wavenumber <- seq(100, 6100, by = 100)
base_a <- dnorm(seq(-3, 3, length.out = length(wavenumber)))
base_b <- rev(cumsum(seq_along(wavenumber)))
spectra <- cbind(base_a, base_a + 0.1, base_a + 0.2,
                 base_b, base_b + 0.1, base_b + 0.2)
colnames(spectra) <- paste0("s", seq_len(ncol(spectra)))
mini <- as_OpenSpecy(
  wavenumber,
  spectra = spectra,
  metadata = data.table::data.table(
    sample_name = colnames(spectra),
    source = rep(c("A", "B"), each = 3),
    label = c("nylon 6", "polyamides", "nylon 6",
```

```
              "pet", "polyesters", "pet"),
    material_class = rep(c("polyamides", "polyesters"), each = 3),
    spectrum_type = rep("ftir", 6),
    intensity_units = rep("absorbance", 6)
  ),
  attributes = list(intensity_unit = "absorbance")
)

name_lookup <- lib_metadata_name_lookup()
name_lookup[name_lookup$canonical_name == "material_color", ]

make_lib_lookup_template(mini, columns = "source", add = "library_type")

source_lookup <- data.frame(
  source = c("A", "B"),
  library_type = c("lab", "field"),
  material = c("nylon 6", "pet")
)
joined <- join_lib_metadata(mini, source_lookup, by = "source",
                            require_complete = TRUE)

hierarchy <- data.frame(
  material = c("nylon 6", "pet"),
  material_class = c("polyamides", "polyesters"),
  material_type = c("plastic", "plastic")
)
joined <- join_material_hierarchy(joined, hierarchy, key_col = "label",
                                  require_complete = TRUE)

deduped <- dedupe_spec(joined)
reduced <- reduce_lib(deduped, group_cols = "material_class",
                      k = 1, min_n = 1)
libs <- build_lib(
  mini,
  recipes = list(
    raw = list(),
    derivative = list(
      conform_spec = FALSE,
      smooth_intens = TRUE,
      smooth_intens_args = list(window = 15, derivative = 1),
      make_rel = TRUE
    )
  ),
  metadata_lookups = source_lookup,
  material_hierarchy = hierarchy,
  restrict_range_args = list(min = 100, max = 6000),
  assess = TRUE,
  dedupe = FALSE
)

model <- suppressWarnings(train_spec_model(
  joined, class_col = "material_class", type_col = NULL, min_n = 2,
  nlambda = 3
```

```
))
assess_lib(libs$raw, class_col = "material_class", nearest = FALSE)
```

`calculate_emissivity` *Calculate spectral emissivity at supplied material temperatures*

Description

Materialize spectral emissivity for a selected or particle-collapsed calibrated-radiance `OpenSpecy` object. Estimate temperature after particle collapse because TES is nonlinear. Values outside `[0, 1]` are retained as calibration and model diagnostics.

Usage

```
calculate_emissivity(x, ...)

## Default S3 method:
calculate_emissivity(x, ...)

## S3 method for class 'OpenSpecy'
calculate_emissivity(x, temperature_k, downwelling, block_size = NULL, ...)
```

Arguments

`x`	A selected or collapsed calibrated-radiance `OpenSpecy` object.
`...`	Additional arguments passed to methods.
`temperature_k`	A positive material temperature in kelvin, supplied as one scalar or one value per spectrum.
`downwelling`	Required scalar, bandwise numeric, or one-spectrum `OpenSpecy` downwelling radiance.
`block_size`	Optional positive whole-number compute block size. `NULL` selects a bounded value automatically.

Value

An aligned `OpenSpecy` object whose spectra and per-spectrum units are emissivity. The supplied temperatures and radiative-transfer model are appended to metadata, and transformation provenance is recorded.

Experimental

`calculate_emissivity()` is experimental. Its radiative-transfer model, diagnostics, defaults, and recorded provenance may change as the method is validated against more calibrated thermal-emission measurements.

See Also

estimate_temperature()

---

check_lib *Manage spectral libraries*

Description

These functions will import the spectral libraries from Open Specy if they were not already downloaded. The CRAN does not allow for deployment of large datasets so this was a workaround that we are using to make sure everyone can easily get Open Specy functionality running on their desktop. Please see the references when using these libraries. These libraries are the accumulation of a massive amount of effort from independant groups and each should be attributed when you are using their data.

Usage

```
check_lib(
  type = c("derivative", "nobaseline", "raw", "medoid_derivative", "medoid_nobaseline",
    "model_derivative", "model_nobaseline"),
  path = "system",
  condition = "warning"
)

get_lib(
  type = c("derivative", "nobaseline", "raw", "medoid_derivative", "medoid_nobaseline",
    "model_derivative", "model_nobaseline"),
  path = "system",
  mode = "wb",
  revision = NULL,
  ...
)

load_lib(type, path = "system")

rm_lib(
  type = c("derivative", "nobaseline", "raw", "medoid_derivative", "medoid_nobaseline",
    "model_derivative", "model_nobaseline"),
  path = "system"
)
```

Arguments

type	library type to check/retrieve; defaults to c("derivative", "nobaseline", "raw", "medoid_derivative", "medoid_nobaseline", "model_derivative", "model_nobaseline") which reads everything.

path	where to save or look for local library files; defaults to "system" pointing to system.file("extdata", package = "OpenSpecy").
condition	determines if check_lib() should warn ("warning", the default) or throw and error ("error").
mode	see ?download.file for details on mode.
revision	optional AWS S3 versionId. A single value applies to every requested type; a named character vector can pin each type separately. If NULL, the current unversioned object is downloaded.
...	further arguments passed to download.file().

Details

check_lib() checks to see if the Open Specy reference library already exists on the users computer. get_lib() downloads the Open Specy library from its AWS distribution. load_lib() will load the library into the global environment for use with the Open Specy functions. rm_lib() removes the libraries from your computer.

Value

check_lib() and get_lib() return messages only; load_lib() returns an OpenSpecy object containing the respective spectral reference library.

Author(s)

Zacharias Steinmetz, Win Cowger

References

Bell IB, Clark RJH, Gibbs PJ (2010). "Raman Spectroscopic Library." *Christopher Ingold Laboratories, University College London, UK*.

Berzinš K, Sales RE, Barnsley JE, Walker G, Fraser-Miller SJ, Gordon KC (2020). "Low-Wavenumber Raman Spectral Database of Pharmaceutical Excipients." *Vibrational Spectroscopy* **107**, 103021. doi:10.5281/zenodo.3614035.

Cabernard L, Roscher L, Lorenz C, Gerdts G, Primpke S (2018). "Comparison of Raman and Fourier Transform Infrared Spectroscopy for the Quantification of Microplastics in the Aquatic Environment." *Environmental Science & Technology* **52**(22), 13279–13288. doi:10.1021/acs.est.8b03438.

Caggiani MC, Cosentino A, Mangone A (2016). "Pigments Checker version 3.0, a handy set for conservation scientists: A free online Raman spectra database." *Microchemical Journal* **129**, 123–132. doi:10.1016/j.microc.2016.06.020.

Chabuka BK, Kalivas JH (2020). "Application of a Hybrid Fusion Classification Process for Identification of Microplastics Based on Fourier Transform Infrared Spectroscopy. Applied Spectroscopy." *Applied Spectroscopy* **74**(9), 1167–1183. doi:10.1177/0003702820923993.

Cowger, W (2023). "Library data." *OSF*. doi:10.17605/OSF.IO/X7DPZ.

Cowger W, Gray A, Christiansen SH, De Frond H, Deshpande AD, Hemabessiere L, Lee E, Mill L, et al. (2020). "Critical Review of Processing and Classification Techniques for Images and Spectra in Microplastic Research." *Applied Spectroscopy*, **74**(9), 989–1010. doi:10.1177/0003702820929064.

Cowger W, Roscher L, Chamas A, Maurer B, Gehrke L, Jebens H, Gerdts G, Primpke S (2023). "High Throughput FTIR Analysis of Macro and Microplastics with Plate Readers." *ChemRxiv Preprint*. doi:10.26434/chemrxiv2023x88ss.

De Frond H, Rubinovitz R, Rochman CM (2021). "µATR-FTIR Spectral Libraries of Plastic Particles (FLOPP and FLOPP-e) for the Analysis of Microplastics." *Analytical Chemistry* **93**(48), 15878–15885. doi:10.1021/acs.analchem.1c02549.

El Mendili Y, Vaitkus A, Merkys A, Gražulis S, Chateigner D, Mathevet F, Gascoin S, Petit S, Bardeau JF, Zanatta M, Secchi M, Mariotto G, Kumar A, Cassetta M, Lutterotti L, Borovin E, Orberger B, Simon P, Hehlen B, Le Guen M (2019). "Raman Open Database: first interconnected Raman–X-ray diffraction open-access resource for material identification." *Journal of Applied Crystallography*, **52**(3), 618–625. doi:10.1107/s1600576719004229.

Johnson TJ, Blake TA, Brauer CS, Su YF, Bernacki BE, Myers TL, Tonkyn RG, Kunkel BM, Ertel AB (2015). "Reflectance Spectroscopy for Sample Identification: Considerations for Quantitative Library Results at Infrared Wavelengths." *International Conference on Advanced Vibrational Spectroscopy (ICAVS 8)*. https://www.osti.gov/biblio/1452877.

Lafuente R, Downs RT, Yang H, Stone N (2016). "The power of databases: The RRUFF project." *Highlights in Mineralogical Crystallography*. doi:10.1515/9783110417104003.

Munno K, De Frond H, O'Donnell B, Rochman CM (2020). "Increasing the Accessibility for Characterizing Microplastics: Introducing New Application-Based and Spectral Libraries of Plastic Particles (SLoPP and SLoPP-E)." *Analytical Chemistry* **92**(3), 2443–2451. doi:10.1021/acs.analchem.9b03626.

Myers TL, Brauer CS, Su YF, Blake TA, Johnson TJ, Richardson RL (2014). "The influence of particle size on infrared reflectance spectra." *Proceedings Volume 9088, Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XX*, 908809. doi:10.1117/12.2053350.

Myers TL, Brauer CS, Su YF, Blake TA, Tonkyn RG, Ertel AB, Johnson TJ, Richardson RL (2015). "Quantitative reflectance spectra of solid powders as a function of particle size." *Applied Optics* **54**(15), 4863–4875. doi:10.1364/ao.54.004863.

Primpke S, Wirth M, Lorenz C, Gerdts G (2018). "Reference database design for the automated analysis of microplastic samples based on Fourier transform infrared (FTIR) spectroscopy." *Analytical and Bioanalytical Chemistry* **410**, 5131—5141. doi:10.1007/s002160181156x.

Roscher L, Fehres A, Reisel L, Halbach M, Scholz-Böttcher B, Gerriets M, Badewien TH, Shiravani G, Wurpts A, Primpke S, Gerdts G (2021). "Abundances of large microplastics (L-MP, 500-5000 µm) in surface waters of the Weser estuary and the German North Sea." *PANGAEA*. doi:10.1594/PANGAEA.938143.

"Handbook of Raman Spectra for geology" (2023).

"Scientific Workgroup for the Analysis of Seized Drugs." (2023). https://swgdrug.org/ir.htm.

Further contribution of spectra: Suja Sukumaran (Thermo Fisher Scientific), Aline Carvalho, Jennifer Lynch (NIST), Claudia Cella and Dora Mehn (JRC), Horiba Scientific, USDA Soil Characterization Data, Archaeometrielabor, and S.B. Engelsen (Royal Vet. and Agricultural University, Denmark). Kimmel Center data was collected and provided by Prof. Steven Weiner (Kimmel Center for Archaeological Science, Weizmann Institute of Science, Israel).

Examples

```
## Not run:
#check to see if you have the library already
```

```
check_lib("derivative")

#get the library stored in your system from online repo
get_lib("derivative")

#load the library into the working environment
spec_lib <- load_lib("derivative")

#for models you should choose either both, ftir, or raman as a list item before use
get_lib("model_derivative")
mod_lib <- load_lib("model_derivative")[["ftir"]]


## End(Not run)
```

---

collapse_spec *Define features*

Description

Functions for analyzing features, like particles, fragments, or fibers, in spectral map oriented OpenSpecy object.

Usage

```
collapse_spec(x, ...)

## Default S3 method:
collapse_spec(x, ...)

## S3 method for class 'OpenSpecy'
collapse_spec(x, fun = median, column = "feature_id", ...)

def_features(x, ...)

## Default S3 method:
def_features(x, ...)

## S3 method for class 'OpenSpecy'
def_features(
  x,
  features,
  shape_kernel = c(3, 3),
  shape_type = "box",
  close = F,
  close_kernel = c(4, 4),
  close_type = "box",
```

```
  img = NULL,
  bottom_left = NULL,
  top_right = NULL,
  ...
)
```

Arguments

x	an OpenSpecy object
fun	function name to collapse by.
column	column name in metadata to collapse by.
features	a logical vector or character vector describing which of the spectra are of features (TRUE) and which are not (FALSE). If a character vector is provided, it should represent the different feature types present in the spectra.
shape_kernel	the width and height of the area in pixels to search for connecting features, c(3,3) is typically used but larger numbers will smooth connections between particles more.
shape_type	character, options are for the shape used to find connections c("box", "disc", "diamond")
close	logical, whether a closing should be performed using the shape kernel before estimating components.
close_kernel	width and height of the area to close if using the close option.
close_type	character, options are for the shape used to find connections c("box", "disc", "diamond")
img	a file location where a visual image is that corresponds to the spectral image.
bottom_left	a two value vector specifying the x,y location in image pixels where the bottom left of the spectral map begins. y values are from the top down while x values are left to right.
top_right	a two value vector specifying the x,y location in the visual image pixels where the top right of the spectral map extent is. y values are from the top down while x values are left to right.
...	additional arguments passed to subfunctions.

Details

def_features() accepts an OpenSpecy object and a logical or character vector describing which pixels correspond to particles. collapse_spec() takes an OpenSpecy object with particle-specific metadata (from def_features()) and collapses the spectra with a function intensities for each unique particle. It also updates the metadata with centroid coordinates, while preserving the feature information on area and Feret max.

Value

An OpenSpecy object appended with metadata about the features or collapsed for the features. All units are in pixels. Metadata described below.

`x` x coordinate of the pixel or centroid if collapsed

`y` y coordinate of the pixel or centroid if collapsed

`feature_id` unique identifier of each feature

`area` area in pixels of the feature

`perimeter` perimeter of the convex hull of the feature

`rectangular_min` area divided by `feret_max`, retained as the legacy rectangular-width approximation

`feret_min` width of the bounding box perpendicular to the `feret_max` axis

`feret_max` largest dimension of the convex hull of the feature

`convex_hull_area` area of the convex hull

`centroid_x` mean x coordinate of the feature

`centroid_y` mean y coordinate of the feature

`first_x` first x coordinate of the feature

`first_y` first y coordinate of the feature

`rand_x` random x coordinate from the feature

`rand_y` random y coordinate from the feature

`r` if using visual imagery overlay, the red band value at that location

`g` if using visual imagery overlay, the green band value at that location

`b` if using visual imagery overlay, the blue band value at that location

Author(s)

Win Cowger, Zacharias Steinmetz

Examples

```
tiny_map <- read_extdata("CA_tiny_map.zip") |> read_any()
identified_map <- def_features(tiny_map, tiny_map$metadata$x == 0)
collapse_spec(identified_map)
```

---

conform_spec *Conform spectra to a standard wavenumber series*

Description

Spectra can be conformed to a standard suite of wavenumbers to be compared with a reference library or to be merged to other spectra.

Usage

```
conform_spec(x, ...)

