# photobiology Version 0.8.4
# User Guide

Pedro J. Aphalo

August 23, 2015

## Contents

# 1 Introduction

We have developed a set of packages to facilitate the calculation of many different quantities that can be derived from spectral irradiance data. The base package in this suite is called photobiology, and is the package described here. There other specialized packages for quantification of ultraviolet radiation and visible radiation (photobiologyWavebands), or plant photoreceptors (photobiologyPlants). Other packages in the suite provide example spectral data for filters (photobiologyFilters), lamps (photobiologyLamps), LEDs (photobiologyLEDs), sunlight (photobiologySun) and broadband sensors (photobiologySensors). In the future it will be submitted to CRAN (Comprehensive R archive network), it is meanwhile available from `https://www.r4photobiology.info/`. There is also a public Git repository at `https://bitbucket.org/aphalo/` from where the source code of the current an earlier versions can be cloned.

Package photobiology provides two sets of functions for many operations: functions programmed following a functional paradigm, and functions using an object-oriented paradigm. The former functions take as arguments numeric vectors and are probably faster. The later ones take 'spectra' objects as arguments, are easier to use, and at least at the moment, to some extent slower. For everyday use 'spectra' objects are recommended, but when maximum performance or flexibility in scripts is desired, the use of the functions taking numeric vectors as arguments may allow optimizations that are not possible with the object-oriented higher level functions.

# 2 Installation and use

The functions in the package photobiology are made available by installing the packages `photobiology` (once) and loading it from the library when needed.

To load the package into the workspace we use `library(photobiology)`.

```
library(photobiology)
library(photobiologyWavebands)
library(photobiologySun)
library(photobiologyFilters)
library(photobiologyReflectors)
```

```
library(photobiologySensors)
library(lubridate)
```

# 3 Spectra

This package defines a family of classes based on data frames which impose some restrictions on the naming of the vectors, something that allows methods and some functions to 'find' the data when passed one of these objects as argument. In addition, as the data is checked when the object is built, there is no need to test for the validity of the data each time a calculation is carried out. The other advantage of using `spct` objects, is that specialized versions of generic functions like `print` and operators like `+` can be defined for spectra. `___spct` objects are `data.frame` objects, as a result of how classes have been derived. In this package we define a *generic* or *base* spectrum class, derived from `data.frame`, from which specialized types of spectra are in turn derived. This 'parenthood' hierarchy means that spectra objects can be used almost anywhere where a `data.frame` is expected.

## 3.1 Printing

Spectral objects are printed in the current version of the package by the function defined in package `dplyr`, consequently, it is possible to the options from this package to control printing. `dplyr.print_max`, the number of rows in the spectral object above which only `dplyr.print_min` rows are printed, are both set to 5, instead of the default 20 and 10.

```
options(dplyr.print_max = 5)
options(dplyr.print_min = 5)
```

For explicit calls to `print()` its argument `n` can be used to control the number of lines printed. If `n` is set to `Inf` the whole spectrum is always printed.

```
print(sun.spct, n = 3)

## Object: source_spct [522 x 3]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## ..      ...       ...       ...
```

## 3.2 Classes

The package defines several classes intended to be used to store different types of spectral data. They are all derived from `generic_spct`, which in turn is derived

Table 1: Classes for spectral data. In addition to the attributes listed, all spectral objects have attributes `normalized` and `scaled`, plus the normal attributes of `data.frame` objects.

| Name | Variables | Attributes |
|---|---|---|
| generic_spct | w.length | |
| cps_spct | w.length, cps | |
| source_spct | w.length, s.e.irrad, s.q.irrad | time.unit, bswf |
| filter_spct | w.length, Tfr, A | Tfr.type |
| reflector_spct | w.length, Rfr | Rfr.type |
| object_spct | w.length, Tfr, Rfr | Tfr.type, Rfr.type |
| response_spct | w.length, s.e.response, s.q.response | time.unit |
| chroma_spct | w.length, x, y, z | |

from `data.frame` and internally created using `dplyr::data_frame`. Table 3 lists them. Attributes are used in objects of these classes to keep *metadata* such as information about units of expression.

The *design* imposes that data from different observations are never present as different *data columns*, if present, additional data columns represent different properties from the same observation event. In most cases, one spectral object corresponds to one spectral observation, but some functions are compatible or can be used to create spectral objects where the spectral data from different observations are stored "longitudinally" and "tagged" with a factor with a level for each observation event. These observations must use consistent units of expression and attribute values.

### 3.3 Data assumptions

An assumption of the package is that wavelengths are always expressed in nanometres ($1\ \text{nm} = 1 \cdot 10^{-9}\,\text{m}$). If the data to be analysed uses different units for wavelengths, e.g. Ångstrom ($1\ \text{Å} = 1 \cdot 10^{-10}\,\text{m}$), the values need to be re-scaled before any calculations. Table 2 lists the units of expression for the different variables listed in Table 3.

Energy irradiances are assumed to be expressed in $\text{W}\,\text{m}^{-2}$ and photon irradiances in $\text{mol}\,\text{m}^{-2}\,\text{s}^{-1}$, that is to say using second as unit for time. This is the default, but it is possible to set the unit for time to day in the case of `source_spct` objects.

The default time unit used is *second*, but *day* and *exposure* can be used by supplying the arguments `"day"` or `"exposure"`[1] to a parameter of the constructor of `source_spct` objects.

The attributes are normally set when an object spectral object is created, either using default values of values supplied as arguments to the constructor.

[1]The meaning of `"exposure"` is the total exposure time, in other words, fluence instead of irradiance.

Table 2: Variables used for spectral data and their units of expression: A: as stored in objects of the spectral classes, B: also recognized by the `set` family of functions for spectra and automatically converted. `time.unit` accepts in addition to the character strings listed in the table, objects of classes `lubridate::duration` and `period`, in addition `numeric` values are interpreted as seconds. `exposure.time` accepts these same values, but not the character strings.

| Variables | Unit of expression | Attribute value |
|---|---|---|
| A: stored | | |
| w.length | nm | |
| cps | $n\,s^{-1}$ | |
| s.e.irrad | $W\,m^{-2}\,nm^{-1}$ | time.unit = "second" |
| s.e.irrad | $J\,m^{-2}\,d^{-1}\,nm^{-1}$ | time.unit = "day" |
| s.e.irrad | varies | time.unit = *duration* |
| s.q.irrad | $mol\,m^{-2}\,s^{-1}\,nm^{-1}$ | time.unit = "second" |
| s.q.irrad | $mol\,m^{-2}\,d^{-1}\,nm^{-1}$ | time.unit = "day" |
| s.q.irrad | $mol\,m^{-2}\,nm^{-1}$ | time.unit = "exposure" |
| s.q.irrad | varies | time.unit = *duration* |
| Tfr | [0,1] | Tfr.type = "total" |
| Tfr | [0,1] | Tfr.type = "internal" |
| A | a.u. | Tfr.type = "internal" |
| Rfr | [0,1] | Rfr.type = "total" |
| Rfr | [0,1] | Rfr.type = "specular" |
| s.e.response | $x\,J^{-1}\,s^{-1}\,nm^{-1}$ | time.unit = "second" |
| s.e.response | $x\,J^{-1}\,d^{-1}\,nm^{-1}$ | time.unit = "day" |
| s.e.response | $x\,J^{-1}\,nm^{-1}$ | time.unit = "exposure" |
| s.e.response | varies | time.unit = *duration* |
| s.q.response | $x\,mol^{-1}\,s^{-1}\,nm^{-1}$ | time.unit = "second" |
| s.q.response | $x\,mol^{-1}\,d^{-1}\,nm^{-1}$ | time.unit = "day" |
| s.q.response | $x\,mol^{-1}\,nm^{-1}$ | time.unit = "exposure" |
| s.q.response | varies | time.unit = *duration* |
| x, y, z | [0,1] | |
| B: converted | | |
| wl → w.length | nm | |
| wavelength → w.length | nm | |
| Tpc → Tfr | [0,100] | Tfr.type = "total" |
| Tpc → Tfr | [0,100] | Tfr.type = "internal" |
| Rpc → Rfr | [0,100] | Rfr.type = "total" |
| Rpc → Rfr | [0,100] | Rfr.type = "specular" |
| counts.per.second → cps | $n\,s^{-1}$ | |

Not respecting these assumptions will yield completely wrong results! It is extremely important to make sure that the wavelengths are in nanometres as this is what all functions expect. If wavelength values are in the wrong units, the action-spectra weights and quantum conversions will be wrongly calculated, and the values returned by most functions completely wrong, without warning.

If spectral irradiance data is in $\mathrm{W\,m^{-2}\,nm^{-1}}$, and the wavelength in nm, as is the case for many Macam spectroradiometers, the data can be used directly and functions in the package will return irradiances in $\mathrm{W\,m^{-2}}$.

If, for example, the spectral irradiance data output by a spectroradiometer is expressed in $\mathrm{mW\,m^{-2}\,nm^{-1}}$, and the wavelengths are in Ångstrom then to obtain correct results when using any of the packages in the suite, we need to rescale the data when creating a new object.

```
# not run
my.spct <- source_spct(w.length = wavelength/10, s.e.irrad = irrad/1000)
```

In the example above, we take advantage of the behaviour of the S language: an operation between a scalar and vector, is equivalent to applying this operation to each member of the vector. Consequently, in the code above, each value from the vector of wavelengths is divided by 10, and each value in the vector of spectral irradiances is divided by 1000.

## 3.4 Querying the class

Before giving examples of how to construct objects to store spectral data we show how to query the class of an object, and how to query the class of a spectrum. Consistently with R design, the package provides 'is' functions for querying the type of spectra objects.

```
is.any_spct(sun.spct)

## [1] TRUE

is.source_spct(sun.spct)

## [1] TRUE
```

In addition function `class.spc` returns directly the spectrum-related class attributes.

```
class_spct(sun.spct)

## [1] "source_spct"  "generic_spct"

class(sun.spct)

## [1] "source_spct"  "generic_spct" "tbl_df"       "tbl"
## [5] "data.frame"
```

## 3.5 Construction

There are basically two different approaches to the creation of spectra by users, a constructor similar to `data.frame` constructor that takes vectors as arguments, and a constructor that converts `list` objects into spectral objects, which works similarly to `as.data.frame` from base R. In contrast to the data frame constructors spectral constructor require the variables or the vector arguments should be suitably named so that they can be recognized.

