Package ‘photobiology’

April 4, 2021

Type Package

Title Photobiological Calculations

Version 0.10.6

Date 2021-04-04


License GPL (>= 2)

Depends R (>= 3.6.0)

Imports stats, polynom (>= 1.4-0), tibble (>= 3.0.4), stringr (>= 1.4.0), lubridate (>= 1.7.8), plyr (>= 1.8.4), dplyr (>= 1.0.2), tidyr (>= 1.1.2), splus2R (>= 1.2-2), zoo (>= 1.8-8), rlang (>= 0.4.8)

Suggests knitr (>= 1.30), rmarkdown (>= 2.4), testthat (>= 2.3.2), roxygen2 (>= 7.1.1)

LazyLoad yes

LazyData yes

ByteCompile true

URL https://docs.r4photobiology.info/photobiology/, https://github.com/aphalo/photobiology

BugReports https://github.com/aphalo/photobiology/issues

Encoding UTF-8

RoxygenNote 7.1.1

VignetteBuilder knitr
**NeedsCompilation**: no

**Author**: Pedro J. Aphalo [aut, cre] (<https://orcid.org/0000-0003-3385-972X>),
Titta K. Kotilainen [ctb] (<https://orcid.org/0000-0002-2822-9734>),
Glenn Davis [ctb],
Agnese Fazio [ctb]

**Maintainer**: Pedro J. Aphalo <pedro.aphalo@helsinki.fi>

**Repository**: CRAN

**Date/Publication**: 2021-04-04 16:00:06 UTC

---

**R topics documented:**

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>photobiology-package</td>
<td>8</td>
</tr>
<tr>
<td>A.illuminant.spect</td>
<td>10</td>
</tr>
<tr>
<td>A2T</td>
<td>11</td>
</tr>
<tr>
<td>absorbance</td>
<td>12</td>
</tr>
<tr>
<td>absorptance</td>
<td>15</td>
</tr>
<tr>
<td>add_attr2tb</td>
<td>18</td>
</tr>
<tr>
<td>Afr2T</td>
<td>21</td>
</tr>
<tr>
<td>any2T</td>
<td>23</td>
</tr>
<tr>
<td>as.calibration_mspct</td>
<td>24</td>
</tr>
<tr>
<td>as.calibration_spect</td>
<td>25</td>
</tr>
<tr>
<td>as.chroma_mspct</td>
<td>26</td>
</tr>
<tr>
<td>as.chroma_spect</td>
<td>27</td>
</tr>
<tr>
<td>as.cps_mspct</td>
<td>28</td>
</tr>
<tr>
<td>as.cps_spect</td>
<td>29</td>
</tr>
<tr>
<td>as.filter_mspct</td>
<td>30</td>
</tr>
<tr>
<td>as.filter_spect</td>
<td>32</td>
</tr>
<tr>
<td>as.generic_mspct</td>
<td>33</td>
</tr>
<tr>
<td>as.generic_spect</td>
<td>35</td>
</tr>
<tr>
<td>as.matrix_mspct</td>
<td>36</td>
</tr>
<tr>
<td>as.object_mspct</td>
<td>36</td>
</tr>
<tr>
<td>as.object_spect</td>
<td>38</td>
</tr>
<tr>
<td>as.raw_mspct</td>
<td>39</td>
</tr>
<tr>
<td>as.raw_spect</td>
<td>40</td>
</tr>
<tr>
<td>as.reflector_mspct</td>
<td>41</td>
</tr>
<tr>
<td>as.reflector_spect</td>
<td>43</td>
</tr>
<tr>
<td>as.response_mspct</td>
<td>44</td>
</tr>
<tr>
<td>as.response_spect</td>
<td>46</td>
</tr>
<tr>
<td>as.solar_date</td>
<td>47</td>
</tr>
<tr>
<td>as.source_mspct</td>
<td>47</td>
</tr>
<tr>
<td>as.source_spect</td>
<td>49</td>
</tr>
<tr>
<td>as_energy</td>
<td>50</td>
</tr>
<tr>
<td>as_quantum</td>
<td>51</td>
</tr>
<tr>
<td>as_quantum_mol</td>
<td>52</td>
</tr>
<tr>
<td>as_tod</td>
<td>52</td>
</tr>
<tr>
<td>average_spect</td>
<td>53</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>beesxyzCMF.spct</td>
<td>54</td>
</tr>
<tr>
<td>black_body.spct</td>
<td>54</td>
</tr>
<tr>
<td>c</td>
<td>55</td>
</tr>
<tr>
<td>calc_multipliers</td>
<td>55</td>
</tr>
<tr>
<td>calc_source_output</td>
<td>57</td>
</tr>
<tr>
<td>ccd.spct</td>
<td>58</td>
</tr>
<tr>
<td>checkTimeUnit</td>
<td>59</td>
</tr>
<tr>
<td>check_spct</td>
<td>59</td>
</tr>
<tr>
<td>check_spectrum</td>
<td>63</td>
</tr>
<tr>
<td>check_w.length</td>
<td>63</td>
</tr>
<tr>
<td>ciev10.spct</td>
<td>64</td>
</tr>
<tr>
<td>ciev2.spct</td>
<td>65</td>
</tr>
<tr>
<td>ciexyzCC10.spct</td>
<td>66</td>
</tr>
<tr>
<td>ciexyzCC2.spct</td>
<td>67</td>
</tr>
<tr>
<td>ciexyzCMF10.spct</td>
<td>68</td>
</tr>
<tr>
<td>ciexyzCMF2.spct</td>
<td>69</td>
</tr>
<tr>
<td>class_spct</td>
<td>70</td>
</tr>
<tr>
<td>clean</td>
<td>70</td>
</tr>
<tr>
<td>clear.spct</td>
<td>75</td>
</tr>
<tr>
<td>clear_body.spct</td>
<td>76</td>
</tr>
<tr>
<td>clip_wl</td>
<td>76</td>
</tr>
<tr>
<td>collect2mspct</td>
<td>78</td>
</tr>
<tr>
<td>color_of</td>
<td>79</td>
</tr>
<tr>
<td>compare_spct</td>
<td>81</td>
</tr>
<tr>
<td>cone_fundamentals10.spct</td>
<td>83</td>
</tr>
<tr>
<td>convertTfrType</td>
<td>84</td>
</tr>
<tr>
<td>convertThickness</td>
<td>85</td>
</tr>
<tr>
<td>convertTimeUnit</td>
<td>86</td>
</tr>
<tr>
<td>convolve_each</td>
<td>87</td>
</tr>
<tr>
<td>copy_attributes</td>
<td>87</td>
</tr>
<tr>
<td>cps2irrad</td>
<td>88</td>
</tr>
<tr>
<td>D2.UV586</td>
<td>89</td>
</tr>
<tr>
<td>D2.UV653</td>
<td>90</td>
</tr>
<tr>
<td>D2.UV654</td>
<td>90</td>
</tr>
<tr>
<td>D2_spectrum</td>
<td>91</td>
</tr>
<tr>
<td>D65.illuminant.spct</td>
<td>91</td>
</tr>
<tr>
<td>day_night</td>
<td>92</td>
</tr>
<tr>
<td>defunct</td>
<td>95</td>
</tr>
<tr>
<td>despike</td>
<td>96</td>
</tr>
<tr>
<td>diffraction_single_slit</td>
<td>103</td>
</tr>
<tr>
<td>dim.generic_mspct</td>
<td>104</td>
</tr>
<tr>
<td>div_generic_spct</td>
<td>104</td>
</tr>
<tr>
<td>div_spectra</td>
<td>105</td>
</tr>
<tr>
<td>drop_user_cols</td>
<td>106</td>
</tr>
<tr>
<td>e2q</td>
<td>108</td>
</tr>
<tr>
<td>e2qmol_multipliers</td>
<td>109</td>
</tr>
<tr>
<td>e2quantum_multipliers</td>
<td>110</td>
</tr>
<tr>
<td>enable_check_spct</td>
<td>110</td>
</tr>
</tbody>
</table>
energy_as_default ....................................................... 111
energy_irradiance .................................................. 112
energy_ratio .......................................................... 113
eq_ratio ................................................................. 114
Extract ................................................................. 116
Extract_mspct ........................................................ 118
e_fluence ............................................................... 119
e_irrad ................................................................. 122
e_ratio ................................................................. 125
e_response ............................................................ 128
FEL.BN.9101.165 ...................................................... 131
FEL_spectrum .......................................................... 131
findMultipleWl ......................................................... 132
find_peaks .............................................................. 133
find_spikes ............................................................. 134
find_wls ................................................................. 135
fit_peaks ............................................................... 137
fluence ................................................................. 138
format.solar_time .................................................... 141
format.tod_time ...................................................... 142
formatted_range ...................................................... 142
fscale ................................................................. 143
fshift ................................................................. 148
generic_mspct ........................................................ 152
getBSWFUsed .......................................................... 153
getFilterProperties ................................................... 154
getHowMeasured ........................................................ 155
getIdFactor ........................................................... 157
getInstrDesc .......................................................... 157
getInstrSettings ....................................................... 158
getMspctVersion ....................................................... 159
getMultipleWl .......................................................... 159
getNormalized .......................................................... 160
getResponseType ....................................................... 161
getRfrType ............................................................ 162
getScaled ............................................................. 162
getSpctVersion ........................................................ 163
getTfrType ............................................................ 164
g getTimeUnit .......................................................... 164
getWhatMeasured ...................................................... 165
getWhenMeasured ..................................................... 166
getWhereMeasured ...................................................... 168
get_attributes ......................................................... 169
get_peaks ............................................................... 171
green_leaf.spct ......................................................... 172
head_tail .............................................................. 173
insert_hinges .......................................................... 175
insert_spct_hinges .................................................... 176
R topics documented:

- integrate_spct .................................................. 176
- integrate_xy .................................................. 177
- interpolate_spct .............................................. 178
- interpolate_spectrum ......................................... 179
- interpolate_wl .................................................. 180
- irrad .............................................................. 182
- irradiance ....................................................... 185
- is.generic_mspct ................................................ 186
- is.generic_spct ................................................ 188
- is.old_spct ..................................................... 189
- is.solar_time ................................................... 189
- is.summary_generic_spct ...................................... 190
- is.waveband ..................................................... 191
- isValidInstrDesc ............................................... 191
- isValidInstrSettings .......................................... 192
- is_absorbance_based ......................................... 192
- is_effective ..................................................... 193
- is_normalized ................................................... 195
- is_photon_based ............................................... 195
- is_scaled ......................................................... 196
- is_tagged ........................................................ 197
- join_mspct ....................................................... 198
- labels .............................................................. 199
- Ler_leaf.spct .................................................... 200
- Ler_leaf_rflt.spct .............................................. 201
- Ler_leaf_trns.spct .............................................. 202
- Ler_leaf_trns_i.spct .......................................... 203
- log ................................................................. 204
- MathFun ........................................................... 205
- merge2object_spct ............................................... 205
- merge_attributes ............................................... 206
- minus-.generic_spct ............................................ 207
- mod-.generic_spct .............................................. 208
- msmsply .......................................................... 208
- mspct_classes .................................................... 209
- na.omit ........................................................... 210
- normalization ..................................................... 212
- normalize ........................................................ 213
- normalized_diff_ind .......................................... 217
- normalize_range_arg .......................................... 219
- opaque.spct ..................................................... 220
- oper_spectra ..................................................... 220
- peaks ............................................................ 222
- photodiode.spct ................................................ 228
- photons_energy_ratio .......................................... 229
- photon_irradiance .............................................. 230
- photon_ratio ..................................................... 232
- plus-.generic_spct ............................................. 233
R topics documented:

polyester.spct ......................................................... 234
print ................................................................. 234
print.solar_time ..................................................... 235
print.summary_generic_spct ........................................ 236
print.tod_time ...................................................... 237
print.waveband ...................................................... 237
prod_spectra ......................................................... 238
q2e ................................................................. 239
qe_ratio ............................................................. 240
q_fluence ............................................................ 243
q_irrad ............................................................... 245
q_ratio ............................................................... 248
q_response .......................................................... 251
r4p_pks ............................................................. 254
rbindspct ........................................................... 255
reflectance .......................................................... 256
relative_AM ......................................................... 259
replace_bad_pixs .................................................... 260
response ............................................................. 262
Rfr_from_n .......................................................... 264
rgb_spct ............................................................. 265
rmDerivedMspct ...................................................... 266
rmDerivedSpct ....................................................... 267
round ............................................................... 268
select_spct_attributes ............................................. 269
setBSWFUsed ........................................................ 270
setFilterProperties ............................................... 270
setGenericSpct ..................................................... 272
setHowMeasured .................................................... 275
setIdFactor ........................................................ 276
setInstrDesc ........................................................ 277
setInstrSettings .................................................... 277
setMultipleWl ....................................................... 278
setNormalized ...................................................... 279
setResponseType ................................................... 279
setRfrType ........................................................ 280
setScaled .......................................................... 281
setTfrType ........................................................ 282
setTimeUnit ......................................................... 283
setWhatMeasured ................................................... 284
setWhenMeasured .................................................. 285
setWhereMeasured ................................................ 286
shared_member_class ............................................. 288
sign ................................................................. 288
slash-.generic_spct ............................................... 289
smooth_spct ......................................................... 289
solar_time .......................................................... 292
source_spct ........................................................ 293
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>spect_attr2tb</td>
<td>297</td>
</tr>
<tr>
<td>spect_classes</td>
<td>298</td>
</tr>
<tr>
<td>spect_metadata</td>
<td>298</td>
</tr>
<tr>
<td>spikes</td>
<td>300</td>
</tr>
<tr>
<td>split2mspect</td>
<td>305</td>
</tr>
<tr>
<td>split_bands</td>
<td>307</td>
</tr>
<tr>
<td>split_energy_irradiance</td>
<td>308</td>
</tr>
<tr>
<td>split_irradiance</td>
<td>310</td>
</tr>
<tr>
<td>split_photon_irradiance</td>
<td>311</td>
</tr>
<tr>
<td>spread</td>
<td>312</td>
</tr>
<tr>
<td>Subset</td>
<td>314</td>
</tr>
<tr>
<td>subset2mspect</td>
<td>315</td>
</tr>
<tr>
<td>subl_spectra</td>
<td>316</td>
</tr>
<tr>
<td>summary</td>
<td>317</td>
</tr>
<tr>
<td>summary_spect_classes</td>
<td>318</td>
</tr>
<tr>
<td>sum_spectra</td>
<td>318</td>
</tr>
<tr>
<td>sun.daily.data</td>
<td>320</td>
</tr>
<tr>
<td>sun.daily.spect</td>
<td>321</td>
</tr>
<tr>
<td>sun.data</td>
<td>322</td>
</tr>
<tr>
<td>sun.spect</td>
<td>323</td>
</tr>
<tr>
<td>sun_angles</td>
<td>324</td>
</tr>
<tr>
<td>s_e_irrad2rgb</td>
<td>326</td>
</tr>
<tr>
<td>s_mean</td>
<td>327</td>
</tr>
<tr>
<td>s_mean_se</td>
<td>329</td>
</tr>
<tr>
<td>s_median</td>
<td>331</td>
</tr>
<tr>
<td>s_prod</td>
<td>332</td>
</tr>
<tr>
<td>s_range</td>
<td>334</td>
</tr>
<tr>
<td>s_sd</td>
<td>335</td>
</tr>
<tr>
<td>s_se</td>
<td>337</td>
</tr>
<tr>
<td>s_sum</td>
<td>339</td>
</tr>
<tr>
<td>s_var</td>
<td>340</td>
</tr>
<tr>
<td>T2A</td>
<td>342</td>
</tr>
<tr>
<td>T2Afr</td>
<td>344</td>
</tr>
<tr>
<td>tag</td>
<td>346</td>
</tr>
<tr>
<td>thin_wl</td>
<td>348</td>
</tr>
<tr>
<td>times-generic_spect</td>
<td>351</td>
</tr>
<tr>
<td>transmittance</td>
<td>351</td>
</tr>
<tr>
<td>Trig</td>
<td>354</td>
</tr>
<tr>
<td>trimInstrDesc</td>
<td>355</td>
</tr>
<tr>
<td>trimInstrSettings</td>
<td>356</td>
</tr>
<tr>
<td>trim_spect</td>
<td>356</td>
</tr>
<tr>
<td>trim_tails</td>
<td>359</td>
</tr>
<tr>
<td>trim_waveband</td>
<td>360</td>
</tr>
<tr>
<td>trim_wl</td>
<td>362</td>
</tr>
<tr>
<td>tz_time_diff</td>
<td>364</td>
</tr>
<tr>
<td>uncollect2spct</td>
<td>364</td>
</tr>
<tr>
<td>untag</td>
<td>366</td>
</tr>
<tr>
<td>upgrade_spect</td>
<td>367</td>
</tr>
</tbody>
</table>
Definitions of classes, methods, operators and functions for use in photobiology and radiation meteorology and climatology. Calculation of effective (weighted) and not-weighted irradiances/doses, fluence rates, transmittance, reflectance, absorptance, absorbance and diverse ratios and other derived quantities from spectral data. Local maxima and minima: peaks, valleys and spikes. Conversion between energy-and photon-based units. Wavelength interpolation. Astronomical calculations related solar angles and day length. Colours and vision. This package is part of the 'r4photobiology' suite, Aphalo, P. J. (2015) <doi:10.19232/uv4pb.2015.1.14>.
Details

Package ‘photobiology’ is at the core of a suite of packages for analysis and plotting of data relevant to photobiology (described at https://www.r4photobiology.info/). The accompanying packages (under development) provide data and definitions that are to a large extent application-area specific while the functions in the present package are widely useful in photobiology and radiation quantification in geophysics and meteorology. Package ‘photobiology’ has its main focus in the characterization of the light environment in a biologically relevant manner and in the manipulation of spectral data to simulate photo-physical, photo-chemical and photo-biological interactions and responses. The focus of package ‘pavo’ (Maia et al., 2003) is in colour perception by animals and assessment of animal coloration. In spite of the different focus, there is some degree of overlap.

Acknowledgements

This work was funded by the Academy of Finland (decision 252548). COST Action FA9604 ‘UV4Growth’ facilitated discussions and exchanges of ideas that lead to the development of this package. The contributions of Andy McLeod, Lars Olof Björn, Nigel Paul, Lasse Ylianttila, T. Matthew Robson and Titta Kotilainen were specially significant. Tutorials by Hadley Wickham and comments on my presentation at UseR!2015 allowed me to significantly improve the coding and functionality.

Note

Code for some of the astronomical calculations has been adapted from that in package ‘pavo’.

Author(s)

Maintainer: Pedro J. Aphalo <pedro.aphalo@helsinki.fi> (ORCID)

Other contributors:

- Titta K. Kotilainen (ORCID) [contributor]
- Glenn Davis <gdavis@gluonics.com> [contributor]
- Agnese Fazio <agnese.fazio@uni-jena.de> [contributor]

References


**A.illuminant.spct**

**CIE A illuminant data**

**Description**

A dataset containing wavelengths at a 5 nm interval (300 nm to 830 nm) and the corresponding spectral energy irradiance normalized to 1 at 560 nm. Spectrum approximates typical, domestic, tungsten-filament lighting and 'corresponds' to a black body a 2856 K. CIE standard illuminant A is intended to represent typical, domestic, tungsten-filament lighting. Original data from [http://files.cie.co.at/204.xls](http://files.cie.co.at/204.xls) downloaded on 2014-07-25 The variables are as follows:

**Usage**

A.illuminant.spct

**Format**

A source spectrum with 96 rows and 2 variables

---

**Examples**

```r
# irradiance of the whole spectrum
irrad(sun.spct)
# photon irradiance 400 nm to 700 nm
q_irrad(sun.spct, waveband(c(400,700)))
# energy irradiance 400 nm to 700 nm
e_irrad(sun.spct, waveband(c(400,700)))
# simulating the effect of a filter on solar irradiance
e_irrad(sun.spct * yellow_gel.spct, waveband(c(400,500)))
e_irrad(sun.spct * yellow_gel.spct, waveband(c(500,700)))
# daylength
sunrise_time(lubridate::today(tzone = "EET"), tz = "EET",
    geocode = data.frame(lat = 60, lon = 25), unit.out = "hour")
day_length(lubridate::today(tzone = "EET"), tz = "EET",
    geocode = data.frame(lat = 60, lon = 25), unit.out = "hour")
# colour as seen by humans
color_of(sun.spct)
color_of(sun.spct * yellow_gel.spct)
# filter transmittance
transmittance(yellow_gel.spct)
transmittance(yellow_gel.spct, waveband(c(400,500)))
transmittance(yellow_gel.spct, waveband(c(500,700)))
```

---

**See Also**

Useful links:

- [https://docs.r4photobiology.info/photobiology/](https://docs.r4photobiology.info/photobiology/)
- [https://github.com/aphalo/photobiology](https://github.com/aphalo/photobiology)
Details

• w.length (nm)
• s.e.irrad (rel. units)

Author(s)

CIE

See Also

Other Spectral data examples: D65.illuminant.spct, Ler_leaf.spct, Ler_leaf_rflt.spct, Ler_leaf_trns.spct, Ler_leaf_trns_i.spct, black_body.spct, ccd.spct, clear.spct, clear_body.spct, filter_cps.mspct, green_leaf.spct, opaque.spct, photodiode.spct, polyester.spct, sun.daily.data, sun.daily.spct, sun.data, sun.spct, white_body.spct, white_led.cps_spct, white_led.raw_spct, white_led.source_spct, yellow_gel.spct

Examples

A.illuminant.spct

A2T

Convert absorbance into transmittance

Description

Function that converts absorbance (a.u.) into transmittance (fraction).

Usage

A2T(x, action, byref, ...)

## Default S3 method:
A2T(x, action = NULL, byref = FALSE, ...)

## S3 method for class 'numeric'
A2T(x, action = NULL, byref = FALSE, ...)

## S3 method for class 'filter_spct'
A2T(x, action = "add", byref = FALSE, ...)

## S3 method for class 'filter_mspct'
A2T(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)
Arguments

- **x**: an R object
- **action**: a character string
- **byref**: logical indicating if new object will be created by reference or by copy of x
- **...**: not used in current version
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x with a column Tfr added and A and Afr possibly deleted except for w.length. If action = "replace", in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- **default**: Default method for generic function
- **numeric**: method for numeric vectors
- **filter_spct**: Method for filter spectra
- **filter_mspct**: Method for collections of filter spectra

See Also

Other quantity conversion functions: Afr2T(), T2Afr(), T2A(), any2T(), as_quantum(), e2qmol_multipliers(), e2quantum_multipliers(), e2q(), q2e()
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'object_spct'
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'filter_mspct'
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ..., attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
absorbance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ..., attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
Arguments

- `spc`: an R object.
- `w.band`: waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
- `quantity`: character string One of "average" or "mean", "total", "contribution", "contribution.pc", "relative" or "relative.pc".
- `wb.trim`: logical Flag indicating if wavebands crossing spectral data boundaries are trimmed or ignored.
- `use.hinges`: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- `...`: other arguments (possibly used by derived methods).
- `naming`: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- `attr2tb`: character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
- `idx`: character Name of the column with the names of the members of the collection of spectra.
- `.parallel`: if TRUE, apply function in parallel, using parallel backend provided by foreach.
- `.paropts`: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the `.export` and `.packages` arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter `w.band`. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- default: Default for generic function
- `filter_spc`: Specialization for filter spectra
absorptance

- **object_spct**: Specialization for object spectra
- **filter_mspct**: Calculates absorbance from a filter_mspct
- **object_mspct**: Calculates absorbance from an object_mspct

**Note**

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

**Examples**

```r
absorbance(polyester.spct, new_waveband(400,700))
absorbance(yellow_gel.spct, new_waveband(400,700))
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3))
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "average")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "total")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "relative")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "relative.pc")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "contribution")
absorbance(yellow_gel.spct, split_bands(c(400,700), length.out = 3),
quantity = "contribution.pc")
```

<table>
<thead>
<tr>
<th>absorptance</th>
<th>Absorptance</th>
</tr>
</thead>
</table>

**Description**

Function to calculate the mean, total, or other summary of absorptance for spectral data stored in a filter_spct or in an object_spct. Absorptance is a different quantity than absorbance.

**Usage**

```r
absorptance(spt, w.band, quantity, wb.trim, use.hinges, ...)
```

## Default S3 method:

```r
absorptance(spt, w.band, quantity, wb.trim, use.hinges, ...)
```

## S3 method for class 'filter_spct'

```r
absorptance(
  spt,
  w.band = NULL,
  quantity = "average",
)```
arguments

spct an R object.
w. band waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.

quantity character string One of "average" or "mean", "total", "contribution", "contribution.pc", "relative" or "relative.pc".

wb.trim logical Flag if wavebands crossing spectral data boundaries are trimmed or ignored.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

... other arguments (possibly used by derived methods).

naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx character Name of the column with the names of the members of the collection of spectra.

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach.

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

• default: Default for generic function
• filter_spect: Specialization for filter spectra
• object_spect: Specialization for object spectra
• filter_mspct: Calculates absorptance from a filter_mspct
• object_mspct: Calculates absorptance from a object_mspct
Note

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

Examples

```r
absorptance(black_body.spct, new_waveband(400,500))
absorptance(white_body.spct, new_waveband(300,400))
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3))
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
  quantity = "average")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
  quantity = "total")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
  quantity = "relative")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
  quantity = "relative.pc")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
  quantity = "contribution")
absorptance(black_body.spct, split_bands(c(400,700), length.out = 3),
  quantity = "contribution.pc")
```

---

**add_attr2tb**  
*Copy attributes from members of a generic_mspct*

**Description**

Copy metadata attributes from members of a generic_mspct object into a tibble or data.frame.

**Usage**

```r
add_attr2tb(
  tb = NULL,
  mspct,
  col.names = NULL,
  idx = "spct.idx",
  unnest = FALSE
)
when_measured2tb(
  mspct,
  tb = NULL,
  col.names = "when.measured",
  idx = "spct.idx"
)
geocode2tb(mspct, tb = NULL, col.names = "geocode", idx = "spct.idx")
```
lonlat2tb(mspct, tb = NULL, col.names = c("lon", "lat"), idx = "spct.idx")

lon2tb(mspct, tb = NULL, col.names = "lon", idx = "spct.idx")

lat2tb(mspct, tb = NULL, col.names = "lat", idx = "spct.idx")

address2tb(mspct, tb = NULL, col.names = "address", idx = "spct.idx")

what_measured2tb(
    mspct,
    tb = NULL,
    col.names = "what.measured",
    idx = "spct.idx"
)

how_measured2tb(mspct, tb = NULL, col.names = "how.measured", idx = "spct.idx")

normalized2tb(mspct, tb = NULL, col.names = "normalized", idx = "spct.idx")

scaled2tb(mspct, tb = NULL, col.names = "scaled", idx = "spct.idx")

instr_desc2tb(mspct, tb = NULL, col.names = "instr.desc", idx = "spct.idx")

instr_settings2tb(
    mspct,
    tb = NULL,
    col.names = "instr.settings",
    idx = "spct.idx"
)

BSWF_used2tb(mspct, tb = NULL, col.names = "BSWF.used", idx = "spct.idx")

filter_properties2tb(
    mspct,
    tb = NULL,
    col.names = "filter.properties",
    idx = "spct.idx"
)

Tfr_type2tb(mspct, tb = NULL, col.names = "Tfr.type", idx = "spct.idx")

Rfr_type2tb(mspct, tb = NULL, col.names = "Rfr.type", idx = "spct.idx")

time_unit2tb(mspct, tb = NULL, col.names = "time.unit", idx = "spct.idx")

comment2tb(mspct, tb = NULL, col.names = "comment", idx = "spct.idx")
add_attr2tb

Arguments

- **tb**: tibble or data.frame to which to add the data (optional).
- **mspct**: generic_mspct Any collection of spectra.
- **col.names**: named character vector Name(s) of metadata attributes to copy, while if named, the names provide the name for the column.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **unnest**: logical Flag controlling if metadata attributes that are lists of values should be returned in a list column or in separate columns.

Details

The attributes are copied to a column in a tibble or data frame. If the `tb` formal parameter receives `NULL` as argument, a new tibble will be created. If an existing `data.frame` or tibble is passed as argument, new columns are added to it. However, the number of rows in the argument passed to `tb` must match the number of spectra in the argument passed to `mspct`. Only in the case of method `add_attr2tb()` if the argument to `col.names` is a named vector, the names of members are used as names for the columns created. This permits setting any valid name for the new columns. If the vector passed to `col.names` has no names the names of the attributes are used for the new columns.

If the fields of the attributes are unnested their names are used as names for the columns.

Valid accepted as argument to `col.names` are `NULL`, "lon", "lat", "address", "geocode", "where.measured", "when.measured", "what.measured", "how.measured", "comment", "normalised", "normalized", "scaled", "bswf.used", "instr.desc", "instr.settings", "filter.properties", "Tfr.type", "Rfr.type", "time.unit".

Value

A tibble With the metadata attributes in separate new variables.

Note

The order of the first two arguments is reversed in `add_attr2tb()` compared to the other functions. This is to allow its use in 'pipes', while the functions for single attributes are expected to be used mostly to create new tibbles.

See Also


Examples

```r
library(dplyr)
```
my.mspct <- source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2))
q_irrad(my.mspct) %>%
  add_attr2tb(my.mspct, c(lat = "latitude",
    lon = "longitude",
    when.measured = "time"))

when_measured2tb(my.mspct)

Afr2T

Convert transmittance into absorptance.

Description

Function that converts transmittance (fraction) into absorptance (fraction). If reflectance (fraction) is available, it allows conversions between internal and total absorptance.

Usage

Afr2T(x, action, byref, clean, ...)

## Default S3 method:
Afr2T(x, action = NULL, byref = FALSE, clean = FALSE, ...)

## S3 method for class 'numeric'
Afr2T(x, action = NULL, byref = FALSE, clean = FALSE, Rfr = NA_real_, ...)

## S3 method for class 'filter_spct'
Afr2T(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'object_spct'
Afr2T(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'filter_mspct'
Afr2T(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ..., .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
Afr2T(
  x,
  action = "add",
  ..., .parallel = FALSE,
  .paropts = NULL
)
Arguments

- **x**: an R object
- **action**: character, Allowed values "replace" and "add"
- **byref**: logical, indicating if new object will be created by reference or by copy of x
- **clean**: logical, replace off-boundary values before conversion
- **Rfr**: numeric vector. Spectral reflectance or reflectance factor. Set to zero if x is internal reflectance.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x with a column Tfr added and other columns possibly deleted except for w.length. If action = "replace", in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- **default**: Default method for generic function
- **numeric**: Default method for generic function
- **filter_spct**: Method for filter spectra
- **object_spct**: Method for object spectra
- **filter_mspct**: Method for collections of filter spectra
- **object_mspct**: Method for collections of object spectra

See Also

Other quantity conversion functions: `A2T()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `e2q()`, `q2e()`

Examples

`T2Afr(Ler_leaf.spct)`
any2T

Convert filter quantities.

Description

Functions that convert or add related physical quantities to filter_spct or object_spct objects. transmittance (fraction) into absorptance (fraction).

Usage

any2T(x, action = "add", clean = FALSE)
any2A(x, action = "add", clean = FALSE)
any2Afr(x, action = "add", clean = FALSE)

Arguments

- **x**: an filter_spct or a filter_mspct object.
- **action**: character. Allowed values "replace" and "add".
- **clean**: logical. Replace off-boundary values before conversion.

Details

These functions are dispatchers for A2T, Afr2T, T2A, and T2Afr. The dispatch is based on the names of the variables stored in x. They do not support in-place modification of x.

Value

A copy of x with the columns for the different quantities added or replaced. If action = "replace", in all cases, the additional columns are removed, even if no column needs to be added.

See Also

Other quantity conversion functions: A2T(), Afr2T(), T2A(), as_quantum(), e2qmol_multipliers(), e2quantum_multipliers(), e2q(), q2e()

Examples

any2Afr(Ler_leaf.spct)
any2T(Ler_leaf.spct)
any2T(polyester.spct)
as.calibration_mspct  

Coerce to a collection-of-spectra

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.calibration_mspct(x, ...)

## Default S3 method:
as.calibration_mspct(x, ...)

## S3 method for class 'data.frame'
as.calibration_mspct(x, ...)

## S3 method for class 'calibration_spct'
as.calibration_mspct(x, ...)

## S3 method for class 'list'
as.calibration_mspct(x, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.calibration_mspct(
  x, 
  w.length, 
  spct.data.var = "irrad.mult", 
  multiplier = 1, 
  byrow = NULL, 
  spct.names = "spct_",
  ... 
)

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

... passed to individual spectrum object constructor

ncol integer Number of ‘virtual’ columns in data

byrow logical If ncol > 1 how to read in the data

w.length numeric A vector of wavelength values sorted in strictly ascending order (nm).

spct.data.var character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
as.calibration_spct

multiplier numeric A multiplier to be applied to the values in x to do unit or scale conversion.

spct.names character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of x converted into a calibration_mspctt object.

Methods (by class)

• default:
  • data.frame:
  • calibration_spct:
  • list:
  • matrix:

Note

When x is a square matrix an explicit argument is needed for byrow to indicate how data in x should be read. In every case the length of the w.length vector must match one of the dimensions of x.

See Also

Other Coercion methods for collections of spectra: as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), split2mspct(), subset2mspct()

as.calibration_spct Coerce to a spectrum

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.calibration_spct(x, ...)

## Default S3 method:
as.calibration_spct(x, ...)

Arguments

x an R object

... other arguments passed to "set" functions
Value

A copy of x converted into a calibration_spct object.

Methods (by class)

• default:

See Also

setGenericSpct

Other constructors of spectral objects: as.chroma_spct(), as.cps_spct(), as.filter_spct(),
as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(),
as.source_spct(), source_spct()

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.chroma_mspct(x, ...)

## Default S3 method:
as.chroma_mspct(x, ...)

## S3 method for class 'data.frame'
as.chroma_mspct(x, ...)

## S3 method for class 'chroma_spct'
as.chroma_mspct(x, ...)

## S3 method for class 'list'
as.chroma_mspct(x, ..., ncol = 1, byrow = FALSE)

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

... passed to individual spectrum object constructor

ncol integer Number of ’virtual’ columns in data

byrow logical If ncol > 1 how to read in the data
as.chroma_spct

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.chroma_spct(x, ...)

## Default S3 method:
as.chroma_spct(x, ...)

Arguments

x an R object

... other arguments passed to "set" functions

Value

A copy of x converted into a chroma_spct object.

Methods (by class)

• default:

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), split2mspct(), subset2mspct()
as.cps_mspct

Coerce to a collection-of-spectra

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.cps_mspct(x, ...)

## Default S3 method:
as.cps_mspct(x, ...)

## S3 method for class 'data.frame'
as.cps_mspct(x, ...)

## S3 method for class 'cps_spct'
as.cps_mspct(x, ...)

## S3 method for class 'list'
as.cps_mspct(x, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.cps_mspct(
  x,
  w.length,
  spct.data.var = "cps",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
...
ncol integer Number of 'virtual' columns in data

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.source_spct(), source_spct()
as.cps_spct

byrow

logical If ncol > 1 how to read in the data

w.length

numeric A vector of wavelength values sorted in strictly ascending order (nm).

spct.data.var

character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.

multiplier

numeric A multiplier to be applied to the values in x to do unit or scale conversion.

spct.names

character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of x converted into a cps_mspct object.

Methods (by class)

• default:
  • data.frame:
  • cps_spct:
  • list:
  • matrix:

Note

When x is a square matrix an explicit argument is needed for byrow to indicate how data in x should be read. In every case the length of the w.length vector must match one of the dimensions of x.

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), split2mspct(), subset2mspct()

as.cps_spct  Coerce to a spectrum

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.cps_spct(x, ...)

## Default S3 method:
as.cps_spct(x, ...)
as.filter.mspct

Arguments

x  an R object
...
other arguments passed to "set" functions

Value

A copy of x converted into a cps_spct object.

Methods (by class)

• default:

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.filter_spct(),
as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(),
as.source_spct(), source_spct()

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.filter.mspct(x, ...)

## Default S3 method:
average

as.filter.mspct(x, ...)

## S3 method for class 'data.frame'
average

as.filter.mspct(x, Tfr.type = c("total", "internal"), strict.range = TRUE, ...)

## S3 method for class 'filter_spct'
average

as.filter.mspct(x, ...)

## S3 method for class 'list'
average

as.filter.mspct(
  x,
  Tfr.type = c("total", "internal"),
  strict.range = TRUE,
  ..., 
  ncol = 1,
as.filter_mspct

byrow = FALSE
)

## S3 method for class 'matrix'
as.filter_mspct(
  x,
  w.length,
  spct.data.var = "Tfr",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

... passed to individual spectrum object constructor

Tfr.type a character string, either "total" or "internal"

strict.range logical Flag indicating how off-range values are handled

ncol integer Number of 'virtual' columns in data

byrow logical If ncol > 1 how to read in the data

w.length numeric A vector of wavelengthvalues sorted in strictly ascending order (nm).

spct.data.var character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.

multiplier numeric A multiplier to be applied to the values in x to do unit or scale conversion.

spct.names character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of x converted into a filter_mspct object.

Methods (by class)

• default:
  • data.frame:
  • filter_spct:
  • list:
  • matrix:
as.filter_spct

Note

When \( x \) is a square matrix an explicit argument is needed for byrow to indicate how data in \( x \) should be read. In every case the length of the w.length vector must match one of the dimensions of \( x \).

See Also

Other Coercion methods for collections of spectra: \texttt{as.calibration_mspct()}, \texttt{as.chroma_mspct()}, \texttt{as.cps_mspct()}, \texttt{as.generic_mspct()}, \texttt{as.object_mspct()}, \texttt{as.raw_mspct()}, \texttt{as.reflector_mspct()}, \texttt{as.response_mspct()}, \texttt{as.source_mspct()}, \texttt{split2mspct()}, \texttt{subset2mspct()}

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

\begin{verbatim}
  as.filter_spct(x, ...)
  # Default S3 method:
  as.filter_spct(
    x, 
    Tfr.type = c("total", "internal"),
    strict.range = getOption("photobiology.strict.range", default = FALSE),
    ...
  )
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} an R object
  \item \texttt{...} other arguments passed to "set" functions
  \item \texttt{Tfr.type} a character string, either "total" or "internal"
  \item \texttt{strict.range} logical Flag indicating whether off-range values result in an error instead of a warning
\end{itemize}

Value

A copy of \( x \) converted into a filter_spct object.

Methods (by class)

\begin{itemize}
  \item default:
as.generic_mspct

See Also

setGenericSpct

Other constructor of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), as.source_spct(), source_spct()

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.generic_mspct(x, ...)

## Default S3 method:
as.generic_mspct(x, ...)

## S3 method for class 'data.frame'
as.generic_mspct(x, force.spct.class = FALSE, ...)

## S3 method for class 'generic_spct'
as.generic_mspct(x, force.spct.class = FALSE, ...)

## S3 method for class 'list'
as.generic_mspct(x, force.spct.class = FALSE, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.generic_mspct(
  x,
  w.length,
  member.class,
  spct.data.var,
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)

mat2mspct(
  x,
  w.length,
  member.class,
  spct.data.var,
Arguments

- `x`: a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
- `...`: passed to individual spectrum object constructor
- `force.spct.class`: logical indicating whether to change the class of members to `generic_spct` or retain the existing class.
- `ncol`: integer Number of 'virtual' columns in data
- `byrow`: logical If `ncol > 1` how to read in the data
- `w.length`: numeric A vector of wavelength values sorted in strictly ascending order (nm).
- `member.class`: character The name of the class of the individual spectra to be constructed.
- `spct.data.var`: character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
- `multiplier`: numeric A multiplier to be applied to the values in `x` to do unit or scale conversion.
- `spct.names`: character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of `x` converted into a `generic_mspct` object.

Methods (by class)

- default:
- data.frame:
- generic_spct:
- list:
- matrix:

Note

Members of `generic_mspct` objects can be heterogeneous: they can belong to any class derived from `generic_spct` and class is not enforced. When `x` is a list of data frames `force.spct.class` = TRUE needs to be supplied. When `x` is a square matrix an explicit argument is needed for `byrow` to indicate how data in `x` should be read. In every case the length of the `w.length` vector must match one of the dimensions of `x`. 
as.generic_spct  

See Also  
Other Coercion methods for collections of spectra:  
as.calibration_mspct(), as.chroma_mspct(),  
as.cps_mspct(), as.filter_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(),  
as.response_mspct(), as.source_mspct(), split2mspct(), subset2mspct()

---

as.generic_spct     Coerce to a spectrum

Description
Return a copy of an R object with its class set to a given type of spectrum.

Usage
as.generic_spct(x, ...)

## Default S3 method:  
as.generic_spct(x, ...)

Arguments
x      an R object

...    other arguments passed to "set" functions

Value
A copy of x converted into a generic_spct object.

Methods (by class)
• default:

See Also
setGenericSpct
Other constructors of spectral objects:  
as.calibration_spct(), as.chroma_spct(), as.cps_spct(),  
as.filter_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(),  
as.source_spct(), source_spct()
### as.object_mspct

Coerce to a collection of spectra.

#### Description

Return a copy of an R object with its class set to a given type of spectrum.

### as.matrix-mspct

Coerce a collection of spectra into a matrix.

#### Description

Convert an object of class generic_mspct or a derived class into an R matrix with wavelengths saved as an attribute and spectral data in rows or columns.

#### Usage

```r
## S3 method for class 'generic_mspct'
as.matrix(x, spct.data.var, byrow = attr(x, "mspct.byrow"), ...)
mspct2mat(x, spct.data.var, byrow = attr(x, "mspct.byrow"), ...)
```

#### Arguments

- `x`: generic_mspct object.
- `spct.data.var`: character The name of the variable containing the spectral data.
- `byrow`: logical. If FALSE (the default) the matrix is filled with the spectra stored by columns, otherwise the matrix is filled by rows.
- `...`: currently ignored.

#### Warning!

This conversion preserves the spectral data but discards almost all the metadata contained in the spectral objects. In other words, a matrix created with this function cannot be used to recreate the original object unless the same metadata is explicitly supplied when converting the matrix into a new collection of spectra.

#### Note

Only collections of spectra containing spectra with exactly the same `w.length` values can be converted. If needed, the spectra can be re-expressed before attempting the conversion to a matrix.
Usage

as.object_mspct(x, ...)

## Default S3 method:
as.object_mspct(x, ...)

## S3 method for class 'data.frame'
as.object_mspct(
x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...
)

## S3 method for class 'object_spct'
as.object_mspct(x, ...)

## S3 method for class 'list'
as.object_mspct(
x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...,
  ncol = 1,
  byrow = FALSE
)

Arguments

x a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

... passed to individual spectrum object constructor

Tfr.type a character string, either "total" or "internal"

Rfr.type a character string, either "total" or "specular"

strict.range logical Flag indicating how off-range values are handled

ncol integer Number of 'virtual' columns in data

byrow logical If ncol > 1 how to read in the data

Value

A copy of x converted into a object_mspct object.

Methods (by class)

* default:
• data.frame:
• object_spct:
• list:

See Also
Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), split2mspct(), subset2mspct()

as.object_spct

## Coerce to a spectrum

### Description
Return a copy of an R object with its class set to a given type of spectrum.

### Usage
as.object_spct(x, ...)

## Default S3 method:
as.object_spct(
x,
Tfr.type = c("total", "internal"),
Rfr.type = c("total", "specular"),
strict.range = getOption("photobiology.strict.range", default = FALSE),
...
)

### Arguments

- `x`:
an R object
- `...`:
other arguments passed to "set" functions
- `Tfr.type`:
a character string, either "total" or "internal"
- `Rfr.type`:
a character string, either "total" or "specular"
- `strict.range`:
logical Flag indicating whether off-range values result in an error instead of a warning

### Value
A copy of `x` converted into a `object_spct` object.

### Methods (by class)

- default:
See Also

`setGenericSpect`  
Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.response_spct()`, `as.source_spct()`, `source_spct()`

---

**as.raw_mspct**  
*Coerce to a collection-of-spectra*

---

**Description**

Return a copy of an R object with its class set to a given type of spectrum.

**Usage**

```r
as.raw_mspct(x, ...)  
## Default S3 method:  
as.raw_mspct(x, ...)