## Default S3 method:
conform_spec(x, ...)

## S3 method for class 'OpenSpecy'
conform_spec(x, range = NULL, res = 5, allow_na = F, type = "interp", ...)
```

Arguments

x	a list object of class OpenSpecy.
range	a vector of new wavenumber values, can be just supplied as a min and max value.
res	spectral resolution adjusted to or NULL if the raw range should be used.
allow_na	logical; should NA values in places beyond the wavenumbers of the dataset be allowed?
type	the type of wavenumber adjustment to make. "interp" results in linear interpolation while "roll" conducts a nearest rolling join of the wavenumbers. "mean_up" assigns source values by midpoint and returns one row for every requested wavenumber: occupied bins are averaged, while requested positions with no source value are linearly interpolated. This can maintain smaller peaks, match either a coarser or finer target axis, and avoid full-size grouping copies. The mean-up option is still experimental.
...	further arguments passed to approx()

Value

adj_intens() returns a data frame containing two columns named "wavenumber" and "intensity"

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

restrict_range() and flatten_range() for adjusting wavenumber ranges; subtr_baseline() for spectral background correction

Examples

```
data("raman_hdpe")
conform_spec(raman_hdpe, c(1000, 2000))
```

`correct_spike` *Detect and correct isolated spikes in spectra*

Description

`correct_spike()` detects isolated positive or negative intensity artifacts and replaces only accepted spike intervals with local interpolation. The default MAD-prominence-width method detects narrow positive and negative peaks, estimates noise from the raw median absolute deviation of first differences, and replaces detected intervals from nearby clean values. The legacy residual method compares each point with a wavenumber-aware local interpolation and scales the residual by a local median absolute deviation. Two additional paper-backed methods use peak prominence and width measured in sample (CCD-pixel) units.

Usage

```
correct_spike(x, ...)

## Default S3 method:
correct_spike(x, ...)

## S3 method for class 'OpenSpecy'
correct_spike(
  x,
  method = c("mad_prominence_width", "residual", "prominence_fwhm",
    "prominence_fwhm_ratio"),
  direction = c("both", "positive", "negative"),
  residual_window = 5L,
  residual_threshold = 8,
  residual_max_width = 1L,
  prominence_threshold = NULL,
  width_threshold = NULL,
  noise_multiplier = 10,
  rel_height = 0.8,
  interpolation_points = 5L,
  interpolation = c("linear", "quadratic"),
  z_threshold = 3.5,
  min_peaks = 20L,
  ...
)
```

Arguments

`x`	an `OpenSpecy` object.
`method`	character; detection method. One of `"mad_prominence_width"` (default), `"residual"`, `"prominence_fwhm"`, or `"prominence_fwhm_ratio"`.
`direction`	character; detect `"both"` positive and negative spikes, only `"positive"` spikes, or only `"negative"` spikes.

`residual_window`
: positive integer; points on each side used by the local residual predictor.

`residual_threshold`
: positive numeric; absolute robust residual score required by the residual method.

`residual_max_width`
: positive integer; widest consecutive candidate interval accepted by the residual method. The one-point default is deliberately conservative.

`prominence_threshold`
: positive numeric or `NULL`; minimum peak prominence. `NULL` uses `noise_multiplier` times the raw MAD of first differences for the default method; it remains required for the manual prominence/FWHM method.

`width_threshold`
: positive numeric or `NULL`; maximum peak FWHM in sample (CCD-pixel) units for the manual prominence/FWHM method. `NULL` uses 2 points for the default method.

`noise_multiplier`
: positive numeric multiplier for the automatic raw MAD prominence threshold used by `method = "mad_prominence_width"`.

`rel_height`
: numeric in `(0, 1]`; prominence fraction at which the interval replaced by paper methods is measured. Coca-Lopez used `0.8` for most examples.

`interpolation_points`
: positive integer; finite, unflagged neighboring points required on each side of an accepted interval. This is the paper's `m` parameter.

`interpolation`
: character; `"linear"` (default) or `"quadratic"` local interpolation.

`z_threshold`
: positive numeric; upper Z-score threshold for automated prominence/FWHM-ratio detection. The paper uses values greater than `3.5`.

`min_peaks`
: integer of at least two; minimum number of measurable peaks used to estimate automated ratio outliers.

`...`
: must be empty. Unexpected arguments are rejected so detector tuning misspellings cannot be silently ignored.

Details

`method = "mad_prominence_width"` reproduces the automated workflow supplied by Nicolas Coca Lopez without requiring `pracma`. With the defaults, peaks must span no more than 2 points and exceed 10 times the raw MAD of first differences. `prominence_threshold` may override that automatic threshold. Detected intervals receive a one-point guard on either side and use a 5-point local interpolation window; at an edge the nearest clean value is used rather than extrapolating a slope.

`method = "prominence_fwhm"` requires user-supplied `prominence_threshold` and `width_threshold` values. These thresholds depend on the material, instrument, spectral resolution, and acquisition settings; the graphene values reported by Coca-Lopez are deliberately not universal defaults. `method = "prominence_fwhm_ratio"` instead treats prominence/FWHM values above `z_threshold` standard deviations as spikes and requires at least `min_peaks` measurable peaks.

Peak widths and flagged intervals follow the prominence contour definition used by `scipy.signal.peak_widths()`: FWHM is measured at `rel_height = 0.5`, while the interval replaced is measured at the requested `rel_height`. The paper-backed modes require `interpolation_points` finite, unflagged samples

on both sides; boundary values are never wrapped. The default method searches that many points on either side and uses the nearest clean value when only one side exists. Close spike intervals are merged before interpolation so one spike cannot be used to repair another. Linear interpolation that materially disagrees with a local quadratic reconstruction over a multi-point interval is rejected to avoid silently truncating an underlying broad band.

Correction proceeds through bounded transactional passes while the detector's correctable count strictly decreases. This lets a newly revealed spike be corrected without rolling back safe earlier replacements. Processing stops on no progress; boundary, interpolation, and band-protection safeguards stay in force, and any remaining safeguarded candidates are recorded rather than forced.

No single-spectrum method can always distinguish a cosmic-ray spike from a genuine band with the same shape. Calibrate paper thresholds on representative standards, especially for narrow-band materials such as calcite and polystyrene, and inspect the `automatic_spike` diagnostic attribute.

Value

An `OpenSpecy` object with accepted spike intervals corrected. The wavenumber axis, spectra dimensions and names, metadata alignment, and existing attributes are preserved. A successful or rejected attempted correction stores an `automatic_spike` attribute containing the method, parameters, corrected and rejected regions, affected spectra, detector counts, pass count, and transaction reason. If nothing is detected, `x` is returned unchanged.

Author(s)

Win Cowger, Nicolas Coca Lopez

References

Coca-Lopez N (2024). "An intuitive approach for spike removal in Raman spectra based on peaks' prominence and width." *Analytica Chimica Acta*, **1295**, 342312. doi:10.1016/j.aca.2024.342312.

Examples

```
wave <- seq(400, 1800, length.out = 101)
values <- sin(wave / 200)
values[51] <- values[51] + 20
spectrum <- as_OpenSpecy(wave, data.frame(sample = values))
corrected <- correct_spike(spectrum)
```

`cor_spec` *Identify and filter spectra*

Description

match_spec() joins two OpenSpecy objects and their metadata based on similarity. cor_spec() correlates two OpenSpecy objects, typically one with knowns and one with unknowns. ident_spec() retrieves the top match values from a correlation matrix and formats them with metadata. get_metadata() retrieves metadata from OpenSpecy objects. max_cor_named() formats the top correlation values from a correlation matrix as a named vector. filter_spec() filters an Open Specy object. fill_spec() adds filler values to an OpenSpecy object where it doesn't have intensities. os_similarity() EXPERIMENTAL, returns a single similarity metric between two OpenSpecy objects based on the method used.

Usage

```
cor_spec(x, ...)

## Default S3 method:
cor_spec(x, ...)

## S3 method for class 'OpenSpecy'
cor_spec(
  x,
  library,
  na.rm = T,
  conform = F,
  type = "roll",
  compute = "optimized",
  ...
)

match_spec(x, ...)

## Default S3 method:
match_spec(x, ...)

## S3 method for class 'OpenSpecy'
match_spec(
  x,
  library,
  na.rm = T,
  conform = F,
  type = "roll",
  top_n = NULL,
  order = NULL,
  top_n_by = NULL,
  batch_size = NULL,
  add_library_metadata = NULL,
  add_object_metadata = NULL,
  compute = "optimized",
  fill = NULL,
```

```
  ...
)

ident_spec(
  cor_matrix,
  x,
  library,
  top_n = NULL,
  add_library_metadata = NULL,
  add_object_metadata = NULL,
  ...
)

get_metadata(x, ...)

## Default S3 method:
get_metadata(x, ...)

## S3 method for class 'OpenSpecy'
get_metadata(x, logic, rm_empty = TRUE, ...)

max_cor_named(cor_matrix, na.rm = T)

filter_spec(x, ...)

## Default S3 method:
filter_spec(x, ...)

## S3 method for class 'OpenSpecy'
filter_spec(x, logic, ...)

ai_classify(x, ...)

## Default S3 method:
ai_classify(x, ...)

## S3 method for class 'OpenSpecy'
ai_classify(x, library, fill = NULL, top_n = 1L, ...)

fill_spec(x, ...)

## Default S3 method:
fill_spec(x, ...)

## S3 method for class 'OpenSpecy'
fill_spec(x, fill, ...)

os_similarity(x, ...)
```

```
## Default S3 method:
os_similarity(x, ...)

## S3 method for class 'OpenSpecy'
os_similarity(x, y, method = "hamming", na.rm = T, ...)
```

Arguments

x	an OpenSpecy object, typically with unknowns.
library	an OpenSpecy or trained model object representing the reference library of spectra or model to use in identification.
na.rm	logical; indicating whether missing values should be removed when calculating correlations. Default is TRUE.
conform	Whether to conform the spectra to the library wavenumbers or not.
type	the type of conformation to make returned by conform_spec()
compute	the compute strategy used for correlation, "optimized" by default will use the current most optimized strategy for Pearson correlation, "base" will use base R's cor()
top_n	integer; specifying the number of top matches to return. For spectral libraries, NULL returns all matches. Model libraries preserve their historical single winning class when NULL; a positive value returns that many ranked class probabilities per spectrum.
order	an OpenSpecy used for sorting, ideally the unprocessed one; NULL skips sorting.
top_n_by	optional single library metadata column name. When supplied for a spectral library, top_n matches are retained independently for each non-missing group (for example, "organization"). It is not supported for trained model libraries.
batch_size	optional positive integer number of query spectra to match per correlation block. For spectral libraries this bounds peak memory and requires a finite top_n; NULL preserves the ordinary dense correlation path unless top_n_by requires grouped matching. It is not supported for trained model libraries.
add_library_metadata	name of a column in the library metadata to be joined; NULL if you don't want to join.
add_object_metadata	name of a column in the object metadata to be joined; NULL if you don't want to join.
fill	an OpenSpecy object with a single spectrum to be used to fill missing values for alignment with AI classification. When omitted and the model library contains a fill object, that stored training filler is used. Finite query values replace the filler; query NAs do not.
cor_matrix	a correlation matrix for object and library, can be returned by cor_spec()
logic	a logical or numeric vector describing which spectra to keep.
rm_empty	logical; whether to remove empty columns in the metadata.

y	an OpenSpecy object to perform similarity search against x.
method	the type of similarity metric to return.
...	additional arguments passed cor().

Value

match_spec() and ident_spec() will return a data.table-class() containing correlations between spectra and the library. The table has three columns: object_id, library_id, and match_val. Each row represents a unique pairwise correlation between a spectrum in the object and a spectrum in the library. If top_n is specified, only the top top_n matches for each object spectrum will be returned. If add_library_metadata is is.character, the library metadata will be added to the output. If add_object_metadata is is.character, the object metadata will be added to the output. filter_spec() returns an OpenSpecy object. fill_spec() returns an OpenSpecy object. cor_spec() returns a correlation matrix. get_metadata() returns a data.table-class() with the metadata for columns which have information. os_similarity() returns a single numeric value representing the type of similarity metric requested. 'wavenumber' similarity is based on the proportion of wavenumber values that overlap between the two objects, 'metadata' is the proportion of metadata column names, 'hamming' is something similar to the hamming distance where we discretize all spectra in the OpenSpecy object by wavenumber intensity values and then relate the wavenumber intensity value distributions by mean difference in min-max normalized space. 'pca' tests the distance between the OpenSpecy objects in PCA space using the first 4 component values and calculating the max-range normalized distance between the mean components. The first two metrics are pretty straightforward and definitely ready to go, the 'hamming' and 'pca' metrics are pretty experimental but appear to be working under our current test cases.

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

adj_intens() converts spectra; get_lib() retrieves the Open Specy reference library; load_lib() loads the Open Specy reference library into an R object of choice

Examples

```
data("test_lib")

unknown <- read_extdata("ftir_ldpe_soil.asp") |>
  read_any() |>
  conform_spec(range = test_lib$wavenumber,
               res = spec_res(test_lib)) |>
  process_spec()
cor_spec(unknown, test_lib)

match_spec(unknown, test_lib, add_library_metadata = "sample_name",
           top_n = 1)
test_lib$metadata[["organization"]] <- rep(
  c("collection_a", "collection_b"), length.out = nrow(test_lib$metadata)
)
```

