Package 'OpenSpecy'

March 15, 2024

Type Package

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

Version 1.0.8

Date 2024-03-14

Description Raman and (FT)IR spectral analysis tool for plastic particles and other environmental samples (Cowger et al. 2021, <doi:10.1021/acs.analchem.1c00123>). 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 (Cowger et al. 2020, <doi:10.1177/0003702820929064>) using match_spec(). A Shiny app is available via run_app() or online at <https://openanalysis.org/openspecy/>.

URL https://github.com/wincowgerDEV/OpenSpecy-package/, http://wincowger.com/OpenSpecy-package/

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

License CC BY 4.0

Encoding UTF-8

LazyLoad true

LazyData true

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports methods, data.table, jsonlite, yaml, osfr, caTools, hyperSpec, mmand, plotly, digest, signal, glmnet, shiny

Suggests knitr, rmarkdown, testthat (>= 3.1.9), config, qs, shinyjs, shinyWidgets, bs4Dash, dplyr, ggplot2, DT, curl, aws.s3, mongolite, loggit

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

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

Maintainer Win Cowger <wincowger@gmail.com>

Repository CRAN

Date/Publication 2024-03-14 23:00:03 UTC

R topics documented:

`adj_intens` *Adjust spectral intensities to absorbance units.*

Description

Converts reflectance or transmittance intensity units to absorbance units.

Usage

```
adj_intens(x, ...)

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

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

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().
...	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. 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 containing the spectral intensities.
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 numeric vectors 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

```
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))
```

---

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, ...)

## 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),
  coords = "gen_grid",
  session_id = 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.
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 data.table 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.
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 `data.table` 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

`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, options transmittance, reflectance, absorbance

`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_units` supported options include `"absorbance"`, `"transmittance"`, or `"reflectance"`

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

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

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

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_units`: supported options include "absorbance", "transmittance", or "reflectance"
`derivative_order`: supported options include "0", "1", or "2"
`baseline`: supported options include "raw" or "nobaseline"
`spectra_type`: supported options include "ftir" or "raman"

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 data.table.
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)
```

---

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", "mediod", "model"),
  path = "system",
  condition = "warning"
)

get_lib(
  type = c("derivative", "nobaseline", "raw", "mediod", "model"),
  path = "system",
  node = "x7dpz",
  conflicts = "overwrite",
  ...
)

load_lib(type, path = "system")

rm_lib(
  type = c("derivative", "nobaseline", "raw", "mediod", "model"),
  path = "system"
)
```

Arguments

type	library type to check/retrieve; defaults to `c("derivative", "nobaseline", "raw", "mediod", "model")` 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"`).
node	the OSF node to be retrieved; should be `"x7dpz"` unless you maintain your own OSF node with spectral libraries.
conflicts	determines what happens when a file with the same name exists at the specified destination. Can be one of the following (see `osf_download()` for details): **"error"** throw an error and abort the file transfer operation. **"skip"** skip the conflicting file(s) and continue transferring the remaining files. **"overwrite" (default)** replace the existing file with the transferred copy.
...	further arguments passed to `osf_download()`.

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 OSF (doi:10.17605/OSF.IO/X7DPZ). `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*. https://www.chem.ucl.ac.uk/resources/raman/.

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). http://www.geologie-lyon.fr/Raman/.

"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 (https://ncsslabdatamart.sc.egov.usda.gov), 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_lib("derivative")
get_lib("derivative")

spec_lib <- load_lib("derivative")

## 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, ...)

def_features(x, ...)

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

## S3 method for class 'OpenSpecy'
def_features(x, features, shape_kernel = c(3, 3), ...)
```

Arguments

x	an OpenSpecy object
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.
...	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 to median 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.

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" only works when Spectra are being aggregated, we take the mean of the intensities within the wavenumber specified. This can maintain smaller peaks and make spectra more similar to it's less resolved relatives. 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))
```

---

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", ...)

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,
  add_library_metadata = NULL,
  add_object_metadata = NULL,
  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, ...)

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 glmnet 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()
top_n	integer; specifying the number of top matches to return. If NULL (default), all matches will be returned.
order	an OpenSpecy used for sorting, ideally the unprocessed one; NULL skips sorting.
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 the AI classification.
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)
```

c_spec *Manage spectral objects*

Description

`c_spec()` concatenates `OpenSpecy` objects. `sample_spec()` samples spectra from an `OpenSpecy` object.