Here we briefly describe the 'as' constructor functions for spectra. In the first example we create an object to store spectral irradiance data for 'light source', by first creting a data frame, and creating the spectral object as a copy of it. In the example below we supply a single value, 1, for the spectral irradiance. This value gets recycled as is normal in R, but of course in real use it is more usual to supply a vector of the same length as the `w.length` vector.

```
my.df <- data.frame(w.length = 400:410, s.e.irrad = 1)
my.spct <- as.source_spct(my.df)
class(my.spct)

## [1] "source_spct"  "generic_spct" "tbl_df"       "tbl"
## [5] "data.frame"

class(my.df)

## [1] "data.frame"

my.spct

## Object: source_spct [11 x 2]
## Wavelength (nm): range 400 to 410, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       400         1
## 2       401         1
## 3       402         1
## 4       403         1
## 5       404         1
## ..      ...       ...
```

We can make a 'generic_spct' copy of any spectrum object.

```
my.g.spct <- as.generic_spct(my.spct)
class(my.g.spct)

## [1] "generic_spct" "tbl_df"       "tbl"          "data.frame"
```

When cronstructing spectral objetcs from numeric vectors the names of the arguments are meaningful and convey information on the nature of the spectral data and basis of expression.

```
source_spct(w.length = 300:305, s.e.irrad = 1)

## Object: source_spct [6 x 2]
## Wavelength (nm): range 300 to 305, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       300         1
## 2       301         1
## 3       302         1
## 4       303         1
## 5       304         1
## ..      ...       ...
```

```
z <- 300:305
y <- 2
source_spct(w.length = z, s.e.irrad = y)

## Object: source_spct [6 x 2]
## Wavelength (nm): range 300 to 305, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       300         2
## 2       301         2
## 3       302         2
## 4       303         2
## 5       304         2
## ..      ...       ...
```

```
w.length <- 300:305
s.e.irrad <- 1
source_spct(w.length, s.e.irrad)

## Object: source_spct [6 x 2]
## Wavelength (nm): range 300 to 305, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       300         1
## 2       301         1
## 3       302         1
## 4       303         1
## 5       304         1
## ..      ...       ...
```

The different constructors have additional arguments to be used in setting non-default values for the attributes. These arguments have the same name as the attributes. Here we used the data frame created in the first chunk of the section.

```
my.d.spct <- as.source_spct(my.df, time.unit = "day")
```

Argument `strict.range` can be used to override or make more strict the validation of the data values.

```
source_spct(w.length = 300:305, s.e.irrad = -1)

## Warning in range_check(x, strict.range = strict.range):  Negative spectral
energy irradiance values; minimun s.e.irrad = -1

## Object: source_spct [6 x 2]
## Wavelength (nm): range 300 to 305, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       300        -1
## 2       301        -1
## 3       302        -1
## 4       303        -1
## 5       304        -1
## ..      ...       ...

source_spct(w.length = 300:305, s.e.irrad = -1, strict.range = NULL)

## Object: source_spct [6 x 2]
## Wavelength (nm): range 300 to 305, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       300        -1
## 2       301        -1
## 3       302        -1
## 4       303        -1
## 5       304        -1
## ..      ...       ...
```

Finally argument `comment` can be used to add a comment to the data at the time of construction.

```
my.cm.spct <- source_spct(w.length = 300:305, s.e.irrad = 1,
                          comment = "This is a comment")
comment(my.cm.spct)

## [1] "This is a comment"
```

## 3.6 Special attributes

Spectral objects have several attributes used to store metadata, such as the time unit used. There are different functions available. One group of functions return a logical value, and another group returns the values of the attributes themselves. A third group of functions can be used to set the attributes, but these are very rarely needed in user code.

For example function `is_effective()` returns `TRUE` if the spectral data has been weighted with a BSWF. The corresponding `getBSWFUsed()` function can be used, in this case to retrieve the name of the BSWF that was used for generating the data. Here we demonstrate with one example, where we use two different `waveband` objects—constructed on-the-fly—, one defining a range

of wavelengths, and another one defining the spectral weighting function for human erythema.

```
is_effective(sun.spct)

## [1] FALSE

is_effective(sun.spct * VIS())

## [1] FALSE

getBSWFUsed(sun.spct * VIS())

## [1] "none"

is_effective(sun.spct * CIE())

## [1] TRUE

getBSWFUsed(sun.spct * CIE())

## [1] "CIE98.298"
```

Similar functions exist for other attributes and spectral clasees.

As seen above, the attributes are set automatically, and consequently function `setBSWFUsed()` and other *set* functions are only of use to programmers extending the package. One exception is when a wrong value has been assigned by mistake.

Sometimes it may be desired to change the time unit used for expresing spectral irradiance or spectral response, and this can be achieved with *conversion* function `convertTimeUnit`. This function both converts spectral data to the new unit of expression and sets the `time.unit` attribute, preserving the validity of the data object.

```
ten.minutes.spct <-
  convertTimeUnit(sun.spct, time.unit = duration(10, "minutes"))
ten.minutes.spct

## Object: source_spct [522 x 3]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 600s (~10 minutes)
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...

getTimeUnit(ten.minutes.spct)

## [1] "600s (~10 minutes)"
```

Table 3: Classes for collections of spectral objects

| Name | Member objects | Attributes |
|---|---|---|
| generic_mspct | generic_spct | names, dim |
| cps_mspct | cps_spct | names, dim |
| source_mspct | source_spct | names, dim |
| filter_mspct | filter_spct | names, dim |
| reflector_mspct | reflector_spct | names, dim |
| object_mspct | object_spct | names, dim |
| response_mspct | response_spct | names, dim |
| chroma_mspct | chroma_spct | names, dim |

Spectral objects created with earlier (pre-release) versions of this package are missing some attributes. For this reason 'summary' and 'plot' functions may not work as expected with them. These *old* objects can be updated by adding the missing attribute using functions `setTimeUnit`, `setBSWFUsed`, `setTfrType` and `setRfrType`. However, in many cases function `update_spct` can be used to set the missing attributes to default values, or the scripts re-run to rebuild the data objects from raw data.

# 4 Collections of spectra

## 4.1 Classes

The package defines several classes intended to be used to store *collections* of different types of spectral data. They are all derived from `generic_mspct`, which in turn is derived from `list`. Table **??** lists them.

Objects of these classes, except for those of class `generic_mspct`, can contain members belonging to one of the classes. Being all other spectral object classes derived from `generic_spct`, `generic_mspct` objects can contain heterogeneous collections of spectra. In all cases, there are no restrictions on the lengths, wavelength range and/or wavelength step size, or attributes other than `class` of the contained spectra. Mimicking R's arrays and matrixes, a `dim` attribute is always present and `dim` methods are provided. These allows the storage of time series of spectral data, or (hyper)spectral image data, or even higher dimensional spectral data. The handling of 1D and 2D spectral collections is already implemented in the summary methods. Handling of 3D and higher dimensional data can be implemented in the future without changing the class definition. By having implemented `dim`, also methods `ncol` and `nrow` are available as they use `dim` internally. Array-like subscripting is **not** implemented.

## 4.2 Construction

### 4.2.1 Constructors

We can construct a collection using a list of spectral objects as a starting point, in this case the spectral transmittance for two glass filters.

```
two_filters.mspct <- filter_mspct(list(gg400 = gg400.spct, og550 = og550.spct))
two_filters.mspct

## Object: filter_mspct [2 x 1]
## --- Member: gg400 ---
## Object: filter_spct [180 x 2]
## Wavelength (nm): range 200 to 5150, step 10 to 50
##
##    w.length   Tfr
## 1       200 1e-05
## 2       210 1e-05
## 3       220 1e-05
## 4       230 1e-05
## 5       240 1e-05
## ..      ...   ...
## --- Member: og550 ---
## Object: filter_spct [180 x 2]
## Wavelength (nm): range 200 to 5150, step 10 to 50
##
##    w.length   Tfr
## 1       200 1e-05
## 2       210 1e-05
## 3       220 1e-05
## 4       230 1e-05
## 5       240 1e-05
## ..      ...   ...
## --- END ---
```

We can also create heterogeneous collections, but this reduces the number of methods that can be used on the resulting collection.

```
mixed.mspct <- generic_mspct(list(filter = clear.spct, source = sun.spct))
```

### 4.2.2 Using 'as' functions

The `as` functions for collections of spectra, not only change the class of the collection object, but also apply the corresponding as functions to the member objects. They copy the original objects and then convert the copy, which is returned.

```
two_gen.mscpt <- as.generic_mspct(two_filters.mspct)
class(two_gen.mscpt)

## [1] "generic_mspct" "matrix"

lapply(two_gen.mscpt, class_spct)
```

```
## $gg400
## [1] "generic_spct"
##
## $og550
## [1] "generic_spct"
```

### 4.2.3 Converting 'tidy' data

Spectral objects containing multiple spectra identified by a factor (class of the argument is replicated to collection members).

```
two_suns.spct <- rbindspct(list(a = sun.spct, b = sun.spct / 2))
subset2mspct(two_suns.spct)

## Object: source_mspct [2 x 1]
## --- Member: a ---
## Object: source_spct [522 x 3]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...
## --- Member: b ---
## Object: source_spct [522 x 3]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...
## --- END ---
```

Data frame containing 'tidy' spectral data (target class and index variable need to be supplied as arguments).

```
test1.df <- data.frame(w.length = rep(200:210, 2),
                       s.e.irrad = rep(c(1, 2), c(11, 11)),
                       spectrum = factor(rep(c("A", "B"), c(11,11))))
subset2mspct(test1.df, member.class = "source_spct", idx.var = "spectrum")

## Object: source_mspct [2 x 1]
## --- Member: A ---
## Object: source_spct [11 x 2]
## Wavelength (nm): range 200 to 210, step 1
## Time unit: 1s
```

```
##
##    w.length s.e.irrad
## 1       200         1
## 2       201         1
## 3       202         1
## 4       203         1
## 5       204         1
## ..      ...       ...
## --- Member: B ---
## Object: source_spct [11 x 2]
## Wavelength (nm): range 200 to 210, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       200         2
## 2       201         2
## 3       202         2
## 4       203         2
## 5       204         2
## ..      ...       ...
## --- END ---
```

### 4.2.4 Converting 'untidy' data frames

Data frame containing 'untidy' or 'wide' spectral data (class is determined by the function used, columns which are not `numeric` are skipped.

```
test2.df <- data.frame(w.length = 200:210, A = 1, B = 2, z = "A")
split2source_mspct(test2.df)

## Object: source_mspct [2 x 1]
## --- Member: A ---
## Object: source_spct [11 x 2]
## Wavelength (nm): range 200 to 210, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       200         1
## 2       201         1
## 3       202         1
## 4       203         1
## 5       204         1
## ..      ...       ...
## --- Member: B ---
## Object: source_spct [11 x 2]
## Wavelength (nm): range 200 to 210, step 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       200         2
## 2       201         2
## 3       202         2
## 4       203         2
## 5       204         2
## ..      ...       ...
## --- END ---
```

```
split2source_mspct(test2.df, spct.data.var = "s.q.irrad")

## Object: source_mspct [2 x 1]
## --- Member: A ---
## Object: source_spct [11 x 2]
## Wavelength (nm): range 200 to 210, step 1
## Time unit: 1s
##
##    w.length s.q.irrad
## 1       200         1
## 2       201         1
## 3       202         1
## 4       203         1
## 5       204         1
## ..      ...       ...
## --- Member: B ---
## Object: source_spct [11 x 2]
## Wavelength (nm): range 200 to 210, step 1
## Time unit: 1s
##
##    w.length s.q.irrad
## 1       200         2
## 2       201         2
## 3       202         2
## 4       203         2
## 5       204         2
## ..      ...       ...
## --- END ---
```

## 4.3 Querying the class

`is.` functions are defined for these classes. R's `class` method can also be used.

```
is.filter_mspct(two_filters.mspct)

## [1] TRUE

class(two_filters.mspct)

## [1] "filter_mspct"  "generic_mspct" "list"
```

In addition to using `class` to query the class of the collection, we can use base R's `lapply` together with `class` or `class_spct` to query the class of each of the members of the collection.

```
is.filter_mspct(mixed.mspct)

## [1] FALSE

is.any_mspct(mixed.mspct)

## [1] TRUE

class(mixed.mspct)
```

```
## [1] "generic_mspct" "list"

lapply(mixed.mspct, class_spct)

## $filter
## [1] "filter_spct"  "generic_spct"
##
## $source
## [1] "source_spct"  "generic_spct"

lapply(mixed.mspct, class)

## $filter
## [1] "filter_spct"  "generic_spct" "data.table"   "data.frame"
##
## $source
## [1] "source_spct"  "generic_spct" "tbl_df"       "tbl"
## [5] "data.frame"
```

# 5 Wavebands

When a range of wavelengths or a range of wavelengths plus a spectral weighting function (SWF) is needed for radiation summaries or transformations, methods, operators and functions defined in package photobiology use `waveband` objects to store these data. A few other bits of information can be included to fine-tune calculations. The waveband definitions do NOT describe whether input spectral irradiances are photon or energy based, nor whether the output irradiance will be based on photon or energy units. All waveband objects belong to the S3 class `waveband`.