```
match_spec(unknown, test_lib, top_n = 1, top_n_by = "organization")
```

---

crowd_lookup *Particle analysis validation metrics*

Description

Helpers for assessing particle crowding, spike recovery, minimum detectable amount (MDA), and batch detection limit (BDL) from particle-count tables.

Usage

```
crowd_lookup(
  x,
  sample_col = "sample_id",
  area_col = "area_um2",
  size_col = "min_length_um",
  material_col = NULL,
  group_cols = sample_col,
  size_threshold = 500,
  surface_area = NULL,
  simulations = 10000,
  seed = NULL,
  na.rm = TRUE
)

recovery_rate(
  x,
  observed_col = "count",
  expected_col = "total_spiked",
  group_cols = NULL,
  pre_recovered_col = NULL,
  na.rm = TRUE
)

minimum_detectable_amount(
  x,
  count_col = "count",
  group_cols = NULL,
  offset = 3,
  md_multiplier = 3.29,
  bdl_multiplier = 4.65,
  spike_replicates = 4,
  round = c("integer", "ceiling", "none"),
  na.rm = TRUE
)
```

```
batch_detection_limit(
  x,
  count_col = NULL,
  offset = 3,
  multiplier = 4.65,
  round = c("integer", "ceiling", "none"),
  ...
)
```

Arguments

x	a data frame or data table.
sample_col	column identifying samples.
area_col	column containing particle area.
size_col	optional column containing particle size. If NULL, size is inferred as sqrt(area_col).
material_col	optional material column to include in crowding groups.
group_cols	columns used for grouped summaries.
size_threshold	particles larger than this size are assessed for possible crowding.
surface_area	optional analyzed surface area used to calculate percent area covered.
simulations	number of simulated small-particle cumulative-area draws.
seed	optional random seed for reproducible crowding simulations.
na.rm	logical; remove missing values from summaries?
observed_col, expected_col	columns with observed and expected spike counts.
pre_recovered_col	optional column with particles recovered before the automated analysis.
count_col	column with blank counts.
offset, md_multiplier, bdl_multiplier	numeric constants used in MDA and BDL formulas.
spike_replicates	number of spike replicates in the MDA formula.
round	one of "integer", "ceiling", or "none" for detection-limit rounding. "integer" matches the historic workflow's as.integer().
multiplier	numeric multiplier used by batch_detection_limit().
...	reserved for future extensions.

Value

A data.table containing the requested summary.

Examples

```
blanks <- data.frame(sample_id = c("b1", "b2", "b3", "b4"),
                     area_bins = "(0,212]",
                     count = c(0, 0, 0, 11))
minimum_detectable_amount(blanks, group_cols = "area_bins")
batch_detection_limit(11)
```

---

c_spec *Manage spectral objects*

Description

`c_spec()` concatenates `OpenSpecy` objects. `sample_spec()` samples spectra from an `OpenSpecy` object. `merge_map()` merge two `OpenSpecy` objects from spectral maps.

Usage

```
c_spec(x, ...)

## Default S3 method:
c_spec(x, ...)

## S3 method for class 'OpenSpecy'
c_spec(x, ...)

## S3 method for class 'list'
c_spec(x, range = "full", res = 6, ...)

sample_spec(x, ...)

## Default S3 method:
sample_spec(x, ...)

## S3 method for class 'OpenSpecy'
sample_spec(x, size = 1, prob = NULL, ...)

merge_map(x, ...)

## Default S3 method:
merge_map(x, ...)

## S3 method for class 'OpenSpecy'
merge_map(x, ...)

## S3 method for class 'list'
merge_map(x, origins = NULL, ...)
```

Arguments

x	a list of OpenSpecy objects or of file paths.
range	a numeric providing your own wavenumber range, "full" to use the widest range represented by any supplied spectrum, or "common" to use only their overlapping range. NULL requires identical wavenumbers. The default is "full".
res	resolution of the output wavenumbers. The default of 6 is intended for reference-library identification workflows.
size	the number of spectra to sample.
prob	probabilities to use for the sampling.
origins	a list with 2 value vectors of x y coordinates for the offsets of each image.
...	further arguments passed to submethods.

Value

c_spec() and sample_spec() return OpenSpecy objects.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

conform_spec() for conforming wavenumbers

Examples

```
# Concatenating spectra
spectra <- lapply(c(read_extdata("raman_hdpe.csv"),
                    read_extdata("ftir_ldpe_soil.asp")), read_any)
full <- c_spec(spectra)
common <- c_spec(spectra, range = "common", res = 6)
range <- c_spec(spectra, range = c(1000, 2000), res = 6)

# Sampling spectra
tiny_map <- read_any(read_extdata("CA_tiny_map.zip"))
sampled <- sample_spec(tiny_map, size = 3)
```

---

estimate_temperature *Estimate material temperature and an emissivity diagnostic*

Description

estimate_temperature() applies an experimental, spectrally smooth temperature-emissivity separation (TES) model to calibrated FTIR thermal-emission radiance. The primary OpenSpecy method evaluates spectra in bounded, BLAS-backed blocks and returns one compact row per input spectrum. It does not create a full emissivity cube.

The input must contain calibrated, surface-leaving spectral radiance in "W m^-2 sr^-1 (cm^-1)^-1". This is not suitable for absorbance, transmittance, reflectance, normalized intensities, or detector counts.

Usage

```
estimate_temperature(x, ...)

## Default S3 method:
estimate_temperature(x, ...)

## S3 method for class 'OpenSpecy'
estimate_temperature(
  x,
  downwelling,
  temperature_range_k,
  fit_range_cm1,
  radiance_uncertainty = NULL,
  emissivity_stat = c("planck_weighted", "mean", "median", "max"),
  block_size = NULL,
  ...
)

## S3 method for class 'FileSpecs'
estimate_temperature(
  x,
  downwelling,
  temperature_range_k,
  fit_range_cm1,
  radiance_uncertainty = NULL,
  emissivity_stat = c("planck_weighted", "mean", "median", "max"),
  block_size = NULL,
  ...
)
```

Arguments

x	An OpenSpecy or FileSpecs object containing calibrated FTIR surface-leaving spectral radiance.
...	Additional arguments passed to methods.
downwelling	Required background/downwelling radiance. Supply a numeric scalar, a numeric vector aligned to x$wavenumber, or a one-spectrum calibrated-radiance

OpenSpecy object on the same axis. Numeric zero explicitly requests the no-background assumption.

`temperature_range_k`
: A finite, increasing two-value temperature search interval in kelvin. The endpoints are rejection boundaries, not valid estimates.

`fit_range_cm1`
: A finite two-value fitting interval in inverse centimetres. Select a range for which the opaque, isothermal, surface-leaving model is valid.

`radiance_uncertainty`
: Optional positive radiance uncertainty, supplied as a numeric scalar/vector or one-spectrum OpenSpecy. It inverse-variance weights spectral curvature and excludes trial-temperature channels whose blackbody/downwelling separation is too small for stable inversion.

`emissivity_stat`
: One scalar emissivity reduction per spectrum: `"planck_weighted"` (the default band-effective directional value), `"mean"`, `"median"`, or `"max"`.

`block_size`
: Optional positive whole number of in-memory spectra per compute block. `NULL` derives a bounded value from the number of fitting bands. It changes memory/time trade-offs, not results.

Details

The surface model is `L = epsilon * B(T) + (1 - epsilon) * L_down`, for an opaque isothermal target. Because temperature-emissivity separation is underdetermined, the selected temperature is conditional on a smooth-emissivity prior. A unique interior roughness minimum is required. Boundary, flat, multiple, unresolved, near-singular, and insufficient-band cases remain aligned but return `NA` estimates and a diagnostic status.

`planck_weighted` integrates the retrieved spectral emissivity over the fitting band using Planck radiance and trapezoidal wavenumber weights. It is a band-effective directional value; it is not necessarily the total hemispherical emissivity listed in material tables. Emissivity also depends on wavelength, temperature, viewing geometry, surface finish, oxidation, particle thickness, and sub-pixel mixing.

No values are clipped to `[0, 1]`. The physical fraction reports how much of the retrieved curve lies in that interval, making calibration/model failures visible. Use direct radiance or established signal/noise contrast as the baseline for particle detection until TES metrics are validated on held-out measurements.

Value

`estimate_temperature()` returns a source-aligned `data.table` with estimated material temperature, one selected emissivity value, roughness, physical- and valid-band fractions, fitting-band provenance, and a status. Only `status == "ok"` rows contain estimates. `calculate_emissivity()` returns an OpenSpecy object with unclipped emissivity spectra.

Experimental

`estimate_temperature()` is experimental. Its model, diagnostics, defaults, and returned columns may change as the method is validated against more calibrated thermal-emission measurements.

References

National Bureau of Standards. *Radiometric temperature measurements: II. Applications* (Technical Note 910-8). https://www.nist.gov/publications/self-study-manual-optical-radiation-measurements-part-

Borel CC (1997). Iterative retrieval of surface emissivity and temperature for a hyperspectral sensor. https://digital.library.unt.edu/ark:/67531/metadc696880/

Wilber AC, Kratz DP, Gupta SK (1999). Surface emissivity maps for use in satellite retrievals of longwave radiation. NASA/TP-1999-209362.

Wu Z, Ren H, Zhang T, Qin Q, Dong J, Ye X (2017). A modified method to prevent false minimums occurring in iterative spectrally smooth temperature emissivity separation. doi:10.1109/IGARSS.2017.8128417.

See Also

`calculate_emissivity()`, `sig_noise()`, `def_features()`

`head.OpenSpecy` *Generic Open Specy Methods*

Description

Methods to visualize and convert `OpenSpecy` objects.

Usage

```
## S3 method for class 'OpenSpecy'
head(x, ...)

## S3 method for class 'OpenSpecy'
print(x, ...)

## S3 method for class 'OpenSpecy'
plot(
  x,
  offset = 0,
  legend_var = NULL,
  pallet = rainbow,
  main = "Spectra Plot",
  xlab = "Wavenumber (1/cm)",
  ylab = "Intensity (a.u.)",
  ...
)

## S3 method for class 'OpenSpecy'
summary(object, ...)

## S3 method for class 'OpenSpecy'
```

```
as.data.frame(x, ...)

## S3 method for class 'OpenSpecy'
as.data.table(x, ...)
```

Arguments

x	an OpenSpecy object.
offset	Numeric value for vertical offset of each successive spectrum. Defaults to 1. If 0, all spectra share the same baseline.
legend_var	Character string naming a metadata column in x$metadata that labels/colors each spectrum. If NULL, spectra won't be labeled.
pallet	The base R graphics color pallet function to use. If NULL will default to all black.
main	Plot text for title.
xlab	Plot x axis text
ylab	Plot y axis text
object	an OpenSpecy object.
...	further arguments passed to the respective default method.

Details

head() shows the first few lines of an OpenSpecy object. print() prints the contents of an OpenSpecy object. plot() produces a plot() of an OpenSpecy

Value

head(), print(), and summary() return a textual representation of an OpenSpecy object. plot() and lines() return a plot. as.data.frame() and as.data.table() convert OpenSpecy objects into tabular data.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

head(), print(), summary(), matplot(), and matlines(), as.data.frame(), as.data.table()

Examples

```
data("raman_hdpe")

# Printing the OpenSpecy object
print(raman_hdpe)

# Displaying the first few lines of the OpenSpecy object
head(raman_hdpe)
```

```
# Plotting the spectra
plot(raman_hdpe)
```

---

human_ts *Create human readable timestamps*

Description

This helper function creates human readable timestamps in the form of `%Y%m%d-%H%M%OS` at the current time.

Usage

```
human_ts()
```

Details

Human readable timestamps are appended to file names and fields when metadata are shared with the Open Specy community.

Value

`human_ts()` returns a character value with the respective timestamp.

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

`format.Date` for date conversion functions

Examples

```
human_ts()
```

io_spec *Read and write spectral data*

Description

Functions for reading and writing spectral data to and from OpenSpecy format. `OpenSpecy` objects are lists with components `wavenumber`, `spectra`, and `metadata`; their supported formats are .json, .csv, and .rds. A file-backed `FileSpecs` method writes a new ENVI pair.

Usage

```
## S3 method for class 'FileSpecs'
write_spec(x, file, method = NULL, ...)

write_spec(x, ...)

## Default S3 method:
write_spec(x, ...)

## S3 method for class 'OpenSpecy'
write_spec(x, file, method = NULL, digits = getOption("digits"), ...)

read_spec(file, method = NULL, ...)

as_hyperSpec(x)
```

Arguments

`x`	an object of class `OpenSpecy` or a file-backed `FileSpecs` descriptor. File-backed objects are exported as a new ENVI header/binary pair and never overwrite source members.
`file`	file path to be read from or written to.
`method`	optional custom reader or `OpenSpecy` writer. `FileSpecs` rejects custom writers so its source-protection guarantee cannot be bypassed. Otherwise defaults to the file extension's method.
`digits`	number of significant digits to use when formatting numeric values; defaults to `getOption("digits")`.
`...`	further arguments passed to the submethods.

Details

Due to floating point number errors there may be some differences in the precision of the numbers returned if using multiple devices for .json and .csv files but the numbers should be nearly identical. `readRDS()` should return the exact same object every time. `write_spec.FileSpecs()` streams one complete rectangular region to a new ENVI BIP pair using float64 spectra and a round-trip-safe wavelength axis. It refuses custom writers, source-member targets, existing outputs, and multi-region or incomplete views.

Value

read_spec() reads data formatted as an OpenSpecy object and returns a list object of class OpenSpecy containing spectral data. write_spec() writes spectral data. For FileSpecs, it invisibly returns the new ENVI header and binary paths; other methods are called for their file-writing side effect. as_hyperspec() converts an OpenSpecy object to a hyperSpec-class object.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

OpenSpecy(); read_text(), read_asp(), read_spa(), read_spc(), and read_jdx() for text files, .asp, .spa, .spa, .spc, and .jdx formats, respectively; read_zip() and read_any() for wrapper functions; saveRDS(); readRDS(); write_json(); read_json();

Examples

```
read_extdata("raman_hdpe.json") |> read_spec()
read_extdata("raman_hdpe.rds") |> read_spec()
read_extdata("raman_hdpe.csv") |> read_spec()

## Not run:
data(raman_hdpe)
write_spec(raman_hdpe, "raman_hdpe.json")
write_spec(raman_hdpe, "raman_hdpe.rds")
write_spec(raman_hdpe, "raman_hdpe.csv")