## S3 method for class 'data.frame'
as.raw_mspct(x, ...)

## S3 method for class 'raw_spct'
as.raw_mspct(x, ...)

## S3 method for class 'list'
as.raw_mspct(x, ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.raw_mspct(
  x,
  w.length,
  spct.data.var = "counts",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)
```

**Arguments**

- `x`  
a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.

- `...`  
  passed to individual spectrum object constructor

- `ncol`  
  integer Number of ‘virtual’ columns in data
Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.raw_spct(x, ...)

## Default S3 method:
as.raw_spct(x, ...)
as.reflector_mspct

Arguments

x an R object

... other arguments passed to "set" functions

Value

A copy of x converted into a raw_spct object.

Methods (by class)

• default:

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(),
as.filter_spct(), as.generic_spct(), as.object_spct(), as.reflector_spct(), as.response_spct(),
as.source_spct(), source_spct()

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.reflector_mspct(x, ...)

## Default S3 method:
as.reflector_mspct(x, ...)

## S3 method for class 'data.frame'
as.reflector_mspct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = TRUE,
  ...
)

## S3 method for class 'reflector_spct'
as.reflector_mspct(x, ...)

## S3 method for class 'list'
as.reflector_mspct(
\texttt{x,}
\texttt{Rfr.type = c("total", "specular"),}
\texttt{strict.range = TRUE,}
\texttt{...}
\texttt{ncol = 1,}
\texttt{byrow = FALSE}
\texttt{)}

\footnotesize
\texttt{## S3 method for class 'matrix'
\texttt{as.reflector_mspct(}
\texttt{  x,}
\texttt{  w.length,}
\texttt{  spct.data.var = "Rfr",}
\texttt{  multiplier = 1,}
\texttt{  byrow = NULL,}
\texttt{  spct.names = "spct_",}
\texttt{  ...}
\texttt{)}

\textbf{Arguments}

\begin{itemize}
\item \textbf{x} \hspace{1cm} \text{a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.}
\item \textbf{...} \hspace{1cm} \text{passed to individual spectrum object constructor}
\item \textbf{Rfr.type} \hspace{1cm} \text{a character string, either "total" or "specular"}
\item \textbf{strict.range} \hspace{1cm} \text{logical Flag indicating how off-range values are handled}
\item \textbf{ncol} \hspace{1cm} \text{integer Number of 'virtual' columns in data}
\item \textbf{byrow} \hspace{1cm} \text{logical If ncol > 1 how to read in the data}
\item \textbf{w.length} \hspace{1cm} \text{numeric A vector of wavelength values sorted in strictly ascending order (nm).}
\item \textbf{spct.data.var} \hspace{1cm} \text{character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.}
\item \textbf{multiplier} \hspace{1cm} \text{numeric A multiplier to be applied to the values in x to do unit or scale conversion.}
\item \textbf{spct.names} \hspace{1cm} \text{character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.}
\end{itemize}

\textbf{Value}

A copy of \texttt{x} converted into a \texttt{reflector_mspct} object.

\textbf{Methods (by class)}

- default:
- data.frame:
- reflector_spct:
• list:
• matrix:

Note
When $x$ is a square matrix an explicit argument is needed for `byrow` to indicate how data in $x$ should be read. In every case the length of the `w.length` vector must match one of the dimensions of $x$.

See Also
Other Coercion methods for collections of spectra: `as.calibration_mspct()`, `as.chroma_mspct()`, `as.cps_mspct()`, `as.filter_mspct()`, `as.generic_mspct()`, `as.object_mspct()`, `as.raw_mspct()`, `as.response_mspct()`, `as.source_mspct()`, `split2mspct()`, `subset2mspct()`

---

**as.reflector_spct**  
*Coerce to a spectrum*

**Description**
Return a copy of an R object with its class set to a given type of spectrum.

**Usage**
```r
as.reflector_spct(x, ...)
```

```r
## Default S3 method:
as.reflector_spct(
x,
Rfr.type = c("total", "specular"),
strict.range = getOption("photobiology.strict.range", default = FALSE),
...
)
```

**Arguments**
- `x`  
an R object
- `...`  
other arguments passed to "set" functions
- `Rfr.type`  
a character string, either "total" or "specular"
- `strict.range`  
logical Flag indicating whether off-range values result in an error instead of a warning

**Value**
A copy of $x$ converted into a `reflector_spct` object.
Methods (by class)

- default:

See Also

setGenericSpct

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.response_spct(), as.source_spct(), source_spct()

---

as.response_mspct  Coerce to a collection-of-spectra

---

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.response_mspct(x, ...)

## Default S3 method:
as.response_mspct(x, ...)

## S3 method for class 'data.frame'
as.response_mspct(x, time.unit = "second", ...)

## S3 method for class 'response_spct'
as.response_mspct(x, ...)

## S3 method for class 'list'
as.response_mspct(x, time.unit = "second", ..., ncol = 1, byrow = FALSE)

## S3 method for class 'matrix'
as.response_mspct(
x, w.length,
spct.data.var = "s.e.response",
multiplier = 1,
byrow = NULL,
spct.names = "spct_",
...)

...
Arguments

- **x**: a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
- **...**: passed to individual spectrum object constructor
- **time.unit**: character A string, "second", "day" or "exposure"
- **ncol**: integer Number of 'virtual' columns in data
- **byrow**: logical If ncol > 1 how to read in the data
- **w.length**: numeric A vector of wavelength values sorted in strictly ascending order (nm).
- **spct.data.var**: character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
- **multiplier**: numeric A multiplier to be applied to the values in x to do unit or scale conversion.
- **spct.names**: character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of x converted into a response_mspct object.

Methods (by class)

- default:
- data.frame:
- response_spct:
- list:
- matrix:

Note

When x is a square matrix an explicit argument is needed for byrow to indicate how data in x should be read. In every case the length of the w.length vector must match one of the dimensions of x.

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.source_mspct(), split2mspct(), subset2mspct()
as.response_spct  

Coerce to a spectrum

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.response_spct(x, ...)

## Default S3 method:
as.response_spct(x, time.unit = "second", ...)

Arguments

- **x**: an R object
- **...**: other arguments passed to "set" functions
- **time.unit**: character string indicating the time unit used for spectral irradiance or exposure ("second", "day" or "exposure") or an object of class duration as defined in package lubridate.

Value

A copy of x converted into a response_spct object.

Methods (by class)

- `default`

See Also

- `setGenericSpct`

Other constructors of spectral objects: `as.calibration_spct()`, `as.chroma_spct()`, `as.cps_spct()`, `as.filter_spct()`, `as.generic_spct()`, `as.object_spct()`, `as.raw_spct()`, `as.reflector_spct()`, `as.source_spct()`, `source_spct()`
as.solar_date

Convert a solar_time object into solar_date object

Description

Convert a solar_time object into solar_date object

Usage

as.solar_date(x, time)

Arguments

x solar_time object.
time an R date time object

Value

For method as.solar_date() a date-time object with the class attr set to "solar.time". This is needed only for unambiguous formatting and printing.

as.source_mspct

Coerce to a collection-of-spectra

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.source_mspct(x, ...)

## Default S3 method:
as.source_mspct(x, ...)

## S3 method for class 'data.frame'
as.source_mspct(
x,
time.unit = c("second", "day", "exposure"),
bswf.used = c("none", "unknown"),
strict.range =getOption("photobiology.strict.range", default = FALSE),
...
)

## S3 method for class 'source_spct'


as.source_mspct(x, ...)

## S3 method for class 'list'
as.source_mspct(
  x,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ..., 
  ncol = 1,
  byrow = FALSE
)

## S3 method for class 'matrix'
as.source_mspct(
  x,
  w.length,
  spct.data.var = "s.e.irrad",
  multiplier = 1,
  byrow = NULL,
  spct.names = "spct_",
  ...
)

Arguments

- **x**: a list of spectral objects or a list of objects such as data frames that can be converted into spectral objects.
- **...**: passed to individual spectrum object constructor
- **time.unit**: character A string, "second", "day" or "exposure"
- **bswf.used**: character
- **strict.range**: logical Flag indicating how off-range values are handled
- **ncol**: integer Number of 'virtual' columns in data
- **byrow**: logical If ncol > 1 how to read in the data
- **w.length**: numeric A vector of wavelength values sorted in strictly ascending order (nm).
- **spct.data.var**: character The name of the variable that will contain the spectral data. This indicates what physical quantity is stored in the matrix and the units of expression used.
- **multiplier**: numeric A multiplier to be applied to the values in x to do unit or scale conversion.
- **spct.names**: character Vector of names to be assigned to collection members, either of length 1, or with length equal to the number of spectra.

Value

A copy of x converted into a source_mspct object.
Methods (by class)

• default:
• data.frame:
• source_spct:
• list:
• matrix:

Note

When x is a square matrix an explicit argument is needed for byrow to indicate how data in x should be read. In every case the length of the w.length vector must match one of the dimensions of x.

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), split2mspct(), subset2mspct()

as.source_spct Coerce to a spectrum

Description

Return a copy of an R object with its class set to a given type of spectrum.

Usage

as.source_spct(x, ...)

## Default S3 method:
as.source_spct(
  x,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  ...
)

Arguments

x an R object

... other arguments passed to "set" functions
time.unit character string indicating the time unit used for spectral irradiance or exposure ("second", "day" or "exposure") or an object of class duration as defined in package lubridate.
as_energy

bswf.used character
strict.range logical Flag indicating whether off-range values result in an error instead of a warning

Value
A copy of x converted into a source_spct object.

Methods (by class)
• default:

See Also
setGenericSpct
Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(), as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(), as.response_spct(), source_spct()

as_energy Convert spectral photon irradiance into spectral energy irradiance

Description
Convert a spectral photon irradiance [mol s-1 m-2 nm-1] into a spectral energy irradiance [W m-2 nm-1].

Usage
as_energy(w.length, s.qmol.irrad)

Arguments
w.length numeric vector of wavelengths (nm).
s.qmol.irrad numeric vector of spectral photon irradiance values.

Value
A numeric vector of spectral (energy) irradiances.

See Also
Other low-level functions operating on numeric vectors: as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()
**as_quantum**

**Examples**

```r
with(sun.spct, as_energy(w.length, s.q.irrad))
```

**as_quantum**  
*Convert spectral energy irradiance into spectral photon irradiance*

**Description**

Convert spectral energy irradiance [W m\(^{-2}\) nm\(^{-1}\)] into spectral photon irradiance expressed as number of photons [s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)]

**Usage**

```r
as_quantum(w.length, s.e.irrad)
```

**Arguments**

- `w.length` numeric vector of wavelengths (nm).
- `s.e.irrad` numeric vector of spectral (energy) irradiance values.

**Value**

A numeric vector of spectral photon irradiances.

**See Also**

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2Afr()`, `T2A()`, `any2T()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `e2q()`, `q2e()`

**Examples**

```r
with(sun.data, as_quantum(w.length, s.e.irrad))
```
as_quantum_mol | Convert spectral energy irradiance into spectral photon irradiance

Description

Convert spectral energy irradiance \([\text{W m}^{-2} \text{ nm}^{-1}]\) into a spectral photon irradiance expressed in number of molds of photons \([\text{mol s}^{-1} \text{ m}^{-2} \text{ nm}^{-1}]\).

Usage

\[
\text{as\_quantum\_mol}(w.\text{length}, \text{s.e.irrad})
\]

Arguments

- \(w.\text{length}\) numeric vector of wavelengths (nm).
- \(s.e.\text{irrad}\) numeric vector of spectral (energy) irradiance values.

Value

a numeric vector of spectral photon irradiances.

See Also

Other low-level functions operating on numeric vectors: \texttt{as\_energy()}, \texttt{calc\_multipliers()}, \texttt{div\_spectra()}, \texttt{energy\_irradiance()}, \texttt{energy\_ratio()}, \texttt{insert\_hinges()}, \texttt{integrate\_xy()}, \texttt{interpolate\_spectrum()}, \texttt{irradiance()}, \texttt{l\_insert\_hinges()}, \texttt{oper\_spectra()}, \texttt{photon\_irradiance()}, \texttt{photon\_ratio()}, \texttt{photons\_energy\_ratio()}, \texttt{prod\_spectra()}, \texttt{s\_e\_irrad2rgb()}, \texttt{split\_energy\_irradiance()}, \texttt{split\_photon\_irradiance()}, \texttt{subt\_spectra()}, \texttt{sum\_spectra()}, \texttt{trim\_tails()}, \texttt{v\_insert\_hinges()}, \texttt{v\_replace\_hinges()}.

Examples

\[
\text{with(sun.data, as\_quantum\_mol(w.\text{length}, s.e.\text{irrad}))}
\]

as_tod | Convert datetime to time-of-day

Description

Convert a datetime into a time of day expressed in hours, minutes or seconds from midnight in local time for a time zone. This conversion is useful when time-series data for different days needs to be compared or plotted based on the local time-of-day.

Usage

\[
\text{as\_tod}(x, \text{unit.out} = \text{"hours"}, \text{tz} = \text{NULL})
\]
average_spct

Arguments

x          a datetime object accepted by lubridate functions
unit.out   character string. One of "tod_time", "hours", "minutes", or "seconds".
tz         character string indicating time zone to be used in output.

Value

A numeric vector of the same length as x. If unit.out = "tod_time" an object of class "tod_time"
which the same as for unit.out = "hours" but with the class attribute set, which dispatches to
special format() nad print() methods.

Examples

library(lubridate)
my_instants <- ymd_hms("2020-05-17 12:05:03") + days(c(0, 30))
my_instants
as_tod(my_instants)
as_tod(my_instants, unit.out = "tod_time")

Description

This function gives the result of integrating spectral data over wavelengths and dividing the result
by the spread or span of the wavelengths.

Usage

average_spct(spct)

Arguments

spct          generic_spct

Value

One or more numeric values with no change in scale factor: e.g. [W m-2 nm-1] -> [W m-2 nm-
1]. Each value in the returned vector corresponds to a variable in the spectral object, except for
wavelength.

Examples

average_spct(sun.spct)
**beesxyzCMF.spct**  
*Honeybee xyz chromaticity colour matching function data*

**Description**
A dataset containing wavelengths at a 5 nm interval (300 nm to 700 nm) and the corresponding x, y, and z chromaticity coordinates. Original data from XXX.  
A chroma_spct object with variables as follows:

**Usage**

```
beesxyzCMF.spct
```

**Format**
A data frame with 81 rows and 4 variables

**Details**
- w.length (nm)
- x
- y
- z

**See Also**

---

**black_body.spct**  
*Theoretical black body*

**Description**
A dataset for a hypothetical object with transmittance 0/1 (0%), reflectance 0/1 (0%)

**Format**
A object_spct object with 4 rows and 3 variables

**Details**
- w.length (nm)
- Tfr (0..1)
- Rfr (0..1)
See Also


---

**c**

*Combine collections of spectra*

**Description**

Combine two or more generic_mspct objects into a single object.

**Usage**

```r
## S3 method for class 'generic_mspct'
c(..., recursive = FALSE, ncol = 1, byrow = FALSE)
```

**Arguments**

- `...`: one or more generic_mspct objects to combine.
- `recursive`: logical ignored as nesting of collections of spectra is not supported.
- `ncol`: numeric Virtual number of columns
- `byrow`: logical When object has two dimensions, how to map member objects to columns and rows.

**Value**

A collection of spectra object belonging to the most derived class shared among the combined objects.

---

**calc_multipliers**

*Spectral weights*

**Description**

Calculate multipliers for selecting a range of wavelengths and optionally applying a biological spectral weighting function (BSWF) and wavelength normalization. This function returns numeric multipliers that can be used to select a waveband and apply a weight.
Usage

calc_multipliers(
  w.length,
  w.band,
  unit.out = "energy",
  unit.in = "energy",
  use.cached.mult = FALSE,
  fill = 0
)

Arguments

w.length numeric vector of wavelengths (nm).
w.band waveband object.
unit.out character A string: "photon" or "energy", default is "energy".
unit.in character A string: "photon" or "energy", default is "energy".
use.cached.mult logical Flag indicating whether multiplier values should be cached between calls.
fill numeric If fill = NA then values returned for wavelengths outside the range of the waveband are set to NA.

Value

a numeric vector of multipliers of the same length as w.length.

See Also

Other low-level functions operating on numeric vectors: as_energy(), as_quantum_mol(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

Examples

with(sun.data, calc_multipliers(w.length, new_waveband(400,700),"photon"))
with(sun.data, calc_multipliers(w.length, new_waveband(400,700),"photon"), use.cached.mult = TRUE)
**Description**

Values calculated by interpolation from user-supplied spectral emission data or by name for light source data included in the packages photobiologySun, photobiologyLamps, or photobiologyLEDs, optionally re-scaling the spectral data values.

**Usage**

```r
calc_source_output(
  w.length.out,
  w.length.in,
  s.irrad.in,
  unit.in = "energy",
  scaled = NULL,
  fill = NA,
  ...
)
```

**Arguments**

- `w.length.out` numeric vector of wavelengths (nm) for output.
- `w.length.in` numeric vector of wavelengths (nm) for input.
- `s.irrad.in` numeric vector of spectral transmittance value (fractions or percent).
- `unit.in` a character vector "energy" or "photon".
- `scaled` NULL, "peak", "area"; div ignored if !is.null(scaled).
- `fill` if NA, no extrapolation is done, and NA is returned for wavelengths outside the range of the input. If NULL then the tails are deleted. If 0 then the tails are set to zero.
- `...` Additional arguments passed to `spline` if called.

**Value**

A `source_spct` with three numeric vectors with wavelength values (w.length), scaled and interpolated spectral energy irradiance (s.e.irrad), scaled and interpolated spectral photon irradiance values (s.q.irrad).

**Note**

This is a convenience function that adds no new functionality but makes it a little easier to plot lamp spectral emission data consistently. It automates interpolation, extrapolation/trimming and scaling.
Examples

```r
with(sun.data,
    calc_source_output(290:1100,
        w.length.in = w.length,
        s.irrad.in = s.e.irrad)
)
```

ccd.spct  
*Spectral response of a back-thinned CCD image sensor.*

Description

A dataset containing wavelengths at a 1 nm interval and spectral response as quantum efficiency for CCD sensor type S11071/S10420 from Hamamatsu (measured without a quartz window). These vectors are frequently used as sensors in high-UV-sensitivity vector spectrometers. Data digitized from manufacturer’s data sheet. The original data is expressed as percent quantum efficiency with a value of 77% at the peak. The data have been re-expressed as fractions of one.

Usage

ccd.spct

Format

A `response_spct` object with 186 rows and 2 variables

Details

- `w.length` (nm).
- `s.q.response` (fractional quantum efficiency)

References


See Also

**checkTimeUnit**

*Examples*

```r
ccd.spct
```

---

**checkTimeUnit**  
*Check the "time.unit" attribute of an existing source_spct object*

---

**Description**

Function to read the "time.unit" attribute

**Usage**

```r
checkTimeUnit(x)
```

**Arguments**

- **x**: a source_spct object

**Value**

x possibly with the time.unit attribute modified

**Note**

if x is not a source_spct or a response_spct object, NA is returned

**See Also**

Other time attribute functions: `convertTfrType()`, `convertThickness()`, `convertTimeUnit()`, `getTimeUnit()`, `setTimeUnit()`

---

**check_spct**  
*Check validity of spectral objects*

---

**Description**

Check that an R object contains the expected data members.
Usage

check_spct(x, byref, strict.range, force = FALSE, ...)

## Default S3 method:
check_spct(x, byref = FALSE, strict.range = NA, force = FALSE, ...)

## S3 method for class 'generic_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = NA,
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  
)

## S3 method for class 'calibration_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  
)

## S3 method for class 'raw_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  
)

## S3 method for class 'cps_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  
)

## S3 method for class 'filter_spct'
check_spct(
check_spct

x,
byref = TRUE,
strict.range = getOption("photobiology.strict.range", default = FALSE),
force = FALSE,
multiple.wl = getMultipleWl(x),
...
)

## S3 method for class 'reflector_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'object_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'response_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = NA,
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

## S3 method for class 'source_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)
## S3 method for class 'chroma_spct'
check_spct(
  x,
  byref = TRUE,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  force = FALSE,
  multiple.wl = getMultipleWl(x),
  ...
)

### Arguments
- **x**: An R object
- **byref**: logical indicating if new object will be created by reference or by copy of `x`
- **strict.range**: logical indicating whether off-range values result in an error instead of a warning, `NA` disables the test.
- **force**: logical If TRUE check is done even if checks are disabled.
- **multiple.wl**: numeric Maximum number of repeated w.length entries with same value.

### Methods (by class)
- **default**: Default for generic function.
- **generic_spct**: Specialization for generic_spct.
- **calibration_spct**: Specialization for calibration_spct.
- **raw_spct**: Specialization for raw_spct.
- **cps_spct**: Specialization for cps_spct.
- **filter_spct**: Specialization for filter_spct.
- **reflector_spct**: Specialization for reflector_spct.
- **object_spct**: Specialization for object_spct.
- **response_spct**: Specialization for response_spct.
- **source_spct**: Specialization for source_spct.
- **chroma_spct**: Specialization for chroma_spct.

### See Also
Other data validity check functions: `check_spectrum()`, `check_w.length()`, `enable_check_spct()`

### Examples
```r
check_spct(sun.spct)
check_spct(sun.spct)
# try(check_spct(-sun.spct))
# try(check_spct((sun.spct[1, "w.length"] <- 1000)))
```
check_spectrum

Description

Checks spectral irradiance data in numeric vectors for compliance with assumptions used in calculations.

Usage

check_spectrum(w.length, s.irrad)

Arguments

w.length numeric vector of wavelengths (nm).
s.irrad numeric Corresponding vector of spectral (energy) irradiances (W m-2 nm-1).

Value

A single logical value indicating whether test was passed or not

See Also

Other data validity check functions: check_spct(), check_w.length(), enable_check_spct()

Examples

with(sun.data, check_spectrum(w.length, s.e.irrad))

click_w.length

Sanity check of wavelengths (internal function).

Description

This function checks a w.length vector for compliance with assumptions used in calculations.

Usage

check_w.length(w.length)

Arguments

w.length numeric array of wavelength (nm)
Value

a single logical value indicating whether test was passed or not

See Also

Other data validity check functions: check_spct(), check_spectrum(), enable_check_spct()

Examples

with(sun.data, photobiology:::check_w.length(w.length))
A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding response values for a 2 degrees target. Original data from http://www.cvrl.org/ downloaded on 2014-04-29. The variables are as follows:

- `w.length (nm)`
- `x`
- `y`
- `z`

**Author(s)**

CIE

**See Also**

ciexyzCC10.spct  

**CIE XYZ chromaticity coordinates (CC) 10 deg data**

**Description**

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z chromaticity coordinates. Derived from proposed CIE 2006 standard. Original data from [http://www.cvrl.org/](http://www.cvrl.org/) downloaded on 2014-04-29 The variables are as follows:

- w.length (nm)
- x
- y
- z

**Usage**

ciexyzCC10.spct

**Format**

A chroma_spct object with 441 rows and 4 variables

**Author(s)**

CIE

**See Also**


**Examples**

ciexyzCC10.spct
Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z chromaticity coordinates. According to proposed CIE 2006 standard. Original data from http://www.cvrl.org/ downloaded on 2014-04-28 The variables are as follows:

- w.length (nm)
- x
- y
- z

Usage

ciexyzCC2.spct

Format

A chroma_spct object with 441 rows and 4 variables

Author(s)

CIE

See Also


Examples

ciexyzCC2.spct
ciexyzCMF10.spct  Linear energy CIE xyz colour matching function (CMF) 10 deg data

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding x, y, and z 10 degrees CMF values. Derived from proposed CIE 2006 standard. Original data from http://www.cvrl.org/ downloaded on 2014-04-29 The variables are as follows:

- w.length (nm)
- x
- y
- z

Usage

ciexyzCMF10.spct

Format

A chroma_spct object with 441 rows and 4 variables

Author(s)

CIE

See Also


Examples

ciexyzCMF10.spct
ciexyzCMF2.spct  

Linear energy CIE \(xyz\) colour matching function (CMF) 2 deg data

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding \(x\), \(y\), and \(z\) 2 degrees CMF values. Derived from proposed CIE 2006 standard. Original data from [http://www.cvrl.org/](http://www.cvrl.org/) downloaded on 2014-04-29 The variables are as follows:

- \(w.length\) (nm)
- \(x\)
- \(y\)
- \(z\)

Usage

```r
ciexyzCMF2.spct
```

Format

A chroma_spct object with 441 rows and 4 variables

Author(s)

CIE

See Also

Other Visual response data examples:  
- `beesxyzCMF.spct`
- `ciev10.spct`
- `ciev2.spct`
- `ciexyzCC10.spct`
- `ciexyzCC2.spct`
- `ciexyzCMF10.spct`
- `cone_fundamentals10.spct`

Examples

```r
ciexyzCMF2.spct
```
class_spct  
*Query which is the class of a spectrum*

**Description**

Functions to check if an object is a generic spectrum, or coerce it if possible.

**Usage**

```r
class_spct(x)
```

**Arguments**

- `x` any R object

**Value**

`class_spct` returns a vector containing all matching `xxxx.spct` classes.

**Examples**

```r
class_spct(sun.spct)
class(sun.spct)
```

---

clean  
*Clean (=replace) off-range values in a spectrum*

**Description**

These functions implement the equivalent of `replace()` but for spectral objects instead of vectors.

**Usage**

```r
clean(x, range, range.s.data, fill, ...)
# Default S3 method:
clean(x, range, range.s.data, fill, ...)
# S3 method for class 'source_spct'
clean(
  x,
  range = x,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
)```
clean

...)

## S3 method for class 'filter_spct'
clean(
  x,
  range = x,
  range.s.data = NULL,
  fill = range.s.data,
  qty.out =getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
clean(x, range = x, range.s.data = c(0, 1), fill = range.s.data, ...)

## S3 method for class 'object_spct'
clean(
  x,
  range = x,
  range.s.data = c(0, 1),
  fill = range.s.data,
  min.Afr = NULL,
  ...
)

## S3 method for class 'response_spct'
clean(
  x,
  range = x,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out =getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'cps_spct'
clean(x, range = x, range.s.data = c(0, NA), fill = range.s.data, ...)

## S3 method for class 'raw_spct'
clean(
  x,
  range = x,
  range.s.data = c(NA_real_, NA_real_),
  fill = range.s.data,
  ...
)
## S3 method for class 'generic_spct'
clean(
  x,
  range = x,
  range.s.data = c(NA_real_, NA_real_),
  fill = range.s.data,
  col.names,
  ...
)

## S3 method for class 'source_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_mspct'
clean(
  x,
  range = NULL,
  range.s.data = NULL,
  fill = range.s.data,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, 1),
  fill = range.s.data,
  ...
)

## S3 method for class 'object_mspct'
clean(
  x,
```r
range = NULL,
range.s.data = c(0, 1),
fill = range.s.data,
min.Afr = NULL,
..., 
.parallel = FALSE,
.paropts = NULL

## S3 method for class 'response_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ..., 
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'cps_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  ..., 
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'raw_mspct'
clean(
  x,
  range = NULL,
  range.s.data = c(0, NA),
  fill = range.s.data,
  ..., 
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'generic_mspct'
clean(
  x,
  range = x,
  range.s.data = c(NA_real_, NA_real_),
  ..., 
.parallel = FALSE,
.paropts = NULL
)
```
fill = range.s.data,
col.names,
..., 
.parallel = FALSE,
.paropts = NULL
)

Arguments

x
an R object
range
numeric vector of wavelengths
range.s.data
numeric vector of length two giving the allowable range for the spectral data.
fill
numeric vector of length 1 or 2, giving the replacement values to use at each extreme of the range.
...
currently ignored
unit.out
character string with allowed values "energy", and "photon", or its alias "quantum"
qty.out
character string with allowed values "energy", and "photon", or its alias "quantum"
min.Afr
numeric. Gives the minimum value accepted for the computed absorptance. The default NULL sets a valid value (Afr >= 0) with a warning. If an integer value is passed to digits values are adjusted silently.
col.names
character. The name of the variable to clean
.parallel
if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts
a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x, possibly with some of the spectral data values replaced by the value passed to fill.

Methods (by class)

- default: Default for generic function
- source_spct: Replace off-range values in a source spectrum
- filter_spct: Replace off-range values in a filter spectrum
- reflector_spct: Replace off-range values in a reflector spectrum
- object_spct: Replace off-range values in an object spectrum
- response_spct: Replace off-range values in a response spectrum
- cps_spct: Replace off-range values in a counts per second spectrum
- raw_spct: Replace off-range values in a raw counts spectrum
- generic_spct: Replace off-range values in a generic spectrum
clear.spct

- source_mspct:
- filter_mspct:
- reflector_mspct:
- object_mspct:
- response_mspct:
- cps_mspct:
- raw_mspct:
- generic_mspct:

Note

In the case of object_spct objects, cleaning is done first on the Rfr and Tfr columns and subsequently Afr estimated and if needed half of deviation of Afr from the expected minimum value subtracted from each of Rfr and Tfr.

<table>
<thead>
<tr>
<th>clear.spct</th>
<th>Theoretical spectrum of a clear material</th>
</tr>
</thead>
</table>

Description

A dataset for a hypothetical object with transmittance 1/1 (100%)

Usage

clear.spct

Format

A filter_spct object with 4 rows and 2 variables

Details

- w.length (nm).
- Tfr (0..1)

See Also


Examples

clear.spct
clear_body.spct  Theoretical clear body

Description

A dataset for a hypothetical object with transmittance 1/1 (100%), reflectance 0/1 (0%)

Format

A object_spct object with 4 rows and 3 variables

Details

- w.length (nm)
- Tfr (0..1)
- Rfr (0..1)

See Also


---

clip_wl  Clip head and/or tail of a spectrum

Description

Clip head and tail of a spectrum based on wavelength limits, no interpolation used at range boundaries.

Usage

clip_wl(x, range, ...)

## Default S3 method:
clip_wl(x, range, ...)

## S3 method for class 'generic_spct'
clip_wl(x, range = NULL, ...)

## S3 method for class 'generic_mspct'
clip_wl(x, range = NULL, ...)
## S3 method for class 'waveband'
clip_wl(x, range = NULL, ...)

## S3 method for class 'list'
clip_wl(x, range = NULL, ...)

### Arguments

- **x**: an R object.
- **range**: a numeric vector of length two, or any other object for which function `range()` will return range of wavelengths expressed in nanometres.
- **...**: ignored (possibly used by derived methods).

### Value

A copy of `x`, most frequently of a shorter length, and never longer.

### Methods (by class)

- `default`: Default for generic function
- `generic_spct`: Clip an object of class "generic_spct" or derived.
- `generic_mspct`: Clip an object of class "generic_mspct" or derived.
- `waveband`: Clip an object of class "waveband".
- `list`: Clip a list (of objects of class "waveband").

### Note

The condition tested is \( wl \geq range[1] \& \& wl < (range[2] + 1e-13) \).

### See Also

Other trim functions: `trim_spct()`, `trim_waveband()`, `trim_wl()`

### Examples

```r
clip_wl(sun.spct, range = c(400, 500))
clip_wl(sun.spct, range = c(NA, 500))
clip_wl(sun.spct, range = c(400, NA))
```
collect2mspct  

**Description**

Form a collection of spectra from separate objects in the parent frame of the call.

**Usage**

```r
collect2mspct(
  .list = NULL,
  pattern = "*\.spct$",
  collection.class = NULL,
  ...)
```

**Arguments**

- `.list`  
  list of R objects

- `pattern`  
  character an optional regular expression, ignored if `.list` is not NULL.

- `collection.class`  
  character vector

- `...`  
  additional named arguments passed down to the collection constructor.

**Details**

This is a convenience function that simplifies the creation of collections from existing objects of class `generic_spct` or a derived class. A list of objects can be passed as argument, or a search pattern. If a list is passed, no search is done. If `collection.class` is NULL, then all objects of class `generic_spct` or of a class derived from it are added to the collection. If objects of only one derived class are to be collected this class or that of the matching collection should be passed as argument to `collection.class`. Objects of other R classes are silently discarded, which simplifies the specification of search patterns. By default, i.e., if `collection.class` is NULL, if all the objects collected belong to the same class then the corresponding collection class will be returned, otherwise a `generic_mspct` object with heterogeneous members will be returned. To force the return of a `generic_mspct` even when the collected spectra all belong to the same class, pass `generic_mspct` as argument to `collection.class`. If the argument to `collection.class` is a vector containing two or more class names, only the matching spectra will be collected, and a `generic_mspct` will be returned. The returned object is created with the constructor for the class, and validated.

**Value**

By default a collection of spectra.
See Also

Other experimental utility functions: `drop_user_cols()`, `thin_wl()`, `uncollect2spct()`

Examples

```r
collect2mspct() # returns empty generic_mspct object

sun1.spct <- sun.spct
sun2.spct <- sun.spct
kk.spct <- 10:30 # ignored
collect2mspct()
collect2mspct(collection.class = "generic_mspct")

pet1.spct <- polyester.spct
collect2mspct()
collect2mspct(collection.class = "source_mspct")
collect2mspct(collection.class = "filter_mspct")
collect2mspct(collection.class = "response_mspct")
```

Description

Equivalent RGB color of an object such as a spectrum, wavelength or waveband.

Usage

```r
color_of(x, ...)

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

## S3 method for class 'numeric'
color_of(x, type = "CMF", chroma.type = type, ...)

## S3 method for class 'list'
color_of(x, short.names = TRUE, type = "CMF", chroma.type = type, ...)

## S3 method for class 'waveband'
color_of(x, short.names = TRUE, type = "CMF", chroma.type = type, ...)

## S3 method for class 'source_spct'
color_of(x, type = "CMF", chroma.type = type, ...)

## S3 method for class 'source_mspct'
color_of(x, ..., idx = "spct.idx")
```
colour_of(x, ...)

color(x, ...)

fast_color_of_wl(x, type = "CMF", ...)

fast_color_of_wb(x, type = "CMF", ...)

Arguments

x
an R object.

... ignored (possibly used by derived methods).

type, chroma.type
character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class chroma_spct for any other trichromic visual system.

short.names logical indicating whether to use short or long names for wavebands

idx character Name of the column with the names of the members of the collection of spectra.

Value

A color definition in hexadecimal format as a character string of 7 characters, "#" followed by the red, blue, and green values in hexadecimal (scaled to 0 .. 255). In the case of the specialization for list, a list of such definitions is returned. In the case of a collection of spectra, a data.frame with one column with such definitions and by default an additional column with names of the spectra as index. In case of missing input the returned value is NA.

Methods (by class)

• default: Default method (returns always "black").

• numeric: Method that returns Color definitions corresponding to numeric values representing a wavelengths in nm.

• list: Method that returns Color of elements in a list.

• waveband: Color at midpoint of a waveband object.

• source_spct:

• source_mspct:

Deprecated

Use of color() is deprecated as this wrapper function may be removed in future versions of the package because of name clashes. Use color_of() instead.
**Note**

When `x` is a list but not a waveband, if a method `color_of` is not available for the class of each element of the list, then `color_of.default` will be called.

Function `fast_color_of_wl()` should be used only when high performance is needed. It speeds up performance by rounding the wavelength values in the numeric vector passed as argument to `x` and then retrieves the corresponding pre-computed color definitions if type is either "CMF" or "CC". In other cases it falls-back to calling `color_of.numeric()`. Returned color definitions always have default names irrespective of names of `x`, which is different from the behavior of `color_of()` methods.

Function `fast_color_of_wb()` accepts waveband objects and lists of waveband objects. If all wavebands are narrow, it issues a vectotized call to `fast_color_of_wl()` with a vector of waveband midpoint wavelengths.

**Examples**

```r
wavelengths <- c(300, 420, 500, 600, NA) # nanometres
color_of(wavelengths)
color_of(waveband(c(300, 400)))
color_of(list(blue = waveband(c(400, 480)), red = waveband(c(600, 700))))
color_of(numeric())
color_of(NA_real_)
color_of(sun.spct)
```

**Description**

Coarse-grained comparison of two spectra

Compare two spectra using a specified summary function pre-applied to wavelength intervals.

**Usage**

```r
compare_spct(
  x,
  w.band = 10,
  .summary.fun = NULL,
  ...,
  .comparison.fun = `\'/`,
  returned.value = "spectrum",
  use.hinges = FALSE,
  short.names = TRUE
)
```
Arguments

x A collection of two spectral objects of the same type.
w.band waveband object or a numeric stepsize in nanometres.
.summary.fun function. The summary function to use. It must be a method accepting object x as first argument.
... additional named arguments passed down to .summary.fun.
.comparison.fun function. The comparison function to use.
.returned.value character One of "data.frame", "spectrum", "tagged.spectrum".
.use.hinges logical Flag indicating whether to insert "hinges" into the returned spectrum when tagging it.
.short.names logical Flag indicating whether to use short or long names for wavebands when tagging.

Details

Summaries are computed for each of the wavebands in w.band by applying function .summary.fun separately to each spectrum, after trimming them to the overlapping wavelength region. Next the matching summaries are compared by means of .comparison.fun. Both the summaries and the result of the comparison are returned. Columns containing summary values are named by concatenating the name each member spectrum with the name of the argument passed to .summary.fun.

Tagging is useful for plotting using wavelength based colours, or when names for wavebands are used as annotations. When tagging is requested, the spectrum is passed to method tag with use.hinges and short.names as additional arguments.

Value

A generic_spct, tagged or not with the wavebands, or a data.frame object containing the summary values per waveband for each spectrum and the result of applying the comparison function to these summaries.

Examples

```r
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)))
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
  w.band = NULL)
compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2)),
  w.band = list(waveband(c(640, 650)), waveband(c(720, 740))))

compare_spct(filter_mspct(list(pet = polyester.spct,
  yllw = yellow_gel.spct)),
  w.band = 50,
  .comparison.fun = `<`
)

head(
  compare_spct(source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2))),
```
Ten-degree cone fundamentals

Description

A dataset containing wavelengths at a 1 nm interval (390 nm to 830 nm) and the corresponding response values for a 2 degrees target. Original data from http://www.cvrl.org/ downloaded on 2014-04-29 The variables are as follows:

Usage

cone_fundamentals10.spct

cone_fundamentals10.mspct

Format

A chroma_spct object with 440 rows and 4 variables
An object of class response_mspct (inherits from generic_mspct, list) with 3 rows and 1 columns.

Details

- w.length (nm)
- x
- y
- z

Value

A chroma_spct object.
A response_mspct object containing the same data in three response_spct objects.

Author(s)

CIE
**convertTfrType**

**See Also**


**Examples**

cone_fundamentals10.spct

---

**Description**

Function to set the "Tfr.type" attribute and simultaneously converting the spectral data to correspond to the new type.

**Usage**

```r
convertTfrType(x, Tfr.type = NULL)
```

**Arguments**

- **x**: a filter_spct, object_spct, filter_mspct or object_mspct object.
- **Tfr.type**: character One of "internal" or "total".

**Details**

Internal transmittance uses as reference the light entering the object while total transmittance takes the incident light as reference. The conversion is possible only if reflectance is known. Either as spectral data in an object_spct object, or a filter_spct object that is under the hood an object_spct, or if a fixed reflectance factor applicable to all wavelengths is known.

**Value**

x possibly with the "thickness" field of the "filter.properties" attribute modified

**Note**

if x is not a filter_spct object, x is returned unchanged. If or x does not have the "filter.properties" attribute set and with no missing data, x is returned with Tfr set to NA values.

**See Also**

Other time attribute functions: ```checkTimeUnit(), convertThickness(), convertTimeUnit(), getTimeUnit(), setTimeUnit()```
**convertThickness**

Convert the "thickness" attribute of an existing filter_spct object.

### Description

Function to set the "thickness" attribute and simultaneously converting the spectral data to correspond to the new thickness.

### Usage

```r
convertThickness(x, thickness = NULL)
```

### Arguments

- **x**
  - a filter_spct, object_spct, filter_mspct or object_mspct object.
- **thickness**
  - numeric (m)

### Details

For spectral transmittance at a different thickness to be exactly computed, it needs to be based on internal transmittance. This function will apply `convertTfrType()` to x if needed, but to succeed metadata should be available. Please, see `convertTfrType`.

### Value

x possibly with the "thickness" field of the "filter.properties" attribute modified

### Note

if x is not a filter_spct object, x is returned unchanged. If or x does not have the "filter.properties" attribute set and with no missing data, x is returned with Tfr set to NA values.

### See Also

Other time attribute functions: `checkTimeUnit()`, `convertTfrType()`, `convertTimeUnit()`, `getTimeUnit()`, `setTimeUnit()`
Examples

```r
my.spct <- polyester.spct
filter_properties(my.spct)
convertThickness(my.spct, thickness = 250e-6)
```

---

**convertTimeUnit**  
*Convert the "time.unit" attribute of an existing source_spct object*

**Description**

Function to set the "time.unit" attribute and simultaneously rescaling the spectral data to be expressed using the new time unit as basis of expression. The change is done by reference ('in place').

**Usage**

```r
convertTimeUnit(x, time.unit = NULL, ...)
```

**Arguments**

- `x`: source_spct or response_spct object
- `time.unit`: a character string, either "second", "hour", "day", "exposure" or "none", or a lubridate::duration
- `...`: (currently ignored)

**Value**

`x` possibly with the `time.unit` attribute modified

**Note**

if `x` is not a source_spct or a response_spct object, or `time.unit` is NULL `x` is returned unchanged, if the existing or new `time.unit` cannot be converted to a duration, then the returned spectrum will contain NAs.

**See Also**

Other time attribute functions: `checkTimeUnit()`, `convertTfrType()`, `convertThickness()`, `getTimeUnit()`, `setTimeUnit()`

**Examples**

```r
my.spct <- sun.spct
my.spct
convertTimeUnit(my.spct, "day")
my.spct
```
convolve_each

Convolve function for collections of spectra

Description

Convolve function for collections of spectra which applies an operation on all the individual members of the collection(s) of spectra.

Usage

\[
\text{convolve_each}(e1, e2, \text{oper} = \times, \text{sep} = ",", ...)\]

Arguments

- **e1**: an object of class `generic_mspct` or `generic_scpt` or numeric
- **e2**: an object of class `generic_mspct` or `generic_scpt` or numeric
- **oper**: function, usually but not necessarily an operator with two arguments.
- **sep**: character Used when pasting the names of members of `e1` and `e2` to form the names of members of the returned collection of spectra.
- **...**: additional arguments passed to `oper` if present.

Note

At least one of `e1` and `e2` must be a `generic_mspct` object or derived.

See Also

Other math operators and functions: `MathFun`, `^\.generic_spct()`, `div\.generic_spct`, `log()`, `minus\.generic_spct`, `mod\.generic_spct`, `plus\.generic_spct`, `round()`, `sign()`, `slash\.generic_spct`, `times\.generic_spct`

---

copy_attributes

Copy attributes

Description

Copy attributes from `x` to `y`. Methods defined for spectral and waveband objects of classes from package 'photobiology'.
Usage

copy_attributes(x, y, which, ...)

## Default S3 method:
copy_attributes(x, y, which = NULL, ...)

## S3 method for class 'generic_spct'
copy_attributes(x, y, which = NULL, which.not = NULL, copy.class = FALSE, ...)

## S3 method for class 'generic_mspct'
copy_attributes(x, y, which = NULL, which.not = NULL, copy.class = FALSE, ...)

## S3 method for class 'waveband'
copy_attributes(x, y, which = NULL, ...)

Arguments

x, y          R objects
which         character Names of attributes to copy, if NULL all those relevant according to
              the class of x is used as default,
...           not used
which.not     character Names of attributes not to be copied. The names passed here are re-
              moved from the list for which, which is most useful when we want to modify
              the default.
copy.class    logical If TRUE class attributes are also copied.

Value

A copy of y with additional attributes set.

Methods (by class)

- default: Default for generic function
- generic_spct:
- generic_mspct:
- waveband:

---

cps2irrad         Conversion from counts per second to physical quantities

Description

Conversion of spectral data expressed as cps into irradiance, transmittance or reflectance.
Usage

cps2irrad(x.sample, pre.fun = NULL, missing.pixels = numeric(0), ...)  
cps2Rfr(x.sample, x.white, x.black = NULL, dyn.range = NULL)  
cps2Tfr(x.sample, x.clear, x.opaque = NULL, dyn.range = NULL)

Arguments

x.sample, x.clear, x.opaque, x.white, x.black  
cps_spct objects.
pre.fun  
function A function applied to x.sample before conversion.
missing.pixels  
integer Index to positions in the detector array or scan missing in x.sample but present in the embedded calibration data. (Use only for emergency recovery of incomplete data!!)
...  
Additional arguments passed to pre.fun.
dyn.range  
numeric The effective dynamic range of the instrument, if NULL it is automatically set based on integration time bracketing.

Value

A source_spct, filter_spct or reflector_spct object containing the spectral values expressed in physical units.

Note

In contrast to other classes defined in package 'photobiology', class "cps_spct" can have more than one column of cps counts in cases where the intention is to merge these values as part of the processing at the time the calibration is applied. However, being these functions the final step in the conversion to physical units, they accept as input only objects with a single "cps" column, as merging is expected to have been already done.

---

D2.UV586

Data for typical calibration lamps

Description

A dataset containing fitted constants to be used as input for function D2_spectrum.

Format

A polynom::polynomial object with 6 constants.

Details

An object of class polynom::polynomial.
**Author(s)**

Lasse Ylianttila (data)

---

**D2.UV653**  
*Data for typical calibration lamps*

**Description**

A dataset containing fitted constants to be used as input for function `D2_spectrum`.

**Format**

A `polynom::polynomial` object with 6 constants.

**Details**

An object of class `polynom::polynomial`.

**Author(s)**

Lasse Ylianttila (data)

---

**D2.UV654**  
*Data for typical calibration lamps*

**Description**

A dataset containing fitted constants to be used as input for function `D2_spectrum`.

**Format**

A `polynom::polynomial` object with 6 constants.

**Details**

An object of class `polynom::polynomial`.

**Author(s)**

Lasse Ylianttila (data)
**D2_spectrum**

*Calculate deuterium lamp output spectrum from fitted constants*

**Description**

Calculate values by means of an nth degree polynomial from user-supplied constants (for example from a lamp calibration certificate).

**Usage**

```r
D2_spectrum(w.length, k = photobiology::D2.UV653, fill = NA_real_)
```

**Arguments**

- `w.length`: numeric vector of wavelengths (nm) for output
- `k`: a polynom:polynomial object with n constants for the polynomial
- `fill`: if NA, no extrapolation is done, and NA is returned for wavelengths outside the range 190 nm to 450 nm. If NULL then the tails are deleted. If 0 then the tails are set to zero, etc. NA is default.

**Value**

A dataframe with four numeric vectors with wavelength values (`w.length`), energy and photon irradiance (`s.e.irrad`, `s.q.irrad`) depending on the argument passed to `unit.out` (`s.irrad`).

**Note**

This function is valid for wavelengths in the range 180 nm to 495 nm, for wavelengths outside this range NAs are returned.

**Examples**

```r
D2_spectrum(200)
D2_spectrum(170:220)
```

---

**D65.illuminant.spct**

*CIE D65 illuminant data*

**Description**

A dataset containing wavelengths at a 5 nm interval (300 nm to 830 nm) and the corresponding spectral energy irradiance normalized to 1 at 560 nm. Spectrum approximates the midday solar spectrum at middle latitude as 'corresponds' to the white point of a black body a 6504 K. Original data from [http://files.cie.co.at/204.xls](http://files.cie.co.at/204.xls) downloaded on 2014-07-25 The variables are as follows:
day_night

Usage

D65.illuminant.spct

Format

A source spectrum with 107 rows and 2 variables

Details

- w.length (nm)
- s.e.irrad (rel. units)

Author(s)

CIE

See Also


Examples

D65.illuminant.spct

day_night

Times for sun positions

Description

Functions for calculating the timing of solar positions, given geographical coordinates and dates. They can be also used to find the time for an arbitrary solar elevation between 90 and -90 degrees by supplying "twilight" angle(s) as argument.

Usage

day_night(
    date = lubridate::now(tzone = "UTC"),
    tz = lubridate::tz(date),
    geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
    twilight = "none",
    unit.out = "hours"
)
day_night

day_night_fast(date, tz, geocode, twilight, unit.out)

noon_time(
    date = lubridate::today(),
    tz = lubridate::tz(date),
    geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
    twilight = "none",
    unit.out = "datetime"
)

sunrise_time(
    date = lubridate::today(),
    tz = lubridate::tz(date),
    geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
    twilight = "sunlight",
    unit.out = "datetime"
)

sunset_time(
    date = lubridate::today(),
    tz = lubridate::tz(date),
    geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
    twilight = "sunlight",
    unit.out = "datetime"
)

day_length(
    date = lubridate::now(),
    tz = "UTC",
    geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
    twilight = "sunlight",
    unit.out = "hours"
)

night_length(
    date = lubridate::now(),
    tz = "UTC",
    geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
    twilight = "sunlight",
    unit.out = "hours"
)

Arguments

date "vector" of POSIXct times or Date objects, any valid TZ is allowed, default is current date at Greenwich.

tz character vector indicating time zone to be used in output.
`geocode` data frame with one or more rows and variables lon and lat as numeric values (degrees). If present, address will be copied to the output.

`twilight` character string, one of "none", "rim", "refraction", "sunlight", "civil", "nautical", "astronomical", or a numeric vector of length one, or two, giving solar elevation angle(s) in degrees (negative if below the horizon).

`unit.out` character string, One of "datetime", "day", "hour", "minute", or "second".

### Details

Twilight names are interpreted as follows. "none": solar elevation = 0 degrees. "rim": upper rim of solar disk at the horizon or solar elevation = -0.53 / 2. "refraction": solar elevation = 0 degrees + refraction correction. "sunlight": upper rim of solar disk corrected for refraction, which is close to the value used by the online NOAA Solar Calculator. "civil": -6 degrees, "naval": -12 degrees, and "astronomical": -18 degrees. Unit names for output are as follows: "day", "hours", "minutes" and "seconds" times for sunrise and sunset are returned as times-of-day since midnight expressed in the chosen unit. "date" or "datetime" return the same times as datetime objects with TZ set (this is much slower than "hours"). Day length and night length are returned as numeric values expressed in hours when "datetime" is passed as argument to unit.out. If twilight is a numeric vector of length two, the element with index 1 is used for sunrise and that with index 2 for sunset.

### Value

A tibble with variables day, tz, twilight.rise, twilight.set, longitude, latitude, address, sunrise, noon, sunset, daylength, nightlength or the corresponding individual vectors.

`noon_time`, `sunrise_time` and `sunset_time` return a vector of POSIXct times

day_length and night_length return numeric a vector giving the length in hours

### Warning

Be aware that R’s Date class does not save time zone metadata. This can lead to ambiguities in the current implementation based on time instants. The argument passed to date should be of class POSIXct, in other words an instant in time, from which the correct date will be computed based on the tz argument.

### Note

This function is an implementation of Meeus equations as used in NOAAs on-line web calculator, which are very precise and valid for a very broad range of dates. For sunrise and sunset the times are affected by refraction in the atmosphere, which does in turn depend on weather conditions. The effect of refraction on the apparent position of the sun is only an estimate based on "typical" conditions. The more tangential to the horizon is the path of the sun, the larger the effect of refraction is on the times of visual occlusion of the sun behind the horizon—i.e. the largest timing errors occur at high latitudes. The computation is not defined for latitudes 90 and -90 degrees, i.e. at the poles. There exists a different R implementation of the same algorithms called "AstroCalcPureR" available as function astrocalc4r in package 'fishmethods'. Although the equations used are almost all the same, the function signatures and which values are returned differ. In particular, the present implementation splits the calculation into two separate functions, one returning angles at given instants in time, and a separate one returning the timing of events for given dates.
In the current implementation functions `sunrise_time`, `noon_time`, `sunset_time` and `day_length` are wrappers on `day_night`, so if more than one quantity is needed it is preferable to directly call `day_night` as it will be faster.

`night_length` returns the length of night-time conditions in one day (00:00:00 to 23:59:59), rather than the length of the night between two consecutive days.

References


A different implementation is available at https://apps-nefsc.fisheries.noaa.gov/AstroCalc4R/ and in R package ‘fishmethods’. In ‘fishmethods’ (= 1.11-0) there is a bug in function astrocalc4r() that affects sunrise and sunset times.

An interactive web page using the same algorithms is available at https://www.esrl.noaa.gov/gmd/grad/solcalc/. There are small differences in the returned times compared to our function that seem to be related to the estimation of atmospheric refraction (about 0.1 degrees).

See Also

`sun_angles`.

Other astronomy related functions: `format.solar_time()`, `format.tod_time()`, `is.solar_time()`, `print.solar_time()`, `print.tod_time()`, `solar_time()`, `sun_angles()`

Examples

library(lubridate)
my.geocode <- data.frame(lat = 60, lon = 25)
day_night(ymd("2015-05-30"), geocode = my.geocode)
day_night(ymd("2015-05-30") + days(1:10), geocode = my.geocode, twilight = "civil")
sunrise_time(ymd("2015-05-30"), geocode = my.geocode)
noon_time(ymd("2015-05-30"), geocode = my.geocode)
sunset_time(ymd("2015-05-30"), geocode = my.geocode)
day_length(ymd("2015-05-30"), geocode = my.geocode)
day_length(ymd("2015-05-30"), geocode = my.geocode, unit.out = "day")

---

Defunct functions and methods

Description

Functions listed here have been removed or deleted, and temporarily replaced by stubs that report this when they are called.
Usage

f_mspct(...)
mutate_mspct(...)
calc_filter_multipliers(...)  
T2T(...)  
getAfrType(...)  
setAfrType(...)

Arguments

... ignored

Note

Function f_mspct() has been renamed msdply().  
Function mutate_mspct() has been renamed msmsply().  
Function calc_filter_multipliers() has been removed.  
Function calc_filter_multipliers() has been removed.  
Method getAfrType() has been removed.  
Method setAfrType() has been removed.

| despike              | Remove spikes from spectrum |

Description

Function that returns an R object with observations corresponding to spikes replaced by values computed from neighboring pixels. Spikes are values in spectra that are unusually high compared to neighbors. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode array detectors.

Usage

despike(x, z.threshold, max.spike.width, window.width, method, na.rm, ...)

## Default S3 method:
despike(
    x,
    z.threshold = NA,
max.spike.width = NA,
window.width = NA,
method = "run.mean",
na.rm = FALSE,
...
)

## S3 method for class 'numeric'
despike(
  x,
z.threshold = 9,
max.spike.width = 8,
window.width = 11,
method = "run.mean",
na.rm = FALSE,
...
)

## S3 method for class 'data.frame'
despike(
  x,
z.threshold = 9,
max.spike.width = 8,
window.width = 11,
method = "run.mean",
na.rm = FALSE,
...,y.var.name = NULL,
var.name = y.var.name
)

## S3 method for class 'generic_spct'
despike(
  x,
z.threshold = 9,
max.spike.width = 8,
window.width = 11,
method = "run.mean",
na.rm = FALSE,
...,y.var.name = NULL,
var.name = y.var.name
)

## S3 method for class 'source_spct'
despike(
  x,
z.threshold = 9,
max.spike.width = 8,
window.width = 11,
method = "run.mean",
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...)

