photobiology Version 0.2.12
User Guide

Pedro J. Aphalo

December 10, 2013

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 especialized packages for quantification of ultraviolet radiation (photobiologyUV), visible radiation (photobiologyVIS), or based on Phytochrome (photobiologyPhy), Cryptochrome (photobiologyCry) (both photoreceptors present in plants), and spectral data for filters (photobiologyFilters). In the future it will be submitted to CRAN (Comprehensive R archive network), it is meanwhile available from `https://bitbucket.org/aphalo/photobiology/downloads`. There is also a public Git repository at `https://bitbucket.org/aphalo/photobiology` from where the source code of the current an earlier versions can be cloned.

2 Installation and use

The functions in the package photobiologyPhy 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(photobiologyUV)
library(photobiologyVIS)
```

3 Spectral data

If our spectral irradiance data is in W m-2 nm-1, and the wavelength in nm, as in the case of the Macam spectroradiometer, the functions will return the effective irradiance in W m-2. In this example we calculated a biologically effective irradiance.

If, for example, the spectral irradiance output by our model of spectrora-diometer is in mW m-2 nm-1, and the wavelengths are in Ångstrom then to obtain the effective irradiance in W m-2we will need to convert the units.

```
energy_irradiance(wavelength/10, irrad/1000)
```

In this example, 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.

If the spectral irradiance is in mW m-2 nm-1then values should be multiplied by 10 to convert them to W m-2 nm-1.

It is very important to make sure that the wavelengths are in nanometers as this is what the functions expect. If the wavelengths are in the wrong units, the action spectra will be wrongly calculated, and the returned value for effective irradiance will be completely wrong.

Here we just use the example data supplied with the package.

4   Defining wavebands

All functions use wavebands as definitions of the range of wave lengths and the spectral weighting function (SWF) to use in the calculations. A few other bits of information may be included to fine-tune calculations. The waveband definifitions 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.

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.

```
my_PAR <- new_waveband(400, 700)

my_CIE_1 <- new_waveband(250, 400, weight = "SWF", SWF.e.fun = CIE.e.fun,
    SWF.norm = 298)
my_CIE_2 <- new_waveband(250, 400, weight = "SWF", SWF.q.fun = CIE.q.fun,
    SWF.norm = 298)
my_CIE_3 <- new_waveband(250, 400, weight = "SWF", SWF.e.fun = CIE.e.fun,
    SWF.q.fun = CIE.q.fun, SWF.norm = 298)
```

5 Calculating irradiance or exposure

There is one basic function for these calculations `irradiance()`, it 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` cab be used to adjust the calculations to expect photon units. The type of unit used for the calculated irradiance (or expossure) is set by the parameter `unit.out` with no default. If no `w.band` parameter is supplied, the whole spectrum spectrum input is used, unweighted, to calculate the total irradiance. If a `w.band` is supplied, then the range of wavelengths specified and SWF if present are used for calculating the irradiance. If the waveband definition does not include a SWF, then the unweighted irradiance is returned, if the definition includes a SWF, then a weighted irradiance is returned.

The functions `photon_irradiance()` and `energy_irradiance()`, just call `irradiance()` with the `unit.out` set to `"photon"` or `"energy"` respectively.

We first load some data to play with.

```
data(sun.data)
attach(sun.data)
```

Then we compare the calculations based on the different wavebands defined in the previous section and the predefined functions in the packages photobiologyVIS and photobiologyUV. The predefined functions have the advantaje of allowing the specification of parameters to modify the `w.band` created. In the example bellow, we use this to set the normalization wavelength.

This is how CIE() is defined in photobiologyUV.

```
CIE <- function(norm = 298) {
    new_waveband(w.low = 250, w.high = 400, weight = "SWF", SWF.e.fun = CIE.e.fun,
        SWF.norm = 298, norm = norm, hinges = c(249.99, 250,
            298, 328, 399.99, 400), wb.name = paste("CIE98",
            as.character(norm), sep = "."))
}
```

Now the example calculations.