## 5.1 Construction

When defining a waveband which uses a SWF, a function can be supplied either based on energy effectiveness, on photon effectiveness, or one function for each one. If only one function is supplied the other one is built automatically, but if performance is a concern it is better to provide two separate functions. Another case when you might want to enter the same function twice, is if you are using an absorptance spectrum as SWF, as the percent of radiation absorbed will be independent of whether photon or energy units are used for the spectral irradiance.

To create a waveband object we use constructor function `waveband`, and optionally giving a name to it.

```
my_PAR <- waveband(c(400, 700), wb.name = "my_PAR")
```

When including a BSWF, we supply, one or two versions of functions returning the weights as a function of wavelength. Several such functions are defined in package photobiologyWavebands as well as constructors using them. Here we

give three examples of how equivalent wavebands can be defined based on a SWF. Although the constructor is smart enough to derive the missing function when only one function is supplied, performance may suffer.

```
my_CIE_1 <-
  waveband(c(250, 400), weight = "SWF", SWF.e.fun = CIE_e_fun, SWF.norm = 298)
my_CIE_2 <-
  waveband(c(250, 400), weight = "SWF", SWF.q.fun = CIE_q_fun, SWF.norm = 298)
my_CIE_3 <-
  waveband(c(250, 400), weight = "SWF", SWF.e.fun = CIE_e_fun,
                         SWF.q.fun = CIE_q_fun, SWF.norm = 298)
```

The first argument to `waveband()` does not need to be a numeric vector of length two. Any R object of a class that supplies a `range()` method definition that can be interpreted as a range of wavelengths in nanometres can be used. As a consequence, when wanting to construct a waveband covering the whole range of a spectrum one can simply supply the spectrum as argument, or to construct an unweighted waveband which covers exactly the same range of wavelengths as an existing effective (weighted) waveband, one can supply a waveband object as an argument.

```
waveband(sun.spct)

## Total
## low (nm) 280
## high (nm) 800
## weighted none

waveband(my_CIE_1)

## range.250.400
## low (nm) 250
## high (nm) 400
## weighted none
```

## 5.2 Querying the class

The function `is.waveband` can the used to query any R object. This function returns a logical value.

```
is.waveband(PAR())

## [1] TRUE

PAR <- PAR()
is.waveband(PAR)

## [1] TRUE
```

Above, we demonstrate that `PAR()` is a waveband constructor returning a waveband object, and `PAR` is a waveband object.

### 5.3 Retrieving properties

The function `is_effective` can the used to query any R object.

```
is_effective(PAR())

## [1] FALSE
```

# 6 Collections of wavebands

In the current implementation there is no special class used for storing collections of `waveband` objects. We simply use base R's `list` class.

## 6.1 Construction

### 6.1.1 List constructor

Just base R's functions used to create a list object.

```
wavebands <- list(waveband(c(300,400)), waveband(c(400,500)))
wavebands

## [[1]]
## range.300.400
## low (nm) 300
## high (nm) 400
## weighted none
##
## [[2]]
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
```

### 6.1.2 Special constructor

The function `split_bands` can be used to generate lists of unweighted wavebands in two different ways: a) it can be used to split a range of wavelengths given by an R object into a series of adjacent wavebands, or b) with a list of objects returning ranges, it can be used to create non-adjacent and even overlapping wavebands.

The code chunk bellow shows an example of two variations of case a). With the default value for `length.out` of `NULL` each numerical value in the input is taken as a wavelength (nm) at the boundary between adjacent wavebands. If a numerical value is supplied to `length.out`, then the whole wavelength range of the input is split into this number of equally spaced adjacent wavebands.

```
split_bands(c(200, 225, 300))

## $wb1
## range.200.225
## low (nm) 200
## high (nm) 225
## weighted none
##
## $wb2
## range.225.300
## low (nm) 225
## high (nm) 300
## weighted none

split_bands(c(200, 225, 300), length.out = 2)

## $wb1
## range.200.250
## low (nm) 200
## high (nm) 250
## weighted none
##
## $wb2
## range.250.300
## low (nm) 250
## high (nm) 300
## weighted none
```

In both examples above, the output is a list of two wavebands, but the 'split' boundaries are at a different wavelength. The chunk bellow gives a few more examples of the use of case a).

```
split_bands(sun.spct, length.out = 2)

## $wb1
## range.280.540
## low (nm) 280
## high (nm) 540
## weighted none
##
## $wb2
## range.540.800
## low (nm) 540
## high (nm) 800
## weighted none

split_bands(PAR(), length.out = 2)

## $wb1
## range.400.550
## low (nm) 400
## high (nm) 550
## weighted none
##
## $wb2
## range.550.700
```

```
## low (nm) 550
## high (nm) 700
## weighted none

split_bands(c(200, 800), length.out = 3)

## $wb1
## range.200.400
## low (nm) 200
## high (nm) 400
## weighted none
##
## $wb2
## range.400.600
## low (nm) 400
## high (nm) 600
## weighted none
##
## $wb3
## range.600.800
## low (nm) 600
## high (nm) 800
## weighted none
```

Now we demonstrate case b). This case is handled by recursion, so each list element can be anything that is a valid input to the function, including a nested list. However, the returned value is always a flat list of wavebands.

```
split_bands(list(A = c(200, 300), B = c(400, 500), C = c(250, 350)))

## $A
## range.200.300
## low (nm) 200
## high (nm) 300
## weighted none
##
## $B
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $C
## range.250.350
## low (nm) 250
## high (nm) 350
## weighted none

split_bands(list(c(100, 150, 200), c(800, 825)))

## $wb.a
## range.100.150
## low (nm) 100
## high (nm) 150
## weighted none
##
```

```
## $<NA>
## range.150.200
## low (nm) 150
## high (nm) 200
## weighted none
##
## $wb.b
## range.800.825
## low (nm) 800
## high (nm) 825
## weighted none
```

In case b) if we supply a numeric value to `length.out`, this value is used recursively for each element of the list.

```
split_bands(list(R = Red(), B = Blue()), length.out  =  2)

## $R
## range.610.685
## low (nm) 610
## high (nm) 685
## weighted none
##
## $<NA>
## range.685.760
## low (nm) 685
## high (nm) 760
## weighted none
##
## $B
## range.450.475
## low (nm) 450
## high (nm) 475
## weighted none
##
## $<NA>
## range.475.500
## low (nm) 475
## high (nm) 500
## weighted none

split_bands(list(c(100, 150, 200), c(800, 825)), length.out = 1)

## $wb.a
## range.100.200
## low (nm) 100
## high (nm) 200
## weighted none
##
## $wb.b
## range.800.825
## low (nm) 800
## high (nm) 825
## weighted none
```

Table 4: Binary operators and operands. Validity and class of result. All operations marked 'Y' are allowed, those marked 'N' are forbidden and return `NA` and issue a warning.

| e1 | + | - | * | / | ^ | e2 | result |
|---|---|---|---|---|---|---|---|
| cps_spct | Y | Y | Y | Y | Y | cps_spct | cps_spct |
| source_spct | Y | Y | Y | Y | Y | source_spct | source_spct |
| filter_spct (T) | N | N | Y | Y | N | filter_spct | filter_spct |
| filter_spct (A) | Y | Y | N | N | N | filter_spct | filter_spct |
| reflector_spct | N | N | Y | Y | N | reflector_spct | reflector_spct |
| object_spct | N | N | N | N | N | object_spct | – |
| response_spct | Y | Y | Y | Y | N | response_spct | response_spct |
| chroma_spct | Y | Y | Y | Y | Y | chroma_spct | chroma_spct |
| cps_spct | Y | Y | Y | Y | Y | numeric | cps_spct |
| source_spct | Y | Y | Y | Y | Y | numeric | source_spct |
| filter_spct | Y | Y | Y | Y | Y | numeric | filter_spct |
| reflector_spct | Y | Y | Y | Y | Y | numeric | reflector_spct |
| object_spct | N | N | N | N | N | numeric | – |
| response_spct | Y | Y | Y | Y | Y | numeric | response_spct |
| chroma_spct | Y | Y | Y | Y | Y | numeric | chroma_spct |
| source_spct | N | N | Y | Y | N | response_spct | response_spct |
| source_spct | N | N | Y | Y | N | filter_spct (T) | source_spct |
| source_spct | N | N | Y | Y | N | filter_spct (A) | source_spct |
| source_spct | N | N | Y | Y | N | reflector_spct | source_spct |
| source_spct | N | N | N | N | N | object_spct | – |
| source_spct | N | N | Y | N | N | waveband (no BSWF) | source_spct |
| source_spct | N | N | Y | N | N | waveband (BSWF) | source_spct |

# 7 Transformations: using operators

## 7.1 Binary operators

The basic maths operators have definitions for spectra. It is possible to sum, subtract, multiply and divide spectra. These operators can be used even if the spectral data is on different arbitrary sets of wavelengths. Operators by default use values expressed in energy units. Only certain operations are meaningful for a given combination of objects belonging to different classes, and meaningless combinations return `NA` also issuing a warning (see Table 4). By default operations are carried out on spectral energy irradiance for `source_spct` objects and transmittance for `filter_spct` objects.

```
sun.spct * sun.spct

## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
```

```
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

When meaningful, operations between different spectra are also allowed. For example, it is possible to simulate the effect of a filter on a light source by multiplying (or convolving) the two spectra.

```
sun.spct * polyester.new.spct
```

```
## Object: source_spct [535 x 2]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.0000         0
## 4  281.8571         0
## 5  282.0000         0
## ..      ...       ...
```

If we have two layers of the filter, this can be approximated using either of these two statements.

```
sun.spct * polyester.new.spct * polyester.new.spct
```

```
## Object: source_spct [535 x 2]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.0000         0
## 4  281.8571         0
## 5  282.0000         0
## ..      ...       ...
```

```
sun.spct * polyester.new.spct^2
```

```
## Object: source_spct [535 x 2]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.0000         0
## 4  281.8571         0
## 5  282.0000         0
## ..      ...       ...
```

Operators are also defined for operations between a spectrum and a numeric vector (with normal recycling).

```
sun.spct * 2
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

```
2 * sun.spct
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

```
sun.spct * c(0,1)
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

There is one special case, for `chroma_spct`: if the numeric operand has length three, containing three *named* values 'x', 'y' and 'z', the corresponding value is used for each of the chromaticity 'columns' in the `chroma_spct`. Un-named values or differently named values are not treated specially.