## S3 method for class 'response_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  filter_qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_spct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
## S3 method for class 'raw_spct'

despike(  
  x,  
  z.threshold = 9,  
  max.spike.width = 8,  
  window.width = 11,  
  method = "run.mean",  
  na.rm = FALSE,  
  ...
)

## S3 method for class 'generic_mspct'

despike(  
  x,  
  z.threshold = 9,  
  max.spike.width = 8,  
  window.width = 11,  
  method = "run.mean",  
  na.rm = FALSE,  
  ...,  
  y.var.name = NULL,  
  var.name = y.var.name,  
  .parallel = FALSE,  
  .paropts = NULL
)

## S3 method for class 'source_mspct'

despike(  
  x,  
  z.threshold = 9,  
  max.spike.width = 8,  
  window.width = 11,  
  method = "run.mean",  
  na.rm = FALSE,  
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),  
  ...,  
  .parallel = FALSE,  
  .paropts = NULL
)

## S3 method for class 'response_mspct'

despike(  
  ...
)
despike

x,
z.threshold = 9,
max.spike.width = 8,
window.width = 11,
method = "run.mean",
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...,
.parallel = FALSE,
.paropts = NULL

## S3 method for class 'filter_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
  ...,
 .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'reflector_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
 .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'cps_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
## S3 method for class 'raw_mspct'
despike(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  window.width = 11,
  method = "run.mean",
  na.rm = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

### Arguments

- **x**: an R object
- **z.threshold**: numeric. Modified Z values larger than `z.threshold` are considered to correspond to spikes.
- **max.spike.width**: integer. Wider regions with high Z values are not detected as spikes.
- **window.width**: integer. The full width of the window used for the running mean used as replacement.
- **method**: character. The name of the method: "run.mean" is running mean as described in Whitaker and Hayes (2018); "adj.mean" is mean of adjacent neighbors (isolated bad pixels only).
- **na.rm**: logical. indicating whether NA values should be treated as spikes and replaced.
- **...**: Arguments passed by name to `find_spikes()`.
- **var.name, y.var.name**: character. Names of columns where to look for spikes to remove.
- **unit.out**: character. One of "energy" or "photon".
- **filter.qty**: character. One of "transmittance" or "absorbance".
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach.
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

### Value

The function returns `x` with rows corresponding to spikes replaced by a local average of adjacent neighbors outside the spike.
Methods (by class)

- default: Default returning always NA.
- numeric: Default function usable on numeric vectors.
- data.frame: Method for "data.frame" objects.
- generic_spct: Method for "generic_spct" objects.
- source_spct: Method for "source_spct" objects.
- filter_spct: Method for "filter_spct" objects.
- reflector_spct: Method for "reflector_spct" objects.
- cps_spct: Method for "cps_spct" objects.
- raw_spct: Method for "raw_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.
- source_mspct: Method for "source_mspct" objects.
- response_mspct: Method for "response_mspct" objects.
- filter_mspct: Method for "filter_mspct" objects.
- reflector_mspct: Method for "reflector_mspct" objects.
- cps_mspct: Method for "cps_mspct" objects.
- raw_mspct: Method for "raw_mspct" objects.

Note

Current algorithm misidentifies steep smooth slopes as spikes, so manual inspection is needed together with adjustment by trial and error of a suitable argument value for z.threshold.

See Also

See the documentation for `find_spikes` and `replace_bad_pixs` for details of the algorithm and implementation.

Examples

```r
white_led.raw_spct[120:125, ]

# find and replace spike at 245.93 nm
despike(white_led.raw_spct,
    z.threshold = 10,
    window.width = 25)[120:125, ]
```
Description

Diffraction of optical radiation passing through a single slit can be computed with function `diffraction_single_slit()`, which implements Fraunhofer’s equation. Diffraction plus interference for a pair of slits can be computed with `diffraction_double_slit()`.

Usage

```r
diffraction_single_slit(w.length, slit.width, angle)
diffraction_double_slit(w.length, slit.width, slit.distance, angle)
```

Arguments

- `w.length` numeric: Wavelength (nm).
- `slit.width` numeric: Width of the slit (m).
- `angle` numeric vector: Angle (radians).
- `slit.distance` numeric: Distance between the centres of the two slits (m).

Value

A numeric vector of the same length as `angle`, containing relative intensities.

Examples

```r
diffraction_single_slit(w.length = 550,
                        slit.width = 1e-5,
                        angle = 0)

# use odd number for length.out so that 0 is in the sequence
angles <- pi * seq(from = -1/2, to = 1/2, length.out = 501)
plot(angles,
     diffraction_single_slit(w.length = 550, # 550 nm
                             slit.width = 6e-6, # 6 um
                             angle = angles),
     type = "l",
     ylab = "Relative irradiance (/1)",
     xlab = "Angle (radian)"
)

plot(angles,
     diffraction_double_slit(w.length = 550, # 550 nm
                             slit.width = 6e-6, # 6 um
                             slit.distance = 18e-6, # 18 um
                             angle = angles),
     type = "l",
     ylab = "Relative irradiance (/1)",
     xlab = "Angle (radian)"
)
```
angle = angles),
type = "l",
ylab = "Relative irradiance (1)",
xlab = "Angle (radian)"

---

**dim.generic_mspct**  
*Dimensions of an Object*

---

**Description**

Retrieve or set the dimension of an object.

**Usage**

```r
## S3 method for class 'generic_mspct'
dim(x)

## S3 replacement method for class 'generic_mspct'
dim(x) <- value
```

**Arguments**

- `x`: A `generic_mspct` object or of a derived class.
- `value`: Either NULL or a numeric vector, which is coerced to integer (by truncation).

**Value**

Either NULL or a numeric vector, which is coerced to integer (by truncation).

---

**div-.generic_spct**  
*Arithmetic Operators*

---

**Description**

Integer-division operator for generic spectra.

**Usage**

```r
## S3 method for class 'generic_spct'
e1 %/% e2
```

**Arguments**

- `e1`: an object of class "generic_spct"
- `e2`: an object of class "generic_spct"
Divide two spectra, even if the wavelengths values differ

**Description**

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are operated upon.

**Usage**

```r
div_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
```

**Arguments**

- `w.length1`: numeric vector of wavelength (nm) of denominator.
- `w.length2`: numeric vector of wavelength (nm) of divisor.
- `s.irrad1`: a numeric vector of spectral values of denominator.
- `s.irrad2`: a numeric vector of spectral values of divisor.
- `trim`: a character string with value "union" or "intersection".
- `na.rm`: a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

**Details**

If `trim="union"` spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If `trim="intersection"` then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If `w.length2==NULL`, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

---

**See Also**

Other math operators and functions: `MathFun`, `^\.generic_spct\()`, `convolve_each\()`, `log\()`, `minus-\.generic_spct`, `mod-\.generic_spct`, `plus-\.generic_spct`, `round\()`, `sign\()`, `slash-\.generic_spct`, `times-\.generic_spct`
Value

a dataframe with two numeric variables.

- **w.length** A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique values, sorted in ascending order.
- **s.irrad** A numeric vector with the sum of the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```r
head(sun.data)
one.data <- with(sun.data, div_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(one.data)
tail(one.data)
```

---

drop_user_cols | Drop user columns

Description

Remove from spectral object additional columns that are user defined.

Usage

```r
drop_user_cols(x, keep.also, ...)
```

---

```r
## Default S3 method:
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'generic_spct'
drop_user_cols(x, keep.also, ...)

## S3 method for class 'source_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'response_spct'
drop_user_cols(x, keep.also = NULL, ...)
```
## S3 method for class 'object_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'filter_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'reflector_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'chroma_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'calibration_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'cps_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'raw_spct'
drop_user_cols(x, keep.also = NULL, ...)

## S3 method for class 'generic_mspct'
drop_user_cols(x, keep.also = NULL, ...)

### Arguments

- **x**: An R object
- **keep.also**: character Additional columns to preserve.
- **...**: needed to allow derivation.

### Value

A copy of x possibly with some columns removed.

### Methods (by class)

- default:
  - generic_spct:
  - source_spct:
  - response_spct:
  - object_spct:
  - filter_spct:
  - reflector_spct:
  - chroma_spct:
  - calibration_spct:
• cps_spct:
• raw_spct:
• generic_mspct:

See Also
Other experimental utility functions: `collect2mspct()`, `thin wl()`, `uncollect2spct()`

---

e2q

Convert energy-based quantities into photon-based quantities.

Description
Function that converts spectral energy irradiance into spectral photon irradiance (molar).

Usage
```r
e2q(x, action, byref, ...)  
## Default S3 method:  
e2q(x, action = "add", byref = FALSE, ...)
```

```r  
## S3 method for class 'source_spct'
e2q(x, action = "add", byref = FALSE, ...)
```

```r  
## S3 method for class 'response_spct'
e2q(x, action = "add", byref = FALSE, ...)
```

```r  
## S3 method for class 'source_mspct'
e2q(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)
```

```r  
## S3 method for class 'response_mspct'
e2q(x, action = "add", byref = FALSE, ..., .parallel = FALSE, .paropts = NULL)
```

Arguments
- `x` - an R object
- `action` - a character string
- `byref` - logical indicating if new object will be created by reference or by copy of `x`
- `...` - not used in current version
- `.parallel` - if TRUE, apply function in parallel, using parallel backend provided by foreach
- `.paropts` - a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the `.export` and `.packages` arguments to supply them so that all cluster nodes have the correct environment set up for computing.
Methods (by class)

- default: Default method
- source_spct: Method for spectral irradiance
- response_spct: Method for spectral responsiveness
- source_mspct: Method for collections of (light) source spectra
- response_mspct: Method for collections of response spectra

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `q2e()`

---

**e2qmol_multipliers** Calculate energy to quantum (mol) multipliers

---

**Description**

Multipliers as a function of wavelength, for converting from energy to photon (quantum) molar units.

**Usage**

```r
e2qmol_multipliers(w.length)
```

**Arguments**

- `w.length`: numeric Vector of wavelengths (nm)

**Value**

A numeric vector of multipliers

**See Also**

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2quantum_multipliers()`, `e2q()`, `q2e()`

**Examples**

```r
with(sun.data, e2qmol_multipliers(w.length))
```
**e2quantum_multipliers**  *Calculate energy to quantum multipliers*

**Description**

Gives multipliers as a function of wavelength, for converting from energy to photon (quantum) units (number of photons as default, or moles of photons).

**Usage**

```r
e2quantum_multipliers(w.length, molar = FALSE)
```

**Arguments**

- `w.length` numeric Vector of wavelengths (nm)
- `molar` logical Flag indicating whether output should be in moles or numbers

**Value**

A numeric vector of multipliers

**See Also**

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2Afr()`, `T2A()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2q()`, `q2e()`

**Examples**

```r
with(sun.data, e2quantum_multipliers(w.length))
with(sun.data, e2quantum_multipliers(w.length, molar = TRUE))
```

---

**enable_check_spct**  *Enable or disable checks*

**Description**

Choose between protection against errors or faster performance by enabling (the default) or disabling data-consistency and sanity checks.

**Usage**

```r
enable_check_spct()
disable_check_spct()
set_check_spct(x)
```
energy_as_default

Arguments

x  logical Flag to enable (TRUE), disable (FALSE) or unset (NULL) option.

Value

The previous value of the option, which can be passed as argument to function set_check_spct() to restore the previous state of the option.

See Also

Other data validity check functions: check_spct(), check_spectrum(), check_w.length()

energy_as_default

Set spectral-data options

Description

Set spectral-data related options easily.

Usage

energy_as_default()
photon_as_default()
quantum_as_default()
Tfr_as_default()
Afr_as_default()
A_as_default()
unset_radiation_unit_default()
unset_filter_qty_default()
unset_user_defaults()

Value

Previous value of the modified option.
energy_irradiance

Calculate (energy) irradiance from spectral irradiance

Description

Energy irradiance for a waveband from a radiation spectrum, optionally applying a "biological spectral weighting function" or BSWF.

Usage

```r
energy_irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

- `w.length`: numeric vector of wavelength (nm).
- `s.irrad`: numeric vector of spectral irradiances, by default as energy (W m\(^{-2}\) nm\(^{-1}\)).
- `w.band`: waveband.
- `unit.in`: a character. Allowed values "photon" or "energy", default is "energy".
- `check.spectrum`: logical Flag indicating whether to sanity check input data, default is TRUE.
- `use.cached.mult`: logical Flag indicating whether multiplier values should be cached between calls.
- `use.hinges`: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wave-bands.

Value

A single numeric value with no change in scale factor: [W m\(^{-2}\) nm\(^{-1}\)] -> [W m\(^{-2}\)].

See Also

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`
Examples

```r
with(sun.data, energy_irradiance(w.length, s.e.irrad))
with(sun.data, energy_irradiance(w.length, s.e.irrad, new_waveband(400,700)))
```

---

**energy_ratio**  
*Energy:energy ratio*

### Description

Energy irradiance ratio between two wavebands for a radiation spectrum.

### Usage

```r
energy_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = NULL
)
```

### Arguments

- `w.length` numeric vector of wavelengths (nm).
- `s.irrad` numeric vector of spectral (energy) irradiances (W m^-2 nm^-1).
- `w.band.num` waveband object used to compute the numerator of the ratio.
- `w.band.denom` waveband object used to compute the denominator of the ratio.
- `unit.in` character Allowed values "energy", and "photon", or its alias "quantum".
- `check.spectrum` logical Flag indicating whether to sanity check input data, default is TRUE.
- `use.cached.mult` logical Flag indicating whether multiplier values should be cached between calls.
- `use.hinges` logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

### Value

a single numeric value giving the unitless ratio.
The default for both `w.band` parameters is a waveband covering the whole range of `w.length`.

### Description

This function returns the energy to mole of photons ratio for each waveband and a light source spectrum.

### Usage

```r
eq_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)  
```

### Examples

```r
with(sun.data,  
    energy_ratio(w.length, s.e.irrad, new_waveband(400,500), new_waveband(400,700)))
```
eq_ratio

eq_ratio(
spt,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[e:q]", ""),
  ...

  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paroits = NULL
)

Arguments

spct    source_spct.
w.band  waveband or list of waveband objects.
scale.factor  numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
wb.trim  logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
use.cached.mult  logical Flag telling whether multiplier values should be cached between calls.
use.hinges  logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
...  other arguments (possibly used by derived methods).
naming  character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
name.tag  character Used to tag the name of the returned values.
attr2tb  character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
idx  character Name of the column with the names of the members of the collection of spectra.
.parallel  if TRUE, apply function in parallel, using parallel backend provided by foreach
.paroits  a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.
Value

Computed values are ratios between energy irradiance and photon irradiance for a given waveband. A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used, with "e:q" prepended. Units [J mol⁻¹].

Methods (by class)

• default: Default for generic function
• source_spct: Method for source_spct objects
• source_mspct: Calculates energy:photon from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other photon and energy ratio functions: e_ratio(), q_ratio(), qe_ratio()

Examples

eq_ratio(sun.spct, new_waveband(400,700))

Extract

Extract or replace parts of a spectrum

Description

Just like extraction and replacement with indexes in base R, but preserving the special attributes used in spectral classes and checking for validity of remaining spectral data.
Usage

```r
## S3 method for class 'generic_spct'
x[i, j, drop = NULL]

## S3 method for class 'raw_spct'
x[i, j, drop = NULL]

## S3 method for class 'cps_spct'
x[i, j, drop = NULL]

## S3 method for class 'source_spct'
x[i, j, drop = NULL]

## S3 method for class 'response_spct'
x[i, j, drop = NULL]

## S3 method for class 'filter_spct'
x[i, j, drop = NULL]

## S3 method for class 'reflector_spct'
x[i, j, drop = NULL]

## S3 method for class 'object_spct'
x[i, j, drop = NULL]

## S3 method for class 'chroma_spct'
x[i, j, drop = NULL]

## S3 replacement method for class 'generic_spct'
x[i, j] <- value

## S3 replacement method for class 'generic_spct'
x$name <- value
```

Arguments

- `x`: spectral object from which to extract element(s) or in which to replace element(s)
- `i`: index for rows,
- `j`: index for columns, specifying elements to extract or replace. Indices are numeric or character vectors or empty (missing) or NULL. Please, see `Extract` for more details.
- `drop`: logical. If TRUE the result is coerced to the lowest possible dimension. The default is FALSE unless the result is a single column.
- `value`: A suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If NULL, deletes the column if a single column is selected.
name  A literal character string or a name (possibly backtick quoted). For extraction, this is normally (see under 'Environments') partially matched to the names of the object.

Details
These methods are just wrappers on the method for data.frame objects which copy the additional attributes used by these classes, and validate the extracted object as a spectral object. When drop is TRUE and the returned object has only one column, then a vector is returned. If the extracted columns are more than one but do not include w.length, a data frame is returned instead of a spectral object.

Value
An object of the same class as x but containing only the subset of rows and columns that are selected. See details for special cases.

Note
If any argument is passed to j, even TRUE, some metadata attributes are removed from the returned object. This is how the extraction operator works with data.frames in R. For the time being we retain this behaviour for spectra, but it may change in the future.

See Also
subset and trim_spct

Examples
sun.spct[sun.spct[["w.length"]]> 400, ]
subset(sun.spct, w.length > 400)

tmp.spct <- sun.spct
tmp.spct[tmp.spct[["s.e.irrad"]]< 1e-5 , "s.e.irrad"] <- 0
e2q(tmp.spct[ , c("w.length", "s.e.irrad")]) # restore data consistency!

Extract_mspct

Extract or replace members of a collection of spectra

Description
Just like extraction and replacement with indexes for base R lists, but preserving the special attributes used in spectral classes.
### e_fluence

**Usage**

```r
## S3 method for class 'generic_mspct'
x[i, drop = NULL]

## S3 replacement method for class 'generic_mspct'
x[i] <- value

## S3 replacement method for class 'generic_mspct'
x$name <- value

## S3 replacement method for class 'generic_mspct'
x[[name]] <- value
```

**Arguments**

- `x`: Collection of spectra object from which to extract member(s) or in which to replace member(s).
- `i`: Index specifying elements to extract or replace. Indices are numeric or character vectors. Please, see `Extract` for more details.
- `drop`: If TRUE the result is coerced to the lowest possible dimension (see the examples). This only works for extracting elements, not for the replacement.
- `value`: A suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If NULL, deletes the column if a single column is selected.
- `name`: A literal character string or a name (possibly backtick quoted). For extraction, this is normally (see under 'Environments') partially matched to the names of the object.

**Details**

This method is a wrapper on base R’s extract method for lists that sets additional attributes used by these classes.

**Value**

An object of the same class as `x` but containing only the subset of members that are selected.

---

<table>
<thead>
<tr>
<th>e_fluence</th>
<th>Energy fluence</th>
</tr>
</thead>
</table>

**Description**

Energy fluence for one or more wavebands of a light source spectrum and a duration of the exposure.
Usage

e_fluence(
  spct,
  w.band,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
e_fluence(
  spct,
  w.band,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
e_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult =getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
)

## S3 method for class 'source_mspct'
e_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult =getOption("photobiology.use.cached.mult", default = FALSE),
use.hinges = NULL,
allow.scaled = FALSE,
...
attr2tb = NULL,
idx = "spct.idx",
.parallel = FALSE,
.parallelopts = NULL
)

Arguments

spct an R object
w.band a list of waveband objects or a waveband object
exposure.time lubridate::duration object.
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
use.cached.mult logical indicating whether multiplier values should be cached between calls
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error
... other arguments (possibly ignored)
naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
idx character Name of the column with the names of the members of the collection of spectra.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The exposure.time is copied to the output as an attribute. Units are as follows: (J) joules per exposure.
Methods (by class)

- default: Default for generic function
- source_spct: Calculate energy fluence from a source_spct object and the duration of the exposure.
- source_mspct: Calculates energy fluence from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other irradiance functions: e_irrad(), fluence(), irrad(), q_fluence(), q_irrad()

Examples

library(lubridate)

e_fluence(sun.spct, waveband(c(400,700)),
exposure.time = lubridate::duration(3, "minutes") )

e_irrad
Energy irradiance

Description

Energy irradiance for one or more wavebands of a light source spectrum.

Usage

e_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)
### Default S3 method:
```
e_irrad(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...)
```

### S3 method for class 'source_spct'
```
e_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...
)
```

### S3 method for class 'source_mspct'
```
e_irrad(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
  naming = "default",
  ...
attr2tb = NULL,
idx = "spct.idx",
.parallel = FALSE,
.paropts = NULL)
```
Arguments

spect an R object.
w.band a list of waveband objects or a waveband object.
quantity character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
time.unit character or lubridate::duration object.
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
use.cached.mult logical indicating whether multiplier values should be cached between calls.
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
... other arguments (possibly used by derived methods).
naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
idx character Name of the column with the names of the members of the collection of spectra.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If units are absolute and time.unit is second, [W m-2 nm-1] -> [W m-2] If time.unit is day, [J d-1 m-2 nm-1] -> [J m-2]; if units are relative, fraction of one or percent.
e_ratio

Methods (by class)

- default: Default for generic function
- source_spct: Calculates energy irradiance from a source_spct object.
- source_mspct: Calculates energy irradiance from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other irradiance functions: e_fluence(), fluence(), irrad(), q_fluence(), q_irrad()

Examples

```r
e_irrad(sun.spct, waveband(c(400,700)))
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3))
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
    quantity = "total")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
    quantity = "average")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
    quantity = "relative")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
    quantity = "relative.pc")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
    quantity = "contribution")
e_irrad(sun.spct, split_bands(c(400,700), length.out = 3),
    quantity = "contribution.pc")
```

---

### e_ratio

**Energy:energy ratio**

**Description**

This function returns the photon ratio for a given pair of wavebands of a light source spectrum.
Usage

e_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## Default S3 method:
e_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)

## S3 method for class 'source_spct'
e_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[e:e]", ""),
  ...
)

## S3 method for class 'source_mspct'
e_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[e:e]", ""),
name.tag = ifelse(naming != "none", "[e:e]", ""),
...,
attr2tb = NULL,
idx = "spect.idx",
.parallel = FALSE,
.paropts = NULL
)

Arguments

spct      source_spct
w.band.num waveband object or a list of waveband objects used to compute the numerator(s)
of the ratio(s).
w.band.denom waveband object or a list of waveband objects used to compute the denominato-
(s) of the ratio(s).
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier
applied to returned values.
wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if
FALSE, they are discarded
use.cached.mult logical Flag telling whether multiplier values should be cached between calls.
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before
integration so as to reduce interpolation errors at the boundaries of the wave-
bands.
... other arguments (possibly used by derived methods).
naming character one of "long", "default", "short" or "none". Used to select the type of
names to assign to returned value.
name.tag character Used to tag the name of the returned values.
attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to
formal parameter col.names.
idx character Name of the column with the names of the members of the collection
of spectra.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach.
.paropts a list of additional options passed into the foreach function when parallel compu-
tation is enabled. This is important if (for example) your code relies on external
data or packages: use the .export and .packages arguments to supply them so
that all cluster nodes have the correct environment set up for computing.

Value

In the case of methods for individual spectra, a numeric vector of adimensional values giving a
energy ratio between integrated energy irradiances for pairs of wavebands, with name attribute set
to the name of the wavebands unless a named list of wavebands is supplied in which case the names
of the list elements are used, with "(e:e)" appended. A data.frame in the case of collections of
spectra, containing one column for each ratio definition, an index column with the names of the
spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Ratio definitions are "assembled" from the arguments passed to w.band.num and w.band.denom. If both arguments are of equal length, then the wavebands are paired to obtain as many ratios as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- default: Default for generic function
- source_spct: Method for source_spct objects
- source_mspct: Calculates energy:energy ratio from a source_mspct object.

Note

Recycling for wavebands takes place when the number of denominator and denominator wavebands differ. The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other photon and energy ratio functions: eq_ratio(), q_ratio(), qe_ratio()

Examples

e_ratio(sun.spct, new_waveband(400,500), new_waveband(400,700))
e_response

wb.trim,
use.hinges,
...
)

## Default S3 method:
e_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## S3 method for class 'response_spct'
e_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)

## S3 method for class 'response_mspct'
e_response(
  spct,
  w.band = NULL,
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...

  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)
Arguments

- **spct**: an R object.
- **w.band**: waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
- **quantity**: character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
- **time.unit**: character or lubridate::duration object.
- **scale.factor**: numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
- **wb.trim**: logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...**: other arguments (possibly used by derived methods).
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach.
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- **default**: Default method for generic function
- **response_spct**: Method for response spectra.
- **response_mspct**: Calculates energy response from a response_mspct
Note
The parameter use.hinges controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

See Also
Other response functions: q_response(), response()

Examples

e_response(ccd.spct, new_waveband(200,300))
e_response(photodiode.spct)

Data for typical calibration lamps

Description
A dataset containing fitted constants to be used as input for function FEL_spectrum.

Format
A numeric vector.

Author(s)
Lasse Ylianttila (data)

Incandescent “FEL” lamp emission spectrum

Description
Calculate values by means of a nth degree polynomial from user-supplied constants (for example from a lamp calibration certificate).

Usage
FEL_spectrum(w.length, k = photobiology::FEL.BN.9101.165, fill = NA_real_)


findMultipleWl

Arguments

- **w.length** numeric vector of wavelengths (nm) for output
- **k** a numeric vector with n constants for the function
- **fill** if NA, no extrapolation is done, and NA is returned for wavelengths outside the range 250 nm to 900 nm. If NULL then the tails are deleted. If 0 then the tails are set to zero, etc. NA is default.

Value

a dataframe with four numeric vectors with wavelength values (w.length), energy and photon irradiance (s.e.irrad, s.q.irrad) depending on the argument passed to unit.out (s.irrad).

Note

This is function is valid for wavelengths in the range 250 nm to 900 nm, for wavelengths outside this range NAs are returned.

Examples

```r
FEL_spectrum(400)
FEL_spectrum(250:900)
```

Description

Find repeated w.length values

Usage

```r
findMultipleWl(x, same.wls = TRUE)
```

Arguments

- **x** a generic_spect object
- **same.wls** logical If TRUE all spectra specked to share same w.length values.

Value

integer Number of spectra, guessed from the number of copies of each individual w.length value.
Description

This function finds all peaks (local maxima) in a spectrum, using a user provided size threshold relative to the tallest peak (global maximum) bellow which found peaks are ignored—i.e., not included in the returned value. This is a wrapper built on top of function `peaks()` from package 'splus2R'.

Usage

```r
find_peaks(x, ignore_threshold = 0, span = 3, strict = TRUE, na.rm = FALSE)
```

Arguments

- `x`: numeric vector
- `ignore_threshold`: numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.
- `span`: integer A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Use `NULL` for the global peak.
- `strict`: logical If TRUE, an element must be strictly greater than all other values in its window to be considered a peak.
- `na.rm`: logical indicating whether NA values should be stripped before searching for peaks.

Value

A logical vector of the same length as `x`. Values that are TRUE correspond to local peaks in the data.

Note

This function is a wrapper built on function `peaks` from `splus2R` and handles non-finite (including `NA`) values differently than `splus2R::peaks`, instead of giving an error they are replaced with the smallest finite value in `x`.

See Also

- `peaks`
- Other peaks and valleys functions: `find_spikes()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `valleys()`, `wls_at_target()`

Examples

```r
with(sun.data, w.length[find_peaks(s.e.irrad)])
```
Description

This function finds spikes in a numeric vector using the algorithm of Whitaker and Hayes (2018). Spikes are values in spectra that are unusually high or low compared to neighbors. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode arrays. Other kinds of accidental "outlayers" will be also detected.

Usage

```r
find_spikes(
  x,
  x.is.delta = FALSE,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE
)
```

Arguments

- `x` numeric vector containing spectral data.
- `x.is.delta` logical Flag indicating if `x` contains already differences.
- `z.threshold` numeric Modified Z values larger than `z.threshold` are considered to be spikes.
- `max.spike.width` integer Wider regions with high Z values are not detected as spikes.
- `na.rm` logical indicating whether NA values should be stripped before searching for spikes.

Details

Spikes are detected based on a modified Z score calculated from the differenced spectrum. The Z threshold used should be adjusted to the characteristics of the input and desired sensitivity. The lower the threshold the more stringent the test becomes, resulting in most cases in more spikes being detected. A modified version of the algorithm is used if a value different from NULL is passed as argument to `max.spike.width`. In such a case, an additional step filters out broader spikes (or falsely detected steep slopes) from the returned values.

Value

A logical vector of the same length as `x`. Values that are TRUE correspond to local spikes in the data.
References


See Also

Other peaks and valleys functions: find_peaks(), get_peaks(), peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

Examples

with(white_led.raw_spct,
  which(find_spikes(counts_3, z.threshold = 30)))

find_wls

Find wavelength values in a spectrum

Description

Find wavelength values corresponding to a target y value in any spectrum. The name of the column of the spectral data to be used to match the target needs to be passed as argument unless the spectrum contains a single numerical variable in addition to "w.length".

Usage

find_wls(
  x,
  target = NULL,
  col.name.x = NULL,
  col.name = NULL,
  .fun = \( \leq \),
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE
)

Arguments

x an R object

target numeric value indicating the spectral quantity value for which wavelengths are to be searched and interpolated if need. The character strings "half.maximum" and "half.range" are also accepted as arguments.

col.name.x character The name of the column in which to the independent variable is stored. Defaults to "w.length" for objects of class "generic_spct" or derived.
col.name character The name of the column in which to search for the target value.

.fun function A binary comparison function or operator.

.interpolate logical Indicating whether the nearest wavelength value in x should be returned or a value calculated by linear interpolation between wavelength values straddling the target.

.idfactor logical or character Generates an index column of factor type. If idfactor = TRUE then the column is auto named spct.idx. Alternatively the column name can be directly passed as argument to idfactor as a character string.

.na.rm logical indicating whether NA values should be stripped before searching for the target.

Value

A spectrum object of the same class as x with fewer rows, possibly even no rows. If FALSE is passed to interpolate a subset of x is returned, otherwise a new object of the same class containing interpolated wavelengths for the target value is returned.

Note

This function is used internally by method wls_at_target(), and these methods should be preferred in user code and scripts.

Examples

```r
find_wls(white_led.source_spct)
find_wls(white_led.source_spct, target = "half.maximum")
find_wls(white_led.source_spct, target = 0.4)
find_wls(white_led.source_spct, target = 0.4, interpolate = TRUE)
find_wls(white_led.source_spct, target = c(0.3, 0.4))
find_wls(white_led.source_spct, target = c(0.3, 0.4), idfactor = "target")
find_wls(white_led.source_spct, target = c(0.3, 0.4), idfactor = TRUE)
find_wls(white_led.source_spct, target = c("HM", "HR"))
find_wls(white_led.source_spct, target = c("HM", "HR"), interpolate = TRUE)

led.df <- as.data.frame(white_led.source_spct)
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length")
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length", target = 0.4)
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length", target = c(0.3, 0.4))
find_wls(led.df, col.name = "s.e.irrad", col.name.x = "w.length", target = 0.4, idfactor = "target")
```
fit_peaks

Refine position and value of extremes by fitting

Description

Functions implementing fitting of peaks in a class-agnostic way. The fitting refines the location of peaks and value of peaks based on the location of maxima and minima supplied. This function is to be used together with find_peaks() or find_valleys().

Usage

```
fit_peaks(
  x,
  peaks.idx,
  span,
  x.col.name = NULL,
  y.col.name,
  method,
  max.span = 5L,
  maximum = TRUE,
  keep.cols = NULL
)

fit_valleys(
  x,
  valleys.idx,
  span,
  x.col.name = NULL,
  y.col.name,
  method,
  max.span = 5L,
  maximum = FALSE,
  keep.cols = NULL
)
```

Arguments

- `x`: generic_spct or data.frame object.
- `peaks.idx, valleys.idx`: logical or integer Indexes into `x` selecting global or local extremes.
- `span`: odd integer The span used when refining the location of maxima or minima of `x`.
- `x.col.name, y.col.name`: character Name of the column of `x` on which to operate.
- `method`: character The method to use for the fit.
max.span odd integer The maximum number of data points used when refining the location of maxima and minima.
maximum logical A flag indicating whether to search for maxima or minima.
keep.cols logical Keep unrecognized columns in data frames

Value
An R object of the same class as x containing the fitted values for the peaks, and optionally the values for at peaks.idx or valleys.idx for other retained columns.

Note
These functions are not meant for everyday use. Use option refine.wl = TRUE of methods peaks() and valleys() instead.

Examples

```r
peaks <- find_peaks(sun.spct["s.e.irrad"], span = 31)
fit_peaks(sun.spct, peaks, span = 31,
          y.col.name = "s.e.irrad", method = "spline")
```

fluence  Fluence

Description
Energy or photon fluence for one or more wavebands of a light source spectrum and a duration of exposure.

Usage

```r
fluence(
  spct,
  w.band,
  unit.out,
  exposure.time,
  scale.factor,
  wb.trim,
  use.cached.multi,
  use.hinges,
  allow.scaled,
  ...
)
```

## Default S3 method:
fluence(
## S3 method for class 'source_spct'
fluence(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
)

## S3 method for class 'source_mspct'
fluence(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
,
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)

**Arguments**

- **spct** an R object.
w.band a list of waveband objects or a waveband object.

unit.out character string with allowed values "energy", and "photon", or its alias "quantum".

exposure.time lubridate::duration object.

scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.

wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.

use.cached.mult logical indicating whether multiplier values should be cached between calls.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

allow.scaled logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.

... other arguments (possibly used by derived methods).

naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx character Name of the column with the names of the members of the collection of spectra.

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, [W m-2 nm-1] -> [mol s-1 m-2] If time.unit is day, [J d-1 m-2 nm-1] -> [mol d-1 m-2]

Methods (by class)

- default: Default for generic function
- source_spct: Calculate photon fluence from a source_spct object and the duration of the exposure
- source_mspct: Calculates fluence from a source_mspct object.
Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mlt=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other irradiance functions: e_fluence(), e_irrad(), irrad(), q_fluence(), q_irrad()

Examples

library(lubridate)
fluence(sun.spct,
  w.band = waveband(c(400,700)),
  exposure.time = lubridate::duration(3, "minutes") )

format.solar_time

Description

Format a solar_time object for pretty printing

Usage

## S3 method for class 'solar_time'
format(x, ..., sep = ":")

Arguments

x      an R object
...
 ignored
sep    character used as separator

See Also

Other astronomy related functions: day_night(), format.tod_time(), is.solar_time(), print.solar_time(), print.tod_time(), solar_time(), sun_angles()
format.tod_time  Encode in a Common Format

Description
Format a tod_time object for pretty printing

Usage
## S3 method for class 'tod_time'
format(x, ..., sep = ":")

Arguments
x an R object
... ignored
sep character used as separator

See Also
Other astronomy related functions: day_night(), format.solar_time(), is.solar_time(), print.solar_time(), print.tod_time(), solar_time(), sun_angles()

formatted_range  Compute range and format it

Description
Compute the range of an R object, and format it as string suitable for printing.

Usage
formatted_range(x, na.rm = TRUE, digits = 3, nsmall = 2, collapse = "..")

Arguments
x an R object
na.rm logical, indicating if NA’s should be omitted.
digits, nsmall numeric, passed to same name parameters of format().
collapse character, passed to same name parameter of paste().

See Also
range, format and paste.
Examples

```r
formatted_range(c(1, 3.5, -0.01))
```
target = 1,
    qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
    set.scaled = target == 1,
    ...
  )

## S3 method for class 'reflector_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  qty.out = NULL,
  set.scaled = target == 1,
  ...
)

## S3 method for class 'raw_spct'
fscale(x, range = NULL, f = "mean", target = 1, set.scaled = target == 1, ...)

## S3 method for class 'cps_spct'
fscale(x, range = NULL, f = "mean", target = 1, set.scaled = target == 1, ...)

## S3 method for class 'generic_spct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  col.names,
  ...
)

## S3 method for class 'source_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  set.scaled = target == 1,
  ...
, .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
fscale(  
  x,  
  range = NULL,  
  f = "mean",  
  target = 1,  
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),  
  set.scaled = target == 1,  
  ...  
  .parallel = FALSE,  
  .paropts = NULL)

## S3 method for class 'filter_mspct'
fscale(  
  x,  
  range = NULL,  
  f = "mean",  
  target = 1,  
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),  
  set.scaled = target == 1,  
  ...  
  .parallel = FALSE,  
  .paropts = NULL)

## S3 method for class 'reflector_mspct'
fscale(  
  x,  
  range = NULL,  
  f = "mean",  
  target = 1,  
  qty.out = NULL,  
  set.scaled = target == 1,  
  ...  
  .parallel = FALSE,  
  .paropts = NULL)

## S3 method for class 'raw_mspct'
fscale(  
  x,  
  range = NULL,  
  f = "mean",  
  target = 1,  
  set.scaled = target == 1,  
  ...  
  .parallel = FALSE,  
  .paropts = NULL)
## S3 method for class 'cps_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  ...,  
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'generic_mspct'
fscale(
  x,
  range = NULL,
  f = "mean",
  target = 1,
  set.scaled = target == 1,
  col.names,
  ...,  
  .parallel = FALSE,
  .paropts = NULL
)

### Arguments

- **x**: An R object
- **...**: additional named arguments passed down to `f`.
- **range**: numeric. An R object on which `range()` returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm)
- **f**: character string. "mean" or "total" for scaling so that this summary value becomes 1 for the returned object, or the name of a function taking `x` as first argument and returning a numeric value.
- **target**: numeric A constant used as target value for scaling.
- **unit.out**: character. Allowed values "energy", and "photon", or its alias "quantum".
- **set.scaled**: logical or NULL Flag indicating if the data is to be marked as "scaled" or not.
- **qty.out**: character. Allowed values "transmittance", and "absorbance".
- **col.names**: character vector containing the names of columns or variables to which to apply the scaling.
- **.parallel**: logical if TRUE, apply function in parallel, using parallel backend provided by foreach.
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x with the original spectral data values replaced with rescaled values, and the "scaled" attribute set to a list describing the scaling applied.
a new object of the same class as x.

Methods (by class)

• default: Default for generic function
• source_spct:
• response_spct:
• filter_spct:
• reflector_spct:
• raw_spct:
• cps_spct:
• generic_spct:
• source_mspct:
• response_mspct:
• filter_mspct:
• reflector_mspct:
• raw_mspct:
• cps_mspct:
• generic_mspct:

Note

Sometimes we rescale a spectrum to a "theoretical" value for the summary, while in other cases we rescale the spectrum to a real-world target value of e.g. a reference energy irradiance. In the first case we say that the data are expressed in relative units, while in the second case we retain actual physical units. To indicate this, this package uses an attribute, which will by default be set assuming the first of these two situations when target == 1 and the second situation otherwise. These defaults can be overridden with an explicit logical argument passed to set.scaled.

See Also

Other rescaling functions: fshift(), getNormalized(), getScaled(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()
Examples

```r
fscale(sun.spct)
fscale(sun.spct, f = "mean") # same as default
fscale(sun.spct, f = "mean", na.rm = TRUE)
fscale(sun.spct, range = c(400, 700)) # default is whole spectrum
fscale(sun.spct, f = e_irrad, range = c(400, 700))
s400.spct <- fscale(sun.spct,
  f = e_irrad,
  range = c(400, 700),
  target = 400) # a target in W m-2
s400.spct
e_irrad(s400.spct, c(400, 700))
```

---

**fshift**  
*Shift the scale of a spectrum using a summary function*

**Description**

The `fshift()` methods return a spectral object of the same class as the one supplied as argument but with the spectral data on a zero-shifted scale. A range of wavelengths is taken as a zero reference and the summary calculated with `f` for this waveband is subtracted. This results in a zero shift (= additive correction) to the values in the returned object. Metadata attributes are retained unchanged.

**Usage**

```r
fshift(x, ...)
```

## Default S3 method:

```r
fshift(x, ...)
```

## S3 method for class 'source_spct'

```r
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)
```

## S3 method for class 'response_spct'

```r
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
)
fshift

...)

## S3 method for class 'filter_spct'

fshift(
x,
  range = c(min(x), min(x) + 10),
  f = "min",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
...)

## S3 method for class 'reflector_spct'

fshift(x, range = c(min(x), min(x) + 10), f = "min", qty.out = NULL, ...)

## S3 method for class 'source_mspct'

fshift(
x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...)

## S3 method for class 'raw_spct'

fshift(x, range = c(min(x), min(x) + 10), f = "mean", qty.out = NULL, ...)

## S3 method for class 'cps_spct'

fshift(x, range = c(min(x), min(x) + 10), f = "mean", qty.out = NULL, ...)

## S3 method for class 'generic_spct'

fshift(x, range = c(min(x), min(x) + 10), f = "mean", col.names, ...)

## S3 method for class 'response_mspct'

fshift(
x,
  range = c(min(x), min(x) + 10),
  f = "mean",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
...,
  parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'

fshift(
x,
  range = c(min(x), min(x) + 10),
f = "min",
 qty.out = getOption("photobiology.filter.qty", default = "transmittance"), ...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'reflector_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  qty.out = NULL,
  ...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'raw_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  ...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'cps_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  ...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'generic_mspct'
fshift(
  x,
  range = c(min(x), min(x) + 10),
  f = "min",
  col.names,
  ...
.parallel = FALSE,
.paropts = NULL
)
Arguments

x An R object
... additional named arguments passed down to f.
range An R object on which range() returns a numeric vector of length 2 with the limits of a range of wavelengths in nm, with min and max wavelengths (nm)
f character string "mean", "min" or "max" for scaling so that this summary value becomes the origin of the spectral data scale in the returned object, or the name of a function taking x as first argument and returning a numeric value.
unit.out character Allowed values "energy", and "photon", or its alias "quantum"
qty.out character Allowed values "transmittance", and "absorbance"
col.names character vector containing the names of columns or variables to which to apply the scale shift.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x with the spectral data values replaced with values zero-shifted.
a new object of the same class as x.

Methods (by class)

- default: Default for generic function
- source_spct:
- response_spct:
- filter_spct:
- reflector_spct:
- source_mspct:
- raw_spct:
- cps_spct:
- generic_spct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- raw_mspct:
- cps_mspct:
- generic_mspct:
See Also

Other rescaling functions: \texttt{fscale()}, \texttt{getNormalized()}, \texttt{getScaled()}, \texttt{is_normalized()}, \texttt{is_scaled()}, \texttt{normalize()}, \texttt{setNormalized()}, \texttt{setScaled()}

\begin{description}
\item[generic_mspct] \textit{Collection-of-spectra constructor}
\end{description}

**Description**

Converts a list of spectral objects into a "multi spectrum" object by setting the class attribute of the list of spectra to the corresponding multi-spect class, check that components of the list belong to the expected class.

**Usage**

\begin{verbatim}
generic_mspct(
  l = NULL,
  class = "generic_spct",
  ncol = 1,
  byrow = FALSE,
  dim = c(length(l)%%ncol, ncol)
)
\end{verbatim}

calibration_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

raw_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

cps_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

source_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

filter_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

reflector_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

object_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

response_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

chroma_mspct(l = NULL, ncol = 1, byrow = FALSE, ...)

**Arguments**

\begin{itemize}
\item[l] list of generic_spct or derived classes
\item[class] character The multi spectrum object class or the expected class for the elements of \texttt{l}
\end{itemize}
getBSWFUsed

<table>
<thead>
<tr>
<th>ncol</th>
<th>integer Number of 'virtual' columns in data</th>
</tr>
</thead>
<tbody>
<tr>
<td>byrow</td>
<td>logical If ncol &gt; 1 how to read in the data</td>
</tr>
<tr>
<td>dim</td>
<td>integer vector of dimensions</td>
</tr>
<tr>
<td>...</td>
<td>ignored</td>
</tr>
</tbody>
</table>

**Functions**

- `calibration_mspct`: Specialization for collections of `calibration_spct` objects.
- `raw_mspct`: Specialization for collections of `raw_spct` objects.
- `cps_mspct`: Specialization for collections of `cps_spct` objects.
- `source_mspct`: Specialization for collections of `source_spct` objects.
- `filter_mspct`: Specialization for collections of `filter_spct` objects.
- `reflector_mspct`: Specialization for collections of `reflector_spct` objects.
- `object_mspct`: Specialization for collections of `object_spct` objects.
- `response_mspct`: Specialization for collections of `response_spct` objects.
- `chroma_mspct`: Specialization for collections of `chroma_spct` objects.

**Note**

Setting `class = source_spct` or `class = source_mspct` makes no difference

**Examples**

```r
filter_mspct(list(polyester.spct, yellow_gel.spct))
```

---

**getBSWFUsed**  
*Get the "bswf.used" attribute*

**Description**

Function to read the "time.unit" attribute of an existing `source_spct` object

**Usage**

```r
getBSWFUsed(x)
```

**Arguments**

- `x` : a `source_spct` object

**Value**

character string
getFilterProperties

Description

Function to read the "filter.properties" attribute of an existing filter_spct or a filter_mspct.

Usage

getFilterProperties(x, return.null, ...)

filter_properties(x, return.null, ...)

## Default S3 method:
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'filter_spct'
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'summary_filter_spct'
getFilterProperties(x, return.null = FALSE, ...)

## S3 method for class 'generic_mspct'
getFilterProperties(x, return.null = FALSE, ..., idx = "spct.idx")

Arguments

x a filter_spct object

return.null logical If true, NULL is returned if the attribute is not set, otherwise the expected list is returned with all fields set to NA.

... Allows use of additional arguments in methods for other classes.

idx character Name of the column with the names of the members of the collection of spectra.

Note

if x is not a source_spct object, NA is returned

See Also

Other BSWF attribute functions: setBSWFUsed()

Examples

getBSWFUsed(sun.spct)
getHowMeasured

Value

A list with fields named "Rfr.constant", "thickness" and "attenuation.mode". If the attribute is not set, and return.null is FALSE, a list with fields set to NA is returned, otherwise, NULL.

Methods (by class)

- default: default
- filter_spct: generic_spct
- summary_filter_spct: summary_generic_spct
- generic_mspct: filter_mspct

Note

The method for collections of spectra returns the a tibble with a column of lists.

See Also


Examples

filter_properties(polyester.spct)

getHowMeasured  Get the "how.measured" attribute

Description

Function to read the "how.measured" attribute of an existing generic_spct or a generic_mspct.

Usage

getHowMeasured(x, ...)

how_measured(x, ...)

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

## S3 method for class 'generic_spct'
getHowMeasured(x, ...)
getHowMeasured

## S3 method for class 'summary_generic_spct'
getHowMeasured(x, ...)

## S3 method for class 'generic_mspct'
getHowMeasured(x, ..., idx = "spct.idx")

### Arguments

- **x**
  - a generic_spct object
- **...**
  - Allows use of additional arguments in methods for other classes.
- **idx**
  - character Name of the column with the names of the members of the collection of spectra.

### Value

- character vector An object containing a description of the data.

### Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct

### Note

- The method for collections of spectra returns the a tibble with a column of character strings.

### See Also


### Examples

- how_measured(sun.spct)
getIdFactor

Get the "idfactor" attribute

Description
Function to read the "idfactor" attribute of an existing generic_spct.

Usage
getIdFactor(x)

Arguments
x
a generic_spct object

Value
character

Note
If x is not a generic_spct or an object of a derived class NA is returned.

See Also
Other idfactor attribute functions: setIdFactor()

Examples
getMultipleWl(sun.spct)

getInstrDesc

Get the "instr.desc" attribute

Description
Function to read the "instr.desc" attribute of an existing generic_spct object.

Usage
getInstrDesc(x)

Arguments
x
a generic_spct object
getInstrSettings

Value

list (depends on instrument type)

See Also

getMspctVersion

Description
Function to read the "mspct.version" attribute of an existing generic_mspct object.

Usage
getMspctVersion(x)

Arguments
x  a generic_mspct object

Value
numeric value

Note
if x is not a generic_mspct object, NA is returned, and if it the attribute is missing, zero is returned with a warning.

getMultipleWl

Description
Function to read the "multiple.wl" attribute of an existing generic_spct.

Usage
getMultipleWl(x)

Arguments
x  a generic_spct object

Value
integer

Note
If x is not a generic_spct or an object of a derived class NA is returned.
getNormalized

See Also
Other multiple.wl attribute functions: setMultipleWl()

Examples
getMultipleWl(sun.spct)

getNormalized
Get the "normalized" attribute

Description
Function to read the "normalized" attribute of an existing generic_spct object.

Usage
getNormalized(x, .force.numeric = FALSE)
getNormalised(x, .force.numeric = FALSE)

Arguments
x a generic_spct object
.force.numeric logical If TRUE always silently return a numeric value, with FALSE encoded as zero, and character values as NA.

Value
numeric or logical (possibly character for objects created with earlier versions).

Note
if x is not a generic_spct object, NA is returned
getNormalised() is a synonym for this getNormalized() method.

See Also
Other rescaling functions: fscale(), fshift(), getScaled(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()
getResponseType

Get the "response.type" attribute

Description

Function to read the "response.type" attribute of an existing response_spct object.

Usage

getResponseType(x)

Arguments

x a response_spct object

Details

Objects of class response_spct() can contain data for a response spectrum or an action spectrum. Response spectra are measured using the same photon (or energy) irradiance at each wavelength. Action spectra are derived from dose response curves at each wavelength, and responsivity at each wavelength is expressed as the reciprocal of the photon fluence required to obtain a fixed level of response.

Value

character string

Note

If x is not a response_spct object, NA is returned.

Examples

getResponseType(ccd.spct)
getResponseType(sun.spct)
getRfrType  
Get the "Rfr.type" attribute

**Description**

Function to read the "Rfr.type" attribute of an existing reflector_spct object or object_spct object.

**Usage**

getRfrType(x)

**Arguments**

- **x**: a source_spct object

**Value**

character string

**Note**

if x is not a filter_spct object, NA is returned

**See Also**

Other Rfr attribute functions: setRfrType()

---

getScaled  
Get the "scaled" attribute

**Description**

Function to read the "scaled" attribute of an existing generic_spct object.

**Usage**

getScaled(x, .force.list = FALSE)

**Arguments**

- **x**: a generic_spct object
- **.force.list**: logical If TRUE always silently return a list, with FALSE encoded field multiplier = 1.

**Value**

logical
getSpctVersion

Note

if x is not a filter_spct object, NA is returned

See Also

Other rescaling functions: fscale(), fshift(), getNormalized(), is_normalized(), is_scaled(), normalize(), setNormalized(), setScaled()

Examples

scaled.spct <- fscale(sun.spct)
getScaled(scaled.spct)

getSpctVersion  Get the "spct.version" attribute

Description

Function to read the "spct.version" attribute of an existing generic_spct object.

Usage

getSpctVersion(x)

Arguments

x a generic_spct object

Value

integer value

Note

if x is not a generic_spct object, NA is returned, and if it the attribute is missing, zero is returned with a warning.
**getTfrType**

*Get the "Tfr.type" attribute*

**Description**

Function to read the "Tfr.type" attribute of an existing filter_spct or object_spct object.

**Usage**

getTfrType(x)

**Arguments**

- **x**: a filter_spct or object_spct object

**Value**

character string

**Note**

If `x` is not a filter_spct or an object_spct object, NA is returned.

**See Also**

Other Tfr attribute functions: setTfrType()

**Examples**

getTfrType(polyester.spct)

---

**getTimeUnit**

*Get the "time.unit" attribute of an existing source_spct object*

**Description**

Function to read the "time.unit" attribute

**Usage**

getTimeUnit(x, force.duration = FALSE)

**Arguments**

- **x**: a source_spct object
- **force.duration**: logical If TRUE a lubridate::duration is returned even if the object attribute is a character string, if no conversion is possible NA is returned.
Value
character string or a lubridate::duration

Note
if x is not a source_spct or a response_spct object, NA is returned

See Also
Other time attribute functions: checkTimeUnit(), convertTfrType(), convertThickness(), convertTimeUnit(), setTimeUnit()

Examples
getTimeUnit(sun.spct)
Value

character vector An object containing a description of the data.

Methods (by class)

• default: default
• generic_spct: generic_spct
• summary_generic_spct: summary_generic_spct
• generic_mspct: generic_mspct

Note

The method for collections of spectra returns the a tibble with a column of character strings.

See Also

Other measurement metadata functions: `add_attr2tb()`, `getFilterProperties()`, `getHowMeasured()`,
`getInstrDesc()`, `getInstrSettings()`, `getWhenMeasured()`, `getWhereMeasured()`,
`get_attributes()`, `isValidInstrDesc()`, `isValidInstrSettings()`, `select_spct_attributes()`,
`setFilterProperties()`, `setHowMeasured()`, `setInstrDesc()`, `setInstrSettings()`, `setWhatMeasured()`,
`setWhereMeasured()`, `spct_attr2tb()`, `spct_metadata()`, `trimInstrDesc()`, `trimInstrSettings()`

Examples

```r
what_measured(sun.spct)
```

---

**getWhenMeasured**

*Get the "when.measured" attribute*

**Description**

Function to read the "when.measured" attribute of an existing generic_spct or a generic_mspct.

**Usage**

```r
getWhenMeasured(x, ...)
when_measured(x, ...)
```

```r
## Default S3 method:
getWhenMeasured(x, ...)
```

```r
## S3 method for class 'generic_spct'
getWhenMeasured(x, ...)
```

---

---
getWhenMeasured

```r
## S3 method for class 'summary_generic_spct'
getWhenMeasured(x, ...)