# Convert an OpenSpecy object to a hyperSpec object
hyper <- as_hyperSpec(raman_hdpe)

## End(Not run)
```

---

lib_metadata_name_lookup

Create and apply metadata-name lookup rules

Description

lib_metadata_name_lookup() returns the default editable rules used to merge synonymous metadata columns. lib_clean_name() converts names to lowercase underscore form. lib_clean_metadata() cleans table names and coalesces columns that map to the same canonical name.

Usage

```
lib_metadata_name_lookup(
  ...,
  regex = NULL,
  defaults = TRUE,
  match_without_underscores = TRUE,
  match_singular_plural = TRUE
)

lib_clean_name(x)

lib_clean_metadata(
  x,
  name_lookup = lib_metadata_name_lookup(),
  clean_values = FALSE
)
```

Arguments

`...`	named character vectors of exact aliases, where each argument name is the canonical name, or data.frame/data.table rule tables with `canonical_name`, `source_name`, and optional `regex` columns.
`regex`	an optional named character vector or named list of regular expressions. Names identify the canonical metadata names.
`defaults`	logical; whether to include OpenSpecy's default semantic aliases before merging user rules.
`match_without_underscores`	logical; whether names that differ only by underscores should match automatically.
`match_singular_plural`	logical; whether names that differ only by one terminal `s` should match automatically.
`x`	a character vector of names for `lib_clean_name()`, or a data.frame/data.table for `lib_clean_metadata()`.
`name_lookup`	a table returned by `lib_metadata_name_lookup()` or a compatible rule table. Use `NULL` to clean names without alias merging.
`clean_values`	logical; whether `lib_clean_metadata()` should also lowercase, trim, ASCII-normalize, and normalize blank/unknown character or factor metadata values.

Details

Exact rules determine a column's target before automatic matching that can ignore underscores and a single terminal plural `s`. When values are coalesced, canonical and mechanically equivalent canonical names come before semantic aliases. Regular-expression rules are applied last to names that remain unmatched. Regex patterns are evaluated against names after `lib_clean_name()` has been applied.

Matching options selected in `lib_metadata_name_lookup()` are stored with the returned table and used by `lib_clean_metadata()`. User rules supplied through `...` are merged with the defaults. Set `defaults = FALSE` to construct a lookup from only user rules.

Value

`lib_metadata_name_lookup()` returns a data.table of rules. `lib_clean_name()` returns a character vector. `lib_clean_metadata()` returns a data.table with cleaned, coalesced columns.

Examples

```
lib_clean_name(c("User Name", "Laser (%)", "Method...3"))

name_lookup <- lib_metadata_name_lookup(
  project_code = "campaign name",
  regex = list(instrument_mode = "^method_[0-9]+$")
)
metadata <- data.frame(
  UserName = c("A", NA),
  user_name = c(NA, "B"),
  Campaign.Name = c("one", "two"),
  Method.23 = c("ftir", "raman")
)
lib_clean_metadata(metadata, name_lookup)
```

---

make_rel *Make spectral intensities relative*

Description

`make_rel()` converts intensities `x` into relative values between 0 and 1 using the standard normalization equation. If `na.rm` is `TRUE`, missing values are removed before the computation proceeds.

Usage

```
make_rel(x, ...)

## Default S3 method:
make_rel(x, na.rm = FALSE, ...)

## S3 method for class 'matrix'
make_rel(x, na.rm = FALSE, ...)

## S3 method for class 'OpenSpecy'
make_rel(x, na.rm = FALSE, ...)
```

Arguments

x	a numeric vector or an R OpenSpecy object
na.rm	logical. Should missing values be removed?
...	further arguments passed to make_rel().

Details

make_rel() is used to retain the relative height proportions between spectra while avoiding the large numbers that can result from some spectral instruments. A finite constant spectrum has no relative contrast and is therefore returned as zero rather than producing non-finite values.

Value

make_rel() returns numeric vectors, numeric matrices with each spectrum normalized by column, or an OpenSpecy object with the normalized intensity data.

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

min() and round(); adj_intens() for log transformation functions; conform_spec() for conforming wavenumbers of an OpenSpecy object to be matched with a reference library

Examples

```
make_rel(c(-1000, -1, 0, 1, 10))
```

---

manage_na *Ignore or remove NA intensities*

Description

Sometimes you want to keep or remove NA values in intensities to allow for spectra with varying shapes to be analyzed together or maintained in a single Open Specy object. For type = "ignore", spectra sharing the same missing-value boundaries are processed together, valid source-specific ranges are retained, and attributes returned by the processing function are copied back to the full object.

Usage

```
manage_na(x, ...)

## Default S3 method:
manage_na(x, lead_tail_only = TRUE, ig = c(NA), ...)

## S3 method for class 'OpenSpecy'
manage_na(x, lead_tail_only = TRUE, ig = c(NA), fun, type = "ignore", ...)
```

Arguments

x	a numeric vector or an R OpenSpecy object.
lead_tail_only	logical whether to only look at leading adn tailing values.
ig	character vector, values to ignore.
fun	the name of the function you want run, this is only used if the "ignore" type is chosen.
type	character of either "ignore" or "remove".
...	further arguments passed to fun.

Value

`manage_na()` return logical vectors of NA locations (if vector provided) or an `OpenSpecy` object with ignored or removed NA values.

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

`OpenSpecy` object to be matched with a reference library `fill_spec()` can be used to fill NA values in Open Specy objects. `restrict_range()` can be used to restrict spectral ranges in other ways than removing NAs.

Examples

```
manage_na(c(NA, -1, NA, 1, 10))
manage_na(c(NA, -1, NA, 1, 10), lead_tail_only = FALSE)
manage_na(c(NA, 0, NA, 1, 10), lead_tail_only = FALSE, ig = c(NA,0))
data(raman_hdpe)
raman_hdpe$spectra[1:10, 1] <- NA

#would normally return all NA without na.rm = TRUE but doesn't here.
manage_na(raman_hdpe, fun = make_rel)

#will remove the first 10 values we set to NA
manage_na(raman_hdpe, type = "remove")
```

`material_percentage_uncertainty` *Calculate material percentage uncertainty from observed particle counts*

Description

Calculates the absolute confidence-interval half-width for an observed material-class percentage from the total number of particles characterized. The calculation is the single-property proportion equation described by Cowger et al. (2024), rearranged to solve for error after observation.

Usage

```
material_percentage_uncertainty(count, percentage, confidence = 0.95)
```

Arguments

`count`	positive whole-number total particle count. May be a scalar or a numeric vector.
`percentage`	observed material-class percentage in the closed interval 0–100. May be a scalar or a numeric vector.
`confidence`	confidence level in the open interval 0–1. Defaults to 0.95. May be a scalar or a numeric vector.

Details

For proportion `p = percentage / 100`, total observed count `n`, and two-tailed normal critical value `z`, the returned half-width is `100 * abs(z) * sqrt(p * (1 - p) / n)`. This is a material-composition uncertainty under representative random particle sampling. It does not include laboratory, spectral-identification, concentration, finite- population, or multiple-property uncertainty. In particular, the published Wald-style equation returns zero at exactly 0 and 100 percent.

Value

A numeric vector containing absolute uncertainty half-widths in percentage points. Inputs must either have length one or share one common length.

Author(s)

Win Cowger

References

Cowger W, Markley LAT, Moore S, Gray AB, Upadhyay K, Koelmans AA (2024). "How many microplastics do you need to (sub)sample?" *Ecotoxicology and Environmental Safety*, **275**, 116243. doi:10.1016/j.ecoenv.2024.116243.

Examples

```
material_percentage_uncertainty(100, c(20, 50, 80))
material_percentage_uncertainty(100, 50, confidence = 0.90)
```

`open_specs` *Open a large spectral source as file-backed Specs*

Description

`FileSpecs` keeps a durable, read-only description of a large H5 or ENVI source. Spectral values are read only when an explicit bounded selection is materialized. The object never stores an open connection or modifies a source member.

Usage

```
open_specs(path, cache_dir = NULL)

## S3 method for class 'FileSpecs'
as_Specs(x, ...)

## S3 method for class 'FileSpecs'
print(x, ...)

## S3 method for class 'FileSpecs'
check_Specs(x, ...)

## S3 method for class 'FileSpecs'
decompress_spec(x, index = NULL, region = NULL, roi = NULL, bands = NULL, ...)

## S3 method for class 'FileSpecs'
split_spec(x, by = "region", ...)

## S3 method for class 'FileSpecs'
cor_spec(x, ...)

## S3 method for class 'FileSpecs'
match_spec(x, ...)

## S3 method for class 'FileSpecs'
def_features(x, ...)

## S3 method for class 'FileSpecs'
collapse_spec(x, ...)
```

Arguments

path	path to an H5 file, an ENVI binary file, or its .hdr file. An explicit two-file ENVI binary/header pair is also accepted when temporary upload names do not share a basename.
cache_dir	directory for immutable derived cache generations. The default uses the user cache directory, never the source directory.
x	a FileSpecs object.
...	additional arguments reserved for methods.
index	positive row positions in the current file-backed view.
region	optional region names to materialize.
roi	optional numeric c(xmin, xmax, ymin, ymax) selection or a list with two-element x and y ranges.
bands	optional positive spectral-band positions to materialize.
by	grouping field; file-backed splitting currently supports only "region".

Value

open_specs() returns a descriptor-only FileSpecs object. decompress_spec() returns a bounded OpenSpecy materialization and split_spec() returns lightweight FileSpecs views.

---

particle_image *Plot particle maps with base graphics*

Description

particle_image() plots particle classifications from an OpenSpecy, Specs, or metadata table. When a visual image is attached with add_visual_image() or supplied directly, particle coordinates are transformed onto the image and drawn as a transparent categorical raster.

Usage

```
particle_image(
  x,
  material_col = "material_class",
  image = NULL,
  bottom_left = NULL,
  top_right = NULL,
  pixel_length = 1,
  origin = c(0, 0),
  palette = NULL,
  alpha = 0.8,
  labels = FALSE,
  label_col = "feature_id",
  main = "Particle Image",
```

```
  xlab = "X",
  ylab = "Y",
  legend = FALSE,
  pch = 15,
  cex = 1,
  ...
)
```

Arguments

x	an OpenSpecy object, Specs object, or metadata table.
material_col	column containing material or class labels.
image	optional image path, array, raster, raw BMP bytes, or image object. If NULL, an attached visual image is used when available.
bottom_left, top_right	optional map extent in image pixel coordinates.
pixel_length	map pixel length in plotting units when no image overlay is used.
origin	numeric length-2 x/y origin offset when no image overlay is used.
palette	named character vector mapping materials to colors.
alpha	transparency for particle raster cells.
labels	logical; draw feature labels when feature IDs are present? The default is FALSE because particle maps quickly become cluttered.
label_col	column used for labels.
main, xlab, ylab	plot labels.
legend	logical; draw a base graphics legend?
pch, cex	retained for compatibility; particle cells are drawn as a raster.
...	additional arguments passed to plot().

Value

Invisibly returns the plotted metadata table.

Examples

```
tiny_map <- read_extdata("CA_tiny_map.zip") |> read_any()
tiny_map$metadata$material_class <- ifelse(tiny_map$metadata$x < 5,
                                           "poly(ethylene)", "mineral")
particle_image(tiny_map, legend = TRUE)
```

peak_ratio *Calculate ratios between spectral intensities at two wavenumbers*

Description

Calculates the ratio between intensities at numerator and denominator wavenumbers for every spectrum in an OpenSpecy object.

Usage

```
peak_ratio(x, ...)

## Default S3 method:
peak_ratio(x, ...)

## S3 method for class 'OpenSpecy'
peak_ratio(x, numerator, denominator, method = c("nearest", "linear"), ...)
```

Arguments

x	an OpenSpecy object.
numerator	a finite numeric scalar giving the numerator wavenumber.
denominator	a finite numeric scalar giving the denominator wavenumber.
method	character; use "nearest" to select measured points or "linear" to interpolate between adjacent measured points.
...	additional arguments passed to methods.

Details

This function evaluates intensities at two user-supplied points; it does not search for local maxima. With method = "nearest", a point exactly halfway between two measured wavenumbers uses the lower wavenumber. Linear interpolation is confined to the two adjacent measured points and never extrapolates beyond the shared wavenumber axis.

For an area-over-area ratio, calculate the numerator and denominator from explicit ranges with area_under_band() and divide the two returned vectors. Named area-under-band indices remain available through that function's index argument.

Value

A named numeric vector with one ratio per spectrum. Names and order match the columns of x$spectra. Ratios with a zero denominator or a non-finite numerator, denominator, or result are returned as NA.

See Also

point_intensity() for a non-ratio measurement at one wavenumber and area_under_band() for individual band areas and named area-over-area indices.

Examples

```
data("raman_hdpe")
peak_ratio(raman_hdpe, numerator = 2880, denominator = 2840)
peak_ratio(raman_hdpe, numerator = 2880, denominator = 2840,
           method = "linear")
```

`plot.OpenSpecyParticleAnalysis` *Plot a recorded particle-analysis diagnostic*

Description

Plot a recorded particle-analysis diagnostic

Usage

```
## S3 method for class 'OpenSpecyParticleAnalysis'
plot(x, sample = 1L, which = NULL, ...)
```

Arguments

`x`	an `OpenSpecyParticleAnalysis` result.
`sample`	sample name or numeric position.
`which`	one of `"particle_image"`, `"particle_heatmap"`, `"particle_heatmap_thresholded"`, `"cor_heatmap"`, `"sn_histogram"`, or `"cor_histogram"`. If `NULL`, the first plot with data is used.
`...`	reserved for future plotting options.

Value

`x` invisibly.

`plotly_spec` *Interactive plots for OpenSpecy objects*

Description

These functions generate heatmaps, spectral plots, and interactive plots for OpenSpecy data.

Usage

```
plotly_spec(x, ...)

## Default S3 method:
plotly_spec(x, ...)

## S3 method for class 'OpenSpecy'
plotly_spec(
  x,
  x2 = NULL,
  model = NULL,
  model_class = NULL,
  model_colorscale = list(c(0, "#d73027"), c(0.5, "#fee08b"), c(1, "#1a9850")),
  model_opacity = 0.28,
  line = list(color = "rgb(255, 255, 255)"),
  line2 = list(dash = "dot", color = "rgb(255,0,0)"),
  font = list(color = "#FFFFFF"),
  plot_bgcolor = "rgba(17, 0, 73, 0)",
  paper_bgcolor = "rgb(0, 0, 0)",
  showlegend = FALSE,
  make_rel = TRUE,
  ...
)

model_class_weights(model, model_class)

heatmap_spec(x, ...)