Operators are also defined for operations between an spectrum and a `waveband` object. The next to code chunks demonstrate how the class of the result depends on whether the `waveband` object describes a range of wavelengths or a range of wavelengths plus a BSWF.

```
sun.spct * UVB()

## Object: source_spct [37 x 2]
## Wavelength (nm): range 280 to 315, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

```
sun.spct * CIE()

## Object: source_spct [122 x 2]
## Wavelength (nm): range 280 to 400, step 0.9285714 to 1
## Time unit: 1s
## Data weighted using 'CIE98.298' BSWF
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

And of course these operations can be combined into more complex statements, including parentheses, when needed. The example below estimates the difference in effective spectral irradiance according to the CIE98 definition, between sunlight and sunlight filtered with a polyester film. Of course, the result is valid only for the solar spectral data used, which corresponds to Southern Finland.

```
sun.spct * CIE() - sun.spct * polyester.new.spct * CIE()

## Object: source_spct [135 x 2]
## Wavelength (nm): range 280 to 400, step 1.023182e-12 to 1
## Time unit: 1s
## Data weighted using 'CIE98.298' BSWF
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.0000         0
## 4  281.8571         0
## 5  282.0000         0
## ..      ...       ...
```

Table 5: Unary operators and maths functions. Validity and class of result. All operations marked 'Y' are allowed, those marked 'N' are not implemented and return `NA` and issue a warning.

| e1 | + | - | log() | log10() | exp() | sqrt() | result |
|---|---|---|---|---|---|---|---|
| cps_spct | Y | Y | Y | Y | Y | Y | cps_spct |
| source_spct | Y | Y | Y | Y | Y | Y | source_spct |
| filter_spct | Y | Y | Y | Y | Y | Y | filter_spct |
| reflector_spct | Y | Y | Y | Y | Y | Y | reflector_spct |
| object_spct | N | N | N | N | N | N | – |
| response_spct | Y | Y | Y | Y | Y | Y | response_spct |
| chroma_spct | Y | Y | Y | Y | Y | Y | chroma_spct |

## 7.2 Unary operators and maths functions

The most common maths functions, as well as unary minus and plus, are also implemented for spectral objects (see Table 5).

```
-sun.spct

## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...

sqrt(sun.spct)

## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

## 7.3 Options

Table 6 lists all the recognized options affecting maths operators and functions, and their default values. Within the suite all functions have a default value

Table 6: Options recognized by functions in the photobiology package and the values they can take.

| Option | default | function |
|---|---|---|
| **Base R** | | |
| digits | 7 | $d - 3$ used by `summary` |
| photobiology.radiation.unit | `"energy"` | output ($\mathrm{W\,m^{-2}\,nm^{-1}}$) |
| | `"photon"` | output ($\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$) |
| photobiology.filter.qty | `"transmittance"` | output (/1) |
| | `"absorptance"` | output (/1) |
| | `"absorbance"` | output (a.u. $\log_1 0$ base) |
| photobiology.use.hinges | `NULL` | guess automatically |
| | `TRUE` | do not insert hinges |
| | `FALSE` | do insert hinges |
| photobiology.auto.hinges.limit | 0.5 | wavelength step (nm) |
| photobiology.waveband.trim | `TRUE` | trim or exclude |
| photobiology.use.cached.mult | `FALSE` | cache intermediate results or not |
| photobiology.verbose | `FALSE` | give verbose output or not |

which is used when the options are undefined. Options are set using base R's function `options`, and queried with functions `options` and `getOption`.

The behaviour of the operators defined in this package depends on the value of two global options. For example, if we would like the operators to operate on spectral photon irradiance and return spectral photon irradiance instead of spectral energy irradiance, this behaviour can be set, and will remain active until unset or reset.

```
options(photobiology.radiation.unit = "photon")
sun.spct * UVB()

## Object: source_spct [37 x 2]
## Wavelength (nm): range 280 to 315, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.q.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...

options(photobiology.radiation.unit = "energy")
sun.spct * UVB()

## Object: source_spct [37 x 2]
## Wavelength (nm): range 280 to 315, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
```

Table 7: Transformation methods for spectra. Key: + available, – not available, f available in the future.

| methods | source | response | filter | reflector | object | chroma |
|---|---|---|---|---|---|---|
| `merge` | + | + | + | + | + | + |
| `rbindspct` | + | + | + | + | + | + |
| `e2q, q2e` | + | + | – | – | – | – |
| `A2T, T2A` | – | – | + | – | – | – |
| `subset` | + | + | + | + | + | + |
| `trim_spct` | + | + | + | + | + | + |
| `interpolate_spct` | + | + | + | + | + | + |
| `f_scale` | + | + | + | + | + | + |
| `normalize` | + | + | + | + | + | + |
| *`math operators`* | + | + | + | + | + | + |
| *`math functions`* | + | + | + | + | + | + |
| `tag` | + | + | + | + | + | + |

Table 8: Transformation methods for collections of spectra. Key: + available, – not available, f available in the future.

| methods | source | response | filter | reflector | object | chroma |
|---|---|---|---|---|---|---|
| `mutate_mspct` | + | + | + | + | + | + |
| `rbindspct` | + | + | + | + | + | + |
| `e2q, q2e` | + | + | – | – | – | – |
| `A2T, T2A` | – | – | + | – | – | – |
| `trim_spct` | f | f | f | f | f | f |
| `interpolate_spct` | f | f | f | f | f | f |
| `f_scale` | f | f | f | f | f | f |
| `normalize` | f | f | f | f | f | f |
| *`math operators`* | f | f | f | f | f | f |
| *`math functions`* | f | f | f | f | f | f |
| `tag` | f | f | f | f | f | f |

```
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

The other options listed in Table 6 can be set similarly, to unset any option, they can be given a `NULL` value.

# 8 Transformations: methods and functions

In this section we describe methods and functions that take one or more spectral objects, and in some cases also waveband objects, as arguments and return another spectral object (Table 7) or that take a collection of spectral objects, and in some cases also waveband objects, as argumnents and return a collection of spectral objects (Table 8).

## 8.1 Manipulating spectra

Sometimes, especially for plotting, we may want to row-bind spectra. When the aim is that the returned object retains its class and other attributes like the time unit. Package photobiology provides function `rbinspct` for row-binding spectra, with the necessary checks for consistency of the bound spectra.

```
# STOPGAP
shade.spct <- sun.spct
```

By default an ID factor named `spct.idx` is added allow to idetify the source of the observations after the binding. If the supplied list has named members, then these names are used as factor levels. If a character value is supplied to as `idfactor` argument, this is used as name for the factor.

```
rbindspct(list(sun.spct, shade.spct))

## Object: source_spct [1,044 x 4]
## Wavelength (nm): range 280 to 800, step -520 to 1
## Time unit: 1s
##
##    w.length s.e.irrad spct.idx s.q.irrad
## 1  280.0000         0   spct_1         0
## 2  280.9286         0   spct_1         0
## 3  281.8571         0   spct_1         0
## 4  282.7857         0   spct_1         0
## 5  283.7143         0   spct_1         0
## ..      ...       ...      ...       ...

rbindspct(list(A = sun.spct, B = shade.spct), idfactor = "site")

## Object: source_spct [1,044 x 4]
## Wavelength (nm): range 280 to 800, step -520 to 1
## Time unit: 1s
##
##    w.length s.e.irrad site s.q.irrad
## 1  280.0000         0    A         0
## 2  280.9286         0    A         0
## 3  281.8571         0    A         0
## 4  282.7857         0    A         0
## 5  283.7143         0    A         0
## ..      ...       ...  ...       ...
```

In the special case when the members of the list are `source_spct` objects containing effective spectral irradiance data, and they are not based on the same BSWF, an additional factor `BSWF` will be automatically added, and the `BSWF` attribute of the resulting spectrum set to `"multiple"`.

```
rbindspct(list(sun.spct * CIE(), sun.spct * GEN.G()))

## Object: source_spct [158 x 4]
## Wavelength (nm): range 280 to 400, step -120 to 1
## Time unit: 1s
## Data weighted using 'multiple' BSWF
```

```
##
##    w.length s.e.irrad spct.idx      BSWF
## 1  280.0000         0   spct_1 CIE98.298
## 2  280.9286         0   spct_1 CIE98.298
## 3  281.8571         0   spct_1 CIE98.298
## 4  282.7857         0   spct_1 CIE98.298
## 5  283.7143         0   spct_1 CIE98.298
## ..      ...       ...      ...       ...
```

Special *Extract* methods for spectral objects have been implemented. These are used by default and preserve the attributes used by this package, except when the returned value is a single column from the spectral object.

```
sun.spct[1:10, ]

## Object: source_spct [10 x 3]
## Wavelength (nm): range 280 to 288.35714, step 0.9285714
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...

sun.spct[1:10, 1]

##  [1] 280.0000 280.9286 281.8571 282.7857 283.7143 284.6429 285.5714
##  [8] 286.5000 287.4286 288.3571

sun.spct[1:10, 1, drop = TRUE]

##  [1] 280.0000 280.9286 281.8571 282.7857 283.7143 284.6429 285.5714
##  [8] 286.5000 287.4286 288.3571

sun.spct[1:10, "w.length", drop = TRUE]

##  [1] 280.0000 280.9286 281.8571 282.7857 283.7143 284.6429 285.5714
##  [8] 286.5000 287.4286 288.3571
```

In contrast to `trim_spct`, `subset` never interpolates or inserts *hinges*. On the other hand, the `subset` argument accepts any logical expression and can be consequently used to do subsetting, for example, based on factors. Both `subset()` and `trim()` methods preserve attributes.

```
subset(sun.spct, s.e.irrad > 0.2)

## Object: source_spct [475 x 3]
## Wavelength (nm): range 324 to 800, step 1 to 3
## Time unit: 1s
##
##    w.length s.e.irrad    s.q.irrad
## 1       324 0.2075508 5.621282e-07
```

```
## 2       325 0.2168055 5.890059e-07
## 3       326 0.2774416 7.560580e-07
## 4       327 0.2851096 7.793375e-07
## 5       328 0.2648573 7.261924e-07
## ..      ...       ...          ...

subset(sun.spct, w.length > 600)

## Object: source_spct [200 x 3]
## Wavelength (nm): range 601 to 800, step 1
## Time unit: 1s
##
##    w.length s.e.irrad    s.q.irrad
## 1       601 0.6295837 3.162962e-06
## 2       602 0.6305890 3.173284e-06
## 3       603 0.6360329 3.205995e-06
## 4       604 0.6578140 3.321284e-06
## 5       605 0.6614323 3.345082e-06
## ..      ...       ...          ...

subset(sun.spct, c(TRUE, rep(FALSE, 99)))

## Object: source_spct [6 x 3]
## Wavelength (nm): range 280 to 779, step 99 to 100
## Time unit: 1s
##
##    w.length s.e.irrad    s.q.irrad
## 1       280 0.0000000 0.000000e+00
## 2       379 0.4131498 1.308919e-06
## 3       479 0.7536975 3.017857e-06
## 4       579 0.6474340 3.133575e-06
## 5       679 0.5798542 3.291202e-06
## ..      ...       ...          ...
```

R's Extract methods `$` and `[[ ]]` can be used to extract whole columns. Replace methods `$<-` and `[<-` have definitions for spectral objects, which allow their safe use. They work identically to those for data frames but check the validity of the spectra after the replacement.