## S3 method for class 'generic_mspct'
getWhenMeasured(x, ..., idx = "spct.idx")
```

### Arguments

- **x**: a `generic_spct` object
- **...**: Allows use of additional arguments in methods for other classes.
- **idx**: character Name of the column with the names of the members of the collection of spectra.

### Value

POSIXct An object with date and time.

### Methods (by class)

- **default**: default
- **generic_spct**: `generic_spct`
- **summary_generic_spct**: `summary_generic_spct`
- **generic_mspct**: `generic_mspct`

### Note

If `x` is not a `generic_spct` or an object of a derived class NA is returned.

The method for collections of spectra returns the a tibble with the correct times in TZ = "UTC".

### See Also


### Examples

```r
when_measured(sun.spct)
```
getWhereMeasured

Get the "where.measured" attribute

Description

Function to read the "where.measured" attribute of an existing generic_spct.

Usage

getWhereMeasured(x, ...)

where_measured(x, ...)

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

## S3 method for class 'generic_spct'
getWhereMeasured(x, ...)

## S3 method for class 'summary_generic_spct'
getWhereMeasured(x, ...)

## S3 method for class 'generic_mspct'
getWhereMeasured(x, ..., idx = "spct.idx", .bind.geocodes = TRUE)

Arguments

x

a generic_spct object

...  
Allows use of additional arguments in methods for other classes.

idx

character Name of the column with the names of the members of the collection of spectra.

.bind.geocodes  
logical In the case of collections of spectra if .bind.geocodes = TRUE, the default, the returned value is a single geocode with one row for each member spectrum. Otherwise the individual geocode data frames are returned in a list column within a tibble.

Value

a data.frame with a single row and at least columns "lon" and "lat", unless expand is set to FALSE.

Methods (by class)

- default: default
- generic_spct: generic_spct
- summary_generic_spct: summary_generic_spct
- generic_mspct: generic_mspct
Note

If x is not a generic_spct or an object of a derived class NA is returned.

See Also

get_attributes

Arguments

x a generic_spct object.

which character vector Names of attributes to retrieve.

... currently ignored

allowed character vector Names of attributes accepted by which.

Details

Vectors of character strings passed as argument to which are parsed so that if the first member string is "-" the remaining members are removed from the allowed; and if it is "=" the remaining members are used if in allowed. If the first member is none of these three strings, the behaviour is the same as if the first string is "=". If which is NULL all the attributes in allowed are used. The string "" means no attributes, and has precedence over any other values in the character vector. The order of the names of annotations has no meaning: the vector is interpreted as a set except for the three possible "operators" at position 1.

Value

Named list of attribute values.

Methods (by class)

• generic_spct: generic_spct

• source_spct: source_spct

• filter_spct: filter_spct

• reflector_spct: reflector_spct

• object_spct: object_spct

• waveband: waveband

See Also

select_spct_attributes

get_peaks

Get peaks and valleys in a spectrum

Description

These functions find peaks (local maxima) or valleys (local minima) in a spectrum, using a user selectable size threshold relative to the tallest peak (global maximum). This a wrapper built on top of function peaks from package splus2R.

Usage

get_peaks(
  x,
  y,
  ignore_threshold = 0,
  span = 5,
  strict = TRUE,
  x_unit = "",
  x_digits = 3,
  na.rm = FALSE
)

get_valleys(
  x,
  y,
  ignore_threshold = 0,
  span = 5,
  strict = TRUE,
  x_unit = "",
  x_digits = 3,
  na.rm = FALSE
)

Arguments

x numeric

y numeric

ignore_threshold numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.

span integer A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Use NULL for the global peak.

strict logical If TRUE, an element must be strictly greater than all other values in its window to be considered a peak.
x_unit character Vector of texts to be pasted at end of labels built from x value at peaks.
x_digits numeric Number of significant digits in wavelength label.
na.rm logical indicating whether NA values should be stripped before searching for peaks.

Value
A data frame with variables w.length and s.irrad with their values at the peaks or valleys plus a character variable of labels.

See Also
Other peaks and valleys functions: find_peaks(), find_spikes(), peaks(), replace_bad_pxls(), spikes(), valleys(), wls_at_target()

Examples
with(sun.spct, get_peaks(w.length, s.e.irrad))
with(sun.spct, get_valleys(w.length, s.e.irrad))

Description
A dataset of spectral reflectance expressed as a fraction of one.

Usage
green_leaf.spct

Format
A reflector_spct object with 226 rows and 2 variables

Details
- w.length (nm)
- Rfr (0..1)

References
Aphalo, P. J. & Lehto, T. Effects of light quality on growth and N accumulation in birch seedlings Tree Physiology, 1997, 17, 125-132
head_tail

See Also

Ler_leaf_trns.spct.Ler_leaf_trns_i.spct.black_body.spct.ccd.spct.clear.spct.clear_body.spct.
sun.data.sun.spct.white_body.spct.white_led.cps_spct.white_led.raw_spct.white_led.source_spct.
yellow_gel.spct

Examples

green_leaf.spct

head_tail

Return the First and Last Part of an Object

Description

Returns the first and last "parts" (rows or members) of a spectrum, dataframe, vector, function, table
or ftable. In other words, the combined output from methods head and tail.

Usage

head_tail(x, n, ...)

## Default S3 method:
head_tail(x, n = 3L, ...)

## S3 method for class 'data.frame'
head_tail(x, n = 3L, ...)

## S3 method for class 'matrix'
head_tail(x, n = 3L, ...)

## S3 method for class 'function'
head_tail(x, n = 6L, ...)

## S3 method for class 'table'
head_tail(x, n = 6L, ...)

## S3 method for class 'ftable'
head_tail(x, n = 6L, ...)

Arguments

x an R object.
integer. If positive, \( n \) rows or members in the returned object are copied from each of "head" and "tail" of \( x \). If negative, all except \( n \) elements of \( x \) from each of "head" and "tail" are returned.

... arguments to be passed to or from other methods.

Details

The value returned by \texttt{head\_tail()} is equivalent to row binding the values returned by \texttt{head()} and \texttt{tail()}, although not implemented in this way. The same specializations as defined in package 'utils' for \texttt{head()} and \texttt{tail()} have been implemented.

Value

An object (usually) like \( x \) but smaller, except when \( n = 0 \). For \texttt{ftable} objects \( x \), a transformed \texttt{format(x)}.

Methods (by class)

- \texttt{default}:
- \texttt{data.frame}:
- \texttt{matrix}:
- \texttt{function}:
- \texttt{table}:
- \texttt{ftable}:

Note

For some types of input, like functions, the output may be confusing, however, we have opted for consistency with existing functions. The code is in part a revision of that of \texttt{head()} and \texttt{tail()} from package 'utils'. I have been missing this method especially when checking spectral data, as both ends are of interest.

\texttt{head\_tail()} methods for function, table and \texttt{ftable} classes, are wrappers for \texttt{head()} method.

See Also

\texttt{head}, and compare the examples and the values returned to the examples below.

Examples

\begin{verbatim}
head_tail(letters)
head_tail(letters, n = -6L)
head_tail(freeny.x, n = 10L)
head_tail(freeny.y)

head_tail(stats::ftable(Titanic))
\end{verbatim}
**insert_hinges**

*Insert wavelength values into spectral data.*

**Description**

Inserting wavelengths values immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data have a large wavelength step size.

**Usage**

```r
insert_hinges(x, y, h)
```

**Arguments**

- `x`: numeric vector (sorted in increasing order)
- `y`: numeric vector
- `h`: a numeric vector giving the wavelengths at which the `y` values should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0))

**Value**

a data.frame with variables `x` and `y`. Unless the hinge values were already present in `y`, each inserted hinge, expands the vectors returned in the data frame by one value.

**Note**

Insertion is a costly operation but I have tried to optimize this function as much as possible by avoiding loops. Earlier this function was implemented in C++, but a bug was discovered and I have now rewritten it using R.

**See Also**

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

**Examples**

```r
with(sun.data,
    insert_hinges(w.length, s.e.irrad,
                  c(399.99, 400.00, 699.99, 700.00)))
```
insert_spct_hinges  
*Insert new wavelength values into a spectrum*

**Description**

Insert new wavelength values into a spectrum interpolating the corresponding spectral data values.

**Usage**

insert_spct_hinges(spct, hinges = NULL, byref = FALSE)

**Arguments**

- **spct**: an object of class "generic_spct"
- **hinges**: numeric vector of wavelengths (nm) at which the s.irrad should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0))
- **byref**: logical indicating if new object will be created by reference or by copy of spct

**Value**

a generic_spct or a derived type with variables w.length and other numeric variables.

**Note**

Inserting wavelengths values "hinges" immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data has a large wavelength step size.

**Examples**

```r
insert_spct_hinges(sun.spct, c(399.99,400.00,699.99,700.00))
insert_spct_hinges(sun.spct,
                   c(199.99,200.00,399.50,399.99,400.00,699.99,
                     700.00,799.99,1000.00))
```

\[\text{integrate_spct}\]

*Integrate spectral data.*

**Description**

This function gives the result of integrating spectral data over wavelengths.

**Usage**

integrate_spct(spct)
**integrate_xy**

**Arguments**

- `spct` - generic_spct

**Value**

One or more numeric values with no change in scale factor: e.g. [W m⁻² nm⁻¹] → [W m⁻²]. Each value in the returned vector corresponds to a variable in the spectral object, except for wavelength. For non-numeric variables the returned value is NA.

**Examples**

```r
integrate_spct(sun.spct)
```

**Description**

This function gives the result of integrating spectral irradiance over wavelengths.

**Usage**

```r
integrate_xy(x, y)
```

**Arguments**

- `x` - numeric vector.
- `y` - numeric vector.

**Value**

a single numeric value with no change in scale factor: e.g. [W m⁻² nm⁻¹] → [W m⁻²]

**See Also**

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges`

**Examples**

```r
with(sun.data, integrate_xy(w.length, s.e.irrad))
```
interpolate_spct  Map a spectrum to new wavelength values.

Description
This function gives the result of interpolating spectral data from the original set of wavelengths to a new one.

Usage
interpolate_spct(spct, w.length.out = NULL, fill = NA, length.out = NULL)

interpolate_mspct(
  mspct,
  w.length.out = NULL,
  fill = NA,
  length.out = NULL,
  .parallel = FALSE,
  .paropts = NULL
)

Arguments
spct  generic_spct
w.length.out  numeric vector of wavelengths (nm)
fill  a value to be assigned to out of range wavelengths
length.out  numeric value
mspct  an object of class "generic_mspct"
.parallel  if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts  a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details
If length.out it is a numeric value, then gives the number of rows in the output, if it is NULL, the values in the numeric vector w.length.out are used. If both are not NULL then the range of w.length.out and length.out are used to generate a vector of wavelength. A value of NULL for fill prevents extrapolation. If both w.length.out and length.out are NULL the input is returned as is. If w.length.out has length equal to zero, zero rows from the input are returned.

Value
A new spectral object of the same class as argument spct.
Note

The default fill = NA fills extrapolated values with NA. Giving NULL as argument for fill deletes wavelengths outside the input data range from the returned spectrum. A numerical value can be also be provided as fill. This function calls interpolate_spectrum for each non-wavelength column in the input spectra object.

Examples

```r
interpolate_spct(sun.spct, 400:500, NA)
interpolate_spct(sun.spct, 400:500, NULL)
interpolate_spct(sun.spct, seq(200, 1000, by=0.1), 0)
interpolate_spct(sun.spct, c(400,500), length.out=201)
```

interpolate_spectrum

Calculate spectral values at a different set of wavelengths

Description

Interpolate/re-express spectral irradiance (or other spectral quantity) values at new wavelengths values. This is a low-level function operating on numeric vectors and called by higher level functions in the package, such as mathematical operators for classes for spectral data.

Usage

```r
interpolate_spectrum(w.length.in, s.irrad, w.length.out, fill = NA, ...)
```

Arguments

- `w.length.in` numeric vector of wavelengths (nm).
- `s.irrad` a numeric vector of spectral values.
- `w.length.out` numeric vector of wavelengths (nm).
- `fill` a value to be assigned to out of range wavelengths.
- `...` additional arguments passed to `spline()`.

Value

a numeric vector of interpolated spectral values.

Note

The current version of interpolate uses `spline` if fewer than 25 data points are available. Otherwise it uses `approx`. In the first case a cubic spline is used, in the second case linear interpolation, which should be faster.
interpolate_wl

Map spectra to new wavelength values.

Description

This function returns the result of interpolating spectral data from the original set of wavelengths to a new one.

Usage

interpolate_wl(x, w.length.out, fill, length.out, ...)

## Default S3 method:
interpolate_wl(x, w.length.out, fill, length.out, ...)

## S3 method for class 'generic_spct'
interpolate_wl(x, w.length.out = NULL, fill = NA, length.out = NULL, ...)

## S3 method for class 'generic_mspct'
interpolate_wl(
  x,
  w.length.out = NULL,
  fill = NA,
  length.out = NULL,
  ...
)

Examples

my.w.length <- 300:700
with(sun.data, interpolate_spectrum(w.length, s.e.irrad, my.w.length))

See Also

splinefun.

Other low-level functions operating on numeric vectors: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()
interpolate_wl

Arguments

- **x**: an R object
- **w.length.out**: numeric vector of wavelengths (nm)
- **fill**: a value to be assigned to out of range wavelengths
- **length.out**: numeric value
- **...**: not used
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Details

If **length.out** it is a numeric value, then gives the number of rows in the output, if it is NULL, the values in the numeric vector **w.length.out** are used. If both are not NULL then the range of **w.length.out** and **length.out** are used to generate a vector of wavelength. A value of NULL for **fill** prevents extrapolation.

Value

A new spectral object of the same class as argument **spct**.

Methods (by class)

- **default**: Default for generic function
- **generic_spct**: Interpolate wavelength in an object of class "generic_spct" or derived.
- **generic_mspct**: Interpolate wavelength in an object of class "generic_mspct" or derived.

Note

The default **fill = NA** fills extrapolated values with NA. Giving NULL as argument for fill deletes wavelengths outside the input data range from the returned spectrum. A numerical value can be also be provided as fill. This function calls interpolate_spectrum for each non-wavelength column in the input spectra object.

Examples

```r
interpolate_wl(sun.spct, 400:500, NA)
interpolate_wl(sun.spct, 400:500, NULL)
interpolate_wl(sun.spct, seq(200, 1000, by=0.1), 0)
interpolate_wl(sun.spct, c(400,500), length.out=201)
```
irrad

Irradiance

Description

This function returns the irradiance for a given waveband of a light source spectrum.

Usage

irrad(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## Default S3 method:
irrad(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  allow.scaled,
  ...
)

## S3 method for class 'source_spct'
irrad(
  spct,
  w.band = NULL,
  unit.out =getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  ...
use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
use.hinges = getOption("photobiology.use.hinges"),
allow.scaled = !quantity %in% c("average", "mean", "total"),
naming = "default",
...)

## S3 method for class 'source_mspct'
irrad(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = !quantity %in% c("average", "mean", "total"),
naming = "default",
...,
attr2tb = NULL,
idx = "spct.idx",
.parallel = FALSE,
.paropts = NULL)

Arguments

spct an R object.
w.band waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are summarized.
unit.out character string with allowed values "energy", and "photon", or its alias "quantum".
quantity character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
time.unit character or lubridate::duration object.
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
use.cached.mult logical indicating whether multiplier values should be cached between calls.
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
allow.scaled: logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.

... other arguments (possibly ignored)

naming: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

attr2tb: character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx: character Name of the column with the names of the members of the collection of spectra.

.parallel: if TRUE, apply function in parallel, using parallel backend provided by foreach.

.paropts: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra. If naming = "long" the names generated reflect both quantity and waveband, if naming = "short", names are based only on the wavebands, and if naming = "none" the returned vector has no names.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, \([\text{W m}^{-2} \text{nm}^{-1}] \rightarrow [\text{mol s}^{-1} \text{m}^{-2}]\) or \([\text{W m}^{-2} \text{nm}^{-1}] \rightarrow [\text{W m}^{-2}]\) If time.unit is day, \([\text{J d}^{-1} \text{m}^{-2} \text{nm}^{-1}] \rightarrow [\text{mol d}^{-1} \text{m}^{-2}]\) or \([\text{J d}^{-1} \text{m}^{-2} \text{nm}^{-1}] \rightarrow [\text{J m}^{-2}]\)

Methods (by class)

- default: Default for generic function
- source_spct: Calculates irradiance from a source_spct object.
- source_mspct: Calculates irradiance from a source_mspct object.

Note

Formal parameter allow.scaled is used internally for calculation of ratios, as rescaling and normalization do not invalidate the calculation of ratios.

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.
irradiance

Photon or energy irradiance from spectral energy or photon irradiance.

Description

Energy or photon irradiance for one or more wavebands of a radiation spectrum.

Usage

irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.out = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)

Arguments

- **w.length**: numeric Vector of wavelength (nm).
- **s.irrad**: numeric vector of spectral (energy) irradiances (W m-2 nm-1).
- **w.band**: waveband or list of waveband objects. The waveband(s) determine the region(s) of the spectrum that are summarized.
- **unit.out**: character. Allowed values "energy", and "photon", or its alias "quantum".
- **unit.in**: character. Allowed values "energy", and "photon", or its alias "quantum".

See Also

Other irradiance functions: `e_fluence()`, `e_irrad()`, `fluence()`, `q_fluence()`, `q_irrad()`

Examples

```r
irrad(sun.spct, waveband(c(400,700)))
irrad(sun.spct, waveband(c(400,700)), "energy")
irrad(sun.spct, waveband(c(400,700)), "photon")
irrad(sun.spct, split_bands(c(400,700), length.out = 3))
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "total")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "average")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative.pc")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution")
irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution.pc")
```
check.spectrum  logical Flag indicating whether to sanity check input data, default is TRUE.
use.cached.mult  logical Flag indicating whether multiplier values should be cached between calls.
use.hinges  logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value
A single numeric value or a vector of numeric values with no change in scale factor: \([\text{W m}^{-2} \text{nm}^{-1}]\) -> \([\text{mol s}^{-1} \text{m}^{-2}]\)

Note
The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum()` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector. The is no reason for setting `use.cpp.code=FALSE` other than for testing the improvement in speed, or in cases where there is no suitable C++ compiler for building the package.

See Also
Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `interpolate_spectrum()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photon_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```r
with(sun.data, irradiance(w.length, s.e.irrad, new_waveband(400,700), "photon"))
```

---

**is.generic_mspct**

**Query class of spectrum objects**

**Description**

Functions to check if an object is of a given type of spectrum, or coerce it if possible.
Usage

is.generic_mspct(x)

is.calibration_mspct(x)

is.raw_mspct(x)

is.cps_mspct(x)

is.source_mspct(x)

is.response_mspct(x)

is.filter_mspct(x)

is.reflector_mspct(x)

is.object_mspct(x)

is.chroma_mspct(x)

is.any_mspct(x)

Arguments

x an R object.

Value

These functions return TRUE if its argument is a of the queried type of spectrum and FALSE otherwise.

Note

Derived types also return TRUE for a query for a base type such as generic_mspct.

Examples

my.mspct <- filter_mspct(list(polyester.spct, yellow_gel.spct))
is.any_mspct(my.mspct)
is.filter_mspct(my.mspct)
is.source_mspct(my.mspct)
is.generic_spct  Query class of spectrum objects

Description
Functions to check if an object is of a given type of spectrum, or coerce it if possible.

Usage
is.generic_spct(x)
is.raw_spct(x)
is.calibration_spct(x)
is.cps_spct(x)
is.source_spct(x)
is.response_spct(x)
is.filter_spct(x)
is.reflector_spct(x)
is.object_spct(x)
is.chroma_spct(x)
is.any_spct(x)

Arguments
x  an R object.

Value
These functions return TRUE if its argument is a of the queried type of spectrum and FALSE otherwise.

Note
Derived types also return TRUE for a query for a base type such as generic_spct.

Examples
is.source_spct(sun.spct)
is.filter_spct(sun.spct)
is.generic_spct(sun.spct)
is.old_spct

Description
Query if an object has old class names

Usage
is.old_spct(object)

Arguments
object an R object

Value
logical

See Also
Other upgrade from earlier versions: upgrade_spct(), upgrade_spectra()

is.solar_time

Description
Query class

Usage
is.solar_time(x)

is.solar_date(x)
is.summary_generic_spct

Query class of spectrum summary objects

Description

Functions to check if an object is of a given type of spectrum, or coerce it if possible.

Usage

is.summary_generic_spct(x)

is.summary_raw_spct(x)

is.summary_cps_spct(x)

is.summary_source_spct(x)

is.summary_response_spct(x)

is.summary_filter_spct(x)

is.summary_reflector_spct(x)

is.summary_object_spct(x)

is.summary_chroma_spct(x)

is.any_summary_spct(x)

Arguments

x an R object.

Value

These functions return TRUE if its argument is of the queried type of spectrum and FALSE otherwise.
Note

Derived types also return TRUE for a query for a base type such as generic_spct.

Examples

```r
sm <- summary(sun.spct)
is.summary_source_spct(sm)
```

is.waveband

Query if it is a waveband

Description

Functions to check if an object is waveband.

Usage

```r
is.waveband(x)
```

Arguments

- `x` any R object

Value

is.waveband returns TRUE if its argument is a waveband and FALSE otherwise.

isValidInstrDesc

Check the "instr.desc" attribute

Description

Function to validate the "instr.settings" attribute of an existing generic_spct object.

Usage

```r
isValidInstrDesc(x)
```

Arguments

- `x` a generic_spct object

Value

logical TRUE if at least instrument name and serial number is found.
is_absorbance_based

Description

Functions to check if an filter spectrum contains spectral absorbance data or spectral transmittance data.

See Also

is_effective

Usage

is_absorbance_based(x)

is_absorptance_based(x)

is_transmittance_based(x)

Arguments

x an R object

Value

is_absorbance_based returns TRUE if its argument is a filter_spct object that contains spectral absorbance data and FALSE if it does not contain such data, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

is_absorptance_based returns TRUE if its argument is a filter_spct object that contains spectral absorptance and FALSE if it does not contain such data, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

is_transmittance_based returns TRUE if its argument is a filter_spct object that contains spectral transmittance data and FALSE if it does not contain such data, but returns NA for any other R object, including those belonging other generic_spct-derived classes.

See Also

Other query units functions: is_photon_based()

Examples

is_absorbance_based(polyester.spct)
my.spct <- T2A(polyester.spct)
is.filter_spct(my.spct)
is_absorbance_based(my.spct)

is_absorptance_based(polyester.spct)

is_transmittance_based(polyester.spct)

is_effective Is an R object "effective"

Description

A generic function for querying if a biological spectral weighting function (BSWF) has been applied to an object or is included in its definition.
is_effective

Usage

is_effective(x)

## Default S3 method:
is_effective(x)

## S3 method for class 'waveband'
is_effective(x)

## S3 method for class 'generic_spct'
is_effective(x)

## S3 method for class 'source_spct'
is_effective(x)

## S3 method for class 'summary_generic_spct'
is_effective(x)

## S3 method for class 'summary_source_spct'
is_effective(x)

Arguments

x an R object

Value

A logical.

Methods (by class)

- default: Default method.
- waveband: Is a waveband object defining a method for calculating effective irradiance.
- generic_spct: Does a source_spct object contain effective spectral irradiance values.
- source_spct: Does a source_spct object contain effective spectral irradiance values.
- summary_generic_spct: Method for "summary_generic_spct".
- summary_source_spct: Method for "summary_source_spct".

See Also

Other waveband attributes: labels(), normalization()

Examples

is_effective(summary(sun.spct))
is_normalized Query whether a generic spectrum has been normalized.

Description
This function tests a generic_spct object for an attribute that signals whether the spectral data has been normalized or not after the object was created.

Usage
is_normalized(x)
is_normalised(x)

Arguments
x An R object.

Value
A logical value. If x is not normalized or x is not a generic_spct object the value returned is FALSE.

Note
is_normalised() is a synonym for this is_normalized() method.

See Also
Other rescaling functions: fscale(), fshift(), getNormalized(), getScaled(), is_scaled(), normalize(), setNormalized(), setScaled()

is_photon_based Query if a spectrum contains photon- or energy-based data.

Description
Functions to check if source_spct and response_spct objects contains photon-based or energy-based data.

Usage
is_photon_based(x)
is_energy_based(x)
is_scaled

Query whether a generic spectrum has been scaled

Description

This function tests a generic_spct object for an attribute that signals whether the spectral data has been rescaled or not after the object was created.

Usage

is_scaled(x)

Arguments

x An R object.
Value
A logical value. If \( x \) is not scaled or \( x \) is not a generic_spct object the value returned is FALSE.

See Also
Other rescaling functions: \texttt{fscale()}, \texttt{fshift()}, \texttt{getNormalized()}, \texttt{getScaled()}, \texttt{is_normalized()}, \texttt{normalize()}, \texttt{setNormalized()}, \texttt{setScaled()}

Examples
\begin{verbatim}
scaled.spct <- fscale(sun.spct)
is_scaled(sun.spct)
is_scaled(scaled.spct)
\end{verbatim}
join_mspct  

Join all spectra in a collection

Description

Join all the spectra contained in a homogenous collection, returning a data frame with spectral-data columns named according to the names of the spectra in the collection. By default a full join is done, filling the spectral data for missing wave lengths in individual spectra with NA.

Usage

```r
join_mspct(x, type, ...) # Default S3 method:
join_mspct(x, type = "full", ...) # S3 method for class 'generic_mspct'
join_mspct(x, type = "full", col.name, ...) # S3 method for class 'source_mspct'
join_mspct(x, type = "full", unit.out = "energy", ...) # S3 method for class 'response_mspct'
join_mspct(x, type = "full", unit.out = "energy", ...) # S3 method for class 'filter_mspct'
join_mspct(x, type = "full", qty.out = "transmittance", ...) # S3 method for class 'reflector_mspct'
join_mspct(x, type = "full", qty.out, ...) # S3 method for class 'object_mspct'
```

Arguments

- `x` A generic_mspct object, or an object of a class derived from generic_mspct.
- `type` character Type of join: "left", "right", "inner" or "full" (default). See details for more information.
- `...` ignored (possibly used by derived methods).
- `col.name` character, name of the column in the spectra to be preserved, in addition to "w.length".
- `unit.out` character Allowed values "energy", and "photon", or its alias "quantum".
- `qty.out` character Allowed values "transmittance", and "absorbance".
Value
An object of class dataframe, with the spectra joined by wave length, with rows in addition sorted by wave length (variable w.length).

Methods (by class)

• default:
  • generic_mspct:
  • source_mspct:
  • response_mspct:
  • filter_mspct:
  • reflector_mspct:
  • object_mspct:

Note
Currently only generic_spct, source_mspct, response_mspct, filter_mspct, reflector_mspct and object_mspct classes have this method implemented.

labels
Find labels from "waveband" object

Description
A function to obtain the name and label of objects of class "waveband".

Usage

## S3 method for class 'waveband'
labels(object, ...)

## S3 method for class 'generic_spct'
labels(object, ...)

Arguments

object an object of class "waveband"
...

not used in current version

Methods (by class)

• generic_spct:

See Also

Other waveband attributes: is_effective(), normalization()
**Examples**

labels(sun.spct)

---

**Ler_leaf.spct**

*Green Arabidopsis leaf reflectance and transmittance.*

**Description**

A dataset of total spectral reflectance and total spectral transmittance expressed as fractions of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

Ler_leaf.spct

**Format**

An *object_spct* object with 2401 rows and 3 variables

**Details**

- w.length (nm)
- Rfr (0..1)
- Tfr (0..1)

**Note**

Measured with a Jaz spectrometer from Ocean Optics (USA) configured with a PX Xenon lamp module and Spectroclip double integrating spheres.

**Author(s)**

Aphalo, P. J. & Wang, F (unpublished data)

**See Also**


**Examples**

Ler_leaf.spct
**Ler_leaf_rflt.spct**

Green Arabidopsis leaf spectral reflectance.

**Description**

A dataset of total spectral reflectance expressed as fractions of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

Ler_leaf_rflt.spct

**Format**

An reflector_spct object with 1750 rows and 2 variables

**Details**

- w.length (nm)
- Rfr (0..1)

**Note**

Measured with a Jaz spectrometer from Ocean Optics (USA) configured with a PX Xenon lamp module and Spectroclip double integrating spheres.

**Author(s)**

Aphalo, P. J. & Wang, F (unpublished data)

**See Also**

Other Spectral data examples: A.illuminant.spct, D65.illuminant.spct, Ler_leaf.spct, Ler_leaf_trns.spct, Ler_leaf_trns_i.spct, black_body.spct, ccd.spct, clear.spct, clear_body.spct, filter_cps.mspct, green_leaf.spct, opaque.spct, photodiode.spct, polyester.spct, sun.daily.data, sun.daily.spct, sun.data, sun.spct, white_body.spct, white_led.cps.spct, white_led.raw.spct, white_led.source.spct, yellow_gel.spct

**Examples**

Ler_leaf_rflt.spct
**Ler_leaf_trns.spct**

---

**Ler_leaf_trns.spct**  
*Green Arabidopsis leaf spectral transmittance.*

---

**Description**

A dataset of total spectral transmittance expressed as a fraction of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

Ler_leaf_trns.spct

**Format**

An *filter_spct* object with 1753 rows and 2 variables

**Details**

- w.length (nm)
- Tfr (0..1)

**Note**

Measured with a Jaz spectrometer from Ocean Optics (USA) configured with a PX Xenon lamp module and Spectroclip double integrating spheres.

**Author(s)**

Aphalo, P. J. & Wang, F (unpublished data)

**See Also**

Other Spectral data examples: *A.illuminant.spct,D65.illuminant.spct,Ler_leaf.spct,Ler_leaf_rflt.spct,Ler_leaf_trns_i.spct,black_body.spct,ccd.spct,clear.spct,clear_body.spct,filter_cps.mspct,green_leaf.spct,opaque.spct,photodiode.spct,polyester.spct,sun.daily.data,sun.daily.spct,sun.data,sun.spct,white_body.spct,white_led.cps_spct,white_led.raw_spct,white_led.source_spct,yellow_gel.spct*

**Examples**

Ler_leaf_trns.spct
**Ler_leaf_trns_i.spct**  

*Green Arabidopsis leaf spectral transmittance.*

---

**Description**

A dataset of internal spectral transmittance expressed as a fraction of one from the upper surface of a leaf of an Arabidopsis thaliana 'Ler' rosette.

**Usage**

```
Ler_leaf_trns_i.spct
```

**Format**

An `filter_spct` object with 2401 rows and 2 variables

**Details**

- `w.length (nm)`
- `Tfr (0..1)`

**Note**

Measured with a Jaz spectrometer from Ocean Optics (USA) configured with a PX Xenon lamp module and Spectroclip double integrating spheres.

**Author(s)**

Aphalo, P. J. & Wang, F (unpublished data)

**See Also**


**Examples**

```
Ler_leaf_trns_i.spct
```
Logarithms and Exponentials

Description

Logarithms and Exponentials for Spectra. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of \( x \) and the current value of output options.

Usage

```r
## S3 method for class 'generic_spct'
log(x, base = exp(1))

log2.generic_spct(x)

log10.generic_spct(x)

## S3 method for class 'generic_spct'
exp(x)
```

Arguments

- **x**: an object of class "generic_spct"
- **base**: a positive number: the base with respect to which logarithms are computed. Defaults to \( e=\exp(1) \).

Value

An object of the same class as \( x \).

Note

In most cases a logarithm of an spectral quantity will yield off-range values. For this reason unless \( x \) is an object of base class generic_spct, checks will not be passed, resulting in warnings or errors.

See Also

Other math operators and functions: MathFun, \(^.\)generic_spct(), convolve_each(), div-.generic_spct, minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), slash-.generic_spct, times-.generic_spct
MathFun

Miscellaneous Mathematical Functions

Description

abs(x) computes the absolute value of x, sqrt(x) computes the (principal) square root of x. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of x and the current value of output options.

Usage

## S3 method for class 'generic_spct'
sqrt(x)

## S3 method for class 'generic_spct'
abs(x)

Arguments

x an object of class "generic_spct"

See Also

Other math operators and functions: ^\.generic_spct(), convolve_each(), div\.generic_spct, log(), minus\.generic_spct, mod\.generic_spct, plus\.generic_spct, round(), sign(), slash\.generic_spct, times\.generic_spct

merge2object_spct

Merge into object_spct

Description

Merge a filter_spct with a reflector_spct returning an object_spct object, even if wavelength values are mismatched.

Usage

merge2object_spct(
  x,
  y,
  by = "w.length",
  ...,
  w.length.out = x[["w.length"]],
  Tfr.type.out = "total"
)
merge_attributes

Arguments

\(x, y\)  
a filter_spect object and a reflector_spect object.

by  
a vector of shared column names in \(x\) and \(y\) to merge on; by defaults to \(w.length\).

...  
other arguments passed to dplyr::inner_join()

\(w.length.out\)  
numeric vector of wavelengths to be used for the returned object (nm).

\(Tfr.type.out\)  
character string indicating whether transmittance values in the returned object should be expressed as "total" or "internal". This applies only to the case when an object_spect is returned.

Value

An object_spect is returned as the result of merging a filter_spect and a reflector_spect object.

Note

If a numeric vector is supplied as argument for \(w.length.out\), the two spectra are interpolated to the new wavelength values before merging. The default argument for \(w.length.out\) is \(x[[w.length]]\).

See Also

join

merge_attributes    Merge and copy attributes

Description

Merge attributes from \(x\) and \(y\) and copy them to \(z\). Methods defined for spectral objects of classes from package 'photobiology'.

Usage

merge_attributes(x, y, z, which, which.not, ...)

## Default S3 method:
merge_attributes(x, y, z, which = NULL, which.not = NULL, ...)

## S3 method for class 'generic_spect'
merge_attributes(
  x,
  y,
  z,
  which = NULL,
  which.not = NULL,
  copy.class = FALSE,
  ...
)

### minus-.generic_spct

**Arguments**

- **x, y, z**: R objects. Objects x and y must be of the same class, z must be an object with a structure valid for this same class.
- **which**: character Names of attributes to copy, if NULL all those relevant according to the class of x are used as default.
- **which.not**: character Names of attributes not to be copied. The names passed here are removed from the list for which, which is most useful when we want to modify the default.
- **...**: not used
- **copy.class**: logical If TRUE class attributes are also copied.

**Value**

A copy of z with additional attributes set.

**Methods (by class)**

- default: Default for generic function
- generic_spct:

```r
 minus-.generic_spct Arithmetric Operators
```

**Description**

Subtraction operator for generic spectra.

**Usage**

```r
## S3 method for class 'generic_spct'
e1 - e2 = NULL
```

**Arguments**

- **e1**: an object of class "generic_spct"
- **e2**: an object of class "generic_spct"

**See Also**

Other math operators and functions: `MathFun`, `^.generic_spct()`, `convolve_each()`, `div-.generic_spct`, `log()`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`
mod-.generic_spct

**Arithmetic Operators**

---

**Description**

Reminder operator for generic spectra.

**Usage**

```r
## S3 method for class 'generic_spct'
e1 %% e2
```

**Arguments**

- `e1` an object of class "generic_spct"
- `e2` an object of class "generic_spct"

**See Also**

Other math operators and functions: `MathFun`, `^.generic_spct()`, `convolve_each()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`

---

**msmsply**

*Multi-spct transform methods*

---

**Description**

Apply a function or operator to a collection of spectra.

**Usage**

```r
msmsply(mspct, .fun, ..., .parallel = FALSE, .paropts = NULL)
msdply(
  mspct,
  .fun,
  ..., 
  idx = NULL,
  col.names = NULL,
  .parallel = FALSE,
  .paropts = NULL
)
mslply(mspct, .fun, ..., .parallel = FALSE, .paropts = NULL)
msaply(mspct, .fun, ..., .drop = TRUE, .parallel = FALSE, .paropts = NULL)
```
**mspct_classes**

**Arguments**

- **mspct**: an object of class `generic_mspct` or a derived class
- **.fun**: a function
- **...**: other arguments passed to `.fun`
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by `foreach`
- **.paropts**: a list of additional options passed into the `foreach` function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the `.export` and `.packages` arguments to supply them so that all cluster nodes have the correct environment set up for computing.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **col.names**: character Names to be used for data columns.
- **.drop**: should extra dimensions of length 1 in the output be dropped, simplifying the output. Defaults to TRUE

**Value**

a collection of spectra in the case of `msmsply`, belonging to a different class than `mspct` if `.fun` modifies the class of the member spectra.
a data frame in the case of `msdply`
a list in the case of `mslply`
an vector in the case of `msaply`

---

**mspct_classes**

*Names of multi-spectra classes*

**Description**

Function that returns a vector containing the names of multi-spectra classes using for collections of spectra.

**Usage**

```r
mspct_classes()
```

**Value**

A character vector of class names.

**Examples**

```r
mspct_classes()
```
Handle Missing Values in Objects

Description

These methods are useful for dealing with NAs in e.g., source_spct, response_spct, filter_spct and reflector_spct.

Usage

## S3 method for class 'generic_spct'
na.omit(object, na.action = "omit", fill = NULL, target.colnames, ...)

## S3 method for class 'source_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'response_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'filter_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'reflector_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'object_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'cps_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'raw_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'chroma_spct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'generic_mspct'
na.omit(object, na.action = "omit", fill = NULL, ...)

## S3 method for class 'generic_spct'
na.exclude(object, na.action = "exclude", fill = NULL, target.colnames, ...)

## S3 method for class 'source_spct'
na.exclude(object, na.action = "exclude", fill = NULL, ...)

## S3 method for class 'response_spct'
Arguments

- **object**: an R object
- **na.action**: character One of "omit", "exclude" or "replace".
- **fill**: numeric Value used to replace NAs unless NULL, in which case interpolation is attempted.
- **target.colnames**: character Vector of names for the target columns to operate upon, if present in object.
- **...**: further arguments other special methods could require.

Details

If `na.omit` removes cases, the row numbers of the cases form the "na.action" attribute of the result, of class "omit".

`na.exclude` differs from `na.omit` only in the class of the "na.action" attribute of the result, which is "exclude".

Note

`na.fail` and `na.pass` do not require a specialisation for spectral objects. R's definitions work as expected with no need to override them. We do not define a method `na.replace`, just pass "replace" as argument. The current implementation replaces by interpolation only individual NAs which are flanked on both sides by valid data. Runs of multiple NAs can only replaced by a constant value passed through parameter `fill`.
normalization

Normalization of an R object

Description
Normalization wavelength of an R object, retrieved from the object's attributes.

Usage

```r
normalization(x)

## Default S3 method:
normalization(x)

## S3 method for class 'waveband'
normalization(x)
```

Arguments

- `x`: an R object

Examples

```r
my_sun.spct <- sun.spct
my_sun.spct[3, "s.e.irrad"] <- NA
my_sun.spct[5, "s.q.irrad"] <- NA

head(my_sun.spct)

# rows omitted
zo <- na.omit(my_sun.spct)
head(zo)
na.action(zo)

# rows excluded
ze <- na.exclude(my_sun.spct)
head(ze)
na.action(ze)

# data in both rows replaced
zr <- na.omit(my_sun.spct, na.action = "replace")
head(zr)
na.action(zr)
```
**normalize**

**Methods (by class)**

- default: Default methods.
- waveband: Normalization of a waveband object.

**See Also**

Other waveband attributes: is_effective(), labels()

---

**normalize**

Normalize spectral data

**Description**

This method returns a spectral object of the same class as the one supplied as argument but with the spectral data normalized to 1.0 at a specific wavelength.

**Usage**

```
normalize(x, ...)

normalise(x, ...)
```

```r
## Default S3 method:
normalize(x, ...)
## S3 method for class 'source_spct'
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  unit.out =getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE
)
```

```r
## S3 method for class 'response_spct'
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  unit.out =getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE
)
```

```r
## S3 method for class 'filter_spct'
```
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  na.rm = FALSE
)

## S3 method for class 'reflector_spct'
normalize(x, ..., range = NULL, norm = "max", qty.out = NULL, na.rm = FALSE)

## S3 method for class 'raw_spct'
normalize(x, ..., range = NULL, norm = "max", na.rm = FALSE)

## S3 method for class 'cps_spct'
normalize(x, ..., range = NULL, norm = "max", na.rm = FALSE)

## S3 method for class 'generic_spct'
normalize(x, ..., range = NULL, norm = "max", col.names, na.rm = FALSE)

## S3 method for class 'source_mspct'
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
normalize(
  x,
  ..., 
  range = NULL,
  norm = "max",
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
normalize(
  x,
...,
  range = NULL,
  norm = "max",
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  na.rm = FALSE,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'reflector_mspct'
normalize(
  x,
  ...
)

## S3 method for class 'raw_mspct'
normalize(
  x,
  ...
)

## S3 method for class 'cps_mspct'
normalize(
  x,
  ...
)

Arguments

x An R object

... not used in current version

range An R object on which range() returns a numeric vector of length 2 with the
limits of a range of wavelengths in nm, with min and max wavelengths (nm) used to set boundaries for search for normalization.

- **norm**: numeric Normalization wavelength (nm) or character string "max", or "min" for normalization at the corresponding wavelength.
- **unit.out**: character Allowed values "energy", and "photon", or its alias "quantum"
- **na.rm**: logical indicating whether NA values should be stripped before calculating the summary (e.g. "max") used for normalization.
- **qty.out**: character string Allowed values are "transmittance", and "absorbance" indicating on which quantity to apply the normalization.
- **col.names**: character vector containing the names of columns or variables to which to apply the normalization.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) you code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

**Details**

By default normalization is done based on the maximum of the spectral data. It is possible to also do the normalization based on a user-supplied wavelength expressed in nanometres or a wavelength retrieved using an arbitrary R function applied to the spectrum. By default the function is applied to the whole spectrum, but by passing a range of wavelengths as input, the search can be limited to a region of interest within the spectrum.

**Value**

A copy of x, with spectral data values normalized to one for the criterion specified by the argument passed to norm.

A copy of x with the values of the spectral quantity rescaled to 1 at the normalization wavelength. If the normalization wavelength is not already present in x, it is added by interpolation—i.e. the returned value may be one row longer than x.

**Methods (by class)**

- **default**: Default for generic function
- **source_spct**: Normalize a source_spct object.
- **response_spct**: Normalize a response spectrum.
- **filter_spct**: Normalize a filter spectrum.
- **reflector_spct**: Normalize a reflector spectrum.
- **raw_spct**: Normalize a raw spectrum.
- **cps_spct**: Normalize a cps spectrum.
- **generic_spct**: Normalize a raw spectrum.
- **source_mspct**: Normalize the members of a source_mspct object.
- response_mspct: Normalize the members of a response_mspct object.
- filter_mspct: Normalize the members of a filter_mspct object.
- reflector_mspct: Normalize the members of a reflector_mspct object.
- raw_mspct: Normalize the members of a raw_mspct object.
- cps_mspct: Normalize the members of a cps_mspct object.

Note

normalise() is a synonym for this normalize() method.

1) By default if x contains one or more NA values and the normalization is based on a summary quantity, the returned spectrum will contain only NA values. If na.rm == TRUE then the summary quantity will be calculated after stripping NA values, and only the values that were NA in x will be NA values in the returned spectrum.

See Also

Other rescaling functions: fscale(), fshift(), getNormalized(), getScaled(), is_normalized(), is_scaled(), setNormalized(), setScaled()

Examples

```r
normalize(sun.spct)
normalise(sun.spct) # equivalent

normalize(sun.spct, norm = "max")
normalize(sun.spct, norm = 400)
```

---

**normalized_diff_ind**  
*Calculate a normalized index.*

**Description**

This method returns a normalized difference index value for an arbitrary pair of wavebands. There are many such indexes in use, such as NDVI (normalized difference vegetation index), NDWI (normalized difference water index), NDMI (normalized difference moisture index), etc., the only difference among them is in the wavebands used.

**Usage**

```r
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
normalised_diff_ind(spct, plus.w.band, minus.w.band, f, ...)
NDxI(spct, plus.w.band, minus.w.band, f, ...)
```
## Default S3 method:
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)

## S3 method for class "generic_spct"
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)

## S3 method for class "generic_mspct"
normalized_diff_ind(spct, plus.w.band, minus.w.band, f, ...)

### Arguments

- **spct**: an R object
- **plus.w.band**: waveband objects The waveband determine the region of the spectrum used in the calculations
- **minus.w.band**: waveband objects The waveband determine the region of the spectrum used in the calculations
- **f**: function used for integration taking spct as first argument and a list of wavebands as second argument.
- **...**: additional arguments passed to f

### Details

f is most frequently `reflectance`, but also `transmittance`, or even `absorbance`, `response`, `irradiance` or a user-defined function can be used if there is a good reason for it. In every case spct should be of the class expected by f. When using two wavebands of different widths do consider passing to f a suitable quantity argument. Wavebands can describe weighting functions if desired.

### Value

A named numeric value for the index, or a tibble depending on whether a spectrum or a collection of spectra is passed as first argument. If the wavelength range of spct does not fully overlap with both wavebands NA is silently returned.

### Methods (by class)

- default: default
- generic_spct:
- generic_mspct:

### Note

Some NDxI indexes are directly based on satellite instrument data, such as those in the Landsat satellites. To simulate such indexes using spectral reflectance as input, waveband definitions provided by package 'photobiologyWavebands' can be used.

`normalized_diff_ind()` is a synonym for `normalized_diff_ind()`.

`NDxI()` is a shorthand for `normalized_diff_ind()`. 
normalize_range_arg

Normalize a range argument into a true numeric range

Description

Several functions in this package and the suite accept a range argument with a flexible syntax. To ensure that all functions and methods behave in the same way this code has been factored out into a separate function.

Usage

normalize_range_arg(arg.range, wl.range, trim = TRUE)

Arguments

arg.range  
a numeric vector of length two, or any other object for which function range() will return a range of wavelengths (nm).

wl.range  
a numeric vector of length two, or any other object for which function range() will return a range of wavelengths (nm), missing values are not allowed.

trim  
logical If TRUE the range returned is bound within wl.range while if FALSE it can be broader.

Details

The arg.range argument can contain NAs which are replaced by the value at the same position in wl.range. In addition a NULL argument for range is converted into wl.range. The wl.range is also the limit to which the returned value is trimmed if trim == TRUE. The idea is that the value supplied as wl.range is the wavelength range of the data.

Value

a numeric vector of length two, guaranteed not to have missing values.

Examples

normalize_range_arg(c(NA, 500), range(sun.spct))
normalize_range_arg(c(300, NA), range(sun.spct))
normalize_range_arg(c(100, 5000), range(sun.spct), FALSE)
normalize_range_arg(c(NA, NA), range(sun.spct))
normalize_range_arg(c(NA, NA), sun.spct)
opaque.spct  Theoretical spectrum of an opaque material

Description
A dataset for a hypothetical object with transmittance 0/1 (0%)

Usage
opaque.spct

Format
A filter_spct object with 4 rows and 2 variables

Details
- w.length (nm).
- Tfr (0..1)

See Also

Examples
opaque.spct

oper_spectra  Binary operation on two spectra, even if the wavelengths values differ

Description
The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added.
**Usage**

```r
eroper_spectra(
    w.length1,
    w.length2 = NULL,
    s.irrad1,
    s.irrad2,
    trim = "union",
    na.rm = FALSE,
    bin.oper = NULL,
    ...
)
```

**Arguments**

- `w.length1`: numeric vector of wavelength (nm)
- `w.length2`: numeric vector of wavelength (nm)
- `s.irrad1`: a numeric vector of spectral values
- `s.irrad2`: a numeric vector of spectral values
- `trim`: a character string with value "union" or "intersection"
- `na.rm`: a logical value, if TRUE, not the default, NAs in the input are replaced with zeros
- `bin.oper`: a function defining a binary operator (for the usual math operators enclose argument in backticks)
- `...`: additional arguments (by name) passed to bin.oper

**Details**

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If w.length2==NULL, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

**Value**

A dataframe with two numeric variables

- `w.length`: A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique values, sorted in ascending order.
- `s.irrad`: A numeric vector with the sum of the two spectral values at each wavelength.

**See Also**

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `photon_irradiance()`, `photon_ratio()`,
peaks

Peaks or local maxima

Function that returns a subset of an R object with observations corresponding to local maxima.

Usage

peaks(x, span, ignore_threshold, strict, na.rm, ...)