## Default S3 method:
heatmap_spec(x, ...)

## S3 method for class 'OpenSpecy'
heatmap_spec(
  x,
  z = NULL,
  sn = NULL,
  cor = NULL,
  min_sn = NULL,
  min_cor = NULL,
  select = NULL,
  font = list(color = "#FFFFFF"),
  plot_bgcolor = "rgba(17, 0, 73, 0)",
  paper_bgcolor = "rgb(0, 0, 0)",
  colorscale = "Viridis",
  showlegend = FALSE,
  type = "interactive",
  ...
)
```

```
interactive_plot(x, ...)

## Default S3 method:
interactive_plot(x, ...)

## S3 method for class 'OpenSpecy'
interactive_plot(
  x,
  x2 = NULL,
  select = NULL,
  line = list(color = "rgb(255, 255, 255)"),
  line2 = list(dash = "dot", color = "rgb(255,0,0)"),
  font = list(color = "#FFFFFF"),
  plot_bgcolor = "rgba(17, 0, 73, 0)",
  paper_bgcolor = "rgb(0, 0, 0)",
  colorscale = "Viridis",
  ...
)
```

Arguments

x	an OpenSpecy object containing metadata and spectral data for the first group.
x2	an optional second OpenSpecy object containing metadata and spectral data for the second group.
model	optional model library returned by train_spec_model(). Supplying this with model_class adds a logistic-regression coefficient background to plotly_spec().
model_class	exact trained class label whose signed logistic coefficients should be displayed. Positive weights are green, values near zero yellow, and negative weights red. These are model weights, not causal peak attributions.
model_colorscale	continuous Plotly colorscale for signed model weights, centered at zero.
model_opacity	opacity of the model-weight background from zero to one.
line	list; line parameter for x; passed to add_trace().
line2	list; line parameter for x2; passed to
font	list; passed to layout().
plot_bgcolor	color value; passed to layout().
paper_bgcolor	color value; passed to layout().
showlegend	whether to show the legend passed to
make_rel	logical, whether to make the spectra relative or use the raw values
z	optional numeric vector specifying the intensity values for the heatmap. If not provided, the function will use the intensity values from the OpenSpecy object.
sn	optional numeric value specifying the signal-to-noise ratio threshold. If provided along with min_sn, regions with SNR below the threshold will be excluded from the heatmap.

cor	optional numeric value specifying the correlation threshold. If provided along with min_cor, regions with correlation below the threshold will be excluded from the heatmap.
min_sn	optional numeric value specifying the minimum signal-to-noise ratio for inclusion in the heatmap. Regions with SNR below this threshold will be excluded.
min_cor	optional numeric value specifying the minimum correlation for inclusion in the heatmap. Regions with correlation below this threshold will be excluded.
select	optional index of the selected spectrum to highlight on the heatmap.
colorscale	colorscale passed to add_trace() can be an array or one of "Blackbody", "Bluered", "Blues", "Cividis", "Earth", "Electric", "Greens", "Greys", "Hot", "Jet", "Picnic", "Portland", "Rainbow", "RdBu", "Reds", "Viridis", "YlGnBu", "YlOrRd".
type	specification for plot type either interactive or static plot_ly().
...	further arguments passed to plot_ly().

Value

A plotly heatmap object displaying the OpenSpecy data. A subplot containing the heatmap and spectra plot. A plotly object displaying the spectra from the OpenSpecy object(s).

Author(s)

Win Cowger, Zacharias Steinmetz

Examples

```
if (interactive()) {
  data("raman_hdpe")
  tiny_map <- read_extdata("CA_tiny_map.zip") |> read_zip()
  plotly_spec(raman_hdpe)

  heatmap_spec(tiny_map, z = tiny_map$metadata$y, showlegend = TRUE)

  sample_spec(tiny_map, size = 12) |>
    interactive_plot(select = 2, x2 = raman_hdpe)
}
```

---

point_intensity *Measure spectral intensity at one wavenumber*

Description

Measures the intensity at one user-supplied wavenumber for every spectrum in an OpenSpecy object.

Usage

```
point_intensity(x, ...)

## Default S3 method:
point_intensity(x, ...)

## S3 method for class 'OpenSpecy'
point_intensity(x, wavenumber, method = c("nearest", "linear"), ...)
```

Arguments

x	an OpenSpecy object.
wavenumber	a finite numeric scalar giving the wavenumber to measure.
method	character; use "nearest" to select the nearest measured point or "linear" to interpolate between adjacent measured points.
...	additional arguments passed to methods.

Details

This function measures a specified spectral point; it does not search for a local maximum. With method = "nearest", a point exactly halfway between two measured wavenumbers uses the lower wavenumber. Linear interpolation is confined to the two adjacent measured points and never extrapolates beyond the shared wavenumber axis.

Value

A named numeric vector with one intensity per spectrum. Names and order match the columns of x$spectra. A non-finite selected or interpolated intensity is returned as NA for that spectrum. If the requested wavenumber is outside the shared axis, all values are returned as NA.

See Also

peak_ratio() for ratios between two point intensities and area_under_band() for measurements over a spectral region.

Examples

```
data("raman_hdpe")
point_intensity(raman_hdpe, wavenumber = 2880)
point_intensity(raman_hdpe, wavenumber = 2880, method = "linear")
```

predict_class_reference *Predict blank class-reference values with reviewed regex rules*

Description

Applies a separate regex reference only to rows whose `material` is blank. Populated exact materials are authoritative and are never overwritten, even when a regex also matches them. A blank row is filled only when all matching patterns name one material. Distinct-material matches remain blank and are reported as clashes. Exact/regex overlaps are allowed and reported for QA. If present, `match_identity` is used for pattern matching while `spectrum_identity` remains the reported exact identity.

Usage

```
predict_class_reference(
  metadata,
  regex_reference,
  return = c("table", "report")
)
```

Arguments

`metadata`	A data.frame or data.table with `spectrum_identity` and `material` columns. Row order is preserved.
`regex_reference`	A data.frame or data.table with unique `pattern` and nonblank `material` columns.
`return`	Return the updated table or an audit containing `data`, `summary`, `predictions`, `clashes`, and `overlaps`.

Value

An updated data.table, or an audit list when `return = "report"`.

process_spec *Process Spectra*

Description

`process_spec()` is a monolithic wrapper function for all spectral processing steps. Intensity operations automatically use `manage_na()` when spectra contain missing values and run the underlying functions directly otherwise. Processing attributes are updated automatically.

Usage

```
process_spec(x, ...)

## Default S3 method:
process_spec(x, ...)

## S3 method for class 'OpenSpecy'
process_spec(
  x,
  active = TRUE,
  adj_intens = FALSE,
  adj_intens_args = list(type = "none"),
  conform_spec = TRUE,
  conform_spec_args = list(range = NULL, res = 5, type = "interp"),
  restrict_range = FALSE,
  restrict_range_args = list(min = 0, max = 6000),
  flatten_range = FALSE,
  flatten_range_args = list(min = 2200, max = 2420),
  subtr_baseline = FALSE,
 subtr_baseline_args = list(type = "polynomial", degree = 8, raw = FALSE, baseline =
    NULL),
  smooth_intens = TRUE,
 smooth_intens_args = list(polynomial = 3, window = 11, derivative = 1, abs = TRUE),
  make_rel = TRUE,
  make_rel_args = list(na.rm = TRUE),
  correct_spike = FALSE,
  correct_spike_args = list(),
  ...
)
```

Arguments

x	an OpenSpecy object.
active	logical; indicating whether to perform processing. If TRUE, the processing steps will be applied. If FALSE, the original data will be returned.
adj_intens	logical; describing whether to adjust the intensity units.
adj_intens_args	named list of arguments passed to adj_intens().
conform_spec	logical; whether to conform the spectra to a new wavenumber range and resolution.
conform_spec_args	named list of arguments passed to conform_spec().
restrict_range	logical; indicating whether to restrict the wavenumber range of the spectra.
restrict_range_args	named list of arguments passed to restrict_range().
flatten_range	logical; indicating whether to flatten the range around the carbon dioxide region.

flatten_range_args
: named list of arguments passed to flatten_range().

subtr_baseline
: logical; indicating whether to subtract the baseline from the spectra.

subtr_baseline_args
: named list of arguments passed to subtr_baseline().

smooth_intens
: logical; indicating whether to apply a smoothing filter to the spectra.

smooth_intens_args
: named list of arguments passed to smooth_intens().

make_rel
: logical; if TRUE spectra are automatically normalized with make_rel().

make_rel_args
: named list of arguments passed to make_rel().

correct_spike
: logical; whether to correct isolated impulse artifacts before conforming, restricting, or otherwise processing spectra.

correct_spike_args
: named list of arguments passed to correct_spike().

...
: further arguments passed to subfunctions.

Value

process_spec() returns an OpenSpecy object with processed spectra based on the specified parameters.

Examples

```
data("raman_hdpe")
plot(raman_hdpe)

# Process spectra with range restriction and baseline subtraction
process_spec(raman_hdpe,
             restrict_range = TRUE,
             restrict_range_args = list(min = 500, max = 3000),
             subtr_baseline = TRUE,
             subtr_baseline_args = list(type = "polynomial",
                                        polynomial = 8)) |>
  plot()

# Process spectra with smoothing and derivative
process_spec(raman_hdpe,
             smooth_intens = TRUE,
             smooth_intens_args = list(
               polynomial = 3,
               window = 11,
               derivative = 1
               )
             ) |>
  plot()
```

raman_hdpe *Sample Raman spectrum*

Description

Raman spectrum of high-density polyethylene (HDPE) provided by Horiba Scientific.

Format

A three-part list of class `OpenSpecy` containing:

`wavenumber`:	spectral wavenumbers [1/cm] (vector of 964 rows)
`spectra`:	absorbance values - (a `data.table` with 964 rows and 1 column)
`metadata`:	spectral metadata

Author(s)

Zacharias Steinmetz, Win Cowger

Source

Horiba Scientific; distributed under the package's CC BY 4.0 license.

References

Cowger W, Gray A, Christiansen SH, De Frond H, Deshpande AD, Hemabessiere L, Lee E, Mill L, et al. (2020). "Critical Review of Processing and Classification Techniques for Images and Spectra in Microplastic Research." *Applied Spectroscopy*, **74**(9), 989–1010. doi:10.1177/0003702820929064.

Examples

```
data(raman_hdpe)
print(raman_hdpe)
```

read_any *Read spectral data from multiple files*

Description

Wrapper functions for reading files in batch.

Usage

```
read_any(file, c_spec = T, c_spec_args = list(range = NULL, res = NULL), ...)

read_many(file, ...)

read_zip(file, ...)
```

Arguments

file	file to be read from or written to.
c_spec	logical, if multiple spectra should be concatenated or not. Multiple spectra will return a list if this is false.
c_spec_args	list of arguments passed to c_spec()
...	further arguments passed to the submethods.

Details

read_any() provides a single function to quickly read in any of the supported formats, it assumes that the file extension will tell it how to process the spectra. OPUS extensions are a period followed only by one or more digits, including multi-digit extensions such as .10. read_zip() provides functionality for reading in spectral map files with ENVI file format or as individual files in a zip folder. If individual files, spectra are concatenated. read_many() provides functionality for reading multiple files in a character vector and will return a list.

Value

All read_*() functions return OpenSpecy objects if a single spectrum or map is provided, otherwise they provide a list of OpenSpecy objects. Map readers can return Specs when that representation is explicitly forwarded.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

read_spec() for submethods. c_spec() for combining lists of Open Specys.

Examples

```
read_extdata("raman_hdpe.csv") |> read_any()
read_extdata("ftir_ldpe_soil.asp") |> read_any()
read_extdata("testdata_zipped.zip") |> read_many()
read_extdata("CA_tiny_map.zip") |> read_many()
```

read_envi *Read ENVI data*

Description

This function allows ENVI data import.

Usage

```
read_envi(
  file,
  header = NULL,
  spectral_smooth = F,
  sigma = c(1, 1, 1),
  representation = c("OpenSpecy", "Specs"),
  background_filter = NULL,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
   organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
   instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,

    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  ...
)
```

Arguments

file	name of the binary file.
header	name of the ASCII header file. If NULL, the name of the header file is guessed by looking for a second file with the same basename as file but with .hdr extension.
spectral_smooth	logical value determines whether spectral smoothing will be performed.
sigma	if spectral_smooth then this option applies the 3d standard deviations for the gaussianSmooth function from the mmand package to describe how spectral smoothing occurs on each dimension. The first two dimensions are x and y, the third is the wavenumbers.
representation	return an ordinary dense "OpenSpecy" (the default) or an in-memory compact "Specs" map.

background_filter
: optional policy returned by specs_background_filter(). In compact mode, BIP maps are read in halo-aware row blocks and only foreground spectra are retained; source mapping 0 records every rejected or non-finite background pixel.

metadata
: a named list of the metadata; see as_OpenSpecy() for details.

...
: further arguments passed to the submethods.

Details

ENVI data usually consists of two files, an ASCII header and a binary data file. The header contains all information necessary for correctly reading the binary file via read.ENVI().

Value

An OpenSpecy object, or a compact Specs object when requested.

Author(s)

Zacharias Steinmetz, Claudia Beleites

See Also

read_spec() for reading .json, .rds, or .csv (OpenSpecy) files; read_text(), read_asp(), read_spa(), read_spc(), and read_jdx() for text files, .asp, .spa, .spa, .spc, and .jdx formats, respectively; read_opus() for reading .0 (OPUS) files; read_zip() and read_any() for wrapper functions; read.ENVI() gaussianSmooth()

---

read_opus *Read spectral data from Bruker OPUS binary files*

Description

Read file(s) acquired with a Bruker Vertex FTIR Instrument. This function is basically a fork of opus_read() from https://github.com/pierreroudier/opusreader.

Usage

```
read_opus(
  file,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
  organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
  instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,
```