## 8.2 Conversions between radiation units

The functions `e2q` and `q2e` can be used on source spectra to convert spectral energy irradiance into spectral photon irradiance and vice versa. A second optional argument sets the action with `"add"` and `"replace"` as possible values. By default these functions use normal reference semantics.

```
e2q(sun.spct, "add")

## Object: source_spct [522 x 3]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
```

```
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...
```

```
e2q(sun.spct, "replace")
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
##
##    w.length s.q.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

For `filter_spct` objects functions `T2A` and `A2T` allow conversion between spectral transmittance and spectral absorbance and vice versa.

## 8.3 Normalizing a spectrum

Function `normalize` permits normalizing a spectrum to a value of one at an arbitrary wavelength (nm) or to the wavelength of either the maximum or the minimum spectral value. It supports the different spectral classes, we use a `source_spct` object as an example.

```
normalize(sun.spct)
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
## Data normalized to 451 nm
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

Which is equivalent to supplying `"max"` as argument to `norm`, it is also possible to give a range within which the maximum should be searched.

```
normalize(sun.spct, range = PAR(), norm = "max")
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
## Data normalized to 451 nm
```

```
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

It is also possible to normalize to an arbitrary wavelength within the range of the data, even if it is not one of the wavelength values present in the spectral object, as interpolation is used when needed.

```
normalize(sun.spct, norm = 600.3)

## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
## Data normalized to 600.3 nm
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

## 8.4 Rescaling a spectrum

Function `f_scale` rescales a spectrum by dividing each spectral data value by a value calculated with a function (f) selected by a character string ("total" or "mean"), or an actual R function which can accept the spectrum object supplied as its first argument.

```
fscale(sun.spct)

## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
## Rescaled to 'mean' = 1
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...

fscale(sun.spct, f = "total")

## Object: source_spct [522 x 2]
```

```
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
## Rescaled to 'total' = 1
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

```
fscale(sun.spct, range = PAR(), f = irrad)
```

```
## Object: source_spct [522 x 2]
## Wavelength (nm): range 280 to 800, step 0.9285714 to 1
## Time unit: 1s
## Rescaled to 'a user supplied R function' = 1
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

In the third example, the spectral data is rescaled so that the corresponding photosynthetically-active irradiance is equal to one.

## 8.5 Wavelength interpolation

Converting spectra available at a given set of wavelengths values to a different one, is frequently needed when operating with several spectra of different origin. One can increase the *apparent* resolution by interpolation, and reduce it by local averaging or smoothing and resampling. The same function works on all `spct` objects, interpolating every column except `w.length` and replacing in this last column the old wavelength values with the new ones supplied as argument. The optional argument `fill.value` control what value is assigned to wavelengths in the new data that are outside the range of the old wavelengths.

```
interpolate_spct(sun.spct, seq(400, 500, by = 0.1))
```

```
## Object: source_spct [1,001 x 3]
## Wavelength (nm): range 400 to 500, step 0.1
## Time unit: 1s
##
##    w.length s.e.irrad    s.q.irrad
## 1     400.0 0.6081049 2.033314e-06
## 2     400.1 0.6099118 2.039879e-06
## 3     400.2 0.6117187 2.046445e-06
## 4     400.3 0.6135257 2.053010e-06
## 5     400.4 0.6153326 2.059575e-06
## ..      ...       ...          ...
```

## 8.6 Trimming

Sometimes it is desirable to change the range of wavelengths included in a spectrum. If we are interested in a given part of the spectrum, there is no need to do calculations or plotting the whole spectrum. Sometimes we may want to expand the range of wavelengths, filling the expansion of all other variables with a certain value (i.e. a number, or NA.) In contrast to indexing, or subsetting, *trimming* ensures that there will be spectral data returned at the boundaries of the trimmed region. These values are obtained by interpolation when they are not already present in the data.

We can supply the arguments `band`, `low.limit`, `high.limit`, and `fill`. Either `band` or `low.limit` and/or `high.limit` arguments should supplied, but not both at once.

```
trim_spct(sun.spct, PAR())
```

```
## Object: source_spct [301 x 3]
## Wavelength (nm): range 400 to 700, step 1
## Time unit: 1s
##
##    w.length s.e.irrad    s.q.irrad
## 1       400 0.6081049 2.033314e-06
## 2       401 0.6261742 2.098967e-06
## 3       402 0.6497388 2.183388e-06
## 4       403 0.6207287 2.091091e-06
## 5       404 0.6370489 2.151395e-06
## ..      ...       ...          ...
```

```
trim_spct(sun.spct, low.limit = 297)
```

```
## Object: source_spct [504 x 3]
## Wavelength (nm): range 297 to 800, step 1
## Time unit: 1s
##
##    w.length    s.e.irrad    s.q.irrad
## 1       297 0.0001533491 3.807181e-10
## 2       298 0.0003669677 9.141345e-10
## 3       299 0.0007845430 1.960893e-09
## 4       300 0.0012645540 3.171207e-09
## 5       301 0.0026237179 6.601607e-09
## ..      ...          ...          ...
```

The default `fill` value is `NULL` which means deleting the values outside the trimmed region. However, it is possible to supply a different argument.

```
trim_spct(sun.spct, low.limit = 297, fill = 0)
```

```
## Object: source_spct [523 x 3]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
```

```
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...
```

In addition, when fill is not `NULL`, expansion is possible.

```
trim_spct(sun.spct, low.limit = 290, fill = 0)

## Object: source_spct [524 x 3]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...
```

## 8.7 Convoluting weights

It is very instructive to look at weighted spectral data to understand how effective irradiances are calculated. Plotting effective spectral irradiance data can be very instructive when analyzing the interaction of photoreceptors and ambient radiation. It can also illustrate what a large effect that small measuring errors can have on the estimated effective irradiances or exposures when SWFs have a steep slope.

### 8.7.1 Individual spectra

The multiplication operator is defined for operations between a `source_spct` and a `waveband`, so this is the easiest way of doing the calculations.

```
sun.spct * CIE()

## Object: source_spct [122 x 2]
## Wavelength (nm): range 280 to 400, step 0.9285714 to 1
## Time unit: 1s
## Data weighted using 'CIE98.298' BSWF
##
##    w.length s.e.irrad
## 1  280.0000         0
## 2  280.9286         0
## 3  281.8571         0
## 4  282.7857         0
## 5  283.7143         0
## ..      ...       ...
```

### 8.7.2 Vectors

It is also possible to use vectors.

```
weighted.s.e.irrad <-
  with(sun.spct,
       s.e.irrad * calc_multipliers(w.length, CIE())
  )
```

## 8.8 Tagging with bands and colours

We call tagging, to the process of adding reference information to spectral data. For example we can add a factor indicating regions or bands in the spectrum. We can add also information on the colour, as seen by humans, for each observed value, or for individual regions or bands of the spectrum. In most cases this additional information is used for annotations when plotting the spectral data.

### 8.8.1 Individual spectra

The function `tag` can be used to tag different parts of a spectrum according to wavebands.

```
tag(sun.spct, PAR(), byref = FALSE)

## Object: source_spct [524 x 5]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad wl.color wb.f
## 1  280.0000         0         0  #000000   NA
## 2  280.9286         0         0  #000000   NA
## 3  281.8571         0         0  #000000   NA
## 4  282.7857         0         0  #000000   NA
## 5  283.7143         0         0  #000000   NA
## ..      ...       ...       ...      ...  ...

tag(sun.spct, UV_bands(), byref = FALSE)

## Object: source_spct [524 x 5]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad wl.color wb.f
## 1  280.0000         0         0  #000000  UVB
## 2  280.9286         0         0  #000000  UVB
## 3  281.8571         0         0  #000000  UVB
## 4  282.7857         0         0  #000000  UVB
## 5  283.7143         0         0  #000000  UVB
## ..      ...       ...       ...      ...  ...
```

The added factor and colour data can be used for further processing or for plotting. Information about the tagging and wavebands is stored in an attribute

`tag.attr` in every tagged spectrum, this yields a more compact output and keeps a 'trace' of the tagging.

```
tg.sun.spct <- tag(sun.spct, PAR(), byref = FALSE)
attr(tg.sun.spct, "spct.tags")

## $time.unit
## [1] "second"
##
## $wb.key.name
## [1] "Bands"
##
## $wl.color
## [1] TRUE
##
## $wb.color
## [1] TRUE
##
## $wb.num
## [1] 1
##
## $wb.colors
## $wb.colors[[1]]
##   PAR.CMF
## "#735B57"
##
##
## $wb.names
## [1] "PAR"
##
## $wb.list
## $wb.list[[1]]
## PAR
## low (nm) 400
## high (nm) 700
## weighted none
```

Additional functions are available which return a tagged spectrum and take as input a list of wavebands, but no spectral data. They 'build' a spectrum from the data in the wavebands, and are useful for plotting the boundaries of wavebands.

```
wb2tagged_spct(UV_bands())

## Object: generic_spct [8 x 9]
## Wavelength (nm): range 100 to 400, step 9.947598e-13 to 180
##
##    w.length s.e.irrad s.q.irrad Tfr Rfl s.e.response wl.color wb.f y
## 1       100         0         0   0   0            0  #000000   NA 0
## 2       100         0         0   0   0            0  #000000  UVC 0
## 3       280         0         0   0   0            0  #000000  UVC 0
## 4       280         0         0   0   0            0  #000000  UVB 0
## 5       315         0         0   0   0            0  #000000  UVB 0
## ..      ...       ...       ... ... ...          ...      ...  ... .

wb2rect_spct(UV_bands())
```

```
## Object: generic_spct [3 x 11]
## Wavelength (nm): range 190 to 357.5, step 60 to 107.5
##
##   w.length s.e.irrad s.q.irrad Tfr Rfl s.e.response wl.color wb.f wl.high
## 1    190.0         0         0   0   0            0  #000000  UVC     280
## 2    297.5         0         0   0   0            0  #000000  UVB     315
## 3    357.5         0         0   0   0            0  #000000  UVA     400
## Variables not shown: wl.low (dbl), y (dbl)
```

Function `wb2tagged_spct` returns a tagged spectrum, with two rows for each waveband, corresponding to the low and high wavelength boundaries, while function `wb2rect_spct` returns a spectrum with only one row per waveband, with `w.length` set to its midpoint but with additional columns `xmin` and `xmax` corresponding to the low and high wavelength boundaries of the wavebands.