## Default S3 method:
peaks(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)

## S3 method for class 'numeric'
peaks(x, span = 5, ignore_threshold = NA, strict = TRUE, na.rm = FALSE, ...)

## S3 method for class 'data.frame'
peaks(
x, 
span = 5, 
itrigor_threshold = 0, 
strict = TRUE, 
na.rm = FALSE, 
x.var.name = NULL, 
y.var.name = NULL, 
var.name = y.var.name, 

peaks

refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'generic_spct'
peaks(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
var.name = NULL,
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'source_spct'
peaks(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'response_spct'
peaks(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'filter_spct'
peaks(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'reflector_spct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'cps_spct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'raw_spct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_mspct'


peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'source_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'response_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  refine.wl = FALSE,
  method = "spline",
  ...
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'filter_mspct'
peaks(
  x,
  span = 5,
peaks

ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
refine.wl = FALSE,
method = "spline",
...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'reflector_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  refine.wl = FALSE,
  method = "spline",
  ...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'cps_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'raw_mspct'
peaks(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
refine.wl = FALSE,
method = "spline",
...
.parallel = FALSE,
.paropts = NULL
)

Arguments

x an R object
span integer A peak is defined as an element in a sequence which is greater than all other elements within a window of width span centered at that element. Use NULL for the global peak.
ignore_threshold numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.
strict logical If TRUE, an element must be strictly greater than all other values in its window to be considered a peak.
na.rm logical indicating whether NA values should be stripped before searching for peaks.
... ignored
var.name, x.var.name, y.var.name character Name of column where to look for peaks.
refine.wl logical Flag indicating if peak location should be refined by fitting a function.
method character String with the name of a method. Currently only spline interpolation is implemented.
unit.out character One of "energy" or "photon"
filter.qty character One of "transmittance" or "absorbance"
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A subset of x with rows corresponding to local maxima.

Methods (by class)

• default: Default returning always NA.
• numeric: Default function usable on numeric vectors.
• data.frame: Method for "data.frame" objects.
• generic_spct: Method for "generic_spct" objects.
photodiode.spct

- source_spct: Method for "source_spct" objects.
- filter_spct: Method for "filter_spct" objects.
- reflector_spct: Method for "reflector_spct" objects.
- cps_spct: Method for "cps_spct" objects.
- raw_spct: Method for "raw_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.
- source_mspct: Method for "source_mspct" objects.
- response_mspct: Method for "cps_mspct" objects.
- filter_mspct: Method for "filter_mspct" objects.
- reflector_mspct: Method for "reflector_mspct" objects.
- cps_mspct: Method for "cps_mspct" objects.
- raw_mspct: Method for "raw_mspct" objects.

Note

Thresholds for ignoring peaks are applied after peaks are searched for, and negative threshold values can in some cases result in no peaks being returned.

See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), replace_bad_pixs(), spikes(), valleys(), wls_at_target()

Examples

peaks(sun.spct, span = 51)
peaks(sun.spct, span = NULL)
peaks(sun.spct, span = 51, refine.wl = TRUE)

peaks(sun.spct)

photodiode.spct  
Spectral response of a GaAsP photodiode

Description

A dataset containing wavelengths at a 1 nm interval and spectral response as \( A/(W/nm) \) for GaAsP photodiode type G6262 from Hamamatsu. Data digitized from manufacturer’s data sheet. The value at the peak is 0.19 \( A/W \).

Usage

photodiode.spct
**photon:energy_ratio**

**Format**

A response_spct object with 94 rows and 2 variables

**Details**

- w.length (nm).
- s.e.response (A/W)

**References**


**See Also**


**Examples**

photodiode.spct

```
photon:energy_ratio(  
  w.length,  
  s.irrad,  
  w.band = NULL,  
  unit.in = "energy",  
  check.spectrum = TRUE,  
  use.cached.mult = FALSE,  
  use.hinges = getOption("photobiology.use.hinges", default = NULL)  
)
```
\textbf{photon\_irradiance}

\textbf{Arguments}

\begin{itemize}
\item \texttt{w.length} \hspace{5mm} numeric vector of wavelength (nm).
\item \texttt{s.irrad} \hspace{5mm} numeric vector of spectral (energy) irradiances (W m\(^{-2}\) nm\(^{-1}\)).
\item \texttt{w.band} \hspace{5mm} waveband object.
\item \texttt{unit.in} \hspace{5mm} character Allowed values "energy", and "photon", or its alias "quantum".
\item \texttt{check.spectrum} \hspace{5mm} logical Flag telling whether to sanity check input data, default is TRUE.
\item \texttt{use.cached.mult} \hspace{5mm} logical Flag telling whether multiplier values should be cached between calls.
\item \texttt{use.hinges} \hspace{5mm} logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
\end{itemize}

\textbf{Value}

A single numeric value giving the ratio moles-photons per Joule.

\textbf{Note}

The default for the \texttt{w.band} parameter is a waveband covering the whole range of \texttt{w.length}.

\textbf{See Also}

Other low-level functions operating on numeric vectors: \texttt{as\_energy()}, \texttt{as\_quantum\_mol()}, \texttt{calc\_multipliers()}, \texttt{div\_spectra()}, \texttt{energy\_irradiance()}, \texttt{energy\_ratio()}, \texttt{insert\_hinges()}, \texttt{integrate\_xy()}, \texttt{interpolate\_spectrum()}, \texttt{irradiance()}, \texttt{l\_insert\_hinges()}, \texttt{oper\_spectra()}, \texttt{photon\_irradiance()}, \texttt{photon\_ratio()}, \texttt{prod\_spectra()}, \texttt{s\_e\_irrad2rgb()}, \texttt{split\_energy\_irradiance()}, \texttt{split\_photon\_irradiance()}, \texttt{subt\_spectra()}, \texttt{sum\_spectra()}, \texttt{trim\_tails()}, \texttt{v\_insert\_hinges()}, \texttt{v\_replace\_hinges()}

\textbf{Examples}

\begin{verbatim}
# photons:energy ratio
with(sun.data, photons\_energy\_ratio(w.length, s.e.irrad, new\_waveband(400,500)))
# photons:energy ratio for whole spectrum
with(sun.data, photons\_energy\_ratio(w.length, s.e.irrad))
\end{verbatim}

---

\begin{tabular}{ll}
\textbf{photon\_irradiance} & \textbf{Photon irradiance} \\
\end{tabular}

\textbf{Description}

This function returns the photon irradiance for a given waveband of a radiation spectrum, optionally applies a BSWF.
Usage

```r
photon_irradiance(
  w.length,
  s.irrad,
  w.band = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

- `w.length` numeric vector of wavelength (nm).
- `s.irrad` numeric vector of spectral irradiances, by default as energy (W m-2 nm-1).
- `w.band` waveband.
- `unit.in` character Values recognized "photon" or "energy".
- `check.spectrum` logical Flag telling whether to sanity check input data, default is TRUE.
- `use.cached.mult` logical Flag telling whether multiplier values should be cached between calls.
- `use.hinges` logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A single numeric value with no change in scale factor: [mol s-1 m-2 nm-1] -> [mol s-1 m-2].

See Also

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s.e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```r
with(sun.data, photon_irradiance(w.length, s.e.irrad))
with(sun.data, photon_irradiance(w.length, s.e.irrad, new_waveband(400,700)))
```
Description

This function gives the photon ratio between two given wavebands of a radiation spectrum.

Usage

```r
photon_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

- `w.length`: numeric vector of wavelength (nm).
- `s.irrad`: numeric vector of spectral (energy or photon) irradiances (W m\(^{-2}\) nm\(^{-1}\)) or (mol s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)).
- `w.band.num`: waveband object used to compute the numerator of the ratio.
- `w.band.denom`: waveband object used to compute the denominator of the ratio.
- `unit.in`: character. Allowed values "energy", and "photon", or its alias "quantum".
- `check.spectrum`: logical Flag telling whether to sanity check input data, default is TRUE.
- `use.cached.mult`: logical Flag telling whether multiplier values should be cached between calls.
- `use.hinges`: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a single numeric value giving the unitless ratio.

Note

The default for both `w.band` parameters is a waveband covering the whole range of `w.length`. 

---

**photon_ratio**

Photo: photon ratio

---

**Description**

This function gives the photon ratio between two given wavebands of a radiation spectrum.

**Usage**

```r
photon_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

**Arguments**

- `w.length`: numeric vector of wavelength (nm).
- `s.irrad`: numeric vector of spectral (energy or photon) irradiances (W m\(^{-2}\) nm\(^{-1}\)) or (mol s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)).
- `w.band.num`: waveband object used to compute the numerator of the ratio.
- `w.band.denom`: waveband object used to compute the denominator of the ratio.
- `unit.in`: character. Allowed values "energy", and "photon", or its alias "quantum".
- `check.spectrum`: logical Flag telling whether to sanity check input data, default is TRUE.
- `use.cached.mult`: logical Flag telling whether multiplier values should be cached between calls.
- `use.hinges`: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

**Value**

a single numeric value giving the unitless ratio.

**Note**

The default for both `w.band` parameters is a waveband covering the whole range of `w.length`. 

---
See Also

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`, `v_replace_hinges()`

Examples

```r
with(sun.data,
    photon_ratio(w.length, s.e.irrad, new_waveband(400,500), new_waveband(400,700)))
```

Description

Division operator for generic spectra.

Usage

```r
## S3 method for class 'generic_spct'
e1 + e2 = NULL
```

Arguments

- `e1`: an object of class "generic_spct"
- `e2`: an object of class "generic_spct"

See Also

Other math operators and functions: `MathFun`, `^.-.generic_spct()`, `convolve_each()`, `div.-.generic_spct`, `log()`, `minus.-.generic_spct`, `mod.-.generic_spct`, `round()`, `sign()`, `slash.-.generic_spct`, `times.-.generic_spct`
polyester.spct  

Transmittance spectrum of clear polyester film

**Description**

A dataset containing the wavelengths at a 1 nm interval and fractional total transmittance for polyester film.

**Usage**

polyester.spct

**Format**

A `filter_spct` object with 611 rows and 2 variables

**Details**

- `w.length` (nm).
- `Tfr (0..1)`

**See Also**


**Examples**

polyester.spct

---

**print**

Print a spectral object

**Description**

Print method for objects of spectral classes.

**Usage**

```r
## S3 method for class 'generic_spct'
print(x, ..., n = NULL, width = NULL)

## S3 method for class 'generic_mspct'
print(x, ..., n = NULL, width = NULL, n.members = 10)
```
### Arguments

- **x**: An object of one of the summary classes for spectra. Not used in current version.
- **...**: Number of rows to show. If NULL, the default, will print all rows if less than option dplyr.print_max. Otherwise, will print dplyr.print_min.
- **width**: Width of text output to generate. This defaults to NULL, which means usegetOption("width") and only display the columns that fit on one screen. You can also set option(dplyr.width = Inf) to override this default and always print all columns.
- **n.members**: numeric Number of members of the collection to print.

### Value

Returns `x` invisibly.

### Methods (by class)

- **generic_mspct**:

### Note

At the moment just a modified copy of dplyr::print.tbl_df.

### Examples

```r
print(sun.spct)
print(sun.spct, n = 5)
```

---

### Description

Print solar time and solar date objects

### Usage

```r
## S3 method for class 'solar_time'
print(x, ...)

## S3 method for class 'solar_date'
print(x, ...)
```
Arguments

x an R object

... passed to format method

Note

Default is to print the underlying POSIXct as a solar time.

See Also

Other astronomy related functions: day_night(), format.solar_time(), format.tod_time(), is.solar_time(), print.tod_time(), solar_time(), sun_angles()
**print.tod_time**  
*Print time-of-day objects*

**Description**

Print time-of-day objects

**Usage**

```r
## S3 method for class 'tod_time'
print(x, ...)
```

**Arguments**

- `x`: an R object
- `...`: passed to format method

**Note**

Default is to print the underlying numeric vector as a solar time.

**See Also**

Other astronomy related functions: `day_night()`, `format.solar_time()`, `format.tod_time()`, `is.solar_time()`, `print.solar_time()`, `solar_time()`, `sun_angles()`

**print.waveband**  
*Print a "waveband" object*

**Description**

A function to more nicely print objects of class "waveband".

**Usage**

```r
## S3 method for class 'waveband'
print(x, ...)
```

**Arguments**

- `x`: an object of class "waveband"
- `...`: not used in current version
Multiply two spectra, even if the wavelengths values differ

Description

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added.

Usage

prod_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)

Arguments

w.length1 numeric vector of wavelength (nm).
w.length2 numeric vector of wavelength (nm).
s.irrad1 a numeric vector of spectral values.
s.irrad2 a numeric vector of spectral values.
trim a character string with value "union" or "intersection".
na.rm a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

Details

If trim=="union" spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If trim=="intersection" then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If w.length2==NULL, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.

Value

a dataframe with two numeric variables

w.length A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique values, sorted in ascending order.
s.irrad A numeric vector with the sum of the two spectral values at each wavelength.
q2e

See Also

Other low-level functions operating on numeric vectors: \texttt{as\_energy()}, \texttt{as\_quantum\_mol()}, \texttt{calc\_multipliers()}, \texttt{div\_spectra()}, \texttt{energy\_irradiance()}, \texttt{energy\_ratio()}, \texttt{insert\_hinges()}, \texttt{integrate\_xy()}, \texttt{interpolate\_spectrum()}, \texttt{irradiance()}, \texttt{l\_insert\_hinges()}, \texttt{oper\_spectra()}, \texttt{photon\_irradiance()}, \texttt{photon\_ratio()}, \texttt{photons\_energy\_ratio()}, \texttt{s\_e\_irrad2rgb()}, \texttt{split\_energy\_irradiance()}, \texttt{split\_photon\_irradiance()}, \texttt{subt\_spectra()}, \texttt{sum\_spectra()}, \texttt{trim\_tails()}, \texttt{v\_insert\_hinges()}, \texttt{v\_replace\_hinges()}

Examples

```r
head(sun.data)
square.sun.data <-
  with(sun.data, prod_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(square.sun.data)
tail(square.sun.data)
```

---

### q2e

Convert photon-based quantities into energy-based quantities

#### Description

Function that converts spectral photon irradiance (molar) into spectral energy irradiance.

#### Usage

```r
q2e(x, action, byref, ...)
```

### Examples

```r
head(sun.data)
square.sun.data <-
  with(sun.data, prod_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(square.sun.data)
tail(square.sun.data)
```
Arguments

- **x**: an R object
- **action**: a character string
- **byref**: logical indicating if new object will be created by reference or by copy of x
- **...**: not used in current version
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Methods (by class)

- **default**: Default method
- **source_spct**: Method for spectral irradiance
- **response_spct**: Method for spectral responsiveness
- **source_mspct**: Method for collections of (light) source spectra
- **response_mspct**: Method for collections of response spectra

See Also

Other quantity conversion functions: \texttt{A2T()}, \texttt{Afr2T()}, \texttt{T2Afr()}, \texttt{T2A()}, \texttt{any2T()}, \texttt{as_quantum()}, \texttt{e2qmol_multipliers()}, \texttt{e2quantum_multipliers()}, \texttt{e2q()}

\begin{verbatim}
qe_ratio

}\[Photon:energy ratio\]

Description

This function returns the photon to energy ratio for each waveband of a light source spectrum.

Usage

\begin{verbatim}
qe_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)

## Default S3 method:
qe_ratio(spct, w.band, scale.factor, wb.trim, use.cached.mult, use.hinges, ...)

## S3 method for class 'source_spct'
qe_ratio(
spct,
w.band = NULL,
scale.factor = 1,
wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
\end{verbatim}
use.cached.mult = FALSE,
use.hinges = NULL,
naming = "short",
name.tag = ifelse(naming != "none", "[q:e]", ""),
...
)

## S3 method for class 'source_mspct'
qe_ratio(
  spct,
  w.band = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[q:e]", ""),
  ...
  attr2tb = NULL,
  idx = "spct.idx",
  .parallel = FALSE,
  .paropts = NULL
)

Arguments

- `spct` source_spct.
- `w.band` waveband or list of waveband objects.
- `scale.factor` numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
- `wb.trim` logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- `use.cached.mult` logical Flag telling whether multiplier values should be cached between calls.
- `use.hinges` logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- `...` other arguments (possibly used by derived methods).
- `naming` character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- `name.tag` character Used to tag the name of the returned values.
- `attr2tb` character vector, see add_attr2tb for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
- `idx` character Name of the column with the names of the members of the collection of spectra.
- `.parallel` if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

Computed values are ratios between photon irradiance and energy irradiance for a given waveband. A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used, with “q:e” prepended. Units [mol J-1].

Methods (by class)

- default: Default for generic function
- source_spct: Method for source_spct objects
- source_mspct: Calculates photon:energy ratio from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other photon and energy ratio functions: e_ratio(), eq_ratio(), q_ratio()

Examples

qe_ratio(sun.spct, new_waveband(400,700))
**q_fluence**

**Photon fluence**

**Description**

Photon irradiance (i.e. quantum irradiance) for one or more waveband of a light source spectrum.

**Usage**

```r
q_fluence(
  spct,  # Spectrum object
  w.band,  # Waveband object
  exposure.time,  # Duration of exposure
  scale.factor,  # Scaling factor
  wb.trim,  # Waveband trim option
  use.cached.mult,  # Use cached multiplication factors
  use.hinges,  # Use hinges for scaling
  allow.scaled,  # Allow scaled input
  ...  # Additional arguments
)
```

## Default S3 method:
```r
default_q_fluence(
  spct,  # Spectrum object
  w.band,  # Waveband object
  exposure.time,  # Duration of exposure
  scale.factor = 1,  # Scaling factor
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),  # Waveband trim option
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),  # Use cached multiplication factors
  use.hinges = NULL,  # Use hinges for scaling
  allow.scaled = FALSE,  # Allow scaled input
  ...  # Additional arguments
)
```

## S3 method for class 'source_spct'
```r
source_spct_q_fluence(
  spct,  # Spectrum object
  w.band = NULL,  # Waveband object
  exposure.time,  # Duration of exposure
  scale.factor = 1,  # Scaling factor
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),  # Waveband trim option
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),  # Use cached multiplication factors
  use.hinges = NULL,  # Use hinges for scaling
  allow.scaled = FALSE,  # Allow scaled input
  naming = "default",  # Naming option
  ...  # Additional arguments
)
```
## S3 method for class 'source_mspct'

```r
q_fluence(
  spct,
  w.band = NULL,
  exposure.time,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
  use.hinges = NULL,
  allow.scaled = FALSE,
  naming = "default",
  ...
)
```

### Arguments

- `spct` an R object.
- `w.band` a list of waveband objects or a waveband object
- `exposure.time` lubridate::duration object.
- `scale.factor` numeric vector of length 1, or length equal to that of `w.band`. Numeric multiplier applied to returned values.
- `wb.trim` logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- `use.cached.mult` logical indicating whether multiplier values should be cached between calls.
- `use.hinges` logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- `allow.scaled` logical indicating whether scaled or normalized spectra as argument to `spct` are flagged as an error.
- `...` other arguments (possibly ignored).
- `naming` character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- `attr2tb` character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
- `idx` character Name of the column with the names of the members of the collection of spectra.
- `.parallel` if TRUE, apply function in parallel, using parallel backend provided by foreach
- `.paropts` a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the `.export` and `.packages` arguments to supply them so that all cluster nodes have the correct environment set up for computing.
q_irrad

Value

One numeric value for each waveband with no change in scale factor, with name attribute set to the name of each waveband unless a named list is supplied in which case the names of the list elements are used. The exposure.time is copied from the spectrum object to the output as an attribute. Units are as follows: moles of photons per exposure.

Methods (by class)

- default: Default for generic function
- source_spct: Calculate photon fluence from a source_spct object and the duration of the exposure
- source_mspct: Calculates photon (quantum) fluence from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other irradiance functions: e_fluence(), e_irrad(), fluence(), irrad(), q_irrad()

Examples

library(lubridate)
q_fluence(sun.spct,
  w.band = waveband(c(400,700)),
  exposure.time = lubridate::duration(3, "minutes") )
## Default S3 method:
q_irrad(
spct,
w.band, quantity, time.unit, scale.factor, wb.trim, use.cached.mult, use.hinges, allow.scaled, ...
)

## S3 method for class 'source_spct'
q_irrad(
spct,
w.band = NULL, quantity = "total", time.unit = NULL, scale.factor = 1,
wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
use.hinges = NULL,
allow.scaled = !quantity %in% c("average", "mean", "total"),
naming = "default",
...)

## S3 method for class 'source_mspct'
q_irrad(
spct,
w.band = NULL, quantity = "total", time.unit = NULL, scale.factor = 1,
wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
use.cached.mult = getOption("photobiology.use.cached.mult", default = FALSE),
use.hinges = NULL,
allow.scaled = !quantity %in% c("average", "mean", "total"),
Arguments

- **spct**: an R object.
- **w.band**: a list of waveband objects or a waveband object.
- **quantity**: character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
- **time.unit**: character or lubridate::duration object.
- **scale.factor**: numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
- **wb.trim**: logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.
- **use.cached.mult**: logical indicating whether multiplier values should be cached between calls.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **allow.scaled**: logical indicating whether scaled or normalized spectra as argument to spct are flagged as an error.
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and
optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used. The time.unit attribute is copied from the spectrum object to the output. Units are as follows: If time.unit is second, [W m-2 nm-1] -> [mol s-1 m-2] If time.units is day, [J d-1 m-2 nm-1] -> [mol d-1 m-2]

Methods (by class)

• default: Default for generic function
• source_spct: Calculates photon irradiance from a source_spct object.
• source_mspct: Calculates photon (quantum) irradiance from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other irradiance functions: e_fluence(), e_irrad(), fluence(), irrad(), q_fluence()

Examples

q_irrad(sun.spct, waveband(c(400,700)))
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3))
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "total")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "average")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "relative.pc")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution")
q_irrad(sun.spct, split_bands(c(400,700), length.out = 3), quantity = "contribution.pc")

q_ratio

**Photon:photon ratio**

Description

This function returns the photon ratio for a given pair of wavebands of a light source spectrum.
Usage

```r
q_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)
```

## Default S3 method:
```r
q_ratio(
  spct,
  w.band.num,
  w.band.denom,
  scale.factor,
  wb.trim,
  use.cached.mult,
  use.hinges,
  ...
)
```

## S3 method for class 'source_spct'
```r
q_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[q:q]", ""),
  ...
)
```

## S3 method for class 'source_mspct'
```r
q_ratio(
  spct,
  w.band.num = NULL,
  w.band.denom = NULL,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.cached.mult = FALSE,
  use.hinges = NULL,
  naming = "short",
  name.tag = ifelse(naming != "none", "[q:q]", ""),
  ...
)
```
name.tag = ifelse(naming != "none", "[q:q]", ""),
...,
attr2tb = NULL,
idx = "spct.idx",
.parallel = FALSE,
.paropts = NULL
)

Arguments

spct an object of class "source_spct".
w.band.num waveband object or a list of waveband objects used to compute the numerator(s) of the ratio(s).
w.band.denom waveband object or a list of waveband objects used to compute the denominator(s) of the ratio(s).
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded
use.cached.mult logical indicating whether multiplier values should be cached between calls
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
... other arguments (possibly ignored)
naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
name.tag character Used to tag the name of the returned values.
attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
idx character Name of the column with the names of the members of the collection of spectra.
.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

In the case of methods for individual spectra, a numeric vector of adimensional values giving a photon ratio between integrated photon irradiances for pairs of wavebands, with name attribute set to the name of the wavebands unless a named list of wavebands is supplied in which case the names of the list elements are used, with "(q:q)" appended. A data.frame in the case of collections of spectra, containing one column for each ratio definition, an index column with the names of the
spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Ratio definitions are "assembled" from the arguments passed to w.band.num and w.band.denom. If both arguments are of equal length, then the wavebands are paired to obtain as many ratios as the number of wavebands in each list. Recycling for wavebands takes place when the number of denominator and numerator wavebands differ.

Methods (by class)

- default: Default for generic function
- source_spct: Method for source_spct objects
- source_mspct: Calculates photon:photon from a source_mspct object.

Note

The last two parameters control speed optimizations. The defaults should be suitable in most cases. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other photon and energy ratio functions: e_ratio(), eq_ratio(), qe_ratio()

Examples

q_ratio(sun.spct, new_waveband(400,500), new_waveband(400,700))

q_response

Photon-based photo-response

Description

This function returns the mean response for a given waveband and a response spectrum.

Usage

q_response(
  spct,
  w.band,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
Arguments

spct an R object.
w.band waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.

quantity character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".

time.unit character or lubridate::duration object.
scale.factor numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.

wb.trim logical if TRUE wavebands crossing spectral data boundaries are trimmed, if FALSE, they are discarded.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

... other arguments (possibly used by derived methods).

naming character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.

attr2tb character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.

idx character Name of the column with the names of the members of the collection of spectra.

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- default: Default method for generic function
- response_mspct: Calculates photon (quantum) response from a response_mspct
Note

The parameter `use_hinges` controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

See Also

Other response functions: `e_response()`, `response()`

Examples

```r
q_response(ccd.spct, new_waveband(200,300))
q_response(photodiode.spct)
```

---

### r4p_pkgs

*Packages in R for Photobiology suite*

**Description**

A dataset containing the names of all the packages in this suite.

**Usage**

```r
r4p_pkgs
```

**Format**

A character vector.

**Details**

A character vector.

**Examples**

```r
r4p_pkgs
```
rbindspct

Row-bind spectra

Description

A wrapper on dplyr::rbind_fill that preserves class and other attributes of spectral objects.

Usage

rbindspct(
  l,
  use.names = TRUE,
  fill = TRUE,
  idfactor = TRUE,
  attrs.source = NULL
)

Arguments

l

A source_mspct, filter_mspct, reflector_mspct, response_mspct, chroma_mspct, cps_mspct, generic_mspct object or a list containing source_spct, filter_spct, reflector_spct, response_spct, chroma_spct, cps_spct, or generic_spct objects.

use.names

logical If TRUE items will be bound by matching column names. By default TRUE for rbindspct. Columns with duplicate names are bound in the order of occurrence, similar to base. When TRUE, at least one item of the input list has to have non-null column names.

fill

logical If TRUE fills missing columns with NAs. By default TRUE. When TRUE, use.names has also to be TRUE, and all items of the input list have to have non-null column names.

idfactor

logical or character Generates an index column of factor type. Default is (idfactor=TRUE) for both lists and _mspct objects. If idfactor=TRUE then the column is auto named spct.idx. Alternatively the column name can be directly provided to idfactor as a character string.

attrs.source

integer Index into the members of the list from which attributes should be copied. If NULL, all attributes are merged.

Details

Each item of l should be a spectrum, including NULL (skipped) or an empty object (0 rows). rbindspc is most useful when there are a variable number of (potentially many) objects to stack. rbindspct always returns at least a generic_spct as long as all elements in l are spectra.
Value

An spectral object of a type common to all bound items containing a concatenation of all the items passed in. If the argument `idfactor` is TRUE, then a factor `spct.idx` will be added to the returned spectral object.

Note

Note that any additional 'user added' attributes that might exist on individual items of the input list will not be preserved in the result. The attributes used by the photobiology package are preserved, and if they are not consistent across the bound spectral objects, a warning is issued.

dplyr::rbind_fill is called internally and the result returned is the highest class in the inheritance hierarchy which is common to all elements in the list. If not all members of the list belong to one of the _spct classes, an error is triggered. The function sets all data in source_spct and response_spct objects supplied as arguments into energy-based quantities, and all data in filter_spct objects into transmittance before the row binding is done. If any member spectrum is tagged, it is untagged before row binding.

Examples

```r
# default, adds factor 'spct.idx' with letters as levels
spct <- rbindspct(list(sun.spct, sun.spct))
spct
class(spct)

# adds factor 'spct.idx' with letters as levels
spct <- rbindspct(list(sun.spct, sun.spct), idfactor = TRUE)
head(spct)
class(spct)

# adds factor 'spct.idx' with the names given to the spectra in the list
# supplied as formal argument 'l' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct), idfactor = TRUE)
head(spct)
class(spct)

# adds factor 'ID' with the names given to the spectra in the list
# supplied as formal argument 'l' as levels
spct <- rbindspct(list(one = sun.spct, two = sun.spct),
                     idfactor = "ID")
head(spct)
class(spct)
```
**Description**

Function to calculate the mean, total, or other summary of reflectance for spectral data stored in a `reflector_spct` or in an `object_spct`.

**Usage**

```r
reflectance(spct, w.band, quantity, wb.trim, use.hinges, ...)
```

## Default S3 method:
reflectance(spct, w.band, quantity, wb.trim, use.hinges, ...)

## S3 method for class 'reflector_spct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'object_spct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

## S3 method for class 'reflector_mspct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ..., attr2tb = NULL, idx = "spct.idx", .parallel = FALSE, .paropts = NULL
)
## S3 method for class 'object_mspct'
reflectance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
  ...
)

### Arguments

- **spct**: an R object
- **w.band**: waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
- **quantity**: character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc"
- **wb.trim**: logical Flag telling if wavebands crossing spectral data boundaries are trimmed or ignored
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...**: other arguments
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

### Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter `w.band`. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and
optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- default: Default for generic function
- reflector_spct: Specialization for reflector_spct
- object_spct: Specialization for object_spct
- reflector_mspct: Calculates reflectance from a reflector_mspct
- object_mspct: Calculates reflectance from a object_mspct

Note

The use.hinges parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

Examples

reflectance(black_body.spct, waveband(c(400,700)))
reflectance(white_body.spct, waveband(c(400,700)))

<table>
<thead>
<tr>
<th>relative_AM</th>
<th>Relative Air Mass (AM)</th>
</tr>
</thead>
</table>

Description

Approximate relative air mass (AM) from sun elevation or sun zenith angle.

Usage

relative_AM(elevation.angle = NULL, zenith.angle = NULL, occluded.value = NA)

Arguments

elevation.angle, zenith.angle
numeric vector Angle in degrees for the sun position. An argument should be passed to one and only one of elevation_angle or zenith_angle.

occluded.value numeric Value to return when elevation angle is negative (sun below the horizon).
Details

This is an implementation of equation (3) in Kasten and Young (1989). This equation is only an approximation to the tabulated values in the same paper. Returned values are rounded to three significant digits.

Note

Although relative air mass is not defined when the sun is not visible, returning a value different from the default NA might be useful in some cases.

References


Examples

```r
relative_AM(c(90, 60, 30, 1, -10))
relative_AM(c(90, 60, 30, 1, -10), occluded.value = Inf)
relative_AM(zenith.angle = 0)
```

replace_bad_pixs  
Replace bad pixels in a spectrum

Description

This function replaces data for bad pixels by a local estimate, by either simple interpolation or using the algorithm of Whitaker and Hayes (2018).

Usage

```r
replace_bad_pixs(
  x,
  bad.pix.idx = FALSE,
  window.width = 11,
  method = "run.mean",
  na.rm = TRUE
)
```

Arguments

- **x**: numeric vector containing spectral data.
- **bad.pix.idx**: logical vector or integer. Index into bad pixels in x.
- **window.width**: integer. The full width of the window used for the running mean.
replace_bad_pixs

method character The name of the method: "run.mean" is running mean as described in Whitaker and Hayes (2018); "adj.mean" is mean of adjacent neighbors (isolated bad pixels only).

na.rm logical Treat NA values as additional bad pixels and replace them.

Details

Simple interpolation replaces values of isolated bad pixels by the mean of their two closest neighbors. The running mean approach allows the replacement of short runs of bad pixels by the running mean of neighboring pixels within a window of user-specified width. The first approach works well for spectra from array spectrometers to correct for hot and dead pixels in an instrument. The second approach is most suitable for Raman spectra in which spikes triggered by radiation are wider than a single pixel but usually not more than five pixels wide.

Value

A logical vector of the same length as x. Values that are TRUE correspond to local spikes in the data.

Note

In the current implementation NA values are not removed, and if they are in the neighborhood of bad pixels, they will result in the generation of additional NAs during their replacement.

References


See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), peaks(), spikes(), valleys(), wls_at_target()

Examples

# in a vector
replace_bad_pixs(c(1, 1, 45, 1, 1), bad.pix.idx = 3)

# before replacement
white_led.raw_spct$counts_3[120:125]

# replacing bad pixels at index positions 123 and 1994
with(white_led.raw_spct,
    replace_bad_pixs(counts_3, bad.pix.idx = c(123, 1994)))[120:125]
response  Integrated response

Description

Calculate average photon- or energy-based photo-response.

Usage

response(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## Default S3 method:
response(
  spct,
  w.band,
  unit.out,
  quantity,
  time.unit,
  scale.factor,
  wb.trim,
  use.hinges,
  ...
)

## S3 method for class 'response_spct'
response(
  spct,
  w.band = NULL,
  unit.out =getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim =getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges =getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)
## S3 method for class 'response_mspct'
response(
  spct,
  w.band = NULL,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  quantity = "total",
  time.unit = NULL,
  scale.factor = 1,
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = getOption("photobiology.use.hinges", default = NULL),
  naming = "default",
  ...
)

### Arguments

- **spct**: an R object of class "generic_spct".
- **w.band**: waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
- **unit.out**: character Allowed values "energy", and "photon", or its alias "quantum".
- **quantity**: character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
- **time.unit**: character or lubridate::duration object.
- **scale.factor**: numeric vector of length 1, or length equal to that of w.band. Numeric multiplier applied to returned values.
- **wb.trim**: logical Flag telling if wavebands crossing spectral data boundaries are trimmed or ignored.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...**: other arguments (possibly used by derived methods).
- **naming**: character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb**: character vector, see add_attr2tb for the syntax for attr2tb passed as is to formal parameter col.names.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach.
.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter w.band. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

Whether returned values are expressed in energy-based or photon-based units depends on unit.out. By default values are only integrated, but depending on the argument passed to parameter quantity they can be re-expressed as relative fractions or percentages. In the case of vector output, names attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

Methods (by class)

- default: Default for generic function
- response_mspct: Calculates response from a response_mspct

Note

The parameter use.hinges controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.

See Also

Other response functions: e_response(), q_response()

Rfr_from_n  

Reflectance at a planar boundary

Description

The reflectance at the planar boundary between two media, or interface, can be computed from the relative refractive index. Reflectance depends on polarization, and the process of reflection can generate polarized light through selective reflection of s and p components. A perfectly flat (i.e. polished) interface creates specular reflection, and this is the case that these functions describe. These function describe a single interface, and for example in a glass pane, a light beam will cross two air-glass interfaces.
Usage

Rfr_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5, p_fraction = 0.5)

Rfr_p_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5)

Rfr_s_from_n(angle_deg, angle = angle_deg/180 * pi, n = 1.5)

Arguments

angle_deg, angle
numeric vector Angle of incidence of the light beam, in degrees or radians. If both are supplied, radians take precedence.

n
numeric vector, or generic_spct object Relative refractive index. The default 1.5 is suitable for crown glass or acrylic interacting with visible light. n depends on wavelength, more or less strongly depending on the material.

p_fraction
numeric in range 0 to 1. Polarization, defaults to 0.5 assuming light that is not polarized.

Details

These functions implement Fresnel’s formulae. All parameters accept vectors as arguments. If both n and angle are vectors with length different from one, they should both have the same length. Reflectance depends on polarization, the $s$ and $p$ components need to be computed separately and added up. Rfr_from_n() is for non-polarized light, i.e., with equal contribution of the two components.

Value

If n is a numeric vector the returned value is a vector of reflectances, while if n is a generic_spct object the returned value is a reflector_spct object.

Examples

Rfr_from_n(0:90)
Rfr_from_n(0:90, p_fraction = 1)
Rfr_from_n(0:90, n = 1.333) # water

rgb_spct

RGB color values

Description

This function returns the RGB values for a source spectrum.
Usage

\[
\text{rgb\_spct}(\text{spct}, \text{sens} = \text{photobiology}\_\text{proposed}\_\text{human}\_\text{CMF}\_\text{CMF2.spct}, \text{color.name} = \text{NULL})
\]

Arguments

- \text{spct}: an object of class "source\_spct"
- \text{sens}: a \text{chroma\_spct} object with variables \text{w.length}, \text{x}, \text{y}, \text{z}, giving the \text{CC} or \text{CMF definition} (default is the proposed human CMF according to CIE 2006.)
- \text{color.name}: character string for naming the \text{rgb} color definition

Value

A color defined using \text{rgb()}. The numeric values of the RGB components can be obtained.

See Also

Other color functions: \text{w\_length2rgb()}, \text{w\_length\_range2rgb()}

Examples

\[
\text{rgb\_spct} (\text{sun.spct})
\]

---

\section*{rmDerivedMspct}

\textit{Remove "generic\_mspct" and derived class attributes.}

Description

Removes from a spectrum object the class attributes "generic\_mspct" and any derived class attribute such as "source\_mspct". \textbf{This operation is done by reference!}

Usage

\[
\text{rmDerivedMspct}(\text{x})
\]

Arguments

- \text{x}: an R object.

Value

A character vector containing the removed class attribute values. This is different to the behaviour of function \text{unlist} in base R!

Note

If \text{x} is an object of any of the multi spectral classes defined in this package, this function changes by reference the multi spectrum object into the underlying list object. Otherwise, it just leaves \text{x} unchanged. The modified \text{x} is also returned invisibly.
See Also
Other set and unset 'multi spectral' class functions: `shared_member_class()`

---

### Description

Removes from a spectrum object the class attributes "generic_spct" and any derived class attribute such as "source_spct". **This operation is done by reference!**

### Usage

```r
rmDerivedSpct(x, keep.classes = NULL)
```

### Arguments

- **x**: an R object.
- **keep.classes**: character vector Names of classes to keep. Can be used to retain base class "generic_spct".

### Value

A character vector containing the removed class attribute values. This is different to the behaviour of function `unlist` in base R!

### Note

If x is an object of any of the spectral classes defined in this package, this function changes by reference the spectrum object into the underlying data.frame object. Otherwise, it just leaves x unchanged.

This function alters x itself by reference. If x is not a generic_spct object, x is not modified.

### See Also

Other set and unset spectral class functions: `setGenericSpct()`

### Examples

```r
my.spct <- sun.spct
removed <- rmDerivedSpct(my.spct)
removed
class(sun.spct)
class(my.spct)
```
round

Rounding of Numbers

Description

ceiling takes a single numeric argument x and returns a numeric vector containing the smallest integers not less than the corresponding elements of x. \floor takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x. \trunc takes a single numeric argument x and returns a numeric vector containing the integers formed by truncating the values in x toward 0. \round rounds the values in its first argument to the specified number of decimal places (default 0). \signif rounds the values in its first argument to the specified number of significant digits. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of x and the current value of output options.

Usage

## S3 method for class 'generic_spct'
round(x, digits = 0)

## S3 method for class 'generic_spct'
signif(x, digits = 6)

## S3 method for class 'generic_spct'
ceiling(x)

## S3 method for class 'generic_spct'
floor(x)

## S3 method for class 'generic_spct'
trunc(x, ...)

Arguments

x an object of class "generic_spct" or a derived class.
digits integer indicating the number of decimal places (round) or significant digits (signif) to be used. Negative values are allowed (see 'Details').

Arguments to be passed to methods.

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, sign(), slash-.generic_spct, times-.generic_spct
select_spct_attributes

Merge user supplied attribute names with default ones

Description

Allow users to add and subtract from default attributes in addition to providing a given set of attributes.

Usage

```r
select_spct_attributes(attributes, attributes.default = spct_attributes())
spct_attributes(.class = "all", attributes = "+")
```

Arguments

- `attributes`, `attributes.default`
  - character vector or a list of character vectors.
- `.class`
  - character Name of spectral class.

Details

Vectors of character strings passed as argument to `attributes` are parsed so that if the first member string is "+", the remaining members are added to those in `attributes.default`; if it is "-" the remaining members are removed from in `attributes.default`; and if it is "=" the remaining members replace those in in `attributes.default`. If the first member is none of these three strings, the behaviour is the same as when the first string is "=". If `attributes` is NULL all the attributes in `attributes.default` are used and if it is "" no attribute names are returned. "" has precedence over other member values. The order of the names of annotations has no meaning: the vector is interpreted as a set except for the three possible "operators" at position 1.

Value

A character vector of attribute names.

See Also

`get_attributes`

setBSWFUsed  

Set the "bswf.used" attribute

Description

Function to set by reference the "time.unit" attribute of an existing source_spct object.

Usage

setBSWFUsed(x, bswf.used = c("none", "unknown"))

Arguments

x  a source_spct object

bswf.used  a character string, either "none" or the name of a BSWF

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a source_spct, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter bswf.used is used only if x does not already have this attribute set. time.unit = "hour" is currently not fully supported.

See Also

Other BSWF attribute functions: getBSWFUsed()

setFilterProperties  

Set the "filter.properties" attribute

Description

Function to set by reference the "filter.properties" attribute of an existing filter_spct object.
Usage

```r
setFilterProperties(
  x,
  filter.properties = NULL,
  pass.null = FALSE,
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA_character_
)
```

```r
filter_properties(x) <- value
```

Arguments

- **x**: a filter_spct object
- **filter.properties, value**: a list with fields named "Rfr.constant", "thickness" and "attenuation.mode".
- **pass.null**: logical If TRUE, the parameters to the next three parameters will be always ignored, otherwise they will be used to build an object of class "filter.properties" when the argument to filter.properties is NULL.
- **Rfr.constant**: numeric The value of the reflection factor (/1).
- **thickness**: numeric The thickness of the material.
- **attenuation.mode**: character One of "reflection", "absorption", "absorption.layer" or "mixed".

Details

Storing filter properties allows inter-conversion between internal and total transmittance, as well as computation of transmittance for arbitrary thickness of the material. Whether computations are valid depend on the homogeneity of the material. The parameter pass.null makes it possible to remove the attribute.

Value

**x**

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a filter_spct object, x is not modified.

The values of "attenuation.mode" "reflection" and "absorption" should be used when one of these processes is clearly the main one; "mixed" is for cases when they both play a role, i.e., when a simple correction using a single value of Rfr across wavelengths is not possible; "absorption.layer" is for cases when a thin absorbing layer is deposited on the surface of a transparent support or enclosed between two sheets of glass or other transparent material. If in doubt, set this to NA to ensure that computation of spectra for other thicknesses remains disabled.
See Also


Examples

my.spct <- polyester.spct
filter_properties(my.spct)
filter_properties(my.spct) <- NULL
filter_properties(my.spct)
filter_properties(my.spct, return.null = TRUE)
filter_properties(my.spct) <- list(Rfr.constant = 0.01,
                                 thickness = 125e-6,
                                 attenuation.mode = "absorption")
filter_properties(my.spct)

---

setGenericSpct

Convert an R object into a spectrum object.

Description

Sets the class attribute of a data.frame or an object of a derived class to "generic_spct".

Usage

setGenericSpct(x, multiple.wl = 1L, idfactor = NULL)

setCalibrationSpct(
  x,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setRawSpct(
  x,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL
)

setCpsSpct(
  x,
setGenericSpct

time.unit = "second",
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL
)

setFilterSpct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.constant = NA_real_,
  thickness = NA_real_,
  attenuation.mode = NA,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL
)

setReflectorSpct(
  x,
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL
)

setObjectSpct(
  x,
  Tfr.type = c("total", "internal"),
  Rfr.type = c("total", "specular"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL
)

setResponseSpct(
  x,
  time.unit = "second",
  response.type = "response",
multiple.wl = 1L,
idfactor = NULL
)

setSourceSpct(
  x,
  time.unit = "second",
  bswf.used = c("none", "unknown"),
  strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
setGenericSpct

    idfactor = NULL

) setChromaSpct(x, multiple.wl = 1L, idfactor = NULL)

Arguments

x data.frame, list or generic_spct and derived classes
multiple.wl numeric Maximum number of repeated w.length entries with same value.
idfactor character Name of factor distinguishing multiple spectra when stored logitudi-
nally (required if multipie.wl > 1).
strict.range logical Flag indicating whether off-range values result in an error instead of a
warning.
time.unit character string indicating the time unit used for spectral irradiance or exposure
("second", "day" or "exposure") or an object of class duration as defined in
package lubridate.
Tfr.type character A string, either "total" or "internal".
Rfr.constant numeric The value of the reflection factor (/1).
thickness numeric The thickness of the material.
attenuation.mode character One of "reflection", "absorption" or "mixed".
Rfr.type character A string, either "total" or "specular".
response.type a character string, either "response" or "action".
bswf.used character A string, either "none" or the name of a BSWF.

Value

x

Functions

- setCalibrationSpct: Set class of a an object to "calibration_spct".
- setRawSpct: Set class of a an object to "raw_spct".
- setCpsSpct: Set class of a an object to "cps_spct".
- setFilterSpct: Set class of an object to "filter_spct".
- setReflectorSpct: Set class of a an object to "reflector_spct".
- setObjectSpct: Set class of an object to "object_spct".
- setResponseSpct: Set class of an object to "response_spct".
- setSourceSpct: Set class of an object to "source_spct".
- setChromaSpct: Set class of an object to "chroma_spct".
Note
This method alters x itself by reference and in addition returns x invisibly.

For non-diffusing materials like glass an approximate Rfr.constant value can be used to interconvert "total" and "internal" transmittance values. Use NA if not known, or not applicable, e.g., for materials subject to internal scattering.

See Also
Other set and unset spectral class functions: rmDerivedSpct()

Examples

my.df <- data.frame(w.length = 300:309, s.e.irrad = rep(100, 10))
is.source_spct(my.df)
setSourceSpct(my.df)
is.source_spct(my.df)

setHowMeasured(x, how.measured)

how_measured(x) <- value

Arguments

x a generic_spct object
how.measured, value a list

Value

x

Note
This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct object, x is not modified.
setIdFactor

See Also

Examples

    my.spct <- sun.spct
    how_measured(my.spct)
    how_measured(my.spct) <- "simulated with a radiation transfer model"
    how_measured(my.spct)

---

setIdFactor Set the "idfactor" attribute

Description

Function to set by reference the "idfactor" attribute of an existing generic_spct or an object of a class derived from generic_spct.

Usage

setIdFactor(x, idfactor)

Arguments

  x a generic_spct object
  idfactor character The name of a factor identifying multiple spectra stored longitudinally.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct or an object of a class derived from generic_spct, x is not modified.

See Also

Other idfactor attribute functions: getIdFactor()
**setInstrDesc**

*Set the "instr.desc" attribute*

---

**Description**

Function to set by reference the "instr.desc" attribute of an existing generic_spct or derived-class object.

**Usage**

```r
setInstrDesc(x, instr.desc)
```

**Arguments**

- `x`: a generic_spct object
- `instr.desc`: a list

**Value**

`x`

**Note**

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a generic_spct object, `x` is not modified.

**See Also**


---

**setInstrSettings**

*Set the "instr.settings" attribute*

---

**Description**

Function to set by reference the "what.measured" attribute of an existing generic_spct or derived-class object.

**Usage**

```r
setInstrSettings(x, instr.settings)
```
Arguments

- `x`: a `generic_spct` object
- `multiple.wl`: a list

Value

- `x`

Note

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a `generic_spct` object, `x` is not modified.

See Also

setNormalized

See Also

Other multiple.wl attribute functions: `getMultipleWl()`

---

**setNormalized**  
*Set the "normalized" attribute*

**Description**

Function to write the "normalized" attribute of an existing generic_spct object.

**Usage**

```r
setNormalized(x, norm = FALSE)
setNormalised(x, norm = FALSE)
```

**Arguments**

- `x`  
a generic_spct object
- `norm`  
numeric or logical

**Note**

if `x` is not a generic_spct object, `x` is not modified.  
`setNormalised()` is a synonym for this `setNormalized()` method.

**See Also**

Other rescaling functions: `fscale()`, `fshift()`, `getNormalized()`, `getScaled()`, `is_normalized()`, `is_scaled()`, `normalize()`, `setScaled()`

---

setResponseType

**Set the "response.type" attribute**

**Description**

Function to set by reference the "response.type" attribute of an existing response_spct object.

**Usage**

```r
setResponseType(x, response.type = c("response", "action"))
```

**Arguments**

- `x`  
a response_spct object
- `response.type`  
a character string, either "response" or "action"
Details

Objects of class response_spct() can contain data for a response spectrum or an action spectrum. Response spectra are measured using the same photon (or energy) irradiance at each wavelength. Action spectra are derived from dose response curves at each wavelength, and responsivity at each wavelength is expressed as the reciprocal of the photon fluence required to obtain a fixed level of response.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a response_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter response.type is used only if x does not already have this attribute set.

Examples

my.spct <- ccd.spct
setResponseTypemy.spct, "action")

setRfrType

Set the "Rfr.type" attribute

Description

Function to set by reference the "Rfr.type" attribute of an existing reflector_spct or object_spct object.

Usage

setRfrType(x, Rfr.type = c("total", "specular"))

Arguments

x a reflector_spct or an object_spct object
Rfr.type a character string, either "total" or "specular"

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a reflector_spct or object_spct object, x is not modified. The behaviour of this function is 'unusual' in that the default for parameter Rfr.type is used only if x does not already have this attribute set.
setScaled

See Also
Other Rfr attribute functions: getRfrType()

Examples
my.spct <- reflector_spct(w.length = 400:409, Rfr = 0.1)
getRfrType(my.spct)
setRfrType(my.spct, "specular")
getRfrType(my.spct)

Description
Function to write the "scaled" attribute of an existing generic_spct object.

Usage
setScaled(x, ...)

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

## S3 method for class 'generic_spct'
setScaled(x, ..., scaled = FALSE)

## S3 method for class 'summary_generic_spct'
setScaled(x, ..., scaled = FALSE)

## S3 method for class 'generic_mspct'
setScaled(x, ..., scaled = FALSE)

Arguments
x
... a generic_spct object.
currently ignored.
scaled logical with FALSE meaning that values are expressed in absolute physical units and TRUE meaning that relative units are used. If NULL the attribute is not modified.

Value
a new object of the same class as x.
Methods (by class)
• default: Default for generic function
• generic_spct: Specialization for generic_spct
• summary_generic_spct: Specialization for summary_generic_spct
• generic_mspct: Specialization for generic_mspct

Note
if x is not a generic_spct object, x is not modified.

See Also
Other rescaling functions: fscale(), fshift(), getNormalized(), getScaled(), is_normalized(), is_scaled(), normalize(), setNormalized()

setTfrType

Set the "Tfr.type" attribute

Description
Function to set by reference the "Tfr.type" attribute of an existing filter_spct or object_spct object

Usage
setTfrType(x, Tfr.type = c("total", "internal"))

Arguments
x a filter_spct or an object_spct object
Tfr.type a character string, either "total" or "internal"

Value
x

Note
This function alters x itself by reference and in addition returns x invisibly. If x is not a filter_spct or an object_spct object, x is not modified The behaviour of this function is 'unusual' in that the default for parameter Tfr.type is used only if x does not already have this attribute set.