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 = NULL, res = 5, ...)

sample_spec(x, ...)

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

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

Arguments

`x`	a list of `OpenSpecy` objects.
`range`	a numeric providing your own wavenumber ranges or character argument called `"common"` to let `c_spec()` find the common wavenumber range of the supplied spectra. `NULL` will interpret the spectra having all the same wavenumber range.
`res`	defaults to `NULL`, the resolution you want the output wavenumbers to be.
`size`	the number of spectra to sample.
`prob`	probabilities to use for the sampling.
`...`	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)
common <- c_spec(spectra, range = "common", res = 5)
range <- c_spec(spectra, range = c(1000, 2000), res = 5)

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

---

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, ...)

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

## 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.
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. summary() produces a result summary of an OpenSpecy object. plot() produces a matplot() of an OpenSpecy object; lines() adds new spectra to it.

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()
```

---

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 '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.

Value

`make_rel()` return numeric vectors (if vector provided) 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.

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]][1:10] <- 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")
```

---

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,
  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,
  ...
)

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,
  ...
)

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.
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 plot_ly().
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".
...	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

```
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)
```

---

process_spec *Process Spectra*

Description

`process_spec()` is a monolithic wrapper function for all spectral processing steps.

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),
  ...
)
```

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 smooth_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().
na.rm	Whether to allow NA or set all NA values to
...	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)) |>
  lines(col = "darkred")

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

---

raman_hdpe *Sample Raman spectrum*

Description

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

Format

An threepart 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

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, ...)

read_many(file, ...)

read_zip(file, ...)
```

Arguments

`file`	file to be read from or written to.
`...`	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. `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 the provide a list of `OpenSpecy` objects.

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),
  share = 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.
share	defaults to NULL; needed to share spectra with the Open Specy community; see share_spec() for details.
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.

Author(s)

Zacharias Steinmetz, Claudia Beleites

See Also

read_spec() for reading .y(a)ml, .json, or .rds (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,
  share = 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"),
  type = "spec",
  digits = 1L,
  atm_comp_minus4offset = FALSE
)
```

Arguments

file	character vector with path to file(s).
share	defaults to NULL; needed to share spectra with the Open Specy community; see share_spec() for details.
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 .y(a)ml, .json, or .rds (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, 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",
  share = 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"),
  ...
)

read_asp(
  file,
  share = 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"),
  ...
```

```
)

read_spa(
  file,
  share = 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"),
  ...
)

read_spc(
  file,
  share = 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"),
  ...
)

read_jdx(
  file,
  share = 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"),
  ...
)

read_extdata(file = 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.
share	defaults to NULL; needed to share spectra with the Open Specy community; see share_spec() for details.
metadata	a named list of the metadata; see as_OpenSpecy() for details.
...	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

All read_*() functions return data frames containing two columns named "wavenumber" and "intensity".

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

read_spec() for reading .y(a)ml, .json, or .rds (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, max, make_rel = TRUE, ...)

flatten_range(x, ...)

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

## S3 method for class 'OpenSpecy'
flatten_range(x, min = 2200, max = 2400, make_rel = TRUE, ...)
```

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?
`...`	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)

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 = "main", test_mode = FALSE, ...)
```

Arguments

path	to store the downloaded app files; defaults to "system" pointing to system.file(package = "OpenSpecy").
log	logical; enables/disables logging to tempdir()
ref	git reference; could be a commit, tag, or branch name. Defaults to "main". Only change this in case of errors.
test_mode	logical; for internal testing only.
...	arguments passed to runApp().

Details

After running this function the Open Specy GUI should open in a separate window or in your computer browser.

Value

This function normally does not return any value, see runGitHub().

Author(s)

Zacharias Steinmetz

See Also

runGitHub()

Examples

```
## Not run:
run_app()

## End(Not run)
```

---

share_spec *Share data with the Open Specy community*

Description

This helper function shares spectral data and metadata with the Open Specy community.

Please note: that share_spec() only provides basic sharing functionality if used interactively. This means that files are only formatted and saved for sharing but are not sent automatically. This only works with hosted instances of Open Specy.

Usage

```
share_spec(x, ...)