Function `is_tagged` can be used to query if an spectrum is tagged or not, and function `untag` removes the tags.

```
tg.sun.spct

## Object: source_spct [524 x 5]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad wl.color wb.f
## 1  280.0000         0         0  #000000   NA
## 2  280.9286         0         0  #000000   NA
## 3  281.8571         0         0  #000000   NA
## 4  282.7857         0         0  #000000   NA
## 5  283.7143         0         0  #000000   NA
## ..      ...       ...       ...      ...  ...

is_tagged(tg.sun.spct)

## [1] TRUE

untag(tg.sun.spct)

## Object: source_spct [524 x 3]
## Wavelength (nm): range 280 to 800, step 1.023182e-12 to 1
## Time unit: 1s
##
##    w.length s.e.irrad s.q.irrad
## 1  280.0000         0         0
## 2  280.9286         0         0
## 3  281.8571         0         0
## 4  282.7857         0         0
## 5  283.7143         0         0
## ..      ...       ...       ...

is_tagged(tg.sun.spct)

## [1] TRUE
```

In the chuck above, we can see how this works, using in this case the default

Table 9: Summary methods for spectra. Key: + available, – not available, f available in the future.

| methods | source | response | filter | reflector | object | chroma |
|---|---|---|---|---|---|---|
| `irrad, e_irrad, q_irrad` | + | – | – | – | – | – |
| `fluence, e_fluence, q_fluence` | + | – | – | – | – | – |
| `ratio, e_ratio, q_ratio` | + | – | – | – | – | – |
| `qe_ratio, eq_ratio` | + | – | – | – | – | – |
| `response, e_response, q_response` | – | + | – | – | – | – |
| `transmittance` | – | – | + | – | + | – |
| `absorptance` | – | – | + | – | + | – |
| `absorbance` | – | – | + | – | + | – |
| `range, min, max` | + | + | + | + | + | + |
| `stepsize, spread, midpoint` | + | + | + | + | + | + |
| `labels` | + | + | + | + | + | + |
| `summary` | + | + | + | + | + | + |
| `peaks` | + | + | + | + | + | + |
| `valleys` | + | + | + | + | + | + |
| `integrate_spct` | + | + | + | + | + | + |
| `average_spct` | + | + | + | + | + | + |
| `color` | + | – | + | + | – | + |

`byref = TRUE` which adds the tags in place, or "by reference", to the spct object supplied as argument.

# 9 Summaries

Summaries can be calculated both from individual spectral objects (Table 9) and from collections of spectral objects (Table 10). They return a *simpler* object than the spectral data in their arguments. For example a vector of numeric values, possibly of length one, in the case of individual spectra, or a data frame containing one row of summary data for each spectrum the collection of multiple spectra supplied as argument.

## 9.1 Summary

Specialized definitions of `summary` and the corresponding `print` methods are available for spectral objects. In the case of `source_spct` objects the `time.unit` attribute makes it possible to print the summary using the correct units.

```
summary(sun.spct)

## wavelength ranges from 280 to 800 nm
## largest wavelength step size is 0.9286 nm
## spectral irradiance ranges from 0 to 0.8205 W m-2 nm-1
## energy irradiance is 269.1 W m-2
## spectral photon irradiance ranges from 0 to 3.375 umol s-1 m-2 nm-1
## photon irradiance is 1255 umol s-1 m-2
```

Table 10: Summary methods for collections of spectra. Key: + available, – not available, f available in the future.

| methods | source | response | filter | reflector | object | chroma |
|---|---|---|---|---|---|---|
| `f_mspct` | + | + | + | + | + | + |
| `irrad, e_irrad, q_irrad` | + | – | – | – | – | – |
| `fluence, e_fluence, q_fluence` | + | – | – | – | – | – |
| `ratio, e_ratio, q_ratio` | + | – | – | – | – | – |
| `qe_ratio, eq_ratio` | + | – | – | – | – | – |
| `response, e_response, q_response` | – | + | – | – | – | – |
| `transmittance` | – | – | + | – | + | – |
| `absorptance` | – | – | + | – | + | – |
| `absorbance` | – | – | + | – | + | – |
| `range, min, max` | + | + | + | + | + | + |
| `stepsize, spread, midpoint` | + | + | + | + | + | + |
| `labels` | – | – | – | – | – | – |
| `summary` | – | – | – | – | – | – |
| `peaks` | f | f | f | f | f | f |
| `valleys` | f | f | f | f | f | f |
| `integrate_spct` | f | f | f | f | f | f |
| `average_spct` | f | f | f | f | f | f |
| `color` | f | f | f | f | f | f |

## 9.2 Wavelength

### 9.2.1 Individual spectra

The 'usual' and a couple of new summary functions are available for spectra, but redefined to return wavelength based summaries in nanometres (nm).

```
range(sun.spct)

## [1] 280 800

min(sun.spct)

## [1] 280

max(sun.spct)

## [1] 800

midpoint(sun.spct)

## [1] 540

spread(sun.spct)

## [1] 520

stepsize(sun.spct)

## [1] 0.9285714 1.0000000
```

### 9.2.2 Collections of spectra

> Not all summary methods are yet implemented for collections of spectra, because this is a new set of classes, just added to the package. See Table 10 where methods be implemented in the *future* are marked with 'f'. Functions `f_mspct` or `plyr::ldply` can be used to apply a function to all the spectra in a collection and obtain the results in a data frame object. Other *apply* functions or `for` loops can also be used if needed.

Collections of spectra can be useful not only for time-series of spectra or spectral images, but also when dealing with a small group of related spectra. In the example below we show how to use a collection of spectra for calculating summaries. The spectra in a collection do **not** need to have been measured at the same wavelength values, or have the same number of rows or even of columns. Consequently, in many cases applying the wavelength summary functions described above to collections of spectra can be useful. The value returned is a data frame, with a number of data columns equal to the length of the returned value by the corresponding method for individual spectra.

```
filtered_sun <-
  source_mspct(
    lapply(list(none = clear.spct,
                ug1 = ug1.spct, ug11 = ug11.spct,
                gg400 = gg400.spct,
                og550 = og550.spct,
                rg665 = rg665.spct, rg830 = rg830.spct),
           `*`,
           y = sun.spct))
range(filtered_sun)

##   spct.idx min.wl max.wl
## 1     none    280    800
## 2      ug1    280    800
## 3     ug11    280    800
## 4    gg400    280    800
## 5    og550    280    800
## 6    rg665    280    800
## 7    rg830    280    800
```

## 9.3 Peaks and valleys

### 9.3.1 Individual spectra

Functions `peaks` and `valleys` take spectra as first argument and return a subset of the spectral object data corresponding to local maxima and local minima of the measured variable. `span` defines the width of the 'window' used as a number of observations.

```
peaks(sun.spct, span = 51)

## Object: source_spct [3 x 2]
## Wavelength (nm): range 451 to 747, step 44 to 252
## Time unit: 1s
##
##   w.length s.e.irrad
## 1      451 0.8204633
## 2      495 0.7899872
## 3      747 0.5025733

valleys(sun.spct, span = 51)

## Object: source_spct [9 x 2]
## Wavelength (nm): range 358 to 761, step 30 to 72
## Time unit: 1s
##
##    w.length s.e.irrad
## 1       358 0.2544907
## 2       393 0.2422023
## 3       431 0.4136900
## 4       487 0.6511654
## 5       517 0.6176652
## ..      ...       ...
```

In the case of `source_spct` and `response_spct` methods `unit.out` can be used to force peaks to be searched using either energy or photon based spectral irradiance. The default is energy, or the option `"photobiology.radiation.unit"` if set.

```
peaks(sun.spct, span = 51, unit.out = "photon")

## Object: source_spct [7 x 2]
## Wavelength (nm): range 451 to 754, step 36 to 90
## Time unit: 1s
##
##    w.length    s.q.irrad
## 1       451 3.093155e-06
## 2       495 3.268822e-06
## 3       531 3.374912e-06
## 4       621 3.355564e-06
## 5       668 3.337584e-06
## ..      ...          ...
```

It is possible to approximately set the width of the windows in nanometres by using function `step_size`. However, here we simply use an odd number of wavelengths 'steps'.

```
peaks(sun.spct, span = 21)

## Object: source_spct [18 x 2]
## Wavelength (nm): range 354 to 774, step 11 to 51
## Time unit: 1s
##
##    w.length s.e.irrad
```

```
## 1       354 0.3758625
## 2       366 0.4491898
## 3       378 0.4969714
## 4       416 0.6761818
## 5       436 0.7336607
## ..      ...       ...
```

Low level functions `find_peaks`, `get_peaks` and `get_valleys` take numeric vectors as arguments.

## 9.4 Irradiance

### 9.4.1 Individual spectra

The code using `spct` objects is simple, to integrate the whole spectrum we can use

```
irrad(sun.spct)

##    Total
## 269.1249
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

and, to integrate a range of wavelengths, in the example, photosynthetically active radiation, we use the predefined waveband constructor `PAR()`.

```
irrad(sun.spct, PAR())

##      PAR
## 196.6343
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

The default for `irrad`, when no argument `unit.out` is supplied, is to return the irradiance value in energy irradiance units, unless the R option `photobiology.radiation.unit` is set.

Functions `e_irrad` and `q_irrad` save some typing, and always return the same type of spectral irradiance quantity, independently of global option `photobiology.radiation.unit`.

```
e_irrad(sun.spct, PAR()) # W m-2

##      PAR
## 196.6343
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

```
q_irrad(sun.spct, PAR()) * 1e6 # umol s-1 m-2

##      PAR
## 894.1352
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

It is also possible to supply a time unit to use as basis of expression for the returned value, but be aware that conversion into a loger time unit is only valid for sources like lamps, which have an output the remains constant in time.

```
irrad(sun.spct, PAR(), time.unit = "hour")

##      PAR
## 707883.4
## attr(,"time.unit")
## [1] "hour"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

irrad(sun.spct, PAR(), time.unit = duration(8, "hours"))

##     PAR
## 5663067
## attr(,"time.unit")
## [1] "28800s (~8 hours)"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

Using a shorter time unit than the original, yields an average value re-expressed on a new time unit base.

```
irrad(sun.daily.spct, PAR(), time.unit = "second")

##      PAR
## 92.16251
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

Lists of wavebands are also accepted as argument.

```
my_wavebands <- list(Red(), Blue(), Green())
e_irrad(sun.spct, my_wavebands) # W m-2

##    Red.ISO   Blue.ISO  Green.ISO
##   79.38159   37.55207   49.26860
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

These functions have an additional argument `quantity`, with default `"total"`, which can take values controlling the output. The value "total" yields **irradiance** in $\mathrm{W\,m^{-2}}$, integrated over wavelengths for each waveband, while "average" yields the mean **spectral irradiance** within each waveband in $\mathrm{W\,m^{-2}\,nm^{-1}}$. The value "contribution" is relative to the irradiance for the whole spectrum, expressed as a fraction of one, while the value "relative" is relative to the sum of the irradiances for the different wavbands given as argument, also expressed as a fraction of one.

```
irrad(sun.spct, UV_bands(), quantity = "total")

##    UVB.ISO    UVA.ISO
##  0.6445078 27.9842061
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

irrad(sun.spct, UV_bands(), quantity = "contribution")

##     UVB.ISO     UVA.ISO
## 0.002394828 0.103982227
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance contribution"

irrad(sun.spct, UV_bands(), quantity = "relative")

##    UVB.ISO    UVA.ISO
## 0.02251264 0.97748736
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance relative"

irrad(sun.spct, UV_bands(), quantity = "average")

##    UVB.ISO    UVA.ISO
## 0.01841451 0.32922595
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance average"
```

### 9.4.2 Collections of spectra

Collections of spectra can be useful not only for time-series of spectra or spectral images, but also when dealing with a small group of related spectra. In the example below we show how to use a collection of spectra for estimating irradiances under different filters set up in sunlight.

One thing to remember, is that operators in R are just functions with special names and call syntax. They can also be called with the usual function call

syntax by enclosing their *name* in backquotes. We use this trick to use the multiplication operator `*` in a call to `lapply` which returns a list, which we convert into a `source_multi_spct` object. After this we just call the `irrad` method on the *collection of spectra* and obtain the result as a data frame with one row per spectrum and one column by waveband.