See Also
Other Tfr attribute functions: getTfrType()
**setTimeUnit**

### Examples

```r
my.spct <- polyester.spct
getTfrType(my.spct)
setTfrType(my.spct, "internal")
getTfrType(my.spct)
```

---

**setTimeUnit**  
*Set the "time.unit" attribute of an existing source_spct object*

---

**Description**

Function to set by reference the "time.unit" attribute

**Usage**

```r
setTimeUnit(
  x,
  time.unit = c("second", "hour", "day", "exposure", "none"),
  override.ok = FALSE
)
```

**Arguments**

- `x`  
  a source_spct object
- `time.unit`  
  character string indicating the time unit used for spectral irradiance or exposure ("second", "day" or "exposure") or an object of class duration as defined in package lubridate.
- `override.ok`  
  logical Flag that can be used to silence warning when overwriting an existing attribute value (used internally)

**Value**

`x`

**Note**

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a source_spct or response_spct object, `x` is not modified. The behaviour of this function is 'unusual' in that the default for parameter `time.unit` is used only if `x` does not already have this attribute set. `time.unit` = "hour" is currently not fully supported.

**See Also**

Other time attribute functions: `checkTimeUnit()`, `convertTfrType()`, `convertThickness()`, `convertTimeUnit()`, `getTimeUnit()`
Examples

```r
my.spct <- sun.spct
setTimeUnit(my.spct, time.unit = "second")
setTimeUnit(my.spct, time.unit = lubridate::duration(1, "seconds"))
```

---

setWhatMeasured: **Set the "what.measured" attribute**

Description

Function to set by reference the "what.measured" attribute of an existing generic_spct or derived-class object.

Usage

```r
setWhatMeasured(x, what.measured)
```

what_measured(x) <- value

Arguments

- `x`: a generic_spct object
- `what.measured`, `value`: a list

Value

`x`

Note

This function alters `x` itself by reference and in addition returns `x` invisibly. If `x` is not a generic_spct object, `x` is not modified.

See Also


Examples

```r
my.spct <- sun.spct
what_measured(my.spct)
what_measured(my.spct) <- "Sun"
what_measured(my.spct)
```
setWhenMeasured  

**Set the "when.measured" attribute**

**Description**

Function to set by reference the "when" attribute of an existing generic_spct or an object of a class derived from generic_spct.

**Usage**

```r
setWhenMeasured(x, when.measured, ...)  
when_measured(x) <- value  

## Default S3 method:  
setWhenMeasured(x, when.measured, ...)  

## S3 method for class 'generic_spct'  
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)  

## S3 method for class 'summary_generic_spct'  
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)  

## S3 method for class 'generic_mspct'  
setWhenMeasured(x, when.measured = lubridate::now(tzone = "UTC"), ...)  
```

**Arguments**

- `x`  
  a generic_spct object  
- `when.measured`, `value`  
  POSIXct to add as attribute, or a list of POSIXct.  
- `...`  
  Allows use of additional arguments in methods for other classes.

**Value**

- `x`

**Methods (by class)**

- default: default  
- generic_spct: generic_spct  
- summary_generic_spct: summary_generic_spct  
- generic_mspct: generic_mspct
Note

This method alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct or an object of a class derived from generic_spct, x is not modified. If when is not a POSIXct object or NULL an error is triggered. A POSIXct describes an instant in time (date plus time-of-day plus time zone).

See Also


Examples

my.spct <- sun.spct
when_measured(my.spct)
when_measured(my.spct) <- lubridate::ymd_hms("2020-01-01 08:00:00")
when_measured(my.spct)

---

setWhereMeasured  Set the "where.measured" attribute

Description

Function to set by reference the "where.measured" attribute of an existing generic_spct or an object of a class derived from generic_spct.

Usage

setWhereMeasured(x, where.measured, lat, lon, address, ...)

where_measured(x) <- value

## Default S3 method:
setWhereMeasured(x, where.measured, lat, lon, address, ...)

## S3 method for class 'generic_spct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)

## S3 method for class 'summary_generic_spct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)

## S3 method for class 'generic_mspct'
setWhereMeasured(x, where.measured = NA, lat = NA, lon = NA, address = NA, ...)


setWhereMeasured

Arguments
x a generic_spct object
where.measured, value
A one row data.frame such as returned by function geocode from package 'ggmap' for a location search.
lata numeric Latitude in decimal degrees North
lon numeric Longitude in decimal degrees West
address character Human readable address
... Allows use of additional arguments in methods for other classes.

Value
x

Methods (by class)
• default: default
• generic_spct: generic_spct
• summary_generic_spct: summary_generic_spct
• generic_mspct: generic_mspct

Note
This method alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct
or an object of a class derived from generic_spct, x is not modified. If where is not a POSIXct
object or NULL an error is triggered. A POSIXct describes an instant in time (date plus time-of-day
plus time zone). As expected passing NULL as argument for where.measured unsets the attribute.
Method for collections of spectra recycles the location information only if it is of length one.

See Also
Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrDesc(), getInstrSettings(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(),
get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(),
setWhenMeasured(), spct_attr2tb(), spct_metadata(), trimInstrDesc(), trimInstrSettings()

Examples

my.spct <- sun.spct
where_measured(my.spct)
where_measured(my.spct) <- data.frame(lon = 0, lat = -60)
where_measured(my.spct)
shared_member_class Classes common to all collection members.

Description

Finds the set intersection among the class attributes of all collection member as a target set of class names.

Usage

shared_member_class(l, target.set = spct_classes())

Arguments

l a list or a generic_mspct object or of a derived class.

target.set character The target set of classes within which to search for classes common to all members.

Value

A character vector containing the class attribute values.

See Also

Other set and unset 'multi spectral' class functions: rmDerivedMspct()

sign Sign

Description

sign returns a vector with the signs of the corresponding elements of x (the sign of a real number is 1, 0, or -1 if the number is positive, zero, or negative, respectively).

Usage

## S3 method for class 'generic_spct'
sign(x)

Arguments

x an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), slash-.generic_spct, times-.generic_spct
slash-.generic_spct  Arithmetic Operators

Description
Division operator for generic spectra.

Usage

```
## S3 method for class 'generic_spct'
e1 / e2
```

Arguments

- `e1`: an object of class "generic_spct"
- `e2`: an object of class "generic_spct"

See Also

Other math operators and functions: MathFun, ^.generic_spct(), convolve_each(), div-.generic_spct, log(), minus-.generic_spct, mod-.generic_spct, plus-.generic_spct, round(), sign(), times-.generic_spct

smooth_spct  Smooth a spectrum

Description
These functions implement one original methods and acts as a wrapper for other common R smoothing functions. The advantage of using this function for smoothing spectral objects is that it simplifies the user interface and sets, when needed, defaults suitable for spectral data.

Usage

```
smooth_spct(x, method, strength, wl.range, ...)

## Default S3 method:
smooth_spct(x, method, strength, wl.range, ...)

## S3 method for class 'source_spct'
smooth_spct(
x,
  method = "custom",
  strength = 1,
  wl.range = NULL,
)```
Arguments

- **x**: an R object.
- **method**: a character string "custom", "lowess", "supsmu".
smooth_spct

strength numeric value to adjust the degree of smoothing. Ignored if method-specific parameters are passed through ... .

wl.range any R object on which applying the method range() yields a vector of two numeric values, describing a range of wavelengths (nm). NA is interpreted as data’s own min or max value.

... other parameters passed to the underlying smoothing functions.

na.rm logical A flag indicating whether NA values should be stripped before the computation proceeds.

Value

A copy of x with spectral data values replaced by smoothed ones.

Methods (by class)

- default: Default for generic function
- source_spct: Smooth a source spectrum
- filter_spct: Smooth a filter spectrum
- reflector_spct: Smooth a reflector spectrum
- response_spct: Smooth a response spectrum
- generic_mspct:

Note

Method "custom" is our home-brewed method which applies strong smoothing to low signal regions of the spectral data, and weaker or no smoothing to the high signal areas. Values very close to zero are set to zero with a limit which depends on the local variation. This method is an ad-hock method suitable for smoothing spectral data obtained with spectrometers. In the cased of methods "lowess" and "supsmu" the current function behaves like a wrapper of the functions of the same names from base R.

Examples

```r
my.spct <- clip_wl(sun.spct, c(400, 500))
smooth_spct(my.spct)
smooth_spct(my.spct, method = "custom", strength = 1)
smooth_spct(my.spct, method = "custom", strength = 4)
smooth_spct(my.spct, method = "supsmu", strength = 4)
```
**solar_time**

---

**Description**

`solar_time` computes from a time and geocode, the time of day expressed in seconds since midnight. `solar_date` returns the same instant in time as a date-time object. Solar time is useful when we want to plot data according to the local solar time of day, irrespective of the date. Solar date is useful when we want to plot a time series stretching for several days using the local solar time but distinguishing between days.

**Usage**

```r
solar_time(
  time = lubridate::now(),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  unit.out = "time"
)
```

**Arguments**

- **time**  
  POSIXct Time, any valid time zone (TZ) is allowed, default is current time

- **geocode**  
  data frame with variables lon and lat as numeric values (degrees).

- **unit.out**  
  character string. One of "datetime", "hour", "minute", or "second".

**Value**

For `solar_time()` numeric value in seconds from midnight but with an additional class attribute "solar.time".

**Warning!**

Returned values are computed based on the time zone of the argument for parameter time. In the case of solar time, this timezone does not affect the result. However, in the case of solar dates the date part may be off by one day, if the time zone does not match the coordinates of the geocode value provided as argument.

**Note**

The algorithm is approximate, it calculates the difference between local solar noon and noon in the time zone of time and uses this value for the whole day when converting times into solar time. Days are not exactly 24 h long. Between successive days the shift is only a few seconds, and this leads to a small jump at midnight.

**See Also**

Other astronomy related functions: `day_night()`, `format.solar_time()`, `format.tod_time()`, `is.solar_time()`, `print.solar_time()`, `print.tod_time()`, `sun_angles()`
Examples

```
BA.geocode <-
data.frame(lon = -58.38156, lat = -34.60368, address = "Buenos Aires, Argentina")
sol_t <- solar_time(lubridate::dmy_hms("21/06/2016 10:00:00", tz = "UTC"),
BA.geocode)
sol_t
class(sol_t)

sol_d <- solar_time(lubridate::dmy_hms("21/06/2016 10:00:00", tz = "UTC"),
BA.geocode,
unit.out = "datetime")
sol_d
class(sol_d)
```

source_spct

**Spectral-object constructor**

Description

These functions can be used to create spectral objects derived from generic_spct. They take as arguments numeric vectors for the data character scalars for attributes, and a logical flag.

Usage

```
source_spct(
  w.length = NULL,
  s.e.irrad = NULL,
  s.q.irrad = NULL,
  time.unit = c("second", "day", "exposure"),
  bswf.used = c("none", "unknown"),
  comment = NULL,
  strict.range = getOption("photobiology.strict.range", default = FALSE),
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)

calibration_spct(
  w.length = NULL,
  irrados.mult = NA_real_,
  comment = NULL,
  instr.desc = NA,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)
```
raw_spct(
  w.length = NULL,
  counts = NA_real_,
  comment = NULL,
  instr.desc = NA,
  instr.settings = NA,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)

cps_spct(
  w.length = NULL,
  cps = NA_real_,
  comment = NULL,
  instr.desc = NA,
  instr.settings = NA,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)

generic_spct(
  w.length = NULL,
  comment = NULL,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)

response_spct(
  w.length = NULL,
  s.e.response = NULL,
  s.q.response = NULL,
  time.unit = c("second", "day", "exposure"),
  response.type = c("response", "action"),
  comment = NULL,
  multiple.wl = 1L,
  idfactor = NULL,
  ...
)

filter_spct(
  w.length = NULL,
  Tfr = NULL,
  Tpc = NULL,
  Afr = NULL,
source_spct

A = NULL,
Tfr.type = c("total", "internal"),
comment = NULL,
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL,
...
)

reflector_spct(
w.length = NULL,
Rfr = NULL,
Rpc = NULL,
Rfr.type = c("total", "specular"),
comment = NULL,
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL,
...
)

object_spct(
w.length = NULL,
Rfr = NULL,
Tfr = NULL,
Afr = NULL,
Tfr.type = c("total", "internal"),
Rfr.type = c("total", "specular"),
comment = NULL,
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL,
...
)

chroma_spct(
w.length = NULL,
x,
y,
z,
comment = NULL,
strict.range = getOption("photobiology.strict.range", default = FALSE),
multiple.wl = 1L,
idfactor = NULL,
...
Arguments

- **w.length**: numeric vector with wavelengths in nanometres
- **s.e.irrad**: numeric vector with spectral energy irradiance in [W m^-2 nm^-1] or [J d^-1 m^-2 nm^-1]
- **s.q.irrad**: numeric vector with spectral photon irradiance in [mol s^-1 m^-2 nm^-1] or [mol d^-1 m^-2 nm^-1].
- **time.unit**: character string indicating the time unit used for spectral irradiance or exposure ("second", "day" or "exposure") or an object of class duration as defined in package lubridate.
- **bswf.used**: character A string indicating the BSWF used, if any, for spectral effective irradiance or exposure ("none" or the name of the BSWF).
- **comment**: character A string to be added as a comment attribute to the object created.
- **strict.range**: logical Flag indicating whether off-range values result in an error instead of a warning.
- **multiple.wl**: numeric Maximum number of repeated w.length entries with same value.
- **idfactor**: character Name of factor distinguishing multiple spectra when stored longitudinally (required if multiple.wl > 1).
- ... other arguments passed to `tibble()`
- **irrad.mult**: numeric vector with multipliers for each detector pixel.
- **instr.desc**: a list
- **counts**: numeric vector with raw counts expressed per scan
- **instr.settings**: a list
- **cps**: numeric vector with linearized raw counts expressed per second
- **s.e.response**: numeric vector with spectral energy irradiance in W m^-2 nm^-1 or J d^-1 m^-2 nm^-1
- **s.q.response**: numeric vector with spectral photon irradiance in mol s^-1 m^-2 nm^-1 or mol d^-1 m^-2 nm^-1
- **response.type**: a character string, either "response" or "action".
- **Tfr**: numeric vector with spectral transmittance as fraction of one
- **Tpc**: numeric vector with spectral transmittance as percent values
- **Afr**: numeric vector of absorptance as fraction of one
- **A**: numeric vector of absorbance values (log10 based a.u.)
- **Tfr.type**: character string indicating whether transmittance and absorptance values are "total" or "internal" values
- **Rfr**: numeric vector with spectral reflectance as fraction of one
- **Rpc**: numeric vector with spectral reflectance as percent values
- **Rfr.type**: character A string, either "total" or "specular".
- **x, y, z**: numeric colour coordinates
Value

A object of class generic_spct or a class derived from it, depending on the function used. In other
words an object of a class with the same name as the constructor function.

Note

The functions can be used to add only one spectral quantity to a spectral object. Some of the
functions have different arguments, for the same quantity expressed in different units. An actual
parameter can be supplied to only one of these formal parameters in a given call to any of these
functions.

"internal" transmittance is defined as the transmittance of the material body itself, while "total"
transmittance includes the effects of surface reflectance on the amount of light transmitted.

See Also

Other constructors of spectral objects: as.calibration_spct(), as.chroma_spct(), as.cps_spct(),
as.filter_spct(), as.generic_spct(), as.object_spct(), as.raw_spct(), as.reflector_spct(),
as.response_spct(), as.source_spct()

spct_attr2tb

Description

Method returning attributes of an object of class generic_spct or derived, or of class waveband.
Only attributes defined and/or set by package 'photobiology' for objects of the corresponding class
are returned.

Usage

spct_attr2tb(
  x,
  which = c("-", "names", "row.names", "spct.tags", "spct.version", "comment"),
  ...
)

Arguments

  x a generic_spct object.
  which character vector Names of attributes to retrieve.
  ... currently ignored

Value

A tibble with the values stored in the attributes whose names were selected through the argument to
which if present in x.
See Also

Other measurement metadata functions: add_attr2tb(), getFilterProperties(), getHowMeasured(),
getInstrDesc(), getInstrSettings(), getWhatMeasured(), getWhenMeasured(), getWhereMeasured(),
get_attributes(), isValidInstrDesc(), isValidInstrSettings(), select_spct_attributes(),
setWhenMeasured(), setWhereMeasured(), spct_metadata(), trimInstrDesc(), trimInstrSettings()
Arguments

- **x**: generic_mspct or generic_spct Any collection of spectra or spectrum.
- **col.names**: named character vector Name(s) of column(s) to create.
- **idx**: character Name of the column with the names of the members of the collection of spectra.
- **na.rm**: logical Flag controlling deletion of columns containing only NA values.
- **unnest**: logical Flag controlling if metadata attributes that are lists of values should be returned in a list column or in separate columns.

Details

Attributes are returned as columns in a tibble. If the argument to `col.names` is a named vector, with the names of members matching the names of attributes, then the values are used as names for the columns created. This permits setting any valid name for the new columns. If the vector passed to `col.names` has no names, then the values are interpreted as the names of the attributes to add, and also used as names for the new columns.

Some metadata values are stored in lists or data frames, these can be returned as a list columns or the individual fields unnested into separate columns.

Value

A tibble With the metadata attributes and an index column.

See Also

- `add_attr2tb` for more details.

Examples

```r
my.mspct <- source_mspct(list(sun1 = sun.spct, sun2 = sun.spct * 2))
spct_metadata(my.mspct)
spct_metadata(sun.spct)
spct_metadata(my.mspct, na.rm = TRUE)
spct_metadata(sun.spct, na.rm = TRUE)
spct_metadata(my.mspct, col.names = c(geocode = "geo", "instr.desc"))
spct_metadata(sun.spct, col.names = c(geocode = "geo", "instr.desc"))
```
spikes(sun.spct, col.names = "where.measured")$where.measured

---

**spikes**  
**Spikes**

**Description**

Function that returns a subset of an R object with observations corresponding to spikes. Spikes are values in spectra that are unusually high compared to neighbors. They are usually individual values or very short runs of similar "unusual" values. Spikes caused by cosmic radiation are a frequent problem in Raman spectra. Another source of spikes are "hot pixels" in CCD and diode arrays.

**Usage**

```r
spikes(x, z.threshold, max.spike.width, na.rm, ...)
```

## Default S3 method:
spikes(x, z.threshold = NA, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class `numeric`
spikes(x, z.threshold = NA, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class `data.frame`
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ..., 
  y.var.name = NULL,
  var.name = y.var.name
)

## S3 method for class `generic_spct`
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ..., 
  var.name = NULL
)

## S3 method for class `source_spct`
spikes(
```
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
... )

## S3 method for class 'response_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  filter.qty = getOption("photobiology.filter_qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
spikes(x, z.threshold = 9, max.spike.width = 8, na.rm = FALSE, ...)

## S3 method for class 'cps_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...
)

## S3 method for class 'raw_spct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...,
```r
var.name = "counts"

## S3 method for class 'generic_mspct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
..., 
var.name = NULL,
.paropts = NULL
)

## S3 method for class 'source_mspct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
..., 
.paropts = NULL
)

## S3 method for class 'response_mspct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
..., 
.paropts = NULL
)

## S3 method for class 'filter_mspct'
spikes(
x,
z.threshold = 9,
max.spike.width = 8,
na.rm = FALSE,
filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
..., 
.paropts = NULL
)```
spikes

paropts = NULL

## S3 method for class 'reflector_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  ...
)

## S3 method for class 'cps_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  var.name = "cps",
  ...
)

## S3 method for class 'raw_mspct'
spikes(
  x,
  z.threshold = 9,
  max.spike.width = 8,
  na.rm = FALSE,
  var.name = "counts",
  ...
)

Arguments

x          an R object
z.threshold numeric Modified Z values larger than z.threshold are considered to correspond to spikes.
max.spike.width integer Wider regions with high Z values are not detected as spikes.
na.rm      logical indicating whether NA values should be stripped before searching for spikes.
...        ignored
spikes

var.name, y.var.name
  character Name of column where to look for spikes.
unit.out
  character One of "energy" or "photon"
filter.qty
  character One of "transmittance" or "absorbance"
.parallel
  if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts
  a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value
A subset of x with rows corresponding to spikes.

Methods (by class)
- default: Default returning always NA.
- numeric: Default function usable on numeric vectors.
- data.frame: Method for "data.frame" objects.
- generic_spct: Method for "generic_spct" objects.
- source_spct: Method for "source_spct" objects.
- filter_spct: Method for "filter_spct" objects.
- reflector_spct: Method for "reflector_spct" objects.
- cps_spct: Method for "cps_spct" objects.
- raw_spct: Method for "raw_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.
- source_mspct: Method for "source_mspct" objects.
- response_mspct: Method for "cps_mspct" objects.
- filter_mspct: Method for "filter_mspct" objects.
- reflector_mspct: Method for "reflector_mspct" objects.
- cps_mspct: Method for "cps_mspct" objects.
- raw_mspct: Method for "raw_mspct" objects.

See Also
See the documentation for find_spikes for details of the algorithm and implementation.
Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), peaks(), replace_bad_pixs(), valleys(), wls_at_target()

Examples
spikes(sun.spct)
split2mspc

Convert a 'wide' or untidy data frame into a collection of spectra

Description

Convert a data frame object into a "multi spectrum" object by constructing a an object of a multi-spect class, converting numeric columns other than wavelength into individual spct objects.

Usage

split2mspct(
  x,
  member.class = NULL,
  spct.data.var = NULL,
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2source_mspct(
  x,
  spct.data.var = "s.e.irrad",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2response_mspct(
  x,
  spct.data.var = "s.e.response",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2filter_mspct(
  x,
  spct.data.var = "Tfr",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
byrow = FALSE, 
...
)

split2reflector_mspct(
  
  x,
  spct.data.var = "Rfr",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2cps_mspct(
  
  x,
  spct.data.var = "cps",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2raw_mspct(
  
  x,
  spct.data.var = "count",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

split2calibration_mspct(
  
  x,
  spct.data.var = "irrad.mult",
  w.length.var = "w.length",
  idx.var = NULL,
  ncol = 1,
  byrow = FALSE,
  ...
)

Arguments

  x        data frame
  member.class character Class of the collection members
split_bands

Character Name of the spectral data argument in the object constructor for member.class

w.length.var

Character Name of column containing wavelength data in nanometres

idx.var

Character Name of column containing data to be copied unchanged to each spct object

ncol

Integer Number of 'virtual' columns in data

byrow

Logical If ncol > 1 how to read in the data

... additional named arguments passed to the member constructor function.

See Also

Other Coercion methods for collections of spectra: as.calibration_mspct(), as.chroma_mspct(), as.cps_mspct(), as.filter_mspct(), as.generic_mspct(), as.object_mspct(), as.raw_mspct(), as.reflector_mspct(), as.response_mspct(), as.source_mspct(), subset2mspct()

Description

Build a list of unweighted 'waveband' objects that can be used as input when calculating irradiances.

Usage

split_bands(
  x, 
  list.names = NULL, 
  short.names = is.null(list.names), 
  length.out = NULL
)

Arguments

x a numeric vector of wavelengths to split at (nm), or a range of wavelengths or a generic_spct or a waveband.

list.names character vector with names for the component wavebands in the returned list (in order of increasing wavelength)

short.names logical indicating whether to use short or long names for wavebands

length.out numeric giving the number of regions to split the range into (ignored if w.length is not numeric).

Value

An un-named list of waveband objects.
split_energy_irradiance

Energy irradiance for split spectrum regions

Description

This function returns the energy irradiance for a series of contiguous wavebands from a radiation-source spectrum. The returned values can be either absolute or relative to their sum.

Usage

split_energy_irradiance(
  w.length,
  s.irrad,
  cut.w.length = range(w.length),
  unit.in = "energy",
  scale = "absolute",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges =getOption("photobiology.use.hinges", default = NULL)
)
split_energy_irradiance

Arguments

w.length numeric vector of wavelengths (nm).
s.irrad numeric vector of spectral (energy or photon) irradiance values (W m\(^{-2}\) nm\(^{-1}\)) or (mol s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)).
cut.w.length numeric vector of wavelengths (nm).
unit.in character string with allowed values "energy", and "photon", or its alias "quantum".
scale character string indicating the scale used for the returned values ("absolute", "relative", "percent").
check.spectrum logical indicating whether to sanity check input data, default is TRUE.
use.cached.mult logical Flag indicating whether multiplier values should be cached between calls.
use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a numeric vector of irradiances with no change in scale factor: [W m\(^{-2}\) nm\(^{-1}\)] -> [W m\(^{-2}\)] or [mol s\(^{-1}\) m\(^{-2}\)] -> [W m\(^{-2}\)] or relative values (fraction of one) if scale = "relative" or scale = "percent".

Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set check.spectrum=FALSE then you should call check_spectrum at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

See Also

Other low-level functions operating on numeric vectors: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

Examples

with(sun.data,
  split_energy_irradiance(w.length, s.e.irrad,
                        cut.w.length = c(300, 400, 500, 600, 700)))
split_irradiance  

Energy or photon irradiance for split spectrum regions

Description
This function returns the energy or photon irradiance for a series of contiguous wavebands from a radiation spectrum. The returned values can be either absolute or relative to their sum.

Usage

split_irradiance(
  w.length,
  s.irrad,
  cut.w.length = range(w.length),
  unit.out = getOption("photobiology.base.unit", default = "energy"),
  unit.in = "energy",
  scale = "absolute",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)

Arguments

- **w.length**  numeric Vector of wavelengths (nm).
- **s.irrad** numeric vector of spectral (energy or photon) irradiances (W m-2 nm-1) or (mol s-1 m-2 nm-1).
- **cut.w.length** numeric Vector of wavelengths (nm).
- **unit.out** character Allowed values "energy", and "photon", or its alias "quantum".
- **unit.in** character Allowed values "energy", and "photon", or its alias "quantum".
- **scale** a character A string indicating the scale used for the returned values ("absolute", "relative", "percent").
- **check.spectrum** logical Flag indicating whether to sanity check input data, default is TRUE.
- **use.cached.mult** logical Flag indicating whether multiplier values should be cached between calls.
- **use.hinges** logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

A numeric vector of irradiances with no change in scale factor: [W m-2 nm-1] -> [mol s-1 m-2] or [mol s-1 m-2 nm-1] -> [mol s-1 m-2] or relative values (as fraction of one if scale == "relative" or percentages if scale == "percent".)
**split_photon_irradiance**

### Note

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set `check.spectrum=FALSE` then you should call `check_spectrum` at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting `use.cached.mult=TRUE`. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the `w.length` vector.

### Examples

```r
with(sun.data,
    split_irradiance(w.length, s.e.irrad,
        cut.w.length = c(300, 400, 500, 600, 700),
        unit.out = "photon"))
```

---

**split_photon_irradiance**

*Photon irradiance for split spectrum regions*

### Description

This function returns the photon irradiance for a series of contiguous wavebands from a radiation spectrum. The returned values can be either absolute or relative to their sum.

### Usage

```r
split_photon_irradiance(
    w.length,
    s.irrad,
    cut.w.length = range(w.length),
    unit.in = "energy",
    scale = "absolute",
    check.spectrum = TRUE,
    use.cached.mult = FALSE,
    use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

### Arguments

- `w.length` numeric vector of wavelengths (nm).
- `s.irrad` numeric vector of spectral (energy or photon) irradiance values (W m⁻² nm⁻¹).
- `cut.w.length` numeric vector of wavelengths (nm).
- `unit.in` character Allowed values "energy", and "photon", or its alias "quantum".
- `scale` a character A string indicating the scale used for the returned values ("absolute", "relative", "percent").
- `check.spectrum` logical Flag indicating whether to sanity check input data, default is TRUE.
spread

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use.cached.mult</td>
<td>logical Flag indicating whether multiplier values should be cached between calls.</td>
</tr>
<tr>
<td>use.hinges</td>
<td>logical Flag indicating whether to insert &quot;hinges&quot; into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.</td>
</tr>
</tbody>
</table>

**Value**

A numeric vector of photon irradiances with no change in scale factor: [W m\(^{-2}\) nm\(^{-1}\)] -> [mol s\(^{-1}\) m\(^{-2}\)], [mol s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)] -> [mol s\(^{-1}\) m\(^{-2}\)] or relative values (fraction of one based on photon units) if scale = "relative" or scale = "percent".

**Note**

The last three parameters control speed optimizations. The defaults should be suitable in most cases. If you set check.spectrum=FALSE then you should call check_spectrum at least once for your spectrum before using any of the other functions. If you will use repeatedly the same SWFs on many spectra measured at exactly the same wavelengths you may obtain some speed up by setting use.cached.mult=TRUE. However, be aware that you are responsible for ensuring that the wavelengths are the same in each call, as the only test done is for the length of the w.length vector.

**See Also**

Other low-level functions operating on numeric vectors: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), s_e_irrad2rgb(), split_energy_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

**Examples**

```r
with(sun.data,
     split_photon_irradiance(w.length, s.e.irrad,
       cut.w.length = c(300, 400, 500, 600, 700)))
with(sun.data,
     split_photon_irradiance(w.length, s.e.irrad))
```

spread  

**Description**

A function that returns the expanse (max(x) - min(x)) for R objects.
**Usage**

spread(x, ...)

wl-expande(x, ...)

expanse(x, ...)

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

## S3 method for class 'numeric'
expanse(x, ...)

## S3 method for class 'waveband'
expanse(x, ...)

## S3 method for class 'generic_spct'
expanse(x, ...)

## S3 method for class 'generic_mspct'
expanse(x, ..., idx = "spct.idx")

**Arguments**

- **x**: an R object
- **...**: not used in current version
- **idx**: character Name of the column with the names of the members of the collection of spectra.

**Value**

A numeric value equal to max(x) - min(x). In the case of spectral objects wavelength difference in nm. For any other R object, according to available definitions of min and max.

**Methods (by class)**

- default: Default method for generic function
- numeric: Method for "numeric"
- waveband: Method for "waveband"
- generic_spct: Method for "generic_spct"
- generic_mspct: Method for "generic_mspct" objects.

**Examples**

expanse(10:20)
expanse(sun.spct)
w1_expandse(sun.spct)
Subset

Subsetting spectra

Description

Return subsets of spectra stored in class generic_spct or derived from it.

Usage

```r
## S3 method for class 'generic_spct'
subset(x, subset, select, drop = FALSE, ...)
```

Arguments

- `x`: object to be subsetted.
- `subset`: logical expression indicating elements or rows to keep: missing values are taken as false.
- `select`: expression, indicating columns to select from a spectrum.
- `drop`: passed on to `[ indexing operator.
- `...`: further arguments to be passed to or from other methods.

Value

An object similar to `x` containing just the selected rows and columns. Depending on the columns remaining after subsetting the class of the object will be simplified to the most derived parent class.

Note

This method is copied from `base::subset.data.frame()` but ensures that all metadata stored in attributes of spectral objects are copied to the returned value.

Examples

```r
subset(sun.spct, w.length > 400)
```
subset2mspct

Convert 'long' or tidy spectral data into a collection of spectra

Description

Convert a data frame object or spectral object into a collection of spectra object of the corresponding class. For data frames converting numeric columns other than wavelength into individual spct objects.

Usage

subset2mspct(
  x,
  member.class = NULL,
  idx.var = attr(x, "idfactor"),
  drop.idx = TRUE,
  ncol = 1,
  byrow = FALSE,
  ...
)

Arguments

x a generic_spct object or a derived class, or a data frame
member.class character string
idx.var character Name of column containing data to be copied unchanged to each spct object
drop.idx logical Flag indicating whether to drop or keep idx.var in the collection members.
ncol integer Number of 'virtual' columns in data
byrow logical If ncol > 1 how to read in the data
... additional named arguments passed to the member constructor function.

Value

A collection of spectral objects, each with attributes set if x is a spectral object in long form with metadata attributes. If this object was created by row binding with 'photobiology' 0.9.14 or later then all metadata for each individual spectrum will be preserved, except for comments which are merged.

Note

A non-null value for member.class is mandatory only when x is a data frame.
See Also

Other Coercion methods for collections of spectra: `as_calibration_mspct()`, `as_chroma_mspct()`, `as_cps_mspct()`, `as_filter_mspct()`, `as_generic_mspct()`, `as_object_mspct()`, `as_raw_mspct()`, `as_reflector_mspct()`, `as_response_mspct()`, `as_source_mspct()`.

---

**subt_spectra**

*Subtract two spectra*

**Description**

The wavelength vectors of the two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added. This is ‘parallel’ operation between two spectra.

**Usage**

```r
subt_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
```

**Arguments**

- `w.length1`: numeric vector of wavelength (nm).
- `w.length2`: numeric vector of wavelength (nm).
- `s.irrad1`: a numeric vector of spectral values.
- `s.irrad2`: a numeric vector of spectral values.
- `trim`: a character string with value "union" or "intersection".
- `na.rm`: a logical value, if TRUE, not the default, NAs in the input are replaced with zeros.

**Details**

If `trim="union"` spectral values are calculated for the whole range of wavelengths covered by at least one of the input spectra, and missing values are set in each input spectrum to zero before addition. If `trim="intersection"` then the range of wavelengths covered by both input spectra is returned, and the non-overlapping regions discarded. If `w.length2==NULL`, it is assumed that both spectra are measured at the same wavelengths, and a simple addition is used, ensuring fast calculation.
Value

a data frame with two numeric variables

w.length A numeric vector with the wavelengths (nm) obtained by "fusing" w.length1 and w.length2. w.length contains all the unique values, sorted in ascending order.

s.irrad A numeric vector with the sum of the two spectral values at each wavelength.

See Also

Other low-level functions operating on numeric vectors: \texttt{as\_energy()}, \texttt{as\_quantum\_mol()}, \texttt{calc\_multipliers()}, \texttt{div\_spectra()}, \texttt{energy\_irradiance()}, \texttt{energy\_ratio()}, \texttt{insert\_hinges()}, \texttt{integrate\_xy()}, \texttt{interpolate\_spectrum()}, \texttt{irradiance()}, \texttt{l\_insert\_hinges()}, \texttt{oper\_spectra()}, \texttt{photon\_irradiance()}, \texttt{photon\_ratio()}, \texttt{photons\_energy\_ratio()}, \texttt{prod\_spectra()}, \texttt{s\_e\_irrad2rgb()}, \texttt{split\_energy\_irradiance()}, \texttt{split\_photon\_irradiance()}, \texttt{sum\_spectra()}, \texttt{trim\_tails()}, \texttt{v\_insert\_hinges()}, \texttt{v\_replace\_hinges()}

Examples

```r
head(sun.data)
zero.data <- with(sun.data, subt_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(zero.data)
tail(zero.data)
```

### Description

Methods of generic function summary for objects of spectral classes.

#### Usage

```r
## S3 method for class 'generic_spect'
summary(object, maxsum = 7, digits = max(3, getOption("digits") - 3), ...)
```

#### Arguments

- **object** An object of one of the spectral classes for which a summary is desired
- **maxsum** integer Indicates how many levels should be shown for factors.
- **digits** integer Used for number formatting with \texttt{format()}.  
- **...** additional arguments affecting the summary produced, ignored in current version

#### Value

A summary object matching the class of object.
Examples

summary(sun.spct)

summary_spct_classes

Function that returns a vector containing the names of spectral summary classes.

Description

Function that returns a vector containing the names of spectral summary classes.

Usage

summary_spct_classes()

Value

A character vector of class names.

sum_spectra

Add two spectra

Description

Merge wavelength vectors of two spectra are merged, and the missing spectral values are calculated by interpolation. After this, the two spectral values at each wavelength are added. This is a 'parallel' operation between two spectra.

Usage

sum_spectra(
  w.length1,
  w.length2 = NULL,
  s.irrad1,
  s.irrad2,
  trim = "union",
  na.rm = FALSE
)
Examples

head(sun.data)
twice.sun.data <- with(sun.data, sum_spectra(w.length, w.length, s.e.irrad, s.e.irrad))
head(twice.sun.data)
tail(twice.sun.data)
sun.daily.data  Daily solar spectral irradiance (simulated)

Description

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance. Values simulated for 2 June 2012, at Helsinki, under clear sky conditions. The variables are as follows:

Usage

sun.daily.data

Format

A data.frame object with 511 rows and 3 variables

Details

- w.length (nm), range 290 to 800 nm.
- s.e.irrad (J d⁻¹ m⁻² nm⁻¹)
- s.q.irrad (mol d⁻¹ m⁻² nm⁻¹)

Author(s)

Anders K. Lindfors (data)

References


See Also


Examples

sun.daily.spct
**Description**

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance. Values simulated for 2 June 2012, at Helsinki, under clear sky conditions. The variables are as follows:

**Usage**

sun.daily.spct

**Format**

A source_spct object with 511 rows and 3 variables

**Details**

- w.length (nm), range 290 to 800 nm.
- s.e.irrad (J d-1 m-2 nm-1)
- s.q.irrad (mol d-1 m-2 nm-1)

**Note**

The simulations are based on libRadTran using hourly mean global radiation measurements to estimate cloud cover. The simulations were for each hour and the results integrated for the whole day.

**Author(s)**

Anders K. Lindfors (data)

**References**


**See Also**

Description

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance and spectral photon irradiance. Values simulated for 22 June 2010, near midday, at Helsinki, under partly cloudy conditions. The variables are as follows:

Usage

sun.data

Format

A data.frame object with 508 rows and 3 variables

Details

- w.length (nm), range 293 to 800 nm.
- s.e.irrad (W m⁻² nm⁻¹)
- s.q.irrad (mol m⁻² nm⁻¹)

Author(s)

Anders K. Lindfors (data)

References


See Also


Examples

sun.data
sun.spct

---

**sun.spct**  
*Solar spectral irradiance (simulated)*

**Description**

A dataset containing the wavelengths at a 1 nm interval and the corresponding spectral (energy) irradiance and spectral photon irradiance. Values simulated for 22 June 2010, near midday, at Helsinki, under partly cloudy conditions. The variables are as follows:

**Usage**

sun.spct

**Format**

A `source_spct` object with 508 rows and 3 variables

**Details**

- `w.length` (nm), range 293 to 800 nm.
- `s.e.irrad` (W m⁻² nm⁻¹)
- `s.q.irrad` (mol m⁻² nm⁻¹)

**Author(s)**

Anders K. Lindfors (data)

**References**


**See Also**

Other Spectral data examples:  
`A.illuminant.spct`,  
`D65.illuminant.spct`,  
`Ler_leaf.spct`,  
`Ler_leaf_rflt.spct`,  
`Ler_leaf_trns.spct`,  
`Ler_leaf_trns_i.spct`,  
`black_body.spct`,  
`ccd.spct`,  
`clear.spct`,  
`clear_body.spct`,  
`filter_cps.mspct`,  
`green_leaf.spct`,  
`opaque.spct`,  
`photodiode.spct`,  
`polyester.spct`,  
`sun.daily.data`,  
`sun.daily.spct`,  
`sun.data`,  
`white_body.spct`,  
`white_led.cps_spct`,  
`white_led.raw_spct`,  
`white_led.source_spct`,  
`yellow_gel.spct`

**Examples**

sun.spct
**Description**

This function returns the solar angles at a given time and location.

**Usage**

```r
sun_angles(
  time = lubridate::now(tzone = "UTC"),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_angles_fast(time, tz, geocode, use.refraction)

sun_elevation(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_zenith_angle(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)

sun_azimuth(
  time = lubridate::now(),
  tz = lubridate::tz(time),
  geocode = tibble::tibble(lon = 0, lat = 51.5, address = "Greenwich"),
  use.refraction = FALSE
)
```

**Arguments**

- **time**: A "vector" of POSIXct Time, with any valid time zone (TZ) is allowed, default is current time.
- **tz**: character string indicating time zone to be used in output.
- **geocode**: data frame with variables lon and lat as numeric values (degrees), nrow > 1, allowed.
- **use.refraction**: logical Flag indicating whether to correct for fraction in the atmosphere.
Details

This function is an implementation of Meeus equations as used in NOAAs on-line web calculator, which are precise and valid for a very broad range of dates (years -1000 to 3000 at least). The apparent solar elevations near sunrise and sunset are affected by refraction in the atmosphere, which does in turn depend on weather conditions. The effect of refraction on the apparent position of the sun is only an estimate based on "typical" conditions for the atmosphere. The computation is not defined for latitudes 90 and -90 degrees, i.e. exactly at the poles.

In the current implementation functions sun_azimuth, sun_elevation, and sun_zenith_angle are wrappers on sun_angles, so if more than one angle is needed it is preferable to directly call sun_angles as it will be faster.

Value

A data.frame with variables time (in same TZ as input), TZ, solartime, longitude, latitude, address, azimuth, and elevation. If a data frame with multiple rows is passed to geocode and a vector of times longer than one is passed to time, sun position for all combinations of locations and times are returned are returned by sun_angles. In contrast, convenience functions returning a vector.

Note

There exists a different R implementation of the same algorithms called "AstroCalcPureR" available as function astrocalc4r in package 'fishmethods'. Although the equations used are almost all the same, the function signatures and which values are returned differ. In particular, the present implementation splits the calculation into two separate functions, one returning angles at given instants in time, and a separate one returning the timing of events for given dates.

References


A different implementation is available at https://apps-nefsc.fisheries.noaa.gov/AstroCalc4R/ and in R package 'fishmethods'. In 'fishmethods' (= 1.11-0) there is a bug in function astrocalc4r() that affects sunrise and sunset times.

An interactive web page using the same algorithms is available at https://www.esrl.noaa.gov/gmd/grad/solcalc/. There are small differences in the returned times compared to our function that seem to be related to the estimation of atmospheric refraction (about 0.1 degrees).

See Also

Other astronomy related functions: day_night(), format.solar_time(), format.tod_time(), is.solar_time(), print.solar_time(), print.tod_time(), solar_time()

Examples

library(lubridate)
sun_angles()
sun_azimuth()
sun_elevation()
sun_zenith_angle()
sun_angles(ymd_hms("2014-09-23 12:00:00"))
sun_angles(ymd_hms("2014-09-23 12:00:00"),
    geocode = data.frame(lat=60, lon=0))
sun_angles(ymd_hms("2014-09-23 12:00:00") + minutes((0:6) * 10))

s_e_irrad2rgb  
Spectral irradiance to rgb color conversion

Description
Calculates rgb values from spectra based on human color matching functions (CMF) or chromaticity coordinates (CC). A CMF takes into account luminous sensitivity, while a CC only the color hue. This function, in contrast to that in package pavo does not normalize the values to equal luminosity, so using a CMF as input gives the expected result. Another difference is that it allows the user to choose the chromaticity data to be used. The data used by default is different, and it corresponds to the whole range of CIE standard, rather than the reduced range 400 nm to 700 nm. The wavelength limits are not hard coded, so the function could be used to simulate vision in other organisms as long as pseudo CMF or CC data are available for the simulation.

Usage
s_e_irrad2rgb(
    w.length,
    s.e.irrad,
    sens = photobiology::ciexyzCMF2.spct,
    color.name = NULL,
    check = TRUE
)

Arguments
w.length  numeric vector of wavelengths (nm).
s.e.irrad numeric vector of spectral irradiance values.
sens  a chroma_spct object with variables w.length, x, y, and z, giving the CC or CMF definition (default is the proposed human CMF according to CIE 2006.).
color.name character string for naming the rgb color definition.
check logical indicating whether to check or not spectral data.

Value
A color defined using rgb. The numeric values of the RGB components can be obtained using function col2rgb.
Note

Very heavily modified from Chad Eliason’s <cme16@zips.uakron.edu> spec2rgb function in package Pavo.

References


Color matching functions obtained from Colour and Vision Research Laboratory online data repository at http://www.cvrl.org/.

See Also

Other low-level functions operating on numeric vectors: as_energy(), as_quantum_mol(), calc_multipliers(), div_spectra(), energy_irradiance(), energy_ratio(), insert_hinges(), integrate_xy(), interpolate_spectrum(), irradiance(), l_insert_hinges(), oper_spectra(), photon_irradiance(), photon_ratio(), photons_energy_ratio(), prod_spectra(), split_energy_irradiance(), split_photon_irradiance(), subt_spectra(), sum_spectra(), trim_tails(), v_insert_hinges(), v_replace_hinges()

Examples

my.color <-
  with(sun.data,
    s_e_irrad2rgb(w.length, s.e.irrad, color.name = "sunWhite"))

col2rgb(my.color)

s_mean

Mean from collection of spectra

Description

A method to compute the mean of values across members of a collections of spectra. Computes the mean at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_mean(x, trim, na.rm, ...)

## Default S3 method:

s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'

s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **trim**: numeric. The fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **...**: Further arguments passed to or from other methods.

Value

If x is a collection of spectral objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spct", containing the mean spectrum.

Methods (by class)

- default:
- source_mspct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- calibration_mspct:
- cps_mspct:
- raw_mspct:
Note

Trimming of extreme values and omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in x must share the same set of wavelengths.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

See mean for the \texttt{mean()} method used for the computations.

\begin{itemize}
\item \texttt{s\_mean\_se(x, na.rm = FALSE, mult = 1, \ldots)}
\end{itemize}

\textbf{Description}

A method to compute the mean of values across members of a collections of spectra. Computes the mean at each wavelength across all the spectra in the collection returning a spectral object.

\textbf{Usage}

\begin{verbatim}
\begin{verbatim}\
\end{verbatim}
\begin{verbatim}\
\end{verbatim}
\end{verbatim}

\begin{verbatim}\
\end{verbatim}
Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **mult**: numeric number of multiples of standard error
- **...**: Further arguments passed to or from other methods.

Value

If `x` is a collection of spectral objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as “filter_spct”, containing the mean spectrum.

Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:
- calibration_mspct:
- cps_mspct:
- raw_mspct:

Note

Trimming of extreme values and omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

See `mean` for the `mean()` method used for the computations.
Description

A method to compute the median of values across members of a collection of spectra. Computes the median at each wavelength across all the spectra in the collection returning a spectral object.

Usage

s_median(x, na.rm, ...)

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

## S3 method for class 'source_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_median(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_median(x, na.rm = FALSE, ...)

Arguments

x An R object. Currently this package defines methods for collections of spectral objects.

na.rm logical. A value indicating whether NA values should be stripped before the computation proceeds.

... Further arguments passed to or from other methods.
Value

If \( x \) is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spct", containing the median spectrum.

Methods (by class)

- default:
  - source_mspct:
  - response_mspct:
  - filter_mspct:
  - reflector_mspct:
  - calibration_mspct:
  - cps_mspct:
  - raw_mspct:

Note

Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in \( x \) must share the same set of wavelengths.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

See median for the \texttt{median()} method used for the computations.

---

\textbf{Description}

A method to compute the product of values across members of a collections of spectra. Computes the product at each wavelength across all the spectra in the collection returning a spectral object.

\textbf{Usage}

\begin{verbatim}
s_prod(x, na.rm, ...)

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

## S3 method for class 'source_mspct'
s_prod(x, na.rm = FALSE, ...)
\end{verbatim}
## S3 method for class 'response_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_prod(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_prod(x, na.rm = FALSE, ...)

### Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **...**: Further arguments passed to or from other methods.

### Value

If `x` is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spct", containing the product of the spectra.

### Methods (by class)

- default:
  - source_mspct:
  - response_mspct:
  - filter_mspct:
  - reflector_mspct:
  - calibration_mspct:
  - cps_mspct:
  - raw_mspct:

### Note

Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths.
A product of spectral irradiance or spectral response is no longer a well defined physical quantity, and these product operations return an object of class generic_spct.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

**See Also**

See `prod` for the `prod()` method used for the computations.

---

### s_range

**Range of a collection of spectra**

**Description**

A method to compute the range of values across members of a collections of spectra. Computes the max and min at each wavelength across all the spectra in the collection returning a spectral object.

**Usage**

```r
s_range(x, na.rm, ...)
```

## Default S3 method:

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'filter_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'source_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'response_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'reflector_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'calibration_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'cps_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```

## S3 method for class 'raw_mspct'

```r
s_range(x, na.rm = FALSE, ...)
```
s_sd

**Arguments**

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **...**: Further arguments passed to or from other methods.

**Value**

If _x_ is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of same class as the members of the collection, such as "filter_spect", containing the mean spectrum.

**Methods (by class)**

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:
- calibration_mspct:
- cps_mspct:
- raw_mspct:

**Note**

Trimming of extreme values and omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in _x_ must share the same set of wavelengths.

Objects of classes raw_spect and cps_spect can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spect members.

**See Also**

See **Extremes** details on the min() and max() methods used for the computations.
Usage

\[ s_{sd}(x, \text{na.rm}, \ldots) \]

## Default S3 method:
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'filter_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'source_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'response_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'reflector_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'calibration_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'cps_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

## S3 method for class 'raw_mspct'
s_{sd}(x, \text{na.rm} = \text{FALSE}, \ldots)

Arguments

- **x**: An R object. Currently this package defines methods for collections of spectral objects.
- **na.rm**: logical. A value indicating whether NA values should be stripped before the computation proceeds.
- **...**: Further arguments passed to or from other methods.

Value

If \( x \) is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of class "generic_spct", containing the standard deviation among the spectra at each wavelength in a column with name ending in ".sd".

Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
s_se

- reflector_mspct:
- calibration_mspct:
- cps_mspct:
- raw_mspct:

Note

Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra
in x must share the same set of wavelengths.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are
implemented for these classes only for the case when all member spectra contain data for a single
scan, or spliced into a single column in the case of cps_spct members.

See Also

See sd for details about sd() methods for other classes.

== s_se

\textit{Standard Error of a collection of spectra}

Description

A method to compute the standard error of values across members of a collections of spectra.
Computes the standard error at each wavelength across all the spectra in the collection returning a
spectral object.

Usage

s_se(x, na.rm, ...)

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

## S3 method for class 'source_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'filter_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_se(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_se(x, na.rm = FALSE, ...)

Arguments

x  An R object. Currently this package defines methods for collections of spectral objects.

na.rm  logical. A value indicating whether NA values should be stripped before the computation proceeds.

...  Further arguments passed to or from other methods.

Value

If \(x\) is a collection spectral of objects, such as a "filter_mspct" object, the returned object is of class "generic_spct", containing the standard error among the spectra at each wavelength in a column with name ending in ".se".