```
photon_irradiance(w.length, s.e.irrad, my_PAR)

## [1] 0.0008938

library(photobiologyVIS)
photon_irradiance(w.length, s.e.irrad, PAR())

## [1] 0.0008938

photon_irradiance(w.length, s.e.irrad, my_CIE_1)

## [1] 2.037e-07

photon_irradiance(w.length, s.e.irrad, my_CIE_2)
```

```
## [1] 2.037e-07
```

```
photon_irradiance(w.length, s.e.irrad, my_CIE_3)
```

```
## [1] 2.037e-07
```

```
energy_irradiance(w.length, s.e.irrad, my_CIE_1)
```

```
## [1] 0.08178
```

```
energy_irradiance(w.length, s.e.irrad, my_CIE_2)
```

```
## [1] 0.08178
```

```
energy_irradiance(w.length, s.e.irrad, my_CIE_3)
```

```
## [1] 0.08178
```

```
energy_irradiance(w.length, s.e.irrad, CIE())
```

```
## [1] 0.08178
```

```
energy_irradiance(w.length, s.e.irrad, CIE(298))
```

```
## [1] 0.08178
```

```
energy_irradiance(w.length, s.e.irrad, CIE(300))
```

```
## [1] 0.1261
```

6   Calculating ratios

The function `waveband_ratio()` takes basically the same parameters as 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 'ratio' functions in the photobiology package.

The derived functions are: `photon_ratio()`, `energy_ratio()`, and `photons_energy_ratio`. The packages photobiologyVIS and photobiologyUV use these to define some convenience functions, and here we give and example for a function not yet implemented, but which you may find as a useful example.

If for example we would like to calcualte the ratio between UVB and PAR radiation, we would use either of the following function calls, depending on which type of units we desire.

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

```
## [1] 0.001708
```

```
energy_ratio(w.length, s.e.irrad, UVB(), PAR())
```

```
## [1] 0.00299
```

If we would like to calculate a conversion factor between PPFD (PAR photon irradiance in mol s-1 m-2) and PAR (energy) irradiance (W m-2) for a light source for which we have spectral data we could use the following code.

```
conv.factor <- photons_energy_ratio(w.length, s.e.irrad, PAR())

PPFD.mol.photon <- 0.001
PAR.energy <- PPFD.mol.photon/conv.factor
print(conv.factor)

## [1] 4.544e-06

print(PPFD.mol.photon * 1e+06)

## [1] 1000

print(PAR.energy)

## [1] 220.1
```

7 Calculating weighted spectral irradiances

This calculation is not very frequently used, but it is very instructive to look at spectral data in this way, as it can make apparent the large effect that small measuring errors can have on the estimated effective irradiances or exposures.

We here plot weighted and unweighted irradiances using simulated solar spectral irradiance data.