```
filtered_sun <-
  source_mspct(
    lapply(list(none = clear.spct,
                ug1 = ug1.spct, ug11 = ug11.spct,
                gg400 = gg400.spct,
                og550 = og550.spct,
                rg665 = rg665.spct, rg830 = rg830.spct),
           `*`,
           y = sun.spct))
irrad(filtered_sun, list(UVA(), VIS()))

##   spct.idx irrad_UVA.ISO irrad_VIS.ISO
## 1     none  2.798421e+01  2.318635e+02
## 2      ug1  1.821686e+01  1.275259e+01
## 3     ug11  1.815902e+01  8.748836e+00
## 4    gg400  1.100107e+00  2.169464e+02
## 5    og550  2.798421e-04  1.156291e+02
## 6    rg665  2.798421e-04  4.455556e+01
## 7    rg830  2.798421e-04  2.875946e-03
```

### 9.4.3 Numeric vectors

The code using numeric vectors is more complicated, but adds some additional flexibility for tweaking performance. Under normal circumstances it is easier to use the functions described above.

Function `irradiance` takes an array of wavelengths (sorted in strictly increasing order), and the corresponding values of spectral irradiance. By default the input is assumed to be in energy units, but parameter `unit.in` can be used to change this default. The type of unit used for the returned quantity is set by `unit.out` with no default. The behaviour with respect to wavebands is as described above for spectral objects. The functions `photon_irradiance()` and `energy_irradiance()`, just call `irradiance()` with the `unit.out` set to `"photon"` or `"energy"` respectively.

The functions taking numerical vectors as arguments can be used with data stored as vectors, or using `with` with data frames, data tables, lists, and spectra objects.

```
with(sun.spct, photon_irradiance(w.length, s.e.irrad, PAR()))

##          PAR
## 0.0008941352
```

The recommended practice is to use `with`, as above.

## 9.5 Fluence

### 9.5.1 Individual spectra

The calculation of fluence values (time-integrated irradiance) is identical to that for irradiance, except that a `exposure.time` argument needs to be supplied. The exposure time must be a `lubridate::duration`, but any argument accepted by `as.duration` can also be used. Functions `fluence`, `e_fluence` and `q_fluence` correspond to `irrad`, `e_irrad` and `q_irrad`,

```
fluence(sun.spct, exposure.time = duration(1, "hours"))

##    Total
## 968849.6
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "3600s (~1 hours)"

fluence(sun.spct, exposure.time = 3600) # seconds

## converting 'time.unit' 3600 into a lubridate::duration

##    Total
## 968849.6
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] 3600

fluence(sun.spct, exposure.time = hms("01:00:00"))

## converting 'time.unit' 1H 0M 0S into a lubridate::duration
## estimate only:  convert periods to intervals for accuracy

##    Total
## 968849.6
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "1H 0M 0S"
```

and, to obtain the photon fluence for a range of wavelengths, in the example, photosynthetically active radiation, we use `PAR()` that is a predefined waveband constructor, for 25 minutes of exposure we use.

```
e_fluence(sun.spct, PAR(), exposure.time = hms("00:25:00"))

## converting 'time.unit' 25M 0S into a lubridate::duration
## estimate only:  convert periods to intervals for accuracy

##      PAR
## 294951.4
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "25M 0S"
```

## 9.6 Photon and energy ratios

### 9.6.1 Individual spectra

The functions described here, in there simplest use, calculate a ratio between two wavebands. The function `q_ratio` returning photon ratios. However both waveband parameters can take lists of wavebands as arguments, with normal recycling rules in effect. The corresponding function `e_ratio` returns energy ratios.

```
q_ratio(sun.spct, UVB(), PAR())

##  UVB.ISO: PAR(q:q)
##        0.001873717
## attr(,"radiation.unit")
## [1] "q:q ratio"

q_ratio(sun.spct,
        list(UVC(), UVB(), UVA()),
        UV())

##  UVB.ISO: UV.ISO.tr.lo(q:q)  UVA.ISO: UV.ISO.tr.lo(q:q)
##                  0.01936939                  0.98063061
## attr(,"radiation.unit")
## [1] "q:q ratio"

q_ratio(sun.spct,
        UVB(),
        list(UV(), PAR()))

##  UVB.ISO: UV.ISO.tr.lo(q:q)           UVB.ISO: PAR(q:q)
##                 0.019369385                 0.001873717
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

Function `qe_ratio`, has only one waveband parameter, and returns the 'photon' to 'energy' ratio, while its complement `eq_ratio` returns the 'energy' to 'photon' ratio.

```
qe_ratio(sun.spct, list(Blue(), Green(), Red()))

##  q:e( Blue.ISO) q:e( Green.ISO)   q:e( Red.ISO)
##    3.968591e-06    4.469290e-06    5.682783e-06
## attr(,"radiation.unit")
## [1] "q:e ratio"
```

### 9.6.2 Collections of spectra

```
q_ratio(filtered_sun, list(UVB(), UVA()), PAR())

##   spct.idx q_ratio_UVB.ISO:PAR(q:q) q_ratio_UVA.ISO:PAR(q:q)
## 1     none             1.873717e-03             9.486227e-02
```

```
## 2      ug1             2.732944e-01             2.315144e+01
## 3     ug11             1.665029e-01             5.844365e+00
## 4    gg400             1.937818e-08             4.223153e-03
## 5    og550             3.627111e-08             1.836329e-06
## 6    rg665             1.655523e-07             8.381559e-06
## 7    rg830             1.873717e-03             9.486227e-02
```

### 9.6.3 Vectors

The function `waveband_ratio()` takes basically the same parameters as `irradiance`, but two waveband definitions instead of one, and two `unit.out` definitions instead of one. This is the base function used in all the vector based 'ratio' functions in the photobiology package.

Similar functions `photon_ratio()`, `energy_ratio()`, and `photons_energy_ratio` return the other ratios described above. In contrast to the functions described in the previous section, these functions only accept individual waveband definitions (not lists of them).

To calculate the photon ratio between UVB and PAR photon irradiance in these to regions we use.

```
with(sun.data,
     photon_ratio(w.length, s.e.irrad, UVB(), PAR())
)

## [1] 0.00187372
```

## 9.7 Normalized difference indexes

## 9.8 Individual spectra

These indexes are frequently used to summarize reflectance data, for example in remote rensing the NDVI (normalized difference vegetation index). Here we give an *unusual* example to demonstrate that function `normalized_diff_ind()` can be used to calculate, or define any similar index.

```
normalized_diff_ind(sun.spct,
                    waveband(c(400, 700)), waveband(c(700, 1100)),
                    irrad)

## NDI  irrad  [ 400.700 ] - [ 700.1100 ]
##                              0.6352382
```

## 9.9 Transmittance, reflectance, absorptance and absorbance

### 9.9.1 Individual spectra

The functions `transmittance`, `absorptance` and `absorbance` take `filter_spct` as argument, while function `reflectance` takes `reflector_spct` objects as ar-

gument. Functions `transmittance`, `reflectance` and `absorptance` are also implemented for `object_spct`. These functions return as default an average value for these quantities **assuming** a light source with a flat spectral energy output, but this can be changed as described above for `irrad()`.

```
transmittance(polyester.new.spct, list(UVB(), UVA(), PAR()))

##     UVB.ISO     UVA.ISO         PAR
## 0.007671429 0.782682353 0.920245000
## attr(,"Tfr.type")
## [1] "total"
## attr(,"radiation.unit")
## [1] "transmittance average"
```

It is more likely that we would like to calculate these values with reference to light of a certain spectral quality. This needs to be calculated by hand, which is not difficult.

```
irrad(sun.spct * polyester.new.spct, list(UVB(), UVA(), PAR()), wb.trim = TRUE) /
  irrad(sun.spct, list(UVB(), UVA(), PAR()), wb.trim = TRUE)

##   UVB.ISO   UVA.ISO       PAR
## 0.0250655 0.8212786 0.9205990
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

### 9.9.2 Collections of spectra

Here we construct a collection of filter spectra, and then we calculate the transmittance of these filters for two wavebands, obtaining the results as a data frame, with one row per filter, and one column per waveband.

```
filters <-
  filter_mspct(
    list(clear = clear.spct,
         ug1 = ug1.spct, ug11 = ug11.spct,
         gg400 = gg400.spct,
         og550 = og550.spct,
         rg665 = rg665.spct, rg830 = rg830.spct))
transmittance(filters, list(UVA(), VIS()))

##   spct.idx transmittance_UVA.ISO transmittance_VIS.ISO
## 1    clear            1.00000000          1.000000e+00
## 2      ug1            0.65514706          7.671034e-02
## 3     ug11            0.69538235          5.161079e-02
## 4    gg400            0.02923176          9.199580e-01
## 5    og550            0.00001000          5.381038e-01
## 6    rg665            0.00001000          2.404373e-01
## 7    rg830            0.00001000          1.315789e-05
```

## 9.10 Integrated response

### 9.10.1 Individual spectra

The functions `response`, `e_response` and `q_response` take `response_spct` objects as arguments, and return the integrated value for each waveband (integrated over wavelength) **assuming** a light source with a flat spectral energy or photon output respectively. If no waveband is supplied as argument, the whole spectrum is integrated.

```
response(Vital_BW_20.spct)

##    Total
## 20.00984
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy response total"
```

When a waveband, or list of wavebands, is supplied the response is calculated for the wavebands.

```
e_response(Vital_BW_20.spct, list(UVB(), UVA()))

##        UVB.ISO  UVA.ISO.tr.hi
##     18.8436051      0.1532823
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy response total"
```

This function has an additional argument `quantity`, with default `"total"`, as described for `irrad()`.

### 9.10.2 Collections of spectra

```
sensors <-
  response_mspct(
    list(BW20 = Vital_BW_20.spct,
         Berger = Berger_UV_erythemal.spct))
response(sensors, list(UVC(), UVB(), UVA()), quantity = "contribution")

##   spct.idx response_UVC.ISO.tr.lo response_UVB.ISO response_UVA.ISO.tr.hi
## 1     BW20            0.050622869        0.9417168            0.007660344
## 2   Berger            0.003949164        0.9790677            0.016983168
```

## 9.11 Integration over wavelengths

When we need to integrate some *non-standard* `numeric` variable stored in a spectral object we can use functions `integrate_spct` or `average_spct`.

### 9.11.1 Calculation from individual spectra

We can integrate the values of arbitrary `numeric` columns other than `w.length` in an spectral object. All spectral classes are derived from `generic_spct`, so the examples in this section apply to objects of any of the derived spectral classes as well.

```
integrate_spct(sun.spct)

##      e.irrad      q.irrad
## 2.691249e+02 1.255336e-03
```

The function `average_spct` integrates every column holding numeric values from a spectrum object, except for `w.length`, and divides the result by the *spread* or width of the wavelength range integrated, returning a value expressed in the same units as the spectral data.

```
average_spct(sun.spct)

##      e.irrad      q.irrad
## 5.175479e-01 2.414107e-06
```

# 10 Astronomy

## 10.1 Position of the sun

In photobiology research we sometimes need to calculate the position on the sun at arbitrary locations and positions. The function `sun_angles` returns the azimuth in degrees eastwards, altitude in degrees above the horizon, solar disk diameter in degrees and sun to earth distance in astronomical units. The time should be a `POSIXct` vector, possibly of length one, and it is easiest to use package `lubridate` for working with time and dates.