Methods (by class)

- default:
- source_mspct:
- response_mspct:
- filter_mspct:
- reflector_mspct:
- calibration_mspct:
- cps_mspct:
- raw_mspct:

Note

Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in \(x\) must share the same set of wavelengths.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.
**s_sum**

*Sum from collection of spectra*

**Description**

A method to compute the sum of values across members of a collections of spectra. Computes the sum at each wavelength across all the spectra in the collection returning a spectral object.

**Usage**

```r
s_sum(x, na.rm, ...)
```

### Default S3 method:
```r
default_s_x
```

### S3 method for class 'filter_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

### S3 method for class 'source_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

### S3 method for class 'response_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

### S3 method for class 'reflector_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

### S3 method for class 'calibration_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

### S3 method for class 'cps_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

### S3 method for class 'raw_mspct'
```r
s_sum(x, na.rm = FALSE, ...)
```

**Arguments**

- **x**
  - An R object. Currently this package defines methods for collections of spectral objects.

- **na.rm**
  - logical. A value indicating whether NA values should be stripped before the computation proceeds.

- **...**
  - Further arguments passed to or from other methods.
Value

If \( x \) is a collection spectral of objects, such as a “filter_mspct” object, the returned object is of same class as the members of the collection, such as “filter_spct”, containing the sum of the spectra.

Methods (by class)

- default:
- filter_mspct:
- source_mspct:
- response_mspct:
- reflector_mspct:
- calibration_mspct:
- cps_mspct:
- raw_mspct:

Note

Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in \( x \) must share the same set of wavelengths.

A sum of transmittances or reflectances is no longer a well defined physical quantity, and these sum operations return an object of class generic_spct.

Objects of classes raw_spct and cps_spct can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of cps_spct members.

See Also

See \texttt{sum} for the \texttt{sum()} method used for the computations.

\begin{tabular}{ll}
\texttt{s.var} & Variance of a collection of spectra \\
\end{tabular}

Description

A method to compute the variance of values across members of a collections of spectra. Computes the variance at each wavelength across all the spectra in the collection returning a spectral object.
s_var

Usage

s_var(x, na.rm, ...)

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

## S3 method for class 'filter_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'source_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'response_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'reflector_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'calibration_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'cps_mspct'
s_var(x, na.rm = FALSE, ...)

## S3 method for class 'raw_mspct'
s_var(x, na.rm = FALSE, ...)

Arguments

x An R object. Currently this package defines methods for collections of spectral objects.

na.rm logical. A value indicating whether NA values should be stripped before the computation proceeds.

... Further arguments passed to or from other methods.

Details

Variance method for collections of spectra. Computes the variance at each wavelength across all the spectra in the collection.

Value

If x is a collection spectral of objects, such as a “filter_mspct” object, the returned object is of class “generic_spect”, containing the variance among the spectra at each wavelength in a column with name ending in “.var”.
# Methods (by class)

- default:
- `filter_mspct`:
- `source_mspct`:
- `response_mspct`:
- `reflector_mspct`:
- `calibration_mspct`:
- `cps_mspct`:
- `raw_mspct`:

## Note

Omission of NAs is done separately at each wavelength. Interpolation is not applied, so all spectra in `x` must share the same set of wavelengths.

Objects of classes `raw_spct` and `cps_spct` can contain data from multiple scans. This functions are implemented for these classes only for the case when all member spectra contain data for a single scan, or spliced into a single column in the case of `cps_spct` members.

## See Also

See `cor` for details about `var()`, which is used for the computations.

## Description

Function that converts transmittance (fraction) into absorbance (a.u.).

## Usage

```r
T2A(x, action, byref, clean, ...)
```

- **Default S3 method:**

```r
T2A(x, action = NULL, byref = FALSE, ...)
```

- **S3 method for class 'numeric':**

```r
T2A(x, action = NULL, byref = FALSE, clean = TRUE, ...)
```

- **S3 method for class 'filter_spct':**

```r
T2A(x, action = "add", byref = FALSE, clean = TRUE, ...)
```

- **S3 method for class 'filter_mspct':**

```r
T2A(
```
Arguments

x          an R object
action     character Allowed values "replace" and "add"
byref      logical indicating if new object will be created by reference or by copy of x
clean      logical replace off-boundary values before conversion
...        not used in current version
.parallel   if TRUE, apply function in parallel, using parallel backend provided by foreach
.paropts   a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x with a column A added and other columns possibly deleted except for w.length. If action = "replace", in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- default: Default method for generic function
- numeric: Method for numeric vectors
- filter_spct: Method for filter spectra
- filter_mspct: Method for collections of filter spectra

See Also

Other quantity conversion functions: A2T(), Afr2T(), T2Afr(), any2T(), as_quantum(), e2qmol_multipliers(), e2quantum_multipliers(), e2q(), q2e()
T2Afr

Convert transmittance into absorptance.

Description

Function that converts transmittance (fraction) into absorptance (fraction). If reflectance (fraction) is available, it allows conversions between internal and total absorptance.

Usage

T2Afr(x, action, byref, clean, ...)

## Default S3 method:
T2Afr(x, action = NULL, byref = FALSE, clean = FALSE, ...)

## S3 method for class 'numeric'
T2Afr(x, action = NULL, byref = FALSE, clean = FALSE, Rfr = NA_real_, ...)

## S3 method for class 'filter_spct'
T2Afr(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'object_spct'
T2Afr(x, action = "add", byref = FALSE, clean = FALSE, ...)

## S3 method for class 'filter_mspct'
T2Afr(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'object_mspct'
T2Afr(
  x,
  action = "add",
  byref = FALSE,
  clean = FALSE,
  ...,
  .parallel = FALSE,
  .paropts = NULL
)
Arguments

- **x**: an R object
- **action**: character. Allowed values "replace" and "add"
- **byref**: logical indicating if new object will be created by reference or by copy of x
- **clean**: logical. Replace off-boundary values before conversion
- **Rfr**: numeric vector. Spectral reflectance or reflectance factor. Set to zero if x is internal reflectance.
- **.parallel**: if TRUE, apply function in parallel, using parallel backend provided by foreach
- **.paropts**: a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A copy of x with a column Afr added and other columns possibly deleted except for w.length. If action = "replace", in all cases, the additional columns are removed, even if no column needs to be added.

Methods (by class)

- default: Default method for generic function
- numeric: Default method for generic function
- filter_spect: Method for filter spectra
- object_spect: Method for object spectra
- filter_mspct: Method for collections of filter spectra
- object_mspct: Method for collections of object spectra

See Also

Other quantity conversion functions: `A2T()`, `Afr2T()`, `T2A()`, `any2T()`, `as_quantum()`, `e2qmol_multipliers()`, `e2quantum_multipliers()`, `e2q()`, `q2e()`

Examples

```r
T2Afr(Ler_leaf.spct)
```
Description

Spectra are tagged by adding variables and attributes containing color definitions, labels, and a factor following the wavebands given in `w.band`. This method is most useful for plotting realistic computed colors from spectral data.

Usage

```r
tag(x, ...)

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

## S3 method for class 'generic_spct'
tag(
x,
w.band = NULL,
wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
use.hinges = TRUE,
short.names = TRUE,
chroma.type = "CMF",
byref = FALSE,
...
)

## S3 method for class 'generic_mspct'
tag(
x,
w.band = NULL,
wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
use.hinges = TRUE,
short.names = TRUE,
chroma.type = "CMF",
byref = FALSE,
..., .parallel = FALSE,
.paropts = NULL
)
```

Arguments

- `x` an R object.
- `...` ignored (possibly used by derived methods).
w.band waveband or list of waveband objects. The waveband(s) determine the region(s) of the spectrum that are tagged.

wb.trim logical Flag telling if wavebands crossing spectral data boundaries are trimmed or ignored.

use.hinges logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

short.names logical Flag indicating whether to use short or long names for wavebands.

chroma.type character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class chroma_spct for any other trichromic visual system.

byref logical Flag indicating if new object will be created by reference or by copy of x.

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach.

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value
A copy of x expanded with additional columns with color-related information.

Methods (by class)
- default: Default method for generic
- generic_spct: Tag one of generic_spct, and derived classes including source_spct, filter_spct, reflector_spct, object_spct, and response_spct.
- generic_mspct: Tag one of generic_mspct, and derived classes including source_mspct, filter_mspct, reflector_mspct, object_mspct, and response_mspct.

Note
NULL as w.band argument does not add any new tags, instead it removes existing tags if present. NA, the default, as w.band argument removes existing waveband tags if present and sets the wl.color variable. If a waveband object or a list of wavebands is supplied as argument then tagging is based on them, and wl.color is also set.

See Also
Other tagging and related functions: is_tagged(), untag(), wb2rect_spct(), wb2spct(), wb2tagged_spct()

Examples

tag(sun.spct)
tag(sun.spct, list(A = waveband(c(300,3005))))
Thin the density of wavelength values

Description
Increase the wavelength step in stored spectral data in featureless regions to save storage space.

Usage
thin_wl(x, ...)

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

## S3 method for class 'generic_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

## S3 method for class 'source_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'response_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),
  ...
)

## S3 method for class 'filter_spct'
thin_wl(
  x,
  max.wl.step = 10,
  max.slope.delta = 0.001,
  qty.out = getOption("photobiology.filter.qty", default = "transmittance"),
  ...
)

## S3 method for class 'reflector_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, ...)
thin_wl

```r
## S3 method for class 'raw_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

## S3 method for class 'cps_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

## S3 method for class 'object_spct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, col.names, ...)

## S3 method for class 'chroma_spct'
thin_wl(x, ...)

## S3 method for class 'calibration_spct'
thin_wl(x, ...)

## S3 method for class 'generic_mspct'
thin_wl(x, max.wl.step = 10, max.slope.delta = 0.001, ...)

## S3 method for class 'chroma_mspct'
thin_wl(x, ...)

## S3 method for class 'calibration_mspct'
thin_wl(x, ...)
```

**Arguments**

- `x` An R object
- `...` additional named arguments passed down to `f`
- `max.wl.step` numeric. Largest allowed wavelength difference between adjacent spectral values in nanometres (nm).
- `max.slope.delta` numeric in 0 to 1. Largest allowed change in relative slope of the spectral quantity per nm between adjacent pairs of values.
- `col.names` character. Name of the column of `x` containing the spectral data to check against `max.slope.delta`. Currently only one column supported.
- `unit.out` character. Allowed values "energy", and "photon", or its alias "quantum".
- `qty.out` character. Allowed values "transmittance", and "absorbance".

**Details**

The algorithm used for spectra is "naive" in an effort to keep it efficient. It works by iteratively attempting to delete every other observation along wavelengths, based on the criteria for maximum wavelength step and maximum relative step in the spectral variable between adjacent data values.

**Value**

An object of the same class as `x` but with a reduced density of wavelength values in those regions where slope is shallow and featureless.
Methods (by class)

- default: Default for generic function
- generic_spct:
- source_spct:
- response_spct:
- filter_spct:
- reflector_spct:
- raw_spct:
- cps_spct:
- object_spct:
- chroma_spct:
- calibration_spct:
- generic_mspct:
- chroma_mspct:
- calibration_mspct:

Note

The value of `max.slope.delta` is expressed as relative change in the slope of spectral variable per nanometre. This means that values between 0.0005 and 0.005 tend to work reasonably well. The best value will depend on the wavelength step of the input and noise in data. A moderate smoothing before thinning can sometimes help in the case of noisy data. The amount of thinning is almost always less than the value of criteria passed as argument as it is based on existing wavelength values. For example if we start with a spectrum with a uniform wavelength step of 1 nm, possible steps in the thinned spectrum are 2, 4, 8, 16, 32, etc. nm. The algorithm, does work with any step sizes, regular or variable in the input. Thinning is most effective for spectra with large "featureless" regions as the algorithm attempts not to discard information, contrary to smoothing or interpolation.

See Also

Other experimental utility functions: `collect2mspct()`, `drop_user_cols()`, `uncollect2spct()`

Examples

```r
nrow(yellow_gel.spct)
wls_stepsizes(yellow_gel.spct)
thinned.spct <- thin_wl(yellow_gel.spct)
nrow(thinned.spct)
wls_stepsizes(thinned.spct)
```
**times-.generic_spct**  
*Arithmetic Operators*

---

**Description**

Multiplication operator for spectra.

**Usage**

```r
## S3 method for class 'generic_spct'
e1 * e2
```

**Arguments**

- `e1`: an object of class "generic_spct"
- `e2`: an object of class "generic_spct"

**See Also**

Other math operators and functions: `MathFun`, `.^-.generic_spct`, `convolve_each()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`

---

**transmittance**  
*Transmittance*

---

**Description**

Summary transmittance for supplied wavebands from filter or object spectrum.

**Usage**

```r
transmittance(spct, w.band, quantity, wb.trim, use.hinges, ...)
```

```r
## Default S3 method:
transmittance(spct, w.band, quantity, wb.trim, use.hinges, ...)
```

```r
## S3 method for class 'filter_spct'
transmittance(
  spct,
  w.band = NULL,
  quantity = "average",
  wb.trim = getOption("photobiology.waveband.trim", default = TRUE),
  use.hinges = NULL,
  naming = "default",
)```
Arguments

spct an R object.

w.band waveband or list of waveband objects or a numeric vector of length two. The waveband(s) determine the region(s) of the spectrum that are summarized. If a numeric range is supplied a waveband object is constructed on the fly from it.
**transmittance** 353

- **quantity** character string One of "total", "average" or "mean", "contribution", "contribution.pc", "relative" or "relative.pc".
- **wb.trim** logical Flag indicating if wavebands crossing spectral data boundaries are trimmed or ignored.
- **use.hinges** logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- **...** ignored (possibly used by derived methods).
- **naming** character one of "long", "default", "short" or "none". Used to select the type of names to assign to returned value.
- **attr2tb** character vector, see `add_attr2tb` for the syntax for `attr2tb` passed as is to formal parameter `col.names`.
- **idx** character Name of the column with the names of the members of the collection of spectra.
- **.parallel** if TRUE, apply function in parallel, using parallel backend provided by foreach.
- **.paropts** a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

**Value**

A named numeric vector in the case of methods for individual spectra, with one value for each waveband passed to parameter `w.band`. A data.frame in the case of collections of spectra, containing one column for each waveband object, an index column with the names of the spectra, and optionally additional columns with metadata values retrieved from the attributes of the member spectra.

By default values are only integrated, but depending on the argument passed to parameter `quantity` they can be re-expressed as relative fractions or percentages. In the case of vector output, `names` attribute is set to the name of the corresponding waveband unless a named list is supplied in which case the names of the list members are used.

**Methods (by class)**

- **default**: Default method
- **filter_spct**: Method for filter spectra
- **object_spct**: Method for object spectra
- **filter_mspct**: Calculates transmittance from a `filter_mspct`
- **object_mspct**: Calculates transmittance from a `object_mspct`

**Note**

The `use.hinges` parameter controls speed optimization. The defaults should be suitable in most cases. Only the range of wavelengths in the wavebands is used and all BSWFs are ignored.
Trig

Trigonometric Functions

Examples

transmittance(polyester.spct, waveband(c(280, 315)))
transmittance(polyester.spct, waveband(c(315, 400)))
transmittance(polyester.spct, waveband(c(400, 700)))

Description

Trigonometric functions for object of generic_spct and derived classes. The functions are applied to the spectral data, not the wavelengths. The quantity in the spectrum to which the function is applied depends on the class of x and the current value of output options.

Usage

```r
## S3 method for class 'generic_spct'
cos(x)

## S3 method for class 'generic_spct'
sin(x)

## S3 method for class 'generic_spct'
tan(x)

## S3 method for class 'generic_spct'
acos(x)

## S3 method for class 'generic_spct'
asin(x)

## S3 method for class 'generic_spct'
atan(x)
```

Arguments

x

an object of class "generic_spct" or a derived class.
Trim the "instr.desc" attribute

Description

Function to trim the "instr.desc" attribute of an existing generic_spct object, discarding all fields except for 'spectrometer.name', 'spectrometer.sn', 'bench.grating', 'bench.slit', and calibration name.

Usage

trimInstrDesc(  
    x,  
    fields = c("time", "spectrometer.name", "spectrometer.sn", "bench.grating", "bench.slit")  
)

Arguments

x a generic_spct object

fields a character vector with the names of the fields to keep, or if first member is "-"*, the names of fields to delete; "*" as first member of the vector makes the function a no-op, leaving the spectrum object unaltered.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct object, x is not modified.

See Also

trimInstrSettings  

Trim the "instr.settings" attribute

Description

Function to trim the "instr.settings" attribute of an existing generic_spct object, by discarding some fields.

Usage

trimInstrSettings(x, fields = "*")

Arguments

x  
a generic_spct object

fields  
a character vector with the names of the fields to keep, or if first member is "-"., the names of fields to delete; "*" as first member of the vector makes the function a no-op, leaving the spectrum object unaltered.

Value

x

Note

This function alters x itself by reference and in addition returns x invisibly. If x is not a generic_spct object, x is not modified.

See Also


trim_spct  

Trim (or expand) head and/or tail of a spectrum

Description

Trim head and tail of a spectrum based on wavelength limits, interpolating the values at the boundaries of the range. Trimming is needed for example to remove short wavelength noise when the measured spectrum extends beyond the known emission spectrum of the measured light source. Occasionally one may want also to expand the wavelength range.
Usage

trim_spct(
  spct,
  range = NULL,
  low.limit = NULL,
  high.limit = NULL,
  use.hinges = TRUE,
  fill = NULL,
  byref = FALSE,
  verbose = getOption("photobiology.verbose")
)

trim_mspct(
  mspct,
  range = NULL,
  low.limit = NULL,
  high.limit = NULL,
  use.hinges = TRUE,
  fill = NULL,
  byref = FALSE,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)

trim2overlap(
  mspct,
  use.hinges = TRUE,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)

extend2extremes(
  mspct,
  use.hinges = TRUE,
  fill = NA,
  verbose = getOption("photobiology.verbose"),
  .parallel = FALSE,
  .paropts = NULL
)

Arguments

spct an object of class "generic_spct".
range a numeric vector of length two, or any other object for which method range() will return a numeric vector of length two.
low.limit shortest wavelength to be kept (defaults to shortest w.length value).
trim_spct

high.limit
longest wavelength to be kept (defaults to longest w.length value).

use.hinges
logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

fill
if fill==NULL then tails are deleted, otherwise tails or s.irrad are filled with the value of fill.

byref
logical indicating if new object will be created by reference or by copy of spct.

verbose
logical.

mspct
an object of class "generic_mspct"

.parallel
if TRUE, apply function in parallel, using parallel backend provided by foreach

.paropts
a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

a spectrum of same class as input with its tails trimmed or expanded.

Note

When expanding a spectrum, if fill==NULL, then expansion is not performed. Range can be "waveband" object, a numeric vector or a list of numeric vectors, or any other user-defined or built-in object for which range() returns a numeric vector of length two, that can be interpreted as wavelengths expressed in nm.

See Also

Other trim functions: clip_wl(), trim_waveband(), trim_wl()

Examples

trim_spct(sun.spct, low.limit=300)
trim_spct(sun.spct, low.limit=300, fill=NULL)
trim_spct(sun.spct, low.limit=300, fill=NA)
trim_spct(sun.spct, low.limit=300, fill=0.0)
trim_spct(sun.spct, range = c(300, 400))
trim_spct(sun.spct, range = c(300, NA))
trim_spct(sun.spct, range = c(NA, 400))
Description

Trim tails of a spectrum based on wavelength limits, interpolating the values at the boundaries. Trimming is needed for example to remove short wavelength noise when the measured spectrum extends beyond the known emission spectrum of the measured light source. Occasionally one may want also to expand the wavelength range.

Usage

```r
trim_tails(
  x,
  y,
  low.limit = min(x),
  high.limit = max(x),
  use.hinges = TRUE,
  fill = NULL,
  verbose = TRUE
)
```

Arguments

- `x`: numeric vector of wavelengths.
- `y`: numeric vector of values for a spectral quantity.
- `low.limit`: smallest x-value to be kept (defaults to smallest x-value in input).
- `high.limit`: largest x-value to be kept (defaults to largest x-value in input).
- `use.hinges`: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- `fill`: if `fill == NULL` then tails are deleted, otherwise tails of `y` are filled with the value of `fill`.
- `verbose`: logical Use to suppress warnings.

Value

A `data.frame` with variables `x` and `y`.

Note

When expanding a spectrum, if `fill == NULL`, expansion is not performed with a warning.
trim_waveband

Trim (or expand) head and/or tail

Description
Trimming of waveband boundaries can be needed when the spectral data do not cover the whole waveband, or wavebands may have to be removed altogether.

Usage
trim_waveband(
  w.band, 
  range = NULL, 
  low.limit = 0, 
  high.limit = Inf, 
  trim = getOption("photobiology.waveband.trim", default = TRUE), 
  use.hinges = TRUE, 
  trunc.labels = getOption("photobiology.brief.trunc.names", default = c("[", "]")) 
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.band</td>
<td>an object of class &quot;waveband&quot; or a list of such objects.</td>
</tr>
<tr>
<td>range</td>
<td>a numeric vector of length two, or any other object for which function range() will return a numeric vector of two wavelengths (nm).</td>
</tr>
<tr>
<td>low.limit</td>
<td>shortest wavelength to be kept (defaults to 0 nm).</td>
</tr>
<tr>
<td>high.limit</td>
<td>longest wavelength to be kept (defaults to Inf nm).</td>
</tr>
<tr>
<td>trim</td>
<td>logical (default is TRUE which trims the wavebands at the boundary, while FALSE discards wavebands that are partly off-boundary).</td>
</tr>
</tbody>
</table>
**trim_waveband**

**use.hinges** logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

**trunc.labels** character vector of length one or two. The first string will be prepended to the waveband name and label on left truncation and the second appended on right truncation. If the vector is of length one, the same string will be used in both cases.

**Details**

This function will accept both individual wavebands or list of wavebands. When the input is a list, wavebands outside the range of the range will be removed from the list, and those partly outside the target range either "trimmed" to this edge truncated if `trim = TRUE` is passed or excluded if `trim = FALSE`). Waveband objects contain a name and a label that are used to label the returned values of calculations that make use of them. When a waveband object is truncated so that the definition changes, the name and label are also modified so that the change is visible when they are used. The name and label have a string prepended or appended, and what strings are used can be set with an R option.

**Value**

The returned value is a waveband object or a list of waveband objects depending on whether a single waveband object or a list of waveband objects was supplied as argument to formal parameter `w.band`. If no waveband is retained, in the first case, a NULL waveband object is returned, and in the second case, a list of length zero is returned. If the input is a named list, names are preserved in the returned list.

**Note**

Modification of the name and label stored in the wavebands passed as input is done so that summaries produced with the modified objects can be recognized as different from those computed using the original definitions when the waveband objects are used. When the input is a named list, the names of the retained members of the list are not modified as these are not part of the definitions.

**See Also**

Other trim functions: `clip_wl()`, `trim_spct()`, `trimwl()`

**Examples**

```r
VIS <- waveband(c(380, 760)) # manometers
trim_waveband(VIS, c(400,700))
trim_waveband(VIS, low.limit = 400)
trim_waveband(VIS, high.limit = 700)
trim_waveband(VIS, c(400,700), trunc.labels = c(">", "<"))
trim_waveband(VIS, c(400,700), trunc.labels = "!")
```
trim wl

Trim head and/or tail of a spectrum

Description

Trim head and tail of a spectrum based on wavelength limits, with interpolation at range boundaries used by default. Expansion is also possible.

Usage

trim wl(x, range, use.hinges, fill, ...)

## Default S3 method:
trim wl(x, range, use.hinges, fill, ...)

## S3 method for class 'generic_spct'
trim wl(x, range = NULL, use.hinges = TRUE, fill = NULL, ...)

## S3 method for class 'generic_mspct'
trim wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  ..., 
  .parallel = FALSE,
  .paropts = NULL
)

## S3 method for class 'waveband'
trim wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  trim = getOption("photobiology.waveband.trim", default = TRUE),
  ...
)

## S3 method for class 'list'
trim wl(
  x,
  range = NULL,
  use.hinges = TRUE,
  fill = NULL,
  trim = getOption("photobiology.waveband.trim", default = TRUE),
  ...
)
Arguments

- **x**
  - an R object.

- **range**
  - a numeric vector of length two, or any other object for which function range() will return two.

- **use.hinges**
  - logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

- **fill**
  - if fill == NULL then tails are deleted, otherwise tails are filled with the value of fill.

- **...**
  - ignored (possibly used by derived methods).

- **.parallel**
  - if TRUE, apply function in parallel, using parallel backend provided by foreach

- **.paropts**
  - a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

- **trim**
  - logical (default is TRUE which trims the wavebands at the boundary, while FALSE discards wavebands that are partly off-boundary).

Value

A copy of x, usually trimmed or expanded to a different length, either shorter or longer. Possibly with some of the original spectral data values replaced with fill.

Methods (by class)

- **default**: Default for generic function
- **generic_spct**: Trim an object of class "generic_spct" or derived.
- **generic_mspct**: Trim an object of class "generic_mspct" or derived.
- **waveband**: Trim an object of class "waveband".
- **list**: Trim a list (of "waveband" objects).

Note

By default the w.length values for the first and last rows in the returned object are the values supplied as range.

trim_wl when applied to waveband objects always inserts hinges when trimming.

See Also

Other trim functions: clip_wl(), trim_spct(), trim_waveband()
Examples

```r
trim_wl(sun.spct, range = c(400, 500))
trim_wl(sun.spct, range = c(NA, 500))
trim_wl(sun.spct, range = c(400, NA))
```

---

**tz_time_diff**

*Time difference between two time zones*

**Description**

Returns the time difference in hours between two time zones at a given instant in time.

**Usage**

```r
tz_time_diff(
  when = lubridate::now(),
  tz.target = lubridate::tz(when),
  tz.reference = "UTC"
)
```

**Arguments**

- `when`: datetime A time instant
- `tz.target, tz.reference`: character Two time zones using names recognized by functions from package 'lubridate'

**Value**

A numeric value.

---

**uncollect2spct**

*Extract all members from a collection*

**Description**

Extract all members from a collection into separate objects in the parent frame of the call.
Usage

uncollect2spct(x, ...)

## Default S3 method:
uncollect2spct(x, ...)

## S3 method for class 'generic_mspct'
uncollect2spct(
  x,
  name.tag = ".spct",
  ignore.case = FALSE,
  check.names = TRUE,
  check.overwrite = TRUE,
  ...
)

Arguments

x An R object

... additional named arguments passed down to f.

name.tag character. A string used as tag for the names of the objects. If of length zero, names of members are used as named of objects. Otherwise the tag is appended, unless already present in the member name.

ignore.case logical. If FALSE, the pattern matching used for name.tag is case sensitive and if TRUE, case is ignored during matching.

check.names logical. If TRUE then the names of the objects created are checked to ensure that they are syntactically valid variable names and unique. If necessary they are adjusted (by make.names) so that they are, and if FALSE names are used as is.

check.overwrite logical. If TRUE trigger an error if an existing object would be overwritten, and if FALSE silently overwrite objects.

Value

Utility used for its side effects, invisibly returns a character vector with the names of the objects created.

Methods (by class)

- default: Default for generic function
- generic_mspct:

See Also

Other experimental utility functions: `collect2mspct()`, `drop_user_cols()`, `thin_wl()`
Examples

```r
my.mscpt <- source_mspct(list(sun1.spct = sun.spct, sun2.spct = sun.spct))
uncollect2spct(my.mscpt)
ls(pattern = "*.spct")
```

##

untag

Remove tags

Description

Remove tags from an R object if present, otherwise return the object unchanged.

Usage

```r
untag(x, ...)
```

##

### Default S3 method:

```r
untag(x, ...)
```

### S3 method for class 'generic_spct'

```r
untag(x, byref = FALSE, ...)
```

### S3 method for class 'generic_mspct'

```r
untag(x, byref = FALSE, ...)
```

Arguments

- `x` an R object.
- `...` ignored (possibly used by derived methods).
- `byref` logical indicating if new object will be created by reference or by copy of `x`

Value

if `x` contains tag data they are removed and the "spct.tags" attribute is set to NA, while if `x` has no tags, it is not modified. In either case, the `byref` argument is respected: in all cases if `byref = FALSE` a copy of `x` is returned.

Methods (by class)

- default: Default for generic function
- `generic_spct`: Specialization for `generic_spct`
- `generic_mspct`: Specialization for `generic_spct`

See Also

Other tagging and related functions: `is_tagged()`, `tag()`, `wb2rect_spct()`, `wb2spct()`, `wb2tagged_spct()`
upgrade_spct

Upgrade one spectral object

Description
Update the spectral class names of objects to those used in photobiology (>= 0.6.0) and add 'version' attribute as used in photobiology (>= 0.70).

Usage
upgrade_spct(object)

Arguments
object generic.spct A single object to upgrade

Value
The modified object (invisibly).

Note
The object is modified by reference. The class names with ending "spct" replaced by their new equivalents ending in "_spct".

See Also
Other upgrade from earlier versions: is.old_spct(), upgrade_spectra()

upgrade_spectra

Upgrade one or more spectral objects

Description
Update the spectral class names of objects to those used in photobiology (>= 0.6.0).

Usage
upgrade_spectra(obj.names = ls(parent.frame()))

Arguments
obj.names char Names of objects to upgrade as a vector of character strings

Value
The modified object (invisibly).
Note

The objects are modified by reference. The class names with ending ".spct" are replaced by their new equivalents ending in "_spct". object.names can safely include names of any R object. Names of objects which do not belong to any of the old .spct classes are ignored. This makes it possible to supply as argument the output from ls, the default, or its equivalent objects.

See Also

Other upgrade from earlier versions: is.old_spct(), upgrade_spct()

---

using_Tfr

*Use photobiology options*

**Description**

Execute an R expression, possibly compound, using a certain setting for spectral data related options.

**Usage**

using_Tfr(expr)
using_Afr(expr)
using_A(expr)
using_energy(expr)
using_photon(expr)
using_quantum(expr)

**Arguments**

expr an R expression to execute.

**Value**

The value returned by the execution of expression.

**References**

Based on withOptions() as offered by Thomas Lumley, and listed in [https://www.burns-stat.com/the-options-mechanism-in-r/](https://www.burns-stat.com/the-options-mechanism-in-r/), section Deep End, of "The Options mechanism in R" by Patrick Burns.
validate_geocode

Validate a geocode

Description
Test validity of a geocode or ensure that a geocode is valid.

Usage
validate_geocode(geocode)

is_valid_geocode(geocode)

length_geocode(geocode)

na_geocode()

Arguments
geocode data.frame with geocode data in columns "lat", "lon", and possibly also "address".

Details
validate_geocode Converts to tibble, checks data bounds, converts address to character if it is not already a character vector, or add character NAs if the address column is missing.
is_valid_geocode Checks if a geocode is valid, returning 0L if not, and the number of row otherwise.

Value
A valid geocode stored in a tibble.
FALSE for invalid, TRUE for valid.
FALSE for invalid, number of rows for valid.
A geo_code tibble with all fields set to suitable NAs.

Examples

validate_geocode(NA)
validate_geocode(data.frame(lon = -25, lat = 66))

is_valid_geocode(NA)
is_valid_geocode(1L)
is_valid_geocode(data.frame(lon = -25, lat = 66))

na_geocode()
Valleys or local minima

Description

Function that returns a subset of an R object with observations corresponding to local maxima.

Usage

valleys(x, span, ignore_threshold, strict, ...)

## Default S3 method:
valleys(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)

## Default S3 method:
valleys(x, span = NA, ignore_threshold = NA, strict = NA, na.rm = FALSE, ...)

## S3 method for class 'numeric'
valleys(x, span = 5, ignore_threshold, strict = TRUE, na.rm = FALSE, ...)

## S3 method for class 'data.frame'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  x.var.name = NULL,
  y.var.name = NULL,
  var.name = y.var.name,
  refine.wl = FALSE,
  method = "spline",
  ...
)

## S3 method for class 'generic_spct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = NULL,
  refine.wl = FALSE,
  method = "spline",
  ...
)
## S3 method for class 'source_spct'
valleys(
    x,
    span = 5,
    ignore_threshold = 0,
    strict = TRUE,
    na.rm = FALSE,
    unit.out = getOption("photobiology.radiation.unit", default = "energy"),
    refine.wl = FALSE,
    method = "spline",
    ...
)

## S3 method for class 'response_spct'
valleys(
    x,
    span = 5,
    ignore_threshold = 0,
    strict = TRUE,
    na.rm = FALSE,
    unit.out = getOption("photobiology.radiation.unit", default = "energy"),
    refine.wl = FALSE,
    method = "spline",
    ...
)

## S3 method for class 'filter_spct'
valleys(
    x,
    span = 5,
    ignore_threshold = 0,
    strict = TRUE,
    na.rm = FALSE,
    filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
    refine.wl = FALSE,
    method = "spline",
    ...
)

## S3 method for class 'reflector_spct'
valleys(
    x,
    span = 5,
    ignore_threshold = 0,
    strict = TRUE,
    na.rm = FALSE,
    refine.wl = FALSE,
method = "spline",
...)

## S3 method for class 'cps_spct'
valleys(
x,
span = 5,
ignore_THRESHOLD = 0,
strict = TRUE,
nna.rm = FALSE,
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'raw_spct'
valleys(
x,
span = 5,
ignore_THRESHOLD = 0,
strict = TRUE,
nna.rm = FALSE,
var_name = "counts",
refine.wl = FALSE,
method = "spline",
...
)

## S3 method for class 'generic_mspct'
valleys(
x,
span = 5,
ignore_THRESHOLD = 0,
strict = TRUE,
nna.rm = FALSE,
var_name = NULL,
refine.wl = FALSE,
method = "spline",
..., .parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'source_mspct'
valleys(
x,
span = 5,
valleys

ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
refine.wl = FALSE,
method = "spline",
...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'response_mspct'
valleys(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
unit.out = getOption("photobiology.radiation.unit", default = "energy"),
refine.wl = FALSE,
method = "spline",
...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'filter_mspct'
valleys(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
filter.qty = getOption("photobiology.filter.qty", default = "transmittance"),
refine.wl = FALSE,
method = "spline",
...
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'reflector_mspct'
valleys(
x,
span = 5,
ignore_threshold = 0,
strict = TRUE,
na.rm = FALSE,
valleys = FALSE,
method = "spline",
..., 
.parallel = FALSE,
.paropts = NULL
)

## S3 method for class 'cps_mspct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "cps",
  refine.wl = FALSE,
  method = "spline",
  ..., 
 .parallel = FALSE,
 .paropts = NULL
)

## S3 method for class 'raw_mspct'
valleys(
  x,
  span = 5,
  ignore_threshold = 0,
  strict = TRUE,
  na.rm = FALSE,
  var.name = "counts",
  refine.wl = FALSE,
  method = "spline",
  ..., 
 .parallel = FALSE,
 .paropts = NULL
)

Arguments

x an R object

span integer A valley is defined as an element in a sequence which is smaller than all other elements within a window of width span centered at that element. Use NULL for the global peak.

ignore_threshold numeric Value between 0.0 and 1.0 indicating the relative size compared to tallest peak threshold below which peaks will be ignored. Negative values set a threshold so that the tallest peaks are ignored, instead of the shortest.

strict logical If TRUE, an element must be strictly greater than all other values in its
window to be considered a peak.

... ignored

na.rm logical indicating whether NA values should be stripped before searching for peaks.

var.name, x.var.name, y.var.name character Name of column where to look for valleys.

refine.wl logical Flag indicating if valley location should be refined by fitting a function.

method character String with the name of a method. Currently only spline interpolation is implemented.

unit.out character One of "energy" or "photon"

filter.qty character One of "transmittance" or "absorbance"

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A subset of x with rows corresponding to local minima.

Methods (by class)

- default: Default function usable on numeric vectors.
- default: Default returning always NA.
- numeric: Default function usable on numeric vectors.
- data.frame: Method for "data.frame" objects.
- generic_spect: Method for "generic_spect" objects.
- source_spect: Method for "source_spect" objects.
- filter_spect: Method for "filter_spect" objects.
- reflector_spect: Method for "reflector_spect".
- cps_spect: Method for "cps_spect" objects.
- generic_mspct: Method for "generic_mspct" objects.
- source_mspct: Method for "source_mspct" objects.
- response_mspct: Method for "cps_mspct" objects.
- filter_mspct: Method for "filter_mspct" objects.
- reflector_mspct: Method for "reflector_mspct" objects.
- cps_mspct: Method for "cps_mspct" objects.
- raw_mspct: Method for "raw_mspct" objects.
See Also
Other peaks and valleys functions: `find_peaks()`, `find_spikes()`, `get_peaks()`, `peaks()`, `replace_bad_pixs()`, `spikes()`, `wls_at_target()`

Examples

```r
call_to_function <- function(sun.spct, span = 50)
call_to_function(sun.spct)
call_to_function(sun.spct)
```

---

`verbose_as_default`  
Set error reporting options

---

Description
Set error reporting related options easily.

Usage

```r
verbose_as_default(flag = TRUE)
strict_range_as_default(flag = TRUE)
```

Arguments

- `flag` logical.

Value

Previous value of the modified option.

---

`v_insert_hinges`  
Insert spectral data values at new wavelength values.

---

Description
Inserting wavelengths values immediately before and after a discontinuity in the SWF, greatly reduces the errors caused by interpolating the weighted irradiance during integration of the effective spectral irradiance. This is specially true when data have a relatively large wavelength step size and/or when the weighting function used has discontinuities in its value or slope. This function differs from `insert_hinges()` in that it returns a vector of y values instead of a tibble.

Usage

```r
v_insert_hinges(x, y, h)
```
v_replace_hinges

v_replace_hinges

Overwrite spectral data values at existing wavelength values.

Description

Overwriting spectral data with interpolated values at wavelengths values containing bad data is needed when cleaning spectral data. This function differs from `insert_hinges()` in that it returns a vector of y values instead of a tibble.

Usage

v_replace_hinges(x, y, h)

Arguments

- **x**: numeric vector (sorted in increasing order).
- **y**: numeric vector.
- **h**: a numeric vector giving the wavelengths at which the y values should be replaced by interpolation, no interpolation is indicated by an empty numeric vector (numeric(0)).

Value

A numeric vector with the numeric values of y with values at the hinges replaced by interpolation of neighbours.

See Also

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_replace_hinges()`
See Also

Other low-level functions operating on numeric vectors: `as_energy()`, `as_quantum_mol()`, `calc_multipliers()`, `div_spectra()`, `energy_irradiance()`, `energy_ratio()`, `insert_hinges()`, `integrate_xy()`, `interpolate_spectrum()`, `irradiance()`, `l_insert_hinges()`, `oper_spectra()`, `photon_irradiance()`, `photon_ratio()`, `photons_energy_ratio()`, `prod_spectra()`, `s_e_irrad2rgb()`, `split_energy_irradiance()`, `split_photon_irradiance()`, `subt_spectra()`, `sum_spectra()`, `trim_tails()`, `v_insert_hinges()`.

---

**water_vp_sat**

**Water vapour pressure**

**Description**

Approximate water pressure in air as a function of temperature, and its inverse the calculation of dewpoint.

**Usage**

```r
water_vp_sat(
  temperature,
  over.ice = FALSE,
  method = "tetens",
  check.range = TRUE
)

water_dp(water.vp, over.ice = FALSE, method = "tetens", check.range = TRUE)

water_fp(water.vp, over.ice = TRUE, method = "tetens", check.range = TRUE)

water_vp2mvc(water.vp, temperature)

water_mvc2vp(water.mvc, temperature)

water_vp2RH(
  water.vp,
  temperature,
  over.ice = FALSE,
  method = "tetens",
  pc = TRUE,
  check.range = TRUE
)

water_RH2vp(
  relative.humidity,
  temperature,
  over.ice = FALSE,
  method = "tetens",
)"
water_vp_sat

    pc = TRUE,
    check.range = TRUE
)

Arguments

- **temperature**: numeric vector of air temperatures (C).
- **over.ice**: logical Is the estimate for equilibrium with liquid water or with ice.
- **method**: character Currently "tetens", modified "magnus", "wexler" and "goff.gratch" equations are supported.
- **check.range**: logical Flag indicating whether to check or not that arguments for temperature are within the range of validity of the method used.
- **water.vp**: numeric vector of water vapour pressure in air (Pa).
- **water.mvc**: numeric vector of water vapour concentration as mass per volume (g m⁻³).
- **pc**: logical flag for result returned as percent or not.
- **relative.humidity**: numeric Relative humidity as fraction of 1.

Details

Function `water_vp_sat()` provides implementations of several well known equations for the estimation of saturation vapor pressure in air. Functions `water_dp()` and `water_fp()` use the inverse of these equations to compute the dew point or frost point from water vapour pressure in air. The inverse functions are either analytical solutions or fitted approximations. None of these functions are solved numerically by iteration.

Method "tetens" implements Tetens’ (1930) equation for the cases of equilibrium with a water and an ice surface. Method "magnus" implements the modified Magnus equations of Alduchov and Eskridge (1996, eqs. 21 and 23). Method "wexler" implements the equations proposed by Wexler (1976, 1977), and their inverse according to Hardy (1998). Method "goff.gratch" implements the equations of Groff and Gratch (1946) with the minor updates of Groff (1956).

The equations are approximations, and in spite of their different names, Tetens’ and Magnus’ equations have the same form with the only difference in the values of the parameters. However, the modified Magnus equation is more accurate as Tetens equation suffers from some bias errors at extreme low temperatures (< -40 C). In contrast Magnus equations with recently fitted values for the parameters are usable for temperatures from -80 C to +50 C over water and -80 C to 0 C over ice. The Groff Gratch equation is more complex and is frequently used as a reference in comparison as it is considered reliable over a broad range of temperatures. Wexler’s equations are computationally simpler and fitted to relatively recent data. There is little difference at temperatures in the range -20 C to +50 C, and differences become large at extreme temperatures. Temperatures outside the range where estimations are highly reliable for each equation return `NA`, unless extrapolation is enabled by passing `FALSE` as argument to parameter `check.range`.

The switch between equations for ice or water cannot be based on air temperature, as it depends on the presence or not of a surface of liquid water. It must be set by passing an argument to parameter `over.ice` which defaults to `FALSE`.

Tetens equation is still very frequently used, and is for example the one recommended by FAO for computing potential evapotranspiration. For this reason it is used as default here.
water_vp_sat

Value

A numeric vector of partial pressures in pascal (P) for water_vp_sat and water_mvc2vp, a numeric vector of dew point temperatures (C) for water_dp and numeric vector of mass per volume concentrations (g m^-3) for water_vp2mvc.

Note

The inverse of the Groff Gratch equation has yet to be implemented.

References


[Equations describing the physical properties of moist air](http://www.conservationphysics.org/atmcalc/atmoclc2.pdf)

Examples

```r
water_vp_sat(20) # C -> Pa
water_vp_sat(temperature = c(0, 10, 20, 30, 40)) # C -> Pa
water_vp_sat(temperature = -10) # over water!!
water_vp_sat(temperature = -10, over.ice = TRUE)
water_vp_sat(temperature = 20) / 100 # C -> mbar

water_vp_sat(temperature = 20, method = "magnus") # C -> Pa
water_vp_sat(temperature = 20, method = "tetens") # C -> Pa
water_vp_sat(temperature = 20, method = "wexler") # C -> Pa
water_vp_sat(temperature = 20, method = "goff.gratch") # C -> Pa

water_vp_sat(temperature = -20, over.ice = TRUE, method = "magnus") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "tetens") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "wexler") # C -> Pa
water_vp_sat(temperature = -20, over.ice = TRUE, method = "goff.gratch") # C -> Pa
```
waveband

waveband

Water_dp(water.vp = 1000) # Pa -> C
water_dp(water.vp = 1000, method = "magnus") # Pa -> C
water_dp(water.vp = 1000, method = "wexler") # Pa -> C
water_dp(water.vp = 500, over.ice = TRUE) # Pa -> C
water_dp(water.vp = 500, method = "wexler", over.ice = TRUE) # Pa -> C
water_fp(water.vp = 300) # Pa -> C
water_dp(water.vp = 300, over.ice = TRUE) # Pa -> C
water_vp2RH(water.vp = 1500, temperature = 20) # Pa, C -> RH %
water_vp2RH(water.vp = 1500, temperature = c(20, 30)) # Pa, C -> RH %
water_vp2RH(water.vp = c(600, 1500), temperature = 20) # Pa, C -> RH %
water_vp2mvc(water.vp = 1000, temperature = 20) # Pa -> g m-3
water_mvc2vp(water.mvc = 30, temperature = 40) # g m-3 -> Pa
water_dp(water.vp = water_mvc2vp(water.mvc = 10, temperature = 30)) # g m-3 -> C

Description

Constructor for "waveband" objects that can be used as input when calculating irradiances.

Usage

waveband(
  x = NULL,
  weight = NULL,
  SWF.e.fun = NULL,
  SWF.q.fun = NULL,
  norm = NULL,
  SWF.norm = NULL,
  hinges = NULL,
  wb.name = NULL,
  wb.label = wb.name
)

new_waveband(
  w.low,
  w.high,
  weight = NULL,
  SWF.e.fun = NULL,
  SWF.q.fun = NULL,
  norm = NULL,
SWF.norm = NULL,
hinges = NULL,
w.name = NULL,
w.label = w.name
)

Arguments

x
any R object on which applying the function range yields an vector of two numeric values, describing a range of wavelengths (nm)

weight
a character string "SWF" or "BSWF", use NULL (the default) to indicate no weighting used when calculating irradiance

SWF.e.fun
a function giving multipliers for a spectral weighting function (energy) as a function of wavelength (nm)

SWF.q.fun
a function giving multipliers for a spectral weighting function (quantum) as a function of wavelength (nm)

norm
a single numeric value indicating the wavelength at which the SWF should be normalized to 1.0, in nm. "NULL" means no normalization.

SWF.norm
a numeric value giving the native normalization wavelength (nm) used by SWF.e.fun and SWF.q.fun

hinges
a numeric vector giving the wavelengths at which the s.irrad should be inserted by interpolation, no interpolation is indicated by an empty vector (numeric(0)), if NULL then interpolation will take place at both ends of the band.

wb.name
character string giving the name for the waveband defined, default is NULL

wb.label
character string giving the label of the waveband to be used for plotting, default is wb.name

w.low
numeric value, wavelength at the short end of the band (nm)

w.high
numeric value, wavelength at the long end of the band (nm)

Value

a waveband object

Functions

- new_waveband: A less flexible variant

See Also

Other waveband constructors: split_bands()

Examples

waveband(c(400,700))

new_waveband(400,700)
waveband_ratio

Photon or energy ratio

Description

This function gives the (energy or photon) irradiance ratio between two given wavebands of a radiation spectrum.

Usage

```r
waveband_ratio(
  w.length,
  s.irrad,
  w.band.num = NULL,
  w.band.denom = NULL,
  unit.out.num = NULL,
  unit.out.denom = unit.out.num,
  unit.in = "energy",
  check.spectrum = TRUE,
  use.cached.mult = FALSE,
  use.hinges = getOption("photobiology.use.hinges", default = NULL)
)
```

Arguments

- **w.length**: numeric Vector of wavelengths (nm)
- **s.irrad**: numeric vector of spectral (energy or photon) irradiances (W m\(^{-2}\) nm\(^{-1}\)) or (mol s\(^{-1}\) m\(^{-2}\) nm\(^{-1}\)).
- **w.band.num**: waveband object used to compute the numerator of the ratio.
- **w.band.denom**: waveband object used to compute the denominator of the ratio.
- **unit.out.num**: character Allowed values "energy", and "photon", or its alias "quantum".
- **unit.out.denom**: character Allowed values "energy", and "photon", or its alias "quantum".
- **unit.in**: character Allowed values "energy", and "photon", or its alias "quantum".
- **check.spectrum**: logical Flag indicating whether to sanity check input data, default is TRUE.
- **use.cached.mult**: logical Flag indicating whether multiplier values should be cached between calls.
- **use.hinges**: logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.

Value

a single numeric value giving the ratio
**Note**

The default for both \texttt{w.band} parameters is a waveband covering the whole range of \texttt{w.length}. From version 9.19 onwards use of this default does not trigger a warning, but instead is used silently.

**Examples**

```r
# photon:photon ratio
with(sun.data,
    waveband_ratio(w.length, s.e.irrad,
                   new_waveband(400,500),
                   new_waveband(400,700), "photon"))

# energy:energy ratio
with(sun.data,
    waveband_ratio(w.length, s.e.irrad,
                   new_waveband(400,500),
                   new_waveband(400,700), "energy"))

# energy:photon ratio
with(sun.data,
    waveband_ratio(w.length, s.e.irrad,
                   new_waveband(400,700),
                   new_waveband(400,700), "energy", "photon"))

# photon:photon ratio waveband : whole spectrum
with(sun.data,
    waveband_ratio(w.length, s.e.irrad,
                   new_waveband(400,500),
                   unit.out.num="photon"))

# photon:photon ratio of whole spectrum should be equal to 1.0
with(sun.data,
    waveband_ratio(w.length, s.e.irrad,
                   unit.out.num="photon"))
```

---

**wb2rect_spct**  
*Create tagged spectrum from wavebands*

**Description**

Create a generic \texttt{spct} object with wavelengths from the range of wavebands in a list. The spectrum is suitable for plotting labels, symbols, rectangles or similar, as the midpoint of each waveband is added to the spectrum.

**Usage**

```r
wb2rect_spct(w.band, short.names = TRUE, chroma.type = "CMF")

fast_wb2rect_spct(w.band, chroma.type = "CMF", simplify = TRUE)
```
Arguments

w.band  waveband or list of waveband objects The waveband(s) determine the wavelengths in variable w.length of the returned spectrum
short.names  logical Flag indicating whether to use short or long names for wavebands
chroma.type  character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class chroma_spct for any other trichromic visual system.
simplify  logical Flag indicating whether to merge neighboring rectangles of equal color. Simplification is done only for narrow wavebands.

Value

A generic.spectrum object with columns w.length, wl.low, wl.hi, wl.color, wb.color and wb.name. The w.length values are the midpoint of the wavebands, wl.low and wl.high give the boundaries of the wavebands, wl.color the color definition corresponding to the wavelength at the center of the waveband and wb.color the color of the waveband as a whole (assuming a flat energy irradiance spectrum). Different spectral data variables are set to zero and added making the returned value compatible with classes derived from generic_spct.