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

## S3 method for class 'OpenSpecy'
share_spec(x, file = NULL, share = "system", credentials = NULL, ...)
```

Arguments

x	a list object of class OpenSpecy.
file	file to share (optional).
share	accepts any local directory to save the spectrum for later sharing via email to <wincowger@gmail.com>; "system" (default) uses the Open Specy package directory at system.file("extdata",package = "OpenSpecy"); if a correct API token exists, "cloud" shares the spectrum with the cloud.
credentials	a named list of credentials for cloud sharing; required if share = "cloud").
...	further arguments passed to the submethods.

Value

share_spec() returns only messages/warnings.

Author(s)

Zacharias Steinmetz, Win Cowger

See Also

read_text(); digest(); sessionInfo()

Examples

```
## Not run:
data("raman_hdpe")
share_spec(raman_hdpe,
           metadata = list(
             user_name = "Win Cowger",
             spectrum_type = "FTIR",
             spectrum_identity = "PE",
             license = "CC BY-NC"
           ))

## 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.), "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 useing 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 sgolay().

Details

For Savitzky-Golay this is a wrapper around the filter function in the signal package to improve integration with other Open Specy 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

`sgolay()`

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")
```

---

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_spec *Split Open Specy objects*

Description

Convert a list of Open Specy objects with any number of spectra into a list of Open Specy objects with one spectrum each.

Usage

```
split_spec(x)
```

Arguments

x	a list of OpenSpecy objects

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

A list of Open Specy objects each with 1 spectrum.

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 a polynomial fitting or accepts a manual baseline.

Usage

```
subtr_baseline(x, ...)

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

## S3 method for class 'OpenSpecy'
subtr_baseline(
  x,
  type = "polynomial",
  degree = 8,
  raw = FALSE,
  baseline,
  make_rel = TRUE,
  ...
)
```

Arguments

`x`	a list object of class `OpenSpecy`.
`type`	one of `"polynomial"` or `"manual"` depending on whether you want spectra to be corrected with a manual baseline or with polynomial baseline fitting.
`degree`	the degree of the polynomial. Must be less than the number of unique points when raw is `FALSE`. Typically a good fit can be found with a 8th order polynomial.
`raw`	if `TRUE`, use raw and not orthogonal polynomials.
`baseline`	an `OpenSpecy` object containing the baseline data to be subtracted.
`make_rel`	logical; if `TRUE` spectra are automatically normalized with `make_rel()`.
`...`	further arguments passed to `poly()`.

Details

This is a translation of Michael Stephen Chen's MATLAB code written for the `imodpolyfit` routine from Zhao et al. 2007.

Value

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

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.

See Also

`poly()`; `smooth_intens()`

Examples

```
data("raman_hdpe")

subtr_baseline(raman_hdpe)
```

---

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")
```

`write_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`. Currently supported formats are .y(a)ml, .json, .csv, or .rds.

Usage

```
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, share = NULL, method = NULL, ...)

as_hyperSpec(x)
```

Arguments

`x`	an object of class `OpenSpecy`.
`file`	file path to be read from or written to.
`method`	optional; function to be used as a custom reader or writer. Defaults to the appropriate function based on the file extension.
`digits`	number of significant digits to use when formatting numeric values; defaults to `getOption("digits")`.
`share`	defaults to `NULL`; needed to share spectra with the Open Specy community; see `share_spec()` for details.
`...`	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 .yaml files but the numbers should be nearly identical. `readRDS()` should return the exact same object every time.

Value

`read_spec()` reads data formatted as an `OpenSpecy` object and returns a list object of class `OpenSpecy` containing spectral data. `write_spec()` writes a file for an object of class `OpenSpecy` containing spectral data. `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_yaml(); read_yaml(); write_json(); read_json();

Examples

```
read_extdata("raman_hdpe.yml") |> read_spec()
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.yml")
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)
```

Index