```
    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  type = "spec",
  digits = 1L,
  atm_comp_minus4offset = FALSE
)
```

Arguments

file	character vector with path to file(s).
metadata	a named list of the metadata; see as_OpenSpecy() for details.
type	character vector of spectra types to extract from OPUS binary file. Default is "spec", which will extract the final spectra, e.g. expressed in absorbance (named AB in Bruker OPUS programs). Possible additional values for the character vector supplied to type are "spec_no_atm_comp" (spectrum of the sample without compensation for atmospheric gases, water vapor and/or carbon dioxide), "sc_sample" (single channel spectrum of the sample measurement), "sc_ref" (single channel spectrum of the reference measurement), "ig_sample" (interferogram of the sample measurement) and "ig_ref" (interferogram of the reference measurement).
digits	Integer that specifies the number of decimal places used to round the wavenumbers (values of x-variables).
atm_comp_minus4offset	Logical whether spectra after atmospheric compensation are read with an offset of -4 bytes from Bruker OPUS files; default is FALSE.

Details

The type of spectra returned by the function when using type = "spec" depends on the setting of the Bruker instrument: typically, it can be either absorbance or reflectance.

The type of spectra to extract from the file can also use Bruker's OPUS software naming conventions, as follows:

- ScSm corresponds to sc_sample
- ScRf corresponds to sc_ref
- IgSm corresponds to ig_sample
- IgRf corresponds to ig_ref

Value

An OpenSpecy object.

Author(s)

Philipp Baumann, Zacharias Steinmetz, Win Cowger

See Also

read_spec() for reading .json, .rds, or .csv (OpenSpecy) files; read_text(), read_asp(), read_spa(), read_spc(), and read_jdx() for text files, .asp, .spa, .spa, .spc, and .jdx formats, respectively; read_text() for reading .dat (ENVI) files; read_zip() and read_any() for wrapper functions; read_opus_raw();

Examples

```
read_extdata("ftir_ps.0") |> read_opus()
```

---

read_opus_raw *Read a Bruker OPUS spectrum binary raw string*

Description

Read single binary acquired with an Bruker Vertex FTIR Instrument

Usage

```
read_opus_raw(rw, type = "spec", atm_comp_minus4offset = FALSE)
```

Arguments

rw	a raw vector
type	character vector of spectra types to extract from OPUS binary file. Default is "spec", which will extract the final spectra, e.g. expressed in absorbance (named AB in Bruker OPUS programs). Possible additional values for the character vector supplied to type are "spec_no_atm_comp" (spectrum of the sample without compensation for atmospheric gases, water vapor and/or carbon dioxide), "sc_sample" (single channel spectrum of the sample measurement), "sc_ref" (single channel spectrum of the reference measurement), "ig_sample" (interferogram of the sample measurement) and "ig_ref" (interferogram of the reference measurement).
atm_comp_minus4offset	logical; whether spectra after atmospheric compensation are read with an offset of -4 bytes from Bruker OPUS files. Default is FALSE.

Details

The type of spectra returned by the function when using type = "spec" depends on the setting of the Bruker instrument: typically, it can be either absorbance or reflectance.

The type of spectra to extract from the file can also use Bruker's OPUS software naming conventions, as follows:

- ScSm corresponds to sc_sample

- ScRf corresponds to sc_ref
- IgSm corresponds to ig_sample
- IgRf corresponds to ig_ref

Value

A list of 10 elements:

metadata a data.frame containing metadata from the OPUS file.

spec if "spec" was requested in the type option, a matrix of the spectrum of the sample (otherwise set to NULL).

spec_no_atm_comp if "spec_no_atm_comp" was requested in the type option, a matrix of the spectrum of the sample without atmospheric compensation (otherwise set to NULL).

sc_sample if "sc_sample" was requested in the type option, a matrix of the single channel spectrum of the sample (otherwise set to NULL).

sc_ref if "sc_ref" was requested in the type option, a matrix of the single channel spectrum of the reference (otherwise set to NULL).

ig_sample if "ig_sample" was requested in the type option, a matrix of the interferogram of the sample (otherwise set to NULL).

ig_ref if "ig_ref" was requested in the type option, a matrix of the interferogram of the reference (otherwise set to NULL).

wavenumbers if "spec" or "spec_no_atm_comp" was requested in the type option, a numeric vector of the wavenumbers of the spectrum of the sample (otherwise set to NULL).

wavenumbers_sc_sample if "sc_sample" was requested in the type option, a numeric vector of the wavenumbers of the single channel spectrum of the sample (otherwise set to NULL).

wavenumbers_sc_ref if "sc_ref" was requested in the type option, a numeric vector of the wavenumbers of the single channel spectrum of the reference (otherwise set to NULL).

Author(s)

Philipp Baumann and Pierre Roudier

See Also

read_opus()

read_text *Read spectral data*

Description

Functions for reading spectral data from external file types. Currently supported reading formats are .csv and other text files, .asp, .spa, .spc, .xyz, and .jdx. Additionally, .0 (OPUS) and .dat (ENVI) files are supported via `read_opus()` and `read_envi()`, respectively. `read_zip()` takes any of the files listed above. Note that proprietary file formats like .0, .asp, and .spa are poorly supported but will likely still work in most cases.

Usage

```
read_text(
  file,
  colnames = NULL,
  method = "fread",
  comma_decimal = TRUE,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
  organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
   instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,

    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  ...
)

read_asp(
  file,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
  organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
   instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,

    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  ...
)
```

```
read_spa(
  file,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
  organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
   instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,

    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  ...
)

read_spc(
  file,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
  organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
   instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,

    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  ...
)

read_jdx(
  file,
 metadata = list(file_name = basename(file), user_name = NULL, contact_info = NULL,
  organization = NULL, citation = NULL, spectrum_type = NULL, spectrum_identity = NULL,
    material_form = NULL, material_phase = NULL, material_producer = NULL,
    material_purity = NULL, material_quality = NULL, material_color = NULL,
    material_other = NULL, cas_number = NULL, instrument_used = NULL,
   instrument_accessories = NULL, instrument_mode = NULL, spectral_resolution = NULL,
    laser_light_used = NULL, number_of_accumulations = NULL,

    total_acquisition_time_s = NULL, data_processing_procedure = NULL,
    level_of_confidence_in_identification = NULL, other_info = NULL, license =
    "CC BY-NC"),
  ...
)
```

```
read_extdata(file = NULL)

read_h5(
  file,
  collapse = FALSE,
  spectral_smooth = FALSE,
  sigma = c(1, 1, 1),
  read_visual = TRUE,
  representation = c("OpenSpecy", "Specs"),
  background_filter = NULL,
  ...
)
```

Arguments

file	file to be read from or written to.
colnames	character vector of length = 2 indicating the column names for the wavenumber and intensity; if NULL columns are guessed.
method	submethod to be used for reading text files; defaults to fread() but read.csv() works as well.
comma_decimal	logical(1) whether commas may represent decimals.
metadata	a named list of the metadata; see
collapse	whether or not to use collapse_spec() by particle_id. For read_h5(), the default is FALSE so region pixels are returned in raw form.
spectral_smooth	logical; whether H5 cubes should be smoothed before matrix conversion.
sigma	numeric vector passed to gaussianSmooth().
read_visual	logical; whether H5 mosaic images should be attached as visual-image attributes when present. H5 region stage coordinates are kept in nanometres and all mosaic tiles intersecting each region are registered for overlays and feature color extraction. as_OpenSpecy() for details.
representation	for H5 maps, return an ordinary dense "OpenSpecy" (the default) or a compact in-memory "Specs".
background_filter	optional H5 map policy returned by specs_background_filter(); rejected/non-finite source pixels receive mapping 0 and foreground spectra retain their read values.
...	further arguments passed to the submethods.

Details

read_spc() and read_jdx() are wrappers around the functions provided by the hyperSpec. Other functions have been adapted various online sources. Metadata is harvested if possible. There are many unique iterations of spectral file formats so there may be bugs in the file conversion. Please contact us if you identify any.

Value

Text and single-spectrum readers return their established objects. `read_h5()` returns an `OpenSpecy` object by default or a compact `Specs` object when requested.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

`read_spec()` for reading .json, .rds, or .csv (OpenSpecy) files; `read_opus()` for reading .0 (OPUS) files; `read_envi()` for reading .dat (ENVI) files; `read_zip()` and `read_any()` for wrapper functions; `read.jdx()`; `read.spc()`

Examples

```
read_extdata("raman_hdpe.csv") |> read_text()
read_extdata("raman_atacamit.spc") |> read_spc()
read_extdata("ftir_ldpe_soil.asp") |> read_asp()
read_extdata("testdata_zipped.zip") |> read_zip()
```

---

restrict_range *Range restriction and flattening for spectra*

Description

`restrict_range()` restricts wavenumber ranges to user specified values. Multiple ranges can be specified by inputting a series of max and min values in order. `flatten_range()` will flatten ranges of the spectra that should have no peaks. Multiple ranges can be specified by inputting the series of max and min values in order.

Usage

```
restrict_range(x, ...)

## Default S3 method:
restrict_range(x, ...)

## S3 method for class 'OpenSpecy'
restrict_range(
  x,
  min = NULL,
  max = NULL,
  make_rel = TRUE,
  automate = FALSE,
  artifact_ratio = 2,
```

```
  tail_n = 5L,
  co2_region = c(2200, 2420),
  max_crop = 0.2,
  saturation = NULL,
  saturation_min_run = NULL,
  saturation_tolerance = sqrt(.Machine$double.eps),
  saturation_guard = 1L,
  max_saturation_loss = 0.7,
  min_remaining = 3L,
  ...
)

flatten_range(x, ...)

## Default S3 method:
flatten_range(x, ...)

## S3 method for class 'OpenSpecy'
flatten_range(
  x,
  min = 2200,
  max = 2400,
  make_rel = TRUE,
  automate = FALSE,
  artifact_ratio = 2,
  tail_n = 5L,
  ...
)
```

Arguments

x	an OpenSpecy object.
min	a vector of minimum values for the range to be flattened.
max	a vector of maximum values for the range to be flattened.
make_rel	logical; should the output intensities be normalized to the range [0, 1] using make_rel() function?
automate	logical; if TRUE, first assess the relevant artifact and only restrict a high tail or flatten a high CO2 region when detected.
artifact_ratio	numeric; minimum artifact-to-control maximum ratio. The default 2 triggers at twice the control-region maximum; higher values make automatic correction less sensitive.
tail_n	integer; number of points defining each spectral tail.
co2_region	numeric length two; carbon dioxide exclusion region used by automatic tail assessment.
max_crop	numeric; maximum fraction of the full wavenumber span that automatic tail restriction may remove across both ends.

`saturation` NULL, "auto", or one finite numeric detector ceiling. Non-NULL values trigger shared saturation restriction.

`saturation_min_run` integer or NULL; minimum adjacent saturated values. The conservative automatic default accepts a two-sample hard plateau; supply an explicit value to calibrate longer plateaus. A known numeric detector ceiling defaults to one sample.

`saturation_tolerance` numeric; relative tolerance used to recognize an effectively constant automatic detector plateau.

`saturation_guard` integer; sampled points added to both sides of each detected saturated interval before the shared union is removed.

`max_saturation_loss` numeric; largest fraction of wavenumber coverage that saturation restriction may remove. The default is 0.70.

`min_remaining` integer; minimum retained wavenumber count after saturation restriction.

`...` additional arguments passed to subfunctions; currently not in use.

Value

An OpenSpecy object with the spectral intensities within specified ranges restricted or flattened.

Author(s)

Win Cowger, Zacharias Steinmetz

See Also

conform_spec() for conforming wavenumbers to be matched with a reference library; adj_intens() for log transformation functions; min() and round()

Examples

```
test_noise <- as_OpenSpecy(x = seq(400,4000, by = 10),
                           spectra = data.frame(intensity = rnorm(361)))
plot(test_noise)

restrict_range(test_noise, min = 1000, max = 2000)
restrict_range(test_noise, automate = TRUE, make_rel = FALSE)

flattened_intensities <- flatten_range(test_noise, min = c(1000, 2000),
                                       max = c(1500, 2500))
plot(flattened_intensities)
```

run_app *Run Open Specy app*

Description

This wrapper function starts the graphical user interface of Open Specy.

Usage

```
run_app(
  path = "system",
  log = TRUE,
  ref = NULL,
  check_local = TRUE,
  test_mode = FALSE,
  launch.browser = getOption("shiny.launch.browser", interactive()),
  ...
)
```

Arguments

path	Shiny app directory, or "system" to launch the bundled app.
log	logical; enables/disables Shiny logging to tempdir().
ref	retained for compatibility with older releases; ignored because the app is bundled with the package.
check_local	retained for compatibility with older releases; ignored because path = "system" always uses the bundled app.
test_mode	logical; for internal testing only.
launch.browser	option for shiny::runApp().
...	arguments passed to shiny::runApp().

Details

By default, run_app() launches the Shiny app bundled with the installed OpenSpecy package at system.file("shiny", package = "OpenSpecy"). Historical GitHub download support has been removed so package installs use the same app files offline. Set path to an explicit app directory when testing a local Shiny app during development.

Value

This function normally does not return any value, see shiny::runApp(). In test_mode, it invisibly returns the resolved app path.

Author(s)

Win Cowger, Zacharias Steinmetz, Garth Covernton

See Also

```
shiny::runApp()
```

Examples

```
## Not run:
run_app()

## End(Not run)
```

`sig_noise` *Calculate signal and noise metrics for OpenSpecy objects*

Description

This function calculates common signal and noise metrics for `OpenSpecy` objects.

Usage

```
sig_noise(x, ...)