Note

Function fast_wb2rect_spct() differs from wb2rect_spct() in that it computes colors for narrow wavebands based on the midpoint wavelength and uses vectorization when possible. It always returns color definitions with short names, which are also used as waveband names for narrow wavebands and merged wavebands. The purpose of merging of rectangles is to speed up rendering and to reduce the size of vector graphics output. This function should be used with care as the color definitions returned are only approximate and original waveband names can be lost.

See Also

Other tagging and related functions: is_tagged(), tag(), untag(), wb2spct(), wb2tagged_spct()
Value

A generic.spectrum object, with columns w.length set to the union of all boundaries and hinges defined in the waveband(s). Different spectral data variables are set to zero and added making the returned value compatible with classes derived from generic_spct.

See Also

Other tagging and related functions: is_tagged(), tag(), untag(), wb2rect_spct(), wb2tagged_spct()

---

**wb2tagged_spct**

Create tagged spectrum from wavebands

Description

Create a tagged generic_spct object with wavelengths from the range of wavebands in a list, and names of the same bands as factor levels, and corresponding color definitions. The spectrum is not suitable for plotting labels, symbols, rectangles or similar, as the midpoint of each waveband is not added to the spectrum.

Usage

```r
wb2tagged_spct(
  w.band, 
  use.hinges = TRUE, 
  short.names = TRUE, 
  chroma.type = "CMF", 
  ... 
)
```

Arguments

- `w.band` waveband or list of waveband objects The waveband(s) determine the region(s) of the spectrum that are tagged and the wavelengths returned in variable w.length.
- `use.hinges` logical Flag indicating whether to insert "hinges" into the spectral data before integration so as to reduce interpolation errors at the boundaries of the wavebands.
- `short.names` logical Flag indicating whether to use short or long names for wavebands.
- `chroma.type` character telling whether "CMF", "CC", or "both" should be returned for human vision, or an object of class chroma_spct for any other trichromic visual system. Ignored (possibly used by derived methods).
- `...` ignored (possibly used by derived methods).

Value

A spectrum as returned by wb2spct but additionally tagged using function tag

See Also

Other tagging and related functions: is_tagged(), tag(), untag(), wb2rect_spct(), wb2spct()
**wb_trim_as_default**

---

**wb_trim_as_default**  
*Set computation options*

---

**Description**

Set computation related options easily.

**Usage**

```r
wb_trim_as_default(flag = TRUE)

use_cached_mult_as_default(flag = TRUE)
```

**Arguments**

- **flag**  
  logical.

**Value**

Previous value of the modified option.

---

**white_body.spct**  
*Theoretical white body*

---

**Description**

A dataset for a hypothetical object with transmittance 0/1 (0%), reflectance 1/1 (100%)

**Format**

A `object_spct` object with 4 rows and 3 variables

**Details**

- w.length (nm)
- Tfr (0..1)
- Rfr (0..1)

**See Also**

white_led.cps_spct  White led bulb spectrum

Description
A dataset containing wavelengths and the corresponding spectral data as counts per second for an Osram warm white led lamp:

Usage
white_led.cps_spct

Format
A data.frame object with 2068 rows and 2 variables

Details
• w.length (nm), range 188 to 1117 nm.
• cps

See Also

Examples
white_led.cps_spct

white_led.raw_spct  White led bulb spectrum

Description
A dataset containing wavelengths and the corresponding spectral data as raw instrument counts for an Osram warm white led lamp, for three different integration times:

Usage
white_led.raw_spct
white_led.source_spct

Format
An object of class raw_spct (inherits from generic_spct, tbl_df, tbl, data.frame) with 2068 rows and 4 columns.

Details
• w.length (nm), range 188 to 1117 nm.
  • counts_1
  • counts_2
  • counts_3
• w.length (nm), range 188 to 1117 nm.
  • cps

See Also

Examples
white_led.raw_spct

white_led.source_spct  White led bulb spectrum

Description
A dataset containing wavelengths and the corresponding spectral irradiance data for an Osram warm white led lamp:

Usage
white_led.source_spct

Format
A source_spct object with 1421 rows and 2 variables

Details
• w.length (nm), range 250 to 900 nm.
  • s.e.irrad (W m-2 nm-1)
See Also


Examples

white_led.source_spct

---

**wls_at_target**  
*Find wavelengths values corresponding to a target spectral value*

**Description**

Find wavelength values corresponding to a target spectral value in a spectrum. The name of the column of the spectral data to be used is inferred from the class of `x` and the argument passed to `unit.out` or `filter.qty` or their defaults that depend on R options set.

**Usage**

```r
wls_at_target(
  x,
  target = NULL,
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
)
```

```r
## Default S3 method:
wls_at_target(
  x,
  target = NULL,
  interpolate = FALSE,
  idfactor = FALSE,
  na.rm = FALSE,
  ...
)
```

```r
## S3 method for class 'data.frame'
wls_at_target(
  x,
  target = "half.maximum",
  interpolate = FALSE,
```
wls_at_target

```r
wls_at_target(  
  x,  
  target = "half.maximum",  
  interpolate = FALSE,  
  idfactor = FALSE,  
  na.rm = FALSE,  
  col.name = NULL,  
  y.var.name = col.name,  
  ...  
)
```

## S3 method for class 'generic_spct'

```r
wls_at_target(  
  x,  
  target = "half.maximum",  
  interpolate = FALSE,  
  idfactor = FALSE,  
  na.rm = FALSE,  
  col.name = NULL,  
  y.var.name = NULL,  
  ...  
)
```

## S3 method for class 'source_spct'

```r
wls_at_target(  
  x,  
  target = "half.maximum",  
  interpolate = FALSE,  
  idfactor = FALSE,  
  na.rm = FALSE,  
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),  
  ...  
)
```

## S3 method for class 'response_spct'

```r
wls_at_target(  
  x,  
  target = "half.maximum",  
  interpolate = FALSE,  
  idfactor = FALSE,  
  na.rm = FALSE,  
  unit.out = getOption("photobiology.radiation.unit", default = "energy"),  
  ...  
)
```

## S3 method for class 'filter_spct'

```r
wls_at_target(  
  x,  
  target = "half.maximum",  
  interpolate = FALSE,  
  idfactor = FALSE,  
  na.rm = FALSE,  
  ...  
)
```
filter qty = getOption("photobiology.filter.qty", default = "transmittance"),
...

## S3 method for class 'reflector_spct'
## wls_at_target
## wls_at_target(
##   x,
##   target = "half.maximum",
##   interpolate = FALSE,
##   idfactor = FALSE,
##   na.rm = FALSE,
##   ...
## )

## S3 method for class 'cps_spct'
## wls_at_target(
##   x,
##   target = "half.maximum",
##   interpolate = FALSE,
##   idfactor = FALSE,
##   na.rm = FALSE,
##   ...
## )

## S3 method for class 'generic_mspct'
## wls_at_target(
##   x,
##   target = "half.maximum",
##   interpolate = FALSE,
##   idfactor = FALSE,
##   na.rm = FALSE,
##   ...
## , parallel = FALSE
## , paropts = NULL
## )

Arguments

x data.frame or spectrum object.
target numeric value indicating the spectral quantity value for which wavelengths are to be searched and interpolated if need. The character string "half.maximum" is also accepted as argument.
interpolate logical Indicating whether the nearest wavelength value in x should be returned or a value calculated by linear interpolation between wavelength values straddling the target.
idfactor logical or character Generates an index column of factor type. If idfactor = TRUE then the column is auto named spct.idx. Alternatively the column name can be directly passed as argument to idfactor as a character string.
wls_at_target

na.rm logical indicating whether NA values should be stripped before searching for the target.

... currently ignored.

x.var.name, y.var.name, col.name character The name of the columns in which to search for the target value. Use of col.name is deprecated, and is a synonym for y.var.name.

unit.out character One of "energy" or "photon"

filter.qty character One of "transmittance" or "absorbance"

.parallel if TRUE, apply function in parallel, using parallel backend provided by foreach

.paropts a list of additional options passed into the foreach function when parallel computation is enabled. This is important if (for example) your code relies on external data or packages: use the .export and .packages arguments to supply them so that all cluster nodes have the correct environment set up for computing.

Value

A data.frame or a spectrum object of the same class as x with fewer rows, possibly even no rows. If FALSE is passed to interpolate a subset of x is returned, otherwise a new object of the same class containing interpolated wavelengths for the target value is returned.

Methods (by class)

• default: Default returning always an empty object of the same class as x.
• data.frame: Method for "data.frame" objects.
• generic_spct: Method for "generic_spct" objects.
• source_spct: Method for "source_spct" objects.
• response_spct: Method for "response_spct" objects.
• filter_spct: Method for "filter_spct" objects.
• reflector_spct: Method for "reflector_spct" objects.
• cps_spct: Method for "cps_spct" objects.
• generic_mspct: Method for "generic_mspct" objects.

Note

When interpolation is used, only column w.length and the column against which the target value was compared are included in the returned object, otherwise, all columns in x are returned. We implement support for data.frame to simplify the coding of 'ggplot2' stats using this function.

See Also

Other peaks and valleys functions: find_peaks(), find_spikes(), get_peaks(), peaks(), replace_bad_pixs(), spikes(), valleys()
Examples

wls_at_target(sun.spct, target = 0.1)
wls_at_target(sun.spct, target = 2e-6, unit.out = "photon")
wls_at_target(polyester.spct, target = "HM")
wls_at_target(polyester.spct, target = "HM", interpolate = TRUE)
wls_at_target(polyester.spct, target = "HM", idfactor = "target")
wls_at_target(polyester.spct, target = "HM", filter.qty = "absorbance")

wl_max

Wavelength maximum

Description

A method specialization that returns the wavelength maximum from objects of classes "waveband" or of class "generic_spct" or derived.

Usage

wl_max(x, na.rm = FALSE)

## S3 method for class 'waveband'
max(..., na.rm = FALSE)

## S3 method for class 'generic_spct'
max(..., na.rm = FALSE)

## S3 method for class 'generic_mspct'
max(..., na.rm = FALSE, idx = "spct.idx")

Arguments

x generic_spct, generic_mspct or waveband object.
na.rm ignored
... not used in current version
idx character Name of the column with the names of the members of the collection of spectra.

Methods (by class)

- generic_spct:
- generic_mspct:

Examples

max(sun.spct)
wl_max(sun.spct)
A function that returns the wavelength (or value) at the center of the wavelength range of a waveband or spectrum object (or numeric vector).

**Usage**

```r
wl_midpoint(x, ...)
midpoint(x, ...)
```

## Default S3 method:

```r
midpoint(x, ...)
```

## S3 method for class `numeric`

```r
midpoint(x, ...)
```

## S3 method for class `waveband`

```r
midpoint(x, ...)
```

## S3 method for class `generic_spct`

```r
midpoint(x, ...)
```

## S3 method for class `generic_mspct`

```r
midpoint(x, ..., idx = "spct.idx")
```

**Arguments**

- `x` an R object
- `...` not used in current version
- `idx` character Name of the column with the names of the members of the collection of spectra.

**Value**

A numeric value equal to \((\max(x) - \min(x)) / 2\). In the case of spectral objects a wavelength in nm. For any other R object, according to available definitions of \(\min\) and \(\max\).

**Methods (by class)**

- `default`: Default method for generic function
- `numeric`: Default method for generic function
- `waveband`: Wavelength at center of a "waveband".
• `generic_spct`: Method for "generic_spct".
• `generic_mspct`: Method for "generic_mspct" objects.

See Also

Other wavelength summaries: `wl_min()`, `wl_range()`, `wl_stepsize()`

Examples

```r
midpoint(10:20)
midpoint(sun.spct)
wl_midpoint(sun.spct)

midpoint(sun.spct)
```

---

### `wl_min`

#### Wavelength minimum

**Description**

A method specialization that returns the wavelength minimum from objects of classes "waveband" or of class "generic_spct" or derived.

**Usage**

```r
wl_min(x, na.rm = FALSE)
```

### S3 method for class 'waveband'

```r
min(..., na.rm = FALSE)
```

### S3 method for class 'generic_spct'

```r
min(..., na.rm = FALSE)
```

### S3 method for class 'generic_mspct'

```r
min(..., na.rm = FALSE, idx = "spct.idx")
```

**Arguments**

- `x`  
generic_spct, generic_mspct or waveband object.
- `na.rm`  
ignored
- `...`  
not used in current version
- `idx`  
character Name of the column with the names of the members of the collection of spectra.
Methods (by class)

• generic_spct:
  • generic_mspct:

See Also

Other wavelength summaries: \texttt{wl_midpoint()}, \texttt{wl_range()}, \texttt{wl_stepsize()}

Examples

\begin{verbatim}
min(sun.spct)
wl_min(sun.spct)
\end{verbatim}

<table>
<thead>
<tr>
<th>\texttt{wl_range}</th>
<th>\textit{Wavelength range}</th>
</tr>
</thead>
</table>

Description

A method specialization that returns the wavelength range from objects of classes "waveband" or of class "generic_spct" or derived.

Usage

\begin{verbatim}
wl_range(x, na.rm = FALSE)
## S3 method for class 'waveband'
range(..., na.rm = FALSE)
## S3 method for class 'generic_spct'
range(..., na.rm = FALSE)
## S3 method for class 'generic_mspct'
range(..., na.rm = FALSE, idx = "spct.idx")
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} \text{generic_spct, generic_mspct or waveband object.}
  \item \texttt{na.rm} \hspace{1cm} \text{ignored}
  \item \texttt{...} \hspace{1cm} \text{a single R object}
  \item \texttt{idx} \hspace{1cm} \text{character Name of the column with the names of the members of the collection of spectra.}
\end{itemize}

Methods (by class)

• \texttt{generic_spct}:
  • \texttt{generic_mspct}:
See Also

Other wavelength summaries: \texttt{wl_midpoint()}, \texttt{wl_min()}, \texttt{wl_stepsize()}

Examples

\begin{verbatim}
range(sun.spct)
wl_range(sun.spct)

range(sun.spct)
\end{verbatim}

\begin{tabular}{ll}
\texttt{wl_stepsize} & \textit{Stepsize} \\
\hline
\end{tabular}

Description

Function that returns the range of step sizes in an object. Range of differences between successive sorted values.

Usage

\begin{verbatim}
wl_stepsize(x, ...)
stepsize(x, ...)

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

### S3 method for class 'numeric'
stepsize(x, ...)

### S3 method for class 'generic_spct'
stepsize(x, ...)

### S3 method for class 'generic_mspct'
stepsize(x, ..., idx = "spct.idx")
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1cm} an R object
\item \texttt{...} \hspace{1cm} not used in current version
\item \texttt{idx} \hspace{1cm} character Name of the column with the names of the members of the collection of spectra.
\end{itemize}

Value

A numeric vector of length 2 with min and maximum stepsize values.
w_length2rgb

Methods (by class)
- default: Default function usable on numeric vectors.
- numeric: Method for numeric vectors.
- generic_spct: Method for "generic_spct" objects.
- generic_mspct: Method for "generic_mspct" objects.

See Also
Other wavelength summaries: \texttt{wl_midpoint()}, \texttt{wl_min()}, \texttt{wl_range()}

Examples
\begin{verbatim}
  steps size(sun.spct)
  wl_stepsize(sun.spct)
  steps size(sun.spct)
\end{verbatim}

\begin{itemize}
  \item \texttt{w_length2rgb(w.length, sens = photobiology::ciexyzCMF2.spct, color.name = NULL)}
\end{itemize}

\textbf{Description}
Calculates rgb values from spectra based on human color matching functions

\textbf{Usage}
\begin{verbatim}
  w_length2rgb(w.length, sens = photobiology::ciexyzCMF2.spct, color.name = NULL)
\end{verbatim}

\textbf{Arguments}
- \texttt{w.length} numeric Vector of wavelengths (nm)
- \texttt{sens} chroma_spct Used as chromaticity definition
- \texttt{color.name} character Used for naming the rgb color definition

\textbf{Value}
A vector of colors defined using \texttt{rgb()}. The numeric values of the RGB components can be obtained using function \texttt{col2rgb()}.

\textbf{See Also}
Other color functions: \texttt{rgb_spct()}, \texttt{w_length_range2rgb()}

w_length_range2rgb

Wavelength range to rgb color conversion

Description

Calculates rgb values from spectra based on human color matching functions

Usage

w_length_range2rgb(
  w.length,
  sens = photobiology::ciexyzCMF2.spct,
  color.name = NULL
)

Arguments

w.length numeric vector of wavelengths (nm) of length 2. If longer, its range is used.
sens chroma_spct Used as the chromaticity definition.
color.name character Used for naming the rgb color definition(s) returned.

Value

A vector of colors defined using rgb(). The numeric values of the RGB components can be obtained by calling function col2rgb.

See Also

Other color functions: rgb_spct(), w_length2rgb()

Examples

col2rgb(w_length_range2rgb(c(500,600)))
col2rgb(w_length_range2rgb(550))
col2rgb(w_length_range2rgb(500:600))
yellow_gel.spct

Transmittance spectrum of yellow theatrical gel.

Description
A dataset containing the wavelengths at a 1 nm interval and fractional total transmittance for polyester film.

Usage
yellow_gel.spct

Format
A filter_spct object with 611 rows and 2 variables

Details
- w.length (nm).
- Tfr (0..1)

See Also

Examples
yellow_gel.spct

^ generic_spct

Arithmetic Operators

Description
Power operator for spectra.

Usage
```r
## S3 method for class 'generic_spct'
e1 ^ e2
```
Arguments

- `e1`: an object of class "generic_spct"
- `e2`: a numeric vector, possibly of length one.

See Also

Other math operators and functions: `MathFun`, `convolve_each()`, `div-.generic_spct`, `log()`, `minus-.generic_spct`, `mod-.generic_spct`, `plus-.generic_spct`, `round()`, `sign()`, `slash-.generic_spct`, `times-.generic_spct`
## Index

* **BSWF attribute functions**
  - getBSWUFused, 153
  - setBSWUFused, 270

* **Coercion methods for collections of spectra**
  - as.calibration_mspct, 24
  - as.chroma_mspct, 26
  - as.cps_mspct, 28
  - as.filter_mspct, 30
  - as.generic_mspct, 33
  - as.object_mspct, 36
  - as.raw_mspct, 39
  - as.reflector_mspct, 41
  - as.response_mspct, 44
  - as.source_mspct, 47
  - split2mspct, 305
  - subset2mspct, 315

* **Rfr attribute functions**
  - getRfrType, 162
  - setRfrType, 280

* **Spectral data examples**
  - A.illuminant.spct, 10
  - black_body.spct, 54
  - ccd.spct, 58
  - clear.spct, 75
  - clear_body.spct, 76
  - D65.illuminant.spct, 91
  - green_leaf.spct, 172
  - Ler_leaf.spct, 200
  - Ler_leaf rflt.spct, 201
  - Ler_leaf trns.spct, 202
  - Ler_leaf trns_i.spct, 203
  - opaque.spct, 220
  - photodiode.spct, 228
  - polyester.spct, 234
  - sun.daily.data, 320
  - sun.daily.spct, 321
  - sun.data, 322
  - sun.spct, 323
  - white_body.spct, 387
  - whiteLed.cps_spct, 388
  - white_led.raw_spct, 388
  - white_led.source_spct, 389
  - yellow_gel.spct, 401

* **Tfr attribute functions**
  - getTfrType, 164
  - setTfrType, 282

* **Visual response data examples**
  - beesxyzCMF.spct, 54
  - ciev10.spct, 64
  - ciev2.spct, 65
  - ciexyzCC10.spct, 66
  - ciexyzCC2.spct, 67
  - ciexyzCMF10.spct, 68
  - ciexyzCMF2.spct, 69
  - cone_fundamentals10.spct, 83

* **astronomy related functions**
  - day_night, 92
  - format.solar_time, 141
  - format.tod_time, 142
  - is.solar_time, 189
  - print.solar_time, 235
  - print.tod_time, 237
  - solar_time, 292
  - sun_angles, 324

* **auxiliary functions**
  - normalize_range_arg, 219

* **collections of spectra classes family**
  - generic_mspct, 152

* **color functions**
  - rgb_spct, 265
  - w_length2rgb, 399
  - w_length_range2rgb, 400

* **constructors of spectral objects**
  - as.calibration_spct, 25
  - as.chroma_spct, 27
  - as.cps_spct, 29
  - as.filter_spct, 32
as.generic_spct, 35
as.object_spct, 38
as.raw_spct, 40
as.reflector_spct, 43
as.response_spct, 46
as.source_spct, 49
source_spct, 293

* conversion of collections of spectra
join_mspct, 198

* data validity check functions
check_spct, 59
check_spectrum, 63
check_w.length, 63
enable_check_spct, 110

* datasets
A.illuminant.spct, 10
beexyzCMF.spct, 54
black_body.spct, 54
ccd.spct, 58
ciev10.spct, 64
ciev2.spct, 65
ciexyzCC10.spct, 66
ciexyzCC2.spct, 67
ciexyzCMF10.spct, 68
ciexyzCMF2.spct, 69
clear.spct, 75
clear_body.spct, 76
cone_fundamentals10.spct, 83
D2.UV586, 89
D2.UV653, 90
D2.UV654, 90
D65.illuminant.spct, 91
FEL.BN.9101.165, 131
green_leaf.spct, 172
Ler_leaf.spct, 200
Ler_leaf.rflt.spct, 201
Ler_leaf.trns.spct, 202
Ler_leaf_trns_i.spct, 203
opaque.spct, 220
photodiode.spct, 228
polyester.spct, 234
r4p_pkg, 254
sun.daily.data, 320
sun.daily.spct, 321
sun.data, 322
sun.spt, 323
white_body.spct, 387
white_led.cps_spct, 388
white_led.raw_spct, 388
white_led.source_spct, 389
yellow_gel.spct, 401

* despike and valleys functions
despike, 96

* experimental utility functions
collect2mspct, 78
drop_user_cols, 106
thin_wl, 348
uncollect2spct, 364

* idfactor attribute functions
getIdFactor, 157
setIdFactor, 276

* internal.
v_insert_hinges, 376
v_replace_hinges, 377

* interpolate functions
interpolate_wl, 180

* irradiance functions
e_fluence, 119
e_irrad, 122
fluence, 138
irrad, 182
q_fluence, 243
q_irrad, 245

* low-level functions operating on numeric vectors.
as_energy, 50
as_quantum_mol, 52
calc_multipliers, 55
div_spectra, 105
ergy_irradiance, 112
ergy_ratio, 113
insert_hinges, 175
integrate_xy, 177
interpolate_spectrum, 179
irradiance, 185
oper_spectra, 220
photon_irradiance, 230
photon_ratio, 232
photons_energy_ratio, 229
prod_spectra, 238
s_e_irrad2rgb, 326
split_energy_irradiance, 308
split_photon_irradiance, 311
subt_spectra, 316
sum_spectra, 318
trim_tails, 359
v_insert_hinges, 376
v_replace_hinges, 377

* math operators and functions
  ^.generic_spct, 401
  convolve_each, 87
  div-.generic_spct, 104
  log, 204
  MathFun, 205
  minus-.generic_spct, 207
  mod-.generic_spct, 208
  plus-.generic_spct, 233
  round, 268
  sign, 288
  slash-.generic_spct, 289
  times-.generic_spct, 351

* measurement metadata functions
  add_attr2tb, 18
  get_attributes, 169
  getFilterProperties, 154
  getHowMeasured, 155
  getInstrDesc, 157
  getInstrSettings, 158
  getWhatMeasured, 165
  getWhenMeasured, 166
  getWhereMeasured, 168
  isValidInstrDesc, 191
  isValidInstrSettings, 192
  select_spct_attributes, 269
  setFilterProperties, 270
  setHowMeasured, 275
  setInstrDesc, 277
  setInstrSettings, 277
  setWhatMeasured, 284
  setWhenMeasured, 285
  setWhereMeasured, 286
  spct_attr2tb, 297
  spct_metadata, 298
  trimInstrDesc, 355
  trimInstrSettings, 356

* multiple.wl attribute functions
  getMultipleWl, 159
  setMultipleWl, 278

* peaks and valleys functions
  find_peaks, 133
  find_spikes, 134
  get_peaks, 171
  peaks, 222
  replace_bad_pixs, 260
  spikes, 300
  valleys, 370
  wls_at_target, 390

* photon and energy ratio functions
  e_ratio, 125
  eq_ratio, 114
  q_ratio, 248
  qe_ratio, 240

* quantity conversion functions
  A2T, 11
  Afr2T, 21
  any2T, 23
  as_quantum, 51
  e2q, 108
  e2qmol_multipliers, 109
  e2quantum_multipliers, 110
  q2e, 239
  T2A, 342
  T2Afr, 344

* query units functions
  is_absorbance_based, 192
  is_photon_based, 195

* rescaling functions
  fscale, 143
  fshift, 148
  getNormalized, 160
  getScaled, 162
  is_normalized, 195
  is_scaled, 196
  normalize, 213
  setNormalized, 279
  setScaled, 281

* response functions
  e_response, 128
  q_response, 251
  response, 262

* response type attribute functions
  setResponseType, 279

* response.type attribute functions
  getResponseType, 161

* set and unset ‘multi spectral’ class functions
  rmDerivedMspct, 266
  shared_member_class, 288

* set and unset spectral class functions
  rmDerivedSpct, 267
  setGenericSpct, 272

* split a spectrum into regions functions
split_irradiance, 310
* tagging and related functions
  is_tagged, 197
tag, 346
untag, 366
wb2rect_spct, 384
wb2spct, 385
wb2tagged_spct, 386
* time attribute functions
  checkTimeUnit, 59
convertTfrType, 84
convertThickness, 85
convertTimeUnit, 86
g getTimeUnit, 164
setTimeUnit, 283
* trim functions
  clip_wl, 76
trim_spct, 356
trim_waveband, 360
trim_wl, 362
* upgrade from earlier versions
  is_old_spct, 189
upgrade_spct, 367
upgrade_spectra, 367
* waveband attributes
  is_effective, 193
labels, 199
normalization, 212
* waveband constructors
  split_bands, 307
waveband, 381
* wavelength summaries
  wl_midpoint, 395
wl_min, 396
wl_range, 397
wl_stepsize, 398
 generic_spct(times-.generic_spct), 351
 generic_spct(plus-.generic_spct), 233
 generic_spct(minus-.generic_spct), 207
 generic_spct(slash-.generic_spct), 289
 generic_spct(plus-.generic_spct), 233
 generic_spct(minus-.generic_spct), 207
 generic_spct(slash-.generic_spct), 289
 [...object_spct(Extract), 116
 [...raw_spct(Extract), 116
 [...reflector_spct(Extract), 116
 [...response_spct(Extract), 116
 [...source_spct(Extract), 116
 [<-generic_mspct(Extract_mspct), 118
 [<-generic_spct(Extract), 116
 [[<-generic_mspct(Extract_mspct), 118
 $<-.generic_mspct(Extract_mspct), 118
 $<-.generic_spct(Extract), 116
 %%.generic_spct(div-.generic_spct), 104
 %%.generic_spct(mod-.generic_spct), 208
 afs.label, 109, 110
 any2A (any2T), 22
 any2Afr (any2T), 23
 any2T, 12, 22, 23, 51, 109, 110, 240, 343, 345
 A_as_default (energy_as_default), 111
 absorbs, 12, 218
 absorptance, 15
 acos.generic_spct (MathFun), 205
 afR function (any2T), 23
 afR2T (any2T), 18
 address2tb (add_attr2tb), 18
 afT2R (any2T), 23
 afT2R (any2T), 23
 any2T, 12, 22, 23, 51, 109, 110, 240, 343, 345
 any2T, 12, 22, 23, 51, 109, 110, 240, 343, 345
 as.calibration_mspct, 24, 27, 29, 32, 35,
 38, 40, 43, 45, 49, 307, 316
 as.calibration_spct, 25, 28, 30, 33, 35, 39,
 41, 44, 46, 50, 297
 as.chroma_mspct, 25, 26, 29, 32, 35, 38, 40,
 43, 45, 49, 307, 316
 as.chroma_spct, 26, 27, 30, 33, 35, 39, 41,
 44, 46, 50, 297
 as.cps_mspct, 25, 27, 28, 32, 35, 38, 40, 43,
 45, 49, 307, 316
 as.cps_spct, 26, 28, 29, 33, 35, 39, 41, 44,
 46, 50, 297
 add_attr2tb, 14, 17, 18, 115, 121, 124, 127,
 130, 140, 155, 156, 158, 166, 167,
 169, 170, 184, 192, 241, 244, 247,
 250, 253, 258, 263, 269, 272,
 276–278, 284, 286, 287, 298, 299,
 353, 355, 356
 A.illuminant_spct, 10, 55, 58, 75, 76, 92,
 173, 200–203, 220, 229, 234,
 320–323, 387–390, 401
 aT2R (any2T), 22
 A_as_default (energy_as_default), 111
 abs.generic_spct (MathFun), 205
 absorbance, 12, 218
 absorptance, 15
 acos.generic_spct (Trig), 354
as.filter_mspct, 25, 27, 29, 30, 35, 38, 40, 43, 45, 49, 307, 316
as.filter_spect, 26, 28, 30, 32, 35, 39, 41, 44, 46, 50, 297
as.generic_mspct, 25, 27, 29, 32, 33, 38, 40, 43, 45, 49, 307, 316
as.generic_spect, 26, 28, 30, 33, 35, 39, 41, 44, 46, 50, 297
as.matrix-mspct, 36
as.matrix.generic_mspct
(as.matrix-mspct), 36
as.object_mspct, 25, 27, 29, 32, 35, 36, 40, 43, 45, 49, 307, 316
as.object_spect, 26, 28, 30, 33, 35, 38, 41, 44, 46, 50, 297
as.raw_mspct, 25, 27, 29, 32, 35, 38, 43, 44, 46, 50, 297
as.raw_spect, 26, 28, 30, 33, 35, 39, 40, 44, 46, 50, 297
as.reflector_mspct, 25, 27, 29, 32, 35, 38, 40, 41, 44, 49, 307, 316
as.reflector_spect, 26, 28, 30, 33, 35, 39, 41, 44, 46, 49, 307, 316
as.response_mspct, 25, 27, 29, 32, 35, 38, 40, 43, 44, 49, 49, 307, 316
as.response_spect, 26, 28, 30, 33, 35, 39, 41, 44, 46, 50, 297
as.solar_date, 47
as.source_mspct, 25, 27, 29, 32, 35, 38, 40, 43, 45, 47, 307, 316
as.source_spect, 26, 28, 30, 33, 35, 39, 41, 44, 46, 50, 297
as_quantum, 12, 22, 23, 51, 109, 110, 240, 343, 345
as_tod, 52
asin.generic_spect(Trig), 354
atan.generic_spect(Trig), 354
average_spect, 53

beesxyzCMF.spect, 54, 64–69, 84
BSWF_used2tb(add_attr2tb), 18
c. 55
calc_filter_multipliers(defunct), 95
calc_source_output, 57
calibration_mspct(generic_mspct), 152
calibration_spect(source_spect), 293
ceiling.generic_spect(round), 268
check_spect, 59, 63, 64, 111
check_spectrum, 62, 63, 64, 111, 309, 311, 312
check_w.length, 62, 63, 63, 111
tcheckTimeUnit, 59, 84–86, 165, 283
chroma_mspct(generic_mspct), 152
chroma_spect(source_spect), 293
cieiv10.spect, 54, 64, 65–69, 84
cieiv2.spect, 54, 64, 65, 66–69, 84
ciexyzCC10.spect, 54, 64, 65, 66, 67–69, 84
ciexyzCC2.spect, 54, 64–66, 67, 68, 69, 84
ciexyzCMF10.spect, 54, 64–67, 68, 69, 84
ciexyzCMF2.spect, 54, 64–68, 69, 84
class_spect, 70
clean, 70
clip_wl, 76, 358, 361, 363
col2rgb, 326, 400
collect2mspect, 78, 108, 350, 365
color(color_of), 79
color_of, 79
colour_of(color_of), 79
comment2tb(add_attr2tb), 18
compare_spect, 81
cone_fundamentals10.mspct
(cone_fundamentals10.spect), 83
cone_fundamentals10.spct, 54, 64–69, 83
cvtColorType, 59, 84, 85, 86, 165, 283
cvtColorThickness, 59, 84, 85, 86, 165, 283
cvtColorTimeUnit, 59, 84, 85, 86, 165, 283
convolve_each, 87, 105, 204, 205, 207, 208, 233, 268, 288, 289, 351, 402
copy_attributes, 87
cor, 342
cos_generic_spct(Trig), 354
cps2irrad, 88
cps2Rfr(cps2irrad), 88
cps2Tfr(cps2irrad), 88
cps_mspct(generic_mspct), 152
cps_spct(source_spct), 293
D2.UV586, 89
D2.UV653, 90
D2.UV654, 90
D2_spectrum, 91
day_length(day_night), 92
day_night, 92, 141, 142, 190, 236, 237, 292, 325
day_night_fast(day_night), 92
defunct, 95
despike, 96
diffracting_double_slit
  (diffracting_single_slit), 103
diffracting_single_slit, 103
dim.generic_mspct, 104
dim<-generic_mspct
  (dim.generic_mspct), 104
disable_check_spct(enable_check_spct), 110
div<-generic_spct, 104
drop_user_cols, 79, 106, 350, 365
e2q, 12, 22, 23, 51, 108, 109, 110, 240, 343, 345
e2qmol_multipliers, 12, 22, 23, 51, 109, 109, 110, 240, 343, 345
e2quantum_multipliers, 12, 22, 23, 51, 109, 110, 240, 343, 345
INDEX

format.solar_time, 95, 141, 142, 190, 236, 237, 292, 325
format.tod_time, 95, 141, 142, 190, 236, 237, 292, 325
formatted_range, 142
fscale, 143, 152, 160, 163, 195, 197, 217, 279, 282
fshift, 147, 148, 160, 163, 195, 197, 217, 279, 282
generic_mspct, 152
generic_spct (source_spct), 293
geocode2tb (add_attr2tb), 18
get_attributes, 20, 155, 156, 158, 166, 167, 169, 169, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
get_peaks, 133, 135, 171, 228, 261, 304, 376, 393
get_valleys (get_peaks), 171
getAfrType (defunct), 95
getBSWFUsed, 153, 270
getFilterProperties, 20, 154, 156, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
getHowMeasured, 20, 155, 155, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
getIdFactor, 157, 276
getInstrDesc, 20, 155, 156, 157, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
getInstrSettings, 20, 155, 156, 158, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
getMspctVersion, 159
getMultipleWl, 159, 279
getNormalised (getNormalized), 160
getNormalized, 147, 152, 160, 163, 195, 197, 217, 279, 282
getResponseType, 161
getRfrType, 162, 281
getScaled, 147, 152, 160, 162, 195, 197, 217, 279, 282
getSpectVersion, 163
getTfrType, 164, 282
g getTimeUnit, 59, 84–86, 164, 283

getWhenMeasured, 20, 155, 156, 158, 166, 166, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
getWhereMeasured, 20, 155, 156, 158, 166, 167, 168, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356

head, 174
head_tail, 173
how_measured (getHowMeasured), 155
how_measured2tb (add_attr2tb), 18
how_measured= (setHowMeasured), 275

insert_spct_hinges, 176
instr_desc2tb (add_attr2tb), 18
instr_settings2tb (add_attr2tb), 18
integrate_spct, 176
interpolate_mspct (interpolate_spct), 178
interpolate_spct, 178
interpolate_wl, 180
irrad, 122, 125, 141, 182, 245, 248
is.any_mspct (is.generic_mspct), 186
is.any_spct (is.generic_spct), 188
is.any_summary_spct
  (is.summary_generic_spct), 190
is.calibration_mspct
  (is.generic_mspct), 186
is.calibration_spct (is.generic_spct), 188
is.chroma_mspct (is.generic_mspct), 186
is.chroma_spct (is.generic_spct), 188
is.cps_mspct (is.generic_mspct), 186
is.cps_spct (is.generic_spct), 188
is.filter_mspct (is.generic_mspct), 186
is.filter_spct (is.generic_spct), 188
is.generic_mspct, 186
is.generic_spct, 188
is.object_mspct (is.generic_mspct), 186
is.object_spct (is.generic_spct), 188
is.old_spct, 189, 367, 368
is.raw_mspct (is.generic_mspct), 186
is.raw_spct (is.generic_spct), 188
is.reflector_mspct (is.generic_mspct), 186
is.reflector_spct (is.generic_spct), 188
is.response_mspct (is.generic_spct), 186
is.response_spct (is.generic_spct), 188
is.solar_date (is.solar_time), 189
is.solar_time, 95, 141, 142, 189, 236, 237, 292, 325
is.source_mspct (is.generic_mspct), 186
is.source_spct (is.generic_spct), 188
is.summary_chroma_spct
  (is.summary_generic_spct), 190
is.summary_cps_spct
  (is.summary_generic_spct), 190
is.summary_filter_spct
  (is.summary_generic_spct), 190
is.summary_generic_spct, 190
is.summary_object_spct
  (is.summary_generic_spct), 190
is.summary_raw_spct
  (is.summary_generic_spct), 190
is.summary_reflector_spct
  (is.summary_generic_spct), 190
is.summary_response_spct
  (is.summary_generic_spct), 190
is.summary_source_spct
  (is.summary_generic_spct), 190
is.waveband, 191
is.absorbance_based, 192, 196
is.absorptance_based
  (is.absorbance_based), 192
is.effective, 193, 199, 213
is.energy_based (is.photon_based), 195
is.normalised (is.normalised), 195
is.normalised, 147, 152, 160, 163, 195, 197, 217, 279, 282
is.photon_based, 193, 195
is.scaled, 147, 152, 160, 163, 195, 196, 217, 279, 282
is.tagged, 197, 347, 366, 385, 386
is_transmittance_based
  (is.absorbance_based), 192
is_valid_geocode (validate_geocode), 369
isValidInstrDesc, 20, 155, 156, 158, 166, 167, 169, 170, 191, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
isValidInstrSettings, 20, 155, 156, 158, 166, 167, 169, 170, 192, 191, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
join, 206
join_mspct, 198
labels, 194, 199, 213
lat2tb (add_attr2tb), 18
length_geocode (validate_geocode), 369
Ler_leaf_spct. 11, 55, 58, 75, 76, 92, 173, 200, 201–203, 220, 229, 234, 320–323, 387–390, 401
log, 87, 105, 204, 205, 207, 208, 233, 268, 288, 289, 351, 402
log10.generic_spct (log), 204
log2.generic_spct (log), 204
lon2tb (add_attr2tb), 18
lonlat2tb (add_attr2tb), 18
mat2mspct (as.generic_mspct), 33
MathFun, 87, 105, 204, 205, 207, 208, 233, 268, 288, 289, 331, 402
max, 313, 395
max (wl_max), 394
mean, 329, 330
median, 332
merge2object_spct, 205
merge_attributes, 206
midpoint (wl_midpoint), 395
min, 313, 395
min (wl_min), 396
minus-.generic_spct, 207
mod-.generic_spct, 208
msaply (msmsaply), 208
msdply (msmsaply), 208
mslply (msmsaply), 208
msmsaply, 208
mspt2mat (as.matrix-mspct), 36
mspct_classes, 209
mutate_mspct (defunct), 95

NA, 142
na.action, 212
na.exclude.chroma_spct (na.omit), 210
na.exclude cps_spct (na.omit), 210
na.exclude.filter_spct (na.omit), 210
na.exclude.generic_mspct (na.omit), 210
na.exclude.generic_spct (na.omit), 210
na.exclude.object_spct (na.omit), 210
na.exclude.raw_spct (na.omit), 210
na.exclude.reflecter_spct (na.omit), 210
na.exclude.response_spct (na.omit), 210
na.exclude.source_spct (na.omit), 210
na.fail, 212
na.omit, 210
na_geocode (validate_geocode), 369
NDvi (normalized_diff_ind), 217
new_waveband (waveband), 381
night_length (day_night), 92
noon_time (day_night), 92
normalise (normalize), 213
normalised_diff_ind
(normalized_diff_ind), 217
normalization, 194, 199, 212
normalize, 147, 152, 160, 163, 195, 197, 213, 279, 282
normalize_range_arg, 219
normalized2tb (add_attr2tb), 18
normalized_diff_ind, 217
object_mspct (generic_mspct), 152
object_spct (source_spct), 293
paste, 142
peaks, 133, 135, 172, 222, 261, 304, 376, 393
photobiology (photobiology-package), 8
photobiology-package, 8
photon_as_default (energy_as_default), 111
plus-.generic_spct, 233
print, 234
print.solar_date (print.solar_time), 235
print.solar_time, 95, 141, 142, 190, 235, 237, 292, 325
print.summary_generic_spct, 236
print.tod_time, 95, 141, 142, 190, 236, 237, 292, 325
print.waveband, 237
prod, 334
q2e, 12, 22, 23, 51, 109, 110, 239, 343, 345
q_fluence, 122, 125, 141, 185, 243, 248
q_irrad, 122, 125, 141, 185, 245, 245
q_ratio, 116, 128, 242, 248
q_response, 131, 251, 264
qe_ratio, 116, 128, 240, 251
quantum_as_default (energy_as_default), 111
r4p_pkgs, 254
range, 142
range (wl_range), 397
raw_mspct (generic_mspct), 152
raw_spct (source_spct), 293
rbindspect, 255
reflectance, 218, 256
reflector_mspct (generic_mspct), 152
reflector_spct (source_spct), 293
relative_AM, 259
replace_bad_pixs, 102, 133, 135, 172, 228, 260, 304, 376, 393
response, 131, 218, 254, 262
response_mspct (generic_mspct), 152
response_spct (source_spct), 293
Rfr_from_n, 264
Rfr_p_from_n (Rfr_from_n), 264
Rfr_s_from_n (Rfr_from_n), 264
Rfr_type2tb (add_attr2tb), 18
rgb, 326
rgb_spct, 265, 399, 400
rmDerivedMspct, 266, 288
rmDerivedSpct, 267, 275
round, 87, 105, 204, 205, 207, 208, 233, 268, 288, 289, 351, 402
s_mean, 327
s_mean_se, 329
s_median, 331
s_prod, 332
s_range, 334
s_sd, 335
s_se, 337
s_sum, 339
s_var, 340
scaled2tb (add_attr2tb), 18
sd, 337
select_spct_attributes, 20, 155, 156, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
set_check_spct (enable_check_spct), 110
setAfrType (defunct), 95
setBSWFUsed, 154, 270
setCalibrationSpct (setGenericSpct), 272
setChromaSpct (setGenericSpct), 272
setCpsSpct (setGenericSpct), 272
setFilterSpct (setGenericSpct), 272
setGenericSpct, 26, 28, 30, 33, 35, 39, 41, 44, 46, 50, 267, 272
setHowMeasured, 20, 155, 156, 158, 166, 167, 169, 170, 192, 269, 272, 275, 277, 278, 284, 286, 287, 298, 299, 355, 356
setIdFactor, 157, 276
setInstrDesc, 20, 155, 156, 158, 166, 167, 169, 170, 192, 269, 272, 276, 277, 278, 284, 286, 287, 298, 299, 355, 356
setInstrSettings, 20, 155, 156, 158, 166, 167, 169, 170, 192, 269, 272, 276, 277, 278, 284, 286, 287, 298, 299, 355, 356
setMultipleWl, 160, 278
setNormalised (setNormalized), 279
setNormalized, 147, 152, 160, 163, 195, 197, 217, 279, 282
setObjectSpct (setGenericSpct), 272
setRawSpct (setGenericSpct), 272
setReflectorSpct (setGenericSpct), 272
setResponseSpct (setGenericSpct), 272
setResponseType, 279
setRfrType, 162, 280
setScaled, 147, 152, 160, 163, 195, 197, 217,
INDEX

279, 281
setSourceSpct (setGenericSpct), 272
setTrType, 164, 282
setTimeZone, 59, 84–86, 165, 283
setWhatMeasured, 20, 155, 156, 158, 166,
167, 169, 170, 192, 269, 272,
276–278, 284, 286, 287, 298, 299,
355, 356
setWhenMeasured, 20, 155, 156, 158, 166,
167, 169, 170, 192, 269, 272,
276–278, 284, 285, 287, 298, 299,
355, 356
shared_member_class, 267, 288
sign, 87, 105, 204, 205, 207, 208, 233, 268,
288, 289, 351, 402
signif.generic_spct (round), 268
sin.generic_spct (Trig), 354
slashgeneric_spct, 289
smooth_spct, 289
solar_time, 95, 141, 142, 190, 236, 237, 292,
325
source_mspct (generic_mspct), 152
source_spct, 26, 28, 30, 33, 35, 39, 41, 44,
46, 50, 293
spct_attr2tb, 20, 155, 156, 158, 166, 167,
169, 170, 192, 269, 272, 276–278,
284, 286, 287, 297, 299, 355, 356
spct_attributes
(select_spct_attributes), 269
spct_classes, 298
spct_metadata, 20, 155, 156, 158, 166, 167,
169, 170, 192, 269, 272, 276–278,
284, 286, 287, 298, 299, 355, 356
spikes, 133, 135, 172, 228, 261, 300, 376, 393
splinefun, 180
split2calibration_mspct (split2mspct), 305
split2cps_mspct (split2mspct), 305
split2filter_mspct (split2mspct), 305
split2mspct, 25, 27, 29, 32, 35, 38, 40, 43,
45, 49, 305, 316
split2raw_mspct (split2mspct), 305
split2reflector_mspct (split2mspct), 305
split2response_mspct (split2mspct), 305
split2source_mspct (split2mspct), 305
split_bands, 307, 382
split_energy_irradiance, 50, 52, 56, 106,
112, 114, 175, 177, 180, 186, 222,
230, 231, 233, 239, 308, 312, 317,
319, 327, 360, 377, 378
split_irradiance, 310
split_photon_irradiance, 50, 52, 56, 106,
112, 114, 175, 177, 180, 186, 222,
230, 231, 233, 239, 309, 311, 317,
319, 327, 360, 377, 378
spread, 312
sqrt.generic_spct (MathFun), 205
steps (wl_steps), 398
strict_range_as_default
(voice_as_default), 376
Subset, 314
subset, 118
subset.generic_spct (Subset), 314
subset2mspct, 25, 27, 29, 32, 35, 38, 40, 43,
45, 49, 307, 315
subt_spectra, 50, 52, 56, 106, 112, 114, 175,
177, 180, 186, 222, 230, 231, 233,
239, 309, 312, 316, 319, 327, 360,
377, 378
sum, 340
sum_spectra, 50, 52, 56, 106, 112, 114, 175,
177, 180, 186, 222, 230, 231, 233,
239, 309, 312, 316, 319, 327, 360,
377, 378
summary, 317
summary_spct_classes, 318
sun.daily.data, 11, 55, 58, 75, 76, 92, 173,
200–203, 220, 229, 234, 320,
321–323, 387–390, 401
sun.daily.spct, 11, 55, 58, 75, 76, 92, 173,
200–203, 220, 229, 234, 320, 321,
322, 323, 387–390, 401
sun.data, 11, 55, 58, 75, 76, 92, 173,
200–203, 220, 229, 234, 320, 321,
322, 323, 387–390, 401
sun_spct, 11, 55, 58, 75, 76, 92, 173,
200–203, 220, 229, 234, 320–322,
323, 387–390, 401
sun_angles, 95, 141, 142, 190, 236, 237, 292,
324
sun_angles_fast (sun_angles), 324
sun_azimuth (sun_angles), 324
sun_elevation (sun_angles), 324
sunZenith_angle (sun_angles), 324
sunrise_time (day_night), 92
sunset_time (day_night), 92
T2A, 12, 22, 23, 51, 109, 110, 240, 342, 345
T2Afr, 12, 22, 23, 51, 109, 110, 240, 343, 344
T2T (defunct), 95
tag, 82, 197, 346, 366, 385, 386
tan.generic_spect (T2T)
sun_zenith_angle (T2Afr), 82, 197, 346, 366, 385, 386
Tfr_as_default (energy_as_default), 111
Tfr_type2tb (add_attr2tb), 18
thin_wl, 79, 108, 348, 365
time_unit2tb (add_attr2tb), 18
times_.generic_spect, 351
transmittance, 218, 351
Trig, 354
trim2overlap (trim_spct), 356
trim_mspct (trim_spct), 356
trim_spct, 77, 118, 356, 361, 363
trim_waveband, 77, 358, 360, 363
trim_wl, 77, 358, 361, 362
trimInstrDesc, 20, 155, 156, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
trimInstrSettings, 20, 155, 156, 158, 166, 167, 169, 170, 192, 269, 272, 276–278, 284, 286, 287, 298, 299, 355, 356
trunc.generic_spect (round), 268
tz_time_diff, 364
unset_filter_qty_default (energy_as_default), 111
unset_radiation_unit_default (energy_as_default), 111
unset_user_defaults (energy_as_default), 111
untag, 197, 347, 366, 385, 386
upgrade_spct, 189, 367, 368
upgrade_spectra, 189, 367, 367
use_cached_mult_as_default (wb.trim_as_default), 387
using_A (using_Tfr), 368
using_Afr (using_Tfr), 368
using_energy (using_Tfr), 368
using_photon (using_Tfr), 368
using_quantum (using_Tfr), 368
using_Tfr, 368
validate_geocode, 369
valleys, 133, 135, 172, 228, 261, 304, 370, 393
verbose_as_default, 376
w_length2rgb, 266, 399, 400
w_length_range2rgb, 266, 399, 400
water_dp (water_vp_sat), 378
water_fp (water_vp_sat), 378
water_mvc2vp (water_vp_sat), 378
water_RH2vp (water_vp_sat), 378
water_vp2mvc (water_vp_sat), 378
water_vp2RH (water_vp_sat), 378
water_vp_sat, 378
waveband, 80, 213, 308, 381
waveband_ratio, 383
wb2rect_spect, 197, 347, 366, 384, 386
wb2spct, 197, 347, 366, 385, 386
wb2tagged_spect, 197, 347, 366, 385, 386
wb.trim_as_default, 387
what_measured (getWhatMeasured), 165
what_measured2tb (add_attr2tb), 18
what_measured<= (setWhatMeasured), 284
when_measured (getWhenMeasured), 166
when_measured2tb (add_attr2tb), 18
when_measured<= (setWhenMeasured), 285
where_measured (getWhereMeasured), 168
where_measured<= (setWhereMeasured), 286
INDEX

wl_expanse (spread), 312
wl_max, 394
wl_midpoint, 395, 397–399
wl_min, 396, 396, 398, 399
wl_range, 396, 397, 397, 399
wl_stepsize, 396–398, 398
wls_at_target, 133, 135, 172, 228, 261, 304, 376, 390