```
weighted.s.e.irrad <- s.e.irrad * calc_multipliers(w.length, GEN.T())
plot(weighted.s.e.irrad/max(weighted.s.e.irrad) ~ w.length, type="l", col="red",
     xlab="Wavelength (nm)",
     ylab="UnWeighted and GEN.T weighted spectral irradiance (relative units)")
lines(s.e.irrad/max(s.e.irrad) ~ w.length, col="black")
```
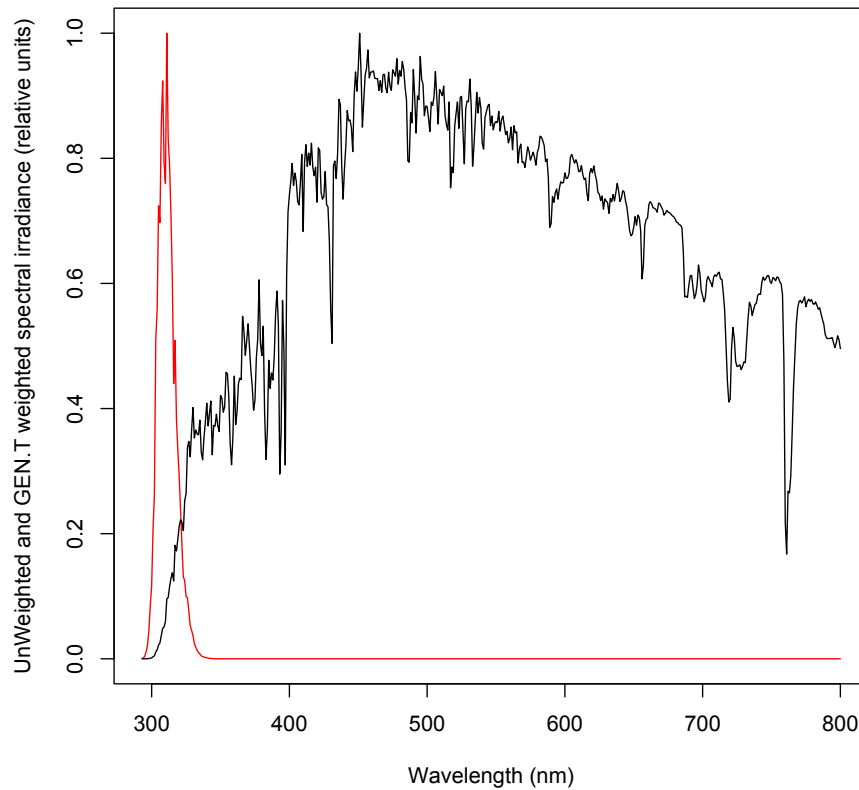
8 Dealing with real 'noisy' spectral data

The first thing to do is to think whether any part of the spectral measurements can be *a priori* known to be equal to zero. For example for the solar spectrum at ground level it is safe to assume that the spectral irradiance is zero for all wavelengths shorter than 293 nm. If the data are noisy, it is best to discard these data before calculating any effective UV doses. This can be easily done as follows if the data are in a data.frame (of course, replacing `w.length` with the name used in your data frame for the wavelengths array:

```
subsetted.sun.data <- subset(sun.data, w.length >= 293)
head(subsetted.sun.data)

##   w.length s.e.irrad s.q.irrad
## 1      293 2.610e-06 6.392e-12
## 2      294 6.142e-06 1.510e-11
## 3      295 2.176e-05 5.366e-11
```

```
## 4       296 6.780e-05 1.678e-10
## 5       297 1.533e-04 3.807e-10
## 6       298 3.670e-04 9.141e-10
```

And then just use the trimmed data in your calculations.

If the data are not in a dataframe, then there are different options: 1) create a data frame from your data, 2) use the function `trim_tails()` from this package, or 3) just use R commands. Here we give an example of the use of `trim_tails()`, and just as an example, we use the same data:

```
trimmed.sun.data <- trim_tails(w.length, s.e.irrad, low = 293)
head(trimmed.sun.data)

##    w.length   s.irrad
## 1      293 2.610e-06
## 2      294 6.142e-06
## 3      295 2.176e-05
## 4      296 6.780e-05
## 5      297 1.533e-04
## 6      298 3.670e-04
```

This function returns a new data.frame and uses always the same variable names for the columns.

```
detach(sun.data)
```

Using real spectral data compared to modelled spectral data for sunlight we show a finally some of the pitfalls, and possible work-arounds, for dealing with noisy measurements, and calculating effective doses from them.

9   Auxiliary function

Sometimes it is needed to add (parallel sum) between two spectra, even if the two spectra have been measured at different wavelengths or wavelength steps. This can happen for example when we would like to calculate the spectrum of a combination of two light sources from the individual spectra. The function `sum_spectra()` can be used in such a case. It's use is described in User Guide of the photobiologyLamps package.

A function `interpolate_spectrum` is also included to facilitate interpolation of spectral values. It is used in the function described in the previous paragraph, but facilitate interpolation. Under the hood it uses R's `spline` function, but it allows more easy control of values to be used for extrapolation.

10   Optimizing performance

When developing the current version of photobiology quite a lot of effort was spent in optimizing performance, as in one of our experiments, we need to process several tens of thoudsands 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 imrovement 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 calling `check_spectrum()`.

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 succesive calls based on the same waveband. However, to achiev 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 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.

The option not to use C++ code is mainly there to be able to test the code and measure the improvement in performance. I see no reason for disabling C++ code, as it does exactly the same calculation but much faster.