## Default S3 method:
sig_noise(x, ...)

## S3 method for class 'OpenSpecy'
sig_noise(
  x,
  metric = "run_sig_over_noise",
  na.rm = TRUE,
  prob = 0.5,
  step = 20,
  breaks = seq(min(unlist(x$spectra)), max(unlist(x$spectra)), length =
   ((nrow(x$spectra)^(1/3)) * (max(unlist(x$spectra)) - min(unlist(x$spectra))))/(2 *
     IQR(unlist(x$spectra)))),
  sig_min = NULL,
  sig_max = NULL,
  noise_min = NULL,
  noise_max = NULL,
  abs = T,
  spatial_smooth = F,
  sigma = c(1, 1),
  threshold = NULL,
  ...
)
```

Arguments

x	an OpenSpecy object.
metric	character; specifying the desired metric to calculate. Options include "sig" (mean intensity), "noise" (standard deviation of intensity), "sig_times_noise" (absolute value of signal times noise), "sig_over_noise" (absolute value of signal / noise), "run_sig_over_noise" (absolute value of signal / noise where signal is estimated as the max intensity and noise is estimated as the height of a low intensity region.), "breakpoint_snr" (ratio between the upper and lower groups around the exact least-squares breakpoint in sorted absolute amplitudes), "log_tot_sig" (sum of the inverse log intensities, useful for spectra in log units), "tot_sig" (sum of intensities), or "entropy" (Shannon entropy of intensities)..
na.rm	logical; indicating whether missing values should be removed when calculating signal and noise. Default is TRUE.
prob	numeric single value; the probability to retrieve for the quantile where the noise will be interpreted with the run_sig_over_noise option.
step	numeric; the step size of the region to look for the run_sig_over_noise option.
breaks	numeric; the number or positions of the breaks for entropy calculation. Defaults to infer a decent value from the data.
sig_min	numeric; the minimum wavenumber value for the signal region.
sig_max	numeric; the maximum wavenumber value for the signal region.
noise_min	numeric; the minimum wavenumber value for the noise region.
noise_max	numeric; the maximum wavenumber value for the noise region.
abs	logical; whether to return the absolute value of the result
spatial_smooth	logical; whether to spatially smooth the sig/noise using the xy coordinates and a gaussian smoother.
sigma	numeric; two value vector describing standard deviation for smoother in each dimension, y is specified first followed by x, should be the same for each in most cases.
threshold	numeric; if NULL, no threshold is set, otherwise use a numeric value to set the target threshold which true signal or noise should be above. The function will return a logical value instead of numeric if a threshold is set.
...	further arguments passed to subfunctions; currently not used.

Value

A numeric vector containing the calculated metric for each spectrum in the OpenSpecy object or logical value if threshold is set describing if the numbers where above or equal to (TRUE) the threshold.

See Also

restrict_range()

Examples

```
data("raman_hdpe")

sig_noise(raman_hdpe, metric = "sig")
sig_noise(raman_hdpe, metric = "noise")
sig_noise(raman_hdpe, metric = "sig_times_noise")
```

`smooth_intens` *Smooth spectral intensities*

Description

This smoother can enhance the signal to noise ratio of the data using a Savitzky-Golay or Whittaker-Henderson filter.

Usage

```
smooth_intens(x, ...)

## Default S3 method:
smooth_intens(x, ...)

## S3 method for class 'OpenSpecy'
smooth_intens(
  x,
  polynomial = 3,
  window = 11,
  derivative = 1,
  abs = TRUE,
  lambda = 1600,
  d = 2,
  type = "sg",
  lag = 2,
  make_rel = TRUE,
  ...
)

calc_window_points(x, ...)

## Default S3 method:
calc_window_points(x, wavenum_width = 70, ...)

## S3 method for class 'OpenSpecy'
calc_window_points(x, wavenum_width = 70, ...)
```

Arguments

x	an object of class OpenSpecy or vector for calc_window_points().
polynomial	polynomial order for the filter
window	number of data points in the window, filter length (must be odd).
derivative	the derivative order if you want to calculate the derivative. Zero (default) is no derivative.
abs	logical; whether you want to calculate the absolute value of the resulting output.
lambda	smoothing parameter for Whittaker-Henderson smoothing, 50 results in rough smoothing and 10^4 results in a high level of smoothing.
d	order of differences to use for Whittaker-Henderson smoothing, typically set to 2.
type	the type of smoothing to use "wh" for Whittaker-Henerson or "sg" for Savitzky-Golay.
lag	the lag to use for the numeric derivative calculation if using Whittaker-Henderson. Greater values lead to smoother derivatives, 1 or 2 is common.
make_rel	logical; if TRUE spectra are automatically normalized with make_rel().
wavenum_width	the width of the window you want in wavenumbers.
...	further arguments passed to the Savitzky-Golay coefficient generator, currently including ts.

Details

For Savitzky-Golay this uses OpenSpecy's internal matrix filter to improve integration with other OpenSpecy functions. A typical good smooth can be achieved with 11 data point window and a 3rd or 4th order polynomial. For Whittaker-Henderson, the code is largely based off of the whittaker() function in the pracma package. In general Whittaker-Henderson is expected to be slower but more robust than Savitzky-Golay.

Value

smooth_intens() returns an OpenSpecy object.

calc_window_points() returns a single numberic vector object of the number of points needed to fill the window and can be passed to smooth_intens(). For many applications, this is more reusable than specifying a static number of points.

Author(s)

Win Cowger, Zacharias Steinmetz

References

Savitzky A, Golay MJ (1964). "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." *Analytical Chemistry*, **36**(8), 1627–1639.

See Also

smooth_intens()

Examples

```
data("raman_hdpe")

smooth_intens(raman_hdpe)

smooth_intens(raman_hdpe, window = calc_window_points(x = raman_hdpe, wavenum_width = 70))

smooth_intens(raman_hdpe, lambda = 1600, d = 2, lag = 2, type = "wh")
```

---

spatial_smooth *Spatial Smoothing of OpenSpecy Objects*

Description

Applies spatial smoothing to an OpenSpecy object using a Gaussian filter.

Usage

```
spatial_smooth(x, sigma = c(1, 1, 1), ...)
```

Arguments

x	an OpenSpecy object.
sigma	a numeric vector specifying the standard deviations for the Gaussian kernel in the x and y dimensions, respectively.
...	further arguments passed to or from other methods.

Details

This function performs spatial smoothing on the spectral data in an OpenSpecy object. It assumes that the spatial coordinates are provided in the metadata element of the object, specifically in the x and y columns, and that there is a col_id column in metadata that matches the column names in the spectra matrix.

Value

An OpenSpecy object with smoothed spectra.

Author(s)

Win Cowger

See Also

as_OpenSpecy(), gaussianSmooth()

Specs *Create compressed Specs objects*

Description

Specs objects store compressed spectral data for large hyperspectral datasets. They use a structure similar to OpenSpecy, but store physical or latent variables, active values, coordinate data, and metadata separately. Version 0.2 objects may represent regular grids and repeated metadata compactly and reserve source mapping 0 for explicitly background-suppressed pixels.

Usage

```
Specs(variables, values, coords = NULL, metadata = NULL, attributes = list())

is_Specs(x)

check_Specs(x, ...)

## Default S3 method:
check_Specs(x, ...)

## S3 method for class 'Specs'
check_Specs(x, ...)

as_Specs(x, ...)

## Default S3 method:
as_Specs(x, ...)

## S3 method for class 'Specs'
as_Specs(
  x,
  model = NULL,
  steps = NULL,
  background_filter = NULL,
  n_components = NULL,
  centers = NULL,
  bits_per_variable = NULL,
  limits = NULL,
  ...
)

## S3 method for class 'OpenSpecy'
```

```
as_Specs(
  x,
  model = NULL,
  steps = c("pca", "hilbert"),
  background_filter = NULL,
  n_components = NULL,
  centers = NULL,
  bits_per_variable = NULL,
  limits = NULL,
  ...
)

fit_specs_pca(x, n_components, center = TRUE, scale. = FALSE, ...)

decompress_spec(x, ...)

## Default S3 method:
decompress_spec(x, ...)

## S3 method for class 'Specs'
decompress_spec(x, expand = TRUE, index = NULL, ...)

## S3 method for class 'Specs'
as_OpenSpecy(x, ...)

encode_specs_hilbert(x, bits_per_variable = NULL, limits = NULL, ...)

decode_specs_hilbert(x, ...)

write_specs(x, file, compress = "xz", ...)

## Default S3 method:
write_specs(x, file, compress = "xz", ...)

## S3 method for class 'Specs'
write_specs(x, file, compress = "xz", ...)

read_specs(file, ...)

specs_background_filter(
  metric = "run_sig_over_noise",
  minimum,
  maximum = Inf,
  sigma = NULL,
  step = 10,
  intensity_type = NULL
)
```

```
specs_source_count(x)

specs_background_mask(x, index = NULL)

specs_source_values(x, index = NULL)

specs_coordinates(x, index = NULL, columns = NULL)

specs_metadata(x, index = NULL, columns = NULL)

## S3 method for class 'FileSpecs'
write_specs(x, file, compress = "xz", ...)

## S3 method for class 'Specs'
cor_spec(x, library, na.rm = TRUE, compute = "optimized", ...)

## S3 method for class 'Specs'
match_spec(
  x,
  library,
  top_n = NULL,
  expand = FALSE,
  top_n_by = NULL,
  add_library_metadata = NULL,
  add_object_metadata = NULL,
  compute = "optimized",
  na.rm = TRUE,
  ...
)

## S3 method for class 'Specs'
def_features(
  x,
  features,
  shape_kernel = c(3, 3),
  shape_type = "box",
  close = FALSE,
  close_kernel = c(4, 4),
  close_type = "box",
  img = NULL,
  bottom_left = NULL,
  top_right = NULL,
  ...
)

## S3 method for class 'Specs'
collapse_spec(x, fun = mean, column = "feature_id", ...)
```

Arguments

variables	vector of latent variable names.
values	numeric matrix with one row per variable and one column per active spectrum or cluster.
coords	coordinate data.frame or data.table; should include x, y, source_id, and value_id; or an internal validated compact regular-grid descriptor returned by map readers.
metadata	metadata data.frame or data.table with one row per column in values.
attributes	list of Specs attributes to attach.
x	an object to test, convert, decompress, or write.
model	optional SpecsPCA model returned by fit_specs_pca(); if omitted and "pca" is in steps, a model is fit from x.
steps	character vector of compression steps. Supported values are "background", "pca", "kmeans", and "hilbert". Background suppression must precede every compression step. K-means can be placed before, between, or after the other compression steps; PCA cannot be placed after Hilbert encoding.
background_filter	optional policy returned by specs_background_filter(). Suppression is explicit and lossy and records the source mask, signal/noise values, reasons, and policy.
n_components	number of PCA components to keep.
centers	initial centers or the number of centers for weighted Lloyd K-means. Source mapping multiplicities supply the weights and mapping 0 is excluded.
bits_per_variable	positive whole number of bits used for each Hilbert-encoded variable. If NULL, the value is inferred from the number of variables to pack the available 64-bit code space.
limits	optional two-column matrix, data frame, or Hilbert model with per-variable minimum and maximum values used for quantization.
center, scale.	arguments passed to prcomp().
expand	logical; if TRUE, decompress or match one row per original coordinate; if FALSE, keep active spectra or clusters.
index	optional positive integer vector selecting spectra to decompress. With expand = TRUE, indexes refer to rows in x$coords; with expand = FALSE, indexes refer to columns in x$values.
file	file path for reading or writing a Specs object.
compress	compression argument passed to saveRDS().
metric	signal/noise metric passed to sig_noise().
minimum, maximum	strict accepted signal/noise bounds.
sigma	optional three-dimensional Gaussian smoothing sigma. NULL classifies the unsmoothed spectra.

step	run-length step passed to sig_noise().
intensity_type	optional intensity units passed to adj_intens() before signal/noise is measured. NULL or "none" preserves uploaded values; "transmittance" and "reflectance" classify absorbance-adjusted values.
columns	optional coordinate or metadata columns to return.
library	a Specs object to match against.
na.rm	logical; should missing values be removed for latent matching?
compute	correlation compute strategy, "optimized" or "base".
top_n	integer; number of top latent matches to return.
top_n_by	optional single library metadata column name; when supplied, retain top_n matches independently within each nonblank group.
add_library_metadata	name of a library metadata column to join.
add_object_metadata	name of an object metadata column to join.
features	logical or character vector with one value per row in x$coords.
shape_kernel, shape_type, close, close_kernel, close_type, img, bottom_left, top_right	arguments passed to the feature-definition routine.
fun	function used to collapse latent values.
column	coordinate column used to group spectra for collapse.
...	additional arguments passed to submethods.

Value

Specs(), as_Specs(), encode_specs_hilbert(), and decode_specs_hilbert() return a Specs object. fit_specs_pca() returns a SpecsPCA model. decompress_spec() returns an exact OpenSpecy object for uncompressed values, an approximate reconstruction after PCA/Hilbert, and an exact zero line for every background-suppressed source. read_specs() returns a Specs object.

Experimental

encode_specs_hilbert() and decode_specs_hilbert() are experimental. Their encoded representation, defaults, and reconstruction guarantees may change as the compression workflow is validated on more hyperspectral data.

Author(s)

Win Cowger

Examples

```
data("raman_hdpe")
specs <- as_Specs(raman_hdpe, n_components = 1)
decompress_spec(specs)
```

spec_res *Spectral resolution*

Description

Helper function for calculating the spectral resolution from `wavenumber` data.

Usage

```
spec_res(x, ...)

## Default S3 method:
spec_res(x, ...)

## S3 method for class 'OpenSpecy'
spec_res(x, ...)
```

Arguments

`x`	a numeric vector with `wavenumber` data or an `OpenSpecy` object.
`...`	further arguments passed to subfunctions; currently not used.

Details

The spectral resolution is the the minimum wavenumber, wavelength, or frequency difference between two lines in a spectrum that can still be distinguished.

Value

`spec_res()` returns a single numeric value.

Author(s)

Win Cowger, Zacharias Steinmetz

Examples

```
data("raman_hdpe")

spec_res(raman_hdpe)
```

split_h5 *Split a large HDF5 spectral map by metadata category*

Description

split_h5() groups an H5 or HDF5 spectral map by a metadata field. The default region-to-H5 path discovers region groups directly and copies them without hashing the complete source, building a per-pixel index, or loading spectral values into R. Other metadata fields use a file-backed FileSpecs descriptor. format = "rds" reads and saves one category at a time; each RDS category must fit in memory.

Usage

```
split_h5(
  file,
  field = "region",
  output_dir = dirname(file),
  format = c("h5", "rds")
)
```

Arguments

file	path to one .h5 or .hdf5 file.
field	metadata field used to group spectra. Defaults to "region".
output_dir	directory in which to save the output files. Defaults to the source file's directory.
format	output format. "h5" (the default) uses native HDF5 object copies for whole-region categories. "rds" materializes one category as an OpenSpecy object before saving it.

Details

Output files are named <input stem>_<field category>.<format>. Characters that are not valid in Windows file names are replaced with underscores. Existing output files are never overwritten. Native H5 output requires each category to contain complete source regions because the source schema stores spectra as three-dimensional regional grids. Use format = "rds" for a field that divides pixels within a region. File-level metadata and selected region groups are retained in H5 outputs. When registered mosaic metadata is available, each H5 output also retains the intersecting image tiles and their corresponding centers so visual-image registration remains self-contained.

Value

Invisibly, a named character vector of output paths. Names are the original field-category labels. H5 outputs remain readable by read_h5() and open_specs(); each RDS file contains one OpenSpecy object.

See Also

open_specs(), decompress_spec()

Examples

```
## Not run:
split_h5("large_map.h5")
split_h5("large_map.h5", field = "particle_id", output_dir = "regions",
         format = "rds")

## End(Not run)
```

---

split_spec *Split Open Specy objects*

Description

Convert a list of Open Specy objects into one-spectrum objects, or split a file-backed map into lightweight region views.

Usage

```
split_spec(x, ...)