```
sun_angles(now(), lat = 34, lon = 0)

## $time
## [1] "2015-08-23 00:52:33 EEST"
##
## $azimuth
## [1] 320.0229
##
## $elevation
## [1] -34.84111
##
## $diameter
## [1] 0.5271885
##
## $distance
## [1] 1.011403

sun_angles(ymd_hms("2014-01-01 0:0:0", tz = "UTC"))
```

```
## $time
## [1] "2014-01-01 UTC"
##
## $azimuth
## [1] 181.9507
##
## $elevation
## [1] -66.96255
##
## $diameter
## [1] 0.5422513
##
## $distance
## [1] 0.9833078
```

## 10.2 Times of sunrise, solar noon and sunset

Functions `sunrise_time`, `sunset_time`, `noon_time`, `day_length` and `night_length` have all the same parameter signature. In addition, function `day_night` returns a list containing all the quantities returned by the other functions. They are all vectorized for the `date` parameter.

We create a vector of dates to use in the examples—default time zone of `ymd` is UTC or GMT.

```
dates <- seq(from = ymd("2015-03-01"), to = ymd("2015-07-1"), length.out = 3)
```

Default latitude is zero (the Equator), the default longitude is zero (Greenwich), and default time zone for the functions in the `photobiology` package is `"UTC"`. Be also aware that for summer dates the times are expressed accrodingly. In the examples below this can be recognized for example, by the time zone being reported as EEST instead of EET for Eastern Europe.

```
noon_time(dates, tz = "UTC", lat =  60)

## [1] "2015-03-01 12:12:48 UTC" "2015-05-01 11:57:17 UTC"
## [3] "2015-07-01 12:03:44 UTC"

noon_time(dates, tz = "CET", lat =  60)

## [1] "2015-03-01 13:12:48 CET"  "2015-05-01 13:57:17 CEST"
## [3] "2015-07-01 14:03:44 CEST"
```

```
day_night(dates, lat =  60)

## $day
## [1] "2015-03-01" "2015-05-01" "2015-07-01"
##
## $sunrise
## [1] "2015-03-01 07:06:26 UTC" "2015-05-01 04:06:51 UTC"
## [3] "2015-07-01 02:52:50 UTC"
```

```
##
## $noon
## [1] "2015-03-01 12:12:48 UTC" "2015-05-01 11:57:17 UTC"
## [3] "2015-07-01 12:03:44 UTC"
##
## $sunset
## [1] "2015-03-01 17:19:30 UTC" "2015-05-01 19:49:05 UTC"
## [3] "2015-07-01 21:14:09 UTC"
##
## $daylength
## [1] 10.21778 15.70382 18.35536
##
## $nightlength
## [1] 13.782215  8.296180  5.644636
```

The default for `date` is the current day.

```
sunrise_time(lat = 60)

## [1] "2015-08-23 04:39:49 UTC"
```

Both latitude and longitude can be supplied, but be aware that if the returned value is desired in the local time coordinates, the time zone should match the longitude.

```
sunrise_time(today(tzone = "UTC"), tz = "UTC", lat = 60, lon = 0)

## [1] "2015-08-22 04:37:28 UTC"

sunrise_time(today(tzone = "EET"), tz = "EET", lat = 60, lon = 25)

## [1] "2015-08-23 05:59:39 EEST"
```

Southern hemisphere latitudes are given as negative numbers.

```
sunrise_time(dates, lat =  60)

## [1] "2015-03-01 07:06:26 UTC" "2015-05-01 04:06:51 UTC"
## [3] "2015-07-01 02:52:50 UTC"

sunrise_time(dates, lat = -60)

## [1] "2015-03-01 05:18:13 UTC" "2015-05-01 07:47:52 UTC"
## [3] "2015-07-01 09:14:28 UTC"
```

The angle used in the twilight calculation can be supplied, either as the name of a standard definition, or as an angle in degrees (negative for sun positions below the horizon). Positive angles can be used when the time of sun occlusion behind a building, mountain, or other obstacle needs to be calculated.

```
sunrise_time(today(tzone = "EET"), tz = "EET", lat = 60, lon = 25,
             twilight = "civil")

## [1] "2015-08-23 05:04:10 EEST"
```

```
sunrise_time(today(tzone = "EET"), tz = "EET", lat = 60, lon = 25,
             twilight = -10)

## [1] "2015-08-23 04:21:50 EEST"

sunrise_time(today(tzone = "EET"), tz = "EET", lat = 60, lon = 25,
             twilight = +12)

## [1] "2015-08-23 07:38:54 EEST"
```

Parameter `unit.out` can be used to obtain the returned value expresed as time-of-day in hours, minutes, or seconds since midnight.

```
sunrise_time(today(tzone = "EET"), tz = "EET", lat = 60, lon = 25,
             unit.out = "hour")

## [1] 5.994421
```

Functions `day_length` and `night_length` return by default the length of time in hours.

```
day_length(dates, lat = 60)

## [1] 10.21778 15.70382 18.35536

night_length(dates, lat = 60)

## [1] 13.782215  8.296180  5.644636
```

Function `day_night` returns a list.

```
day_night(dates, lat = 60)

## $day
## [1] "2015-03-01" "2015-05-01" "2015-07-01"
##
## $sunrise
## [1] "2015-03-01 07:06:26 UTC" "2015-05-01 04:06:51 UTC"
## [3] "2015-07-01 02:52:50 UTC"
##
## $noon
## [1] "2015-03-01 12:12:48 UTC" "2015-05-01 11:57:17 UTC"
## [3] "2015-07-01 12:03:44 UTC"
##
## $sunset
## [1] "2015-03-01 17:19:30 UTC" "2015-05-01 19:49:05 UTC"
## [3] "2015-07-01 21:14:09 UTC"
##
## $daylength
## [1] 10.21778 15.70382 18.35536
##
## $nightlength
## [1] 13.782215  8.296180  5.644636

day_night(dates, lat = 60, unit.out = "hour")
```

```
## $day
## [1] "2015-03-01" "2015-05-01" "2015-07-01"
##
## $sunrise
## [1] 7.107340 4.114251 2.880713
##
## $noon
## [1] 12.21353 11.95495 12.06228
##
## $sunset
## [1] 17.32512 19.81807 21.23608
##
## $daylength
## [1] 10.21778 15.70382 18.35536
##
## $nightlength
## [1] 13.782215  8.296180  5.644636
```

# 11 RGB colours

Two functions allow calculation of simulated colour of light sources as R colour definitions. Three different functions are available, one for monochromatic light taking as argument wavelength values, and one for polychromatic light taking as argument spectral energy irradiances and the corresponding wave length values. The third function can be used to calculate a representative RGB colour for a band of the spectrum represented as a range of wavelength, based on the assumption of a flat energy irradiance across the range. By default CIE coordinates for *typical* human vision are used, but the functions have a parameter that can be used for supplying a different chromaticity definition.

Examples for monochromatic light:

```
w_length2rgb(550) # green

## wl.550.nm
## "#00FF00"

w_length2rgb(630) # red

## wl.630.nm
## "#FF0000"

w_length2rgb(c(550, 630, 380, 750)) # vectorized

## wl.550.nm wl.630.nm wl.380.nm wl.750.nm
## "#00FF00" "#FF0000" "#000000" "#000000"
```

Examples for wavelength ranges:

```
w_length_range2rgb(c(400,700))

## 400-700 nm
##  "#735B57"
```

Examples for spectra as vectors, in this case for the solar spectrum:

```
with(sun.spct, s_e_irrad2rgb(w.length, s.e.irrad))

## [1] "#544F4B"

with(sun.spct, s_e_irrad2rgb(w.length, s.e.irrad, sens = ciexyzCMF2.spct))

## [1] "#544F4B"
```

Examples with `source_spct` objects.

```
rgb_spct(sun.spct)

## [1] "#544F4B"

rgb_spct(sun.spct, sens = ciexyzCMF2.spct)

## [1] "#544F4B"
```

And also a `color` method for `source_spct`.

```
color(sun.spct)

## source CMF  source CC
##  "#544F4B"  "#B63C37"

color(sun.spct * rg630.spct)

## source CMF  source CC
##  "#4A0000"  "#FF0000"
```

# 12 Optimizing performance

When developing the current version of photobiology quite a lot of effort was spent in optimizing performance, especially of the functions accepting vectors as arguments, as in one of our experiments, we need to process several hundred thousands of measured spectra. The defaults should provide good performance in most cases, however, some further improvements are achievable, when a series of different calculations are done on the same spectrum, or when a series of spectra measured at exactly the same wavelengths are used for calculating weighted irradiances or exposures.

In the case of doing calculations repeatedly on the same spectrum, a small improvement in performance can be achieved by setting the parameter `check.spectrum = FALSE` for all but the first call to `irradiance()`, or `photon_irradiance()`, or `energy_irradiance()`, or the equivalent function for ratios. It is also possible to set this parameter to FALSE in all calls, and do the check beforehand by explicitly calling `check_spectrum()`.

Table 11: Data sets included in the package: spectra. The CIE standard illuminant data in this package are normalized to one at $\lambda = 560\,\text{nm}$, while in the CIE standard they are normalized to 100 at the same wavelength.

| Object | class | units | data description |
|---|---|---|---|
| sun.spct | source_spct | $\text{W m}^{-2}\,\text{nm}^{-1}$ | solar spectral irradiance |
| sun.daily.spct | source_spct | $\text{J m}^{-2}\,\text{d}^{-1}\,\text{nm}^{-1}$ | solar spectral exposure |
| sun.data | data.frame | $\text{W m}^{-2}\,\text{nm}^{-1}$ | solar spectral irradiance |
| sun.daily.data | data.frame | $\text{J m}^{-2}\,\text{d}^{-1}\,\text{nm}^{-1}$ | solar spectral exposure |
| D65.illuminant.spct | source_spct | (norm. 560 nm) | CIE standard |
| A.illuminant.spct | source_spct | (norm. 560 nm) | CIE standard |

Table 12: Data sets included in the package: chromaticity data

| Object | class | data description |
|---|---|---|
| ciexyzCC2.spct | chroma_spct | human chromaticity coordinates 2° |
| ciexyzCC10.spct | chroma_spct | human chromaticity coordinates 10° |
| ciexyzCMF2.spct | chroma_spct | human colour matching function 2° |
| ciexyzCMF10.spct | chroma_spct | human colour matching function 10° |
| ciev2.spct | chroma_spct | human luminous efficiency 2° |
| ciev10.spct | chroma_spct | human luminous efficiency 10° |
| beesxyzCMF.spct | chroma_spct | bee colour matching function |

In the case of calculating weighted irradiances on many spectra having exactly the same wavelength values, then a significant improvement in the performance can be achieved by setting `use.cached.mult = TRUE`, as this reuses the multipliers calculated during successive calls based on the same waveband. However, to achieve this increase in performance, the tests to ensure that the wavelength values have not changed, have to be kept to the minimum. Currently only the length of the wavelength array is checked, and the cached values discarded and recalculated if the length changes. For this reason, this is not the default, and when using caching the user is responsible for making sure that the array of wavelengths has not changed between calls.

You can use the package `microbenchmark` to time the code and find the parts that slow it down. I have used it, and also I have used profiling to optimize the code for speed. The choice of defaults is based on what is best when processing a moderate number of spectra, say less than a few hundreds, as opposed to many thousands.

# 13 Example data

A few example spectra are included in this package for use in examples and vignettes, and testing (Tables 11 and 12).