## Default S3 method:
split_spec(x, ...)
```

Arguments

x	a list of OpenSpecy objects or another supported spectral object.
...	arguments passed to class methods.

Details

Function will accept a list of Open Specy objects of any length and will split them to their individual components. For example a list of two objects, an Open Specy with only one spectrum and an Open Specy with 50 spectra will return a list of length 51 each with Open Specy objects that only have one spectrum.

Value

For ordinary inputs, a list of Open Specy objects each with one spectrum. For FileSpecs, a named list of descriptor-only FileSpecs region views that share the immutable source and cache.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

c_spec() for combining OpenSpecy objects. collapse_spec() for summarizing OpenSpecy objects.

Examples

```
data("test_lib")
data("raman_hdpe")
listed <- list(test_lib, raman_hdpe)
test <- split_spec(listed)
test2 <- split_spec(list(test_lib))
```

---

subtr_baseline *Automated background subtraction for spectral data*

Description

This baseline correction routine iteratively finds the baseline of a spectrum using polynomial or Fill Peaks methods, or accepts a manual baseline.

Usage

```
subtr_baseline(x, ...)

## Default S3 method:
subtr_baseline(
  x,
  y,
  type = "polynomial",
  degree = 8,
  raw = FALSE,
  full = T,
  remove_peaks = T,
  refit_at_end = F,
  crop_boundaries = F,
  iterations = 10,
  peak_width_mult = 3,
  termination_diff = 0.05,
  degree_part = 2,
  bl_x = NULL,
  bl_y = NULL,
  make_rel = TRUE,
  lambda = 4,
  hwi = 50,
  it = 10,
  int = NULL,
```

```
  ...
)

## S3 method for class 'OpenSpecy'
subtr_baseline(
  x,
  type = "polynomial",
  degree = 8,
  raw = FALSE,
  full = T,
  remove_peaks = T,
  refit_at_end = F,
  crop_boundaries = F,
  iterations = 10,
  peak_width_mult = 3,
  termination_diff = 0.05,
  degree_part = 2,
  baseline = list(wavenumber = NULL, spectra = NULL),
  make_rel = TRUE,
  lambda = 4,
  hwi = 50,
  it = 10,
  int = NULL,
  ...
)
```

Arguments

x	a list object of class OpenSpecy or a vector of wavenumbers.
y	a vector of spectral intensities.
type	one of "polynomial", "fill_peaks", or "manual", depending on the desired baseline correction method.
degree	the degree of the full spectrum polynomial. Must be less than the number of unique points when raw is FALSE. Typically, a good fit can be found with an 8th order polynomial.
raw	if TRUE, use raw and not orthogonal polynomials.
full	logical, whether to use the full spectrum as in "imodpoly" or to partition as in "smodpoly".
remove_peaks	logical, whether to remove peak regions during first iteration.
refit_at_end	logical, whether to refit a polynomial to the end result (TRUE) or to use linear approximation.
crop_boundaries	logical, whether to smartly crop the boundaries to match the spectra based on peak proximity.
iterations	the number of iterations for polynomial baseline correction. For type = "fill_peaks", this value is used only when it is omitted.

`peak_width_mult`
: scaling factor for the width of peak detection regions.

`termination_diff`
: scaling factor for the ratio of difference in residual standard deviation to terminate iterative fitting with.

`degree_part`
: the degree of the polynomial for `"smodpoly"`. Must be less than the number of unique points.

`bl_x`
: a vector of wavenumbers for the baseline.

`bl_y`
: a vector of spectral intensities for the baseline.

`make_rel`
: logical; if `TRUE`, spectra are automatically normalized with `make_rel()`.

`lambda`
: non-negative numeric scalar controlling the second-derivative penalty for initial Fill Peaks smoothing.

`hwi`
: positive integer half-width, in subsampled buckets, of the initial Fill Peaks suppression window.

`it`
: positive integer number of Fill Peaks suppression iterations. When omitted, `iterations` is used.

`int`
: optional integer number of Fill Peaks subsampling buckets. If `NULL`, it is derived as approximately one tenth of the spectrum length, bounded between three and the number of spectral points.

`baseline`
: an `OpenSpecy` object containing the baseline data to be subtracted (only for `"manual"`).

`...`
: further arguments passed to methods.

Details

This function supports `"polynomial"` automated baseline correction, `"fill_peaks"` iterative local-window baseline suppression, and `"manual"` subtraction of a user-provided baseline. Polynomial default settings are closest to `"imodpoly"` for iterative polynomial fitting based on Zhao et al. (2007). Additionally options recommended by `"smodpoly"` for segmented iterative polynomial fitting with enhanced peak detection from the S-Modpoly algorithm (https://github.com/jackma123-rgb/S-Modpoly). Fill Peaks is implemented in base R using a banded solver for the initial second-derivative smoothing step.

Value

The `OpenSpecy` method returns an `OpenSpecy` object with corrected spectra, the original shared wavenumber axis, aligned metadata, and existing attributes. The default vector method returns a numeric vector of corrected intensities.

Author(s)

Win Cowger, Zacharias Steinmetz

References

Chen MS (2020). Michaelstchen/ModPolyFit. *MATLAB*. Retrieved from https://github.com/michaelstchen/modPolyFit (Original work published July 28, 2015)

Zhao J, Lui H, McLean DI, Zeng H (2007). "Automated Autofluorescence Background Subtraction Algorithm for Biomedical Raman Spectroscopy." *Applied Spectroscopy*, **61**(11), 1225–1232. doi:10.1366/000370207782597003.

Jackma123 (2023). S-Modpoly: Segmented modified polynomial fitting for spectral baseline correction. *GitHub Repository*. Retrieved from https://github.com/jackma123-rgb/S-Modpoly.

Liland KH (2015). 4S Peak Filling – baseline estimation by iterative mean suppression. *MethodsX*, **2**, 135–140. doi:10.1016/j.mex.2015.02.009.

See Also

poly(); smooth_intens()

Examples

```
data("raman_hdpe")

# Use polynomial
subtr_baseline(raman_hdpe, type = "polynomial", degree = 8)

subtr_baseline(raman_hdpe, type = "polynomial", iterations = 5)

# Use Fill Peaks with explicit tuning
subtr_baseline(raman_hdpe, type = "fill_peaks", lambda = 4,
               hwi = 50, it = 10, make_rel = FALSE)

# Use manual
bl <- raman_hdpe
bl$spectra[, "intensity"] <- bl$spectra[, "intensity"] / 2
subtr_baseline(raman_hdpe, type = "manual", baseline = bl)
```

---

test_lib *Test reference library*

Description

Reference library with 29 FTIR and 28 Raman spectra used for examples and internal testing.

Format

An OpenSpecy object; sample_name is the class of the spectra.

Author(s)

Win Cowger

Examples

```
data("test_lib")
```

Index

∗ **data**
raman_hdpe, 75
test_lib, 105
-, *75*

add_trace, *69, 70*
add_visual_image, 4, *16*, *64*
adj_intens, 5, *7, 8*, *25*, *46*, *60*, *73*, *87*
adj_intens(), *98*
adj_neg, *6*
adj_neg *(*adj_res*)*, 7
adj_res, 7
adj_wave, 8
ai_classify *(*cor_spec*)*, 42
approx, *39*
area_under_band, 9, *66*, *71*
as.data.frame, *54*
as.data.frame.OpenSpecy *(*head.OpenSpecy*)*, 53
as.data.table, *54*
as.data.table.OpenSpecy *(*head.OpenSpecy*)*, 53
as_hyperSpec *(*io_spec*)*, 56
as_OpenSpecy, 13, *78, 79*, *84*, *94*
as_OpenSpecy.Specs *(*Specs*)*, 94
as_Specs *(*Specs*)*, 94
as_Specs.FileSpecs *(*open_specs*)*, 63
assess_lib *(*build_lib*)*, 21
assess_spec, 11, *25*, *27*
automate_particle_analysis, 17

basename, *14*
batch_detection_limit *(*crowd_lookup*)*, 47
build_lib, 21
build_model_lib *(*build_lib*)*, 21

c_spec, 49, *76*, *102*
calc_window_points *(*smooth_intens*)*, 91
calculate_emissivity, 32
calculate_emissivity(), *53*
check_lib, 33
check_OpenSpecy *(*as_OpenSpecy*)*, 13
check_Specs *(*Specs*)*, 94
check_Specs.FileSpecs *(*open_specs*)*, 63
collapse_spec, *20*, 36, *84*, *102*
collapse_spec.FileSpecs *(*open_specs*)*, 63
collapse_spec.Specs *(*Specs*)*, 94
conform_res, *6*
conform_res *(*adj_res*)*, 7
conform_spec, *8*, 38, *50*, *60*, *73*, *87*
cor, *46*
cor_spec, 42
cor_spec.FileSpecs *(*open_specs*)*, 63
cor_spec.Specs *(*Specs*)*, 94
correct_spike, 40, *74*
crowd_lookup, 47

data.table, *75*
decode_specs_hilbert *(*Specs*)*, 94
decompress_spec *(*Specs*)*, 94
decompress_spec(), *101*
decompress_spec.FileSpecs *(*open_specs*)*, 63
dedupe_spec *(*build_lib*)*, 21
def_features, *20*
def_features *(*collapse_spec*)*, 36
def_features(), *53*
def_features.FileSpecs *(*open_specs*)*, 63
def_features.Specs *(*Specs*)*, 94
detect_image_origin, *19*
detect_image_origin *(*add_visual_image*)*, 4
digest, *26*
download.file, *34*

encode_specs_hilbert *(*Specs*)*, 94
estimate_temperature, 50
estimate_temperature(), *33*

FileSpecs, *19*, *100*

fill_spec (cor_spec), 42
filter_spec (cor_spec), 42
fit_specs_pca (Specs), 94
flatten_range, *39*, *74*
flatten_range (restrict_range), 85
format.Date, *55*
fread, *84*

gaussianSmooth, *78*, *84*, *94*
gen_grid (as_OpenSpecy), 13
get_lib, *46*
get_lib (check_lib), 33
get_metadata (cor_spec), 42
getOption, *56*
glmnet, *27*

head, *54*
head.OpenSpecy, 53
heatmap_spec (plotly_spec), 67
human_ts, 55
hyperSpec, *84*

ident_spec (cor_spec), 42
interactive_plot (plotly_spec), 67
io_spec, 56
is_empty_vector (adj_res), 7
is_OpenSpecy (as_OpenSpecy), 13
is_Specs (Specs), 94

join_lib_metadata (build_lib), 21
join_material_hierarchy (build_lib), 21

layout, *69*
lib_clean_metadata, *27*
lib_clean_metadata
(lib_metadata_name_lookup), 57
lib_clean_name
(lib_metadata_name_lookup), 57
lib_metadata_name_lookup, *25*, 57
load_lib, *46*
load_lib (check_lib), 33

make_lib_lookup_template (build_lib), 21
make_rel, *6*, *27*, 59, *74*, *92*, *104*
manage_na, *27*, 60, *72*
match_spec, *19*, *29*, *30*
match_spec (cor_spec), 42
match_spec.FileSpecs (open_specs), 63
match_spec.Specs (Specs), 94
material_percentage_uncertainty, 62

matlines, *54*
matplot, *54*
max_cor_named (cor_spec), 42
mean_replace (adj_res), 7
merge_map (c_spec), 49
min, *8*, *60*, *87*
minimum_detectable_amount
(crowd_lookup), 47
model_class_weights (plotly_spec), 67

open_specs, 63
open_specs(), *100*, *101*
OpenSpecy, *56*, *57*, *75*
OpenSpecy (as_OpenSpecy), 13
os_similarity (cor_spec), 42

particle_image, 64
peak_ratio, 66, *71*
plot, *54*, *65*
plot.OpenSpecy (head.OpenSpecy), 53
plot.OpenSpecyParticleAnalysis, 67
plot_ly, *70*
plotly_spec, *30*, 67
point_intensity, *66*, 70
poly, *105*
prcomp, *97*
predict_class_reference, *27*, 72
print, *54*
print.FileSpecs (open_specs), 63
print.OpenSpecy (head.OpenSpecy), 53
process_spec, *20*, *24*, *27*, 72
prune_lib, *25*
prune_lib (build_lib), 21

raman_hdpe, 75
read.csv, *84*
read.ENVI, *78*
read.jdx, *85*
read.spc, *85*
read_any, *24*, *27*, *57*, 75, *78*, *80*, *85*
read_asp, *57*, *78*, *80*
read_asp (read_text), 82
read_envi, 77, *82*, *85*
read_extdata (read_text), 82
read_h5 (read_text), 82
read_h5(), *100*
read_jdx, *57*, *78*, *80*
read_jdx (read_text), 82
read_json, *57*

read_many (read_any), 75
read_opus, *78*, 78, *81*, *82*, *85*
read_opus_raw, *80*, 80
read_spa, *57*, *78*, *80*
read_spa (read_text), 82
read_spc, *57*, *78*, *80*
read_spc (read_text), 82
read_spec, *16*, *76*, *78*, *80*, *85*
read_spec (io_spec), 56
read_specs (Specs), 94
read_text, *57*, *78*, *80*, 82
read_zip, *57*, *78*, *80*, *82*, *85*
read_zip (read_any), 75
readRDS, *56*, *57*
rebuild_lib_artifacts (build_lib), 21
recovery_rate (crowd_lookup), 47
reduce_lib (build_lib), 21
restrict_range, *25*, *39*, *73*, 85
restrict_range(), *90*
rm_lib (check_lib), 33
round, *7*, *8*, *60*, *87*
run_app, 88

sample_spec (c_spec), 49
saveRDS, *57*, *97*
sig_noise, *12*, *20*, *25*, 89
sig_noise(), *53*, *97*, *98*
smooth_intens, *74*, 91, *93*, *105*
spatial_smooth, 93
spec_res, 99
Specs, 94
specs_background_filter, *78*, *84*
specs_background_filter (Specs), 94
specs_background_mask (Specs), 94
specs_coordinates (Specs), 94
specs_metadata (Specs), 94
specs_source_count (Specs), 94
specs_source_values (Specs), 94
split_h5, 100
split_spec, 101
split_spec.FileSpecs (open_specs), 63
subtr_baseline, *6*, *39*, *74*, 102
summary, *54*
summary.OpenSpecy (head.OpenSpecy), 53

test_lib, 105
train_spec_model, *69*
train_spec_model (build_lib), 21

visual_image (add_visual_image), 4

write_json, *57*
write_spec (io_spec), 56
write_specs (Specs), 94