R for Photobiology

A handbook

Pedro J. Aphalo,
Andreas Albert
and
Titta Kotilainen

Git: tag ‘none’, committed with hash e26ac79 on 2015-06-11 19:31:11 +0300
by Pedro J. Aphalo
PDF created 16th June 2015

Contents

Preface

This is just a very early draft of a handbook that will accompany the release of the suite of R packages for photobiology (r4photobiology).

Acknowledgements

We thank Stefano Catola, Paula Salonen, David Israel, Neha Rai, Tendry Randriamanana, Saara Harkikainen, Christian Bianchi-Strømme and . . . for very useful comments and suggestions. We specially thank Matt Robson for exercising the packages with huge amounts of spectral data and giving detailed feedback on problems, and in particular for describing needs and proposing new features.

List of abbreviations and symbols

For quantities and units used in photobiology we follow, as much as possible, the recommendations of the Commission Internationale de l'Éclairage as described by (Sliney 2007).

Symbol	**Definition**
α	(%).
Δe	water vapour pressure difference (Pa).
ϵ	emittance ($\mathrm{W\,m^{-2}}$).
λ	wavelength (nm).
θ	solar zenith angle (degrees).
ν	frequency (Hz or $\mathrm{s^{-1}}$).
ρ	(%).
σ	Stefan-Boltzmann constant.
τ	(%).
χ	water vapour content in the air ($\mathrm{g\,m^{-3}}$).
A	(absorbance units).
ANCOVA	analysis of covariance.
ANOVA	analysis of variance.
BSWF	.
c	speed of light in a vacuum.
CCD	charge coupled device, a type of light detector.
CDOM	coloured dissolved organic matter.
CFC	chlorofluorocarbons.
c.i.	confidence interval.
CIE	Commission Internationale de l'Éclairage; or erythemal action spectrum standardized by CIE.
CTC	closed-top chamber.
DAD	diode array detector, linear light detector based on photodiodes.
DBP	dibutylphthalate.
DC	direct current.
DIBP	diisobutylphthalate.
DNA(N)	UV action spectrum for 'naked' DNA.
DNA(P)	UV action spectrum for DNA in plants.
DOM	dissolved organic matter.
DU	Dobson units.
e	water vapour partial pressure (Pa).
E	(energy) irradiance ($\mathrm{W\,m^{-2}}$).
$E(\lambda)$	spectral (energy) irradiance ($\mathrm{W\,m^{-2}\,nm^{-1}}$).

E_0	fluence rate, also called scalar irradiance ($\mathrm{W\,m^{-2}}$).
ESR	early stage researcher.
FACE	free air carbon-dioxide enhancement.
FEL	a certain type of 1000 W incandescent lamp.
FLAV	UV action spectrum for accumulation of flavonoids.
FWHM	full-width half-maximum.
GAW	Global Atmosphere Watch.
GEN	generalized plant action spectrum, also abreviated as GPAS (Caldwell 1971).
GEN(G)	mathematical formulation of GEN by (Green et al. 1974) .
GEN(T)	mathematical formulation of GEN by (Thimijan et al. 1978).
h	Planck's constant.
h'	Planck's constant per mole of photons.
H	exposure, frequently called dose by biologists ($\mathrm{kJ\,m^{-2}\,d^{-1}}$).
H^{BE}	biologically effective (energy) exposure ($\mathrm{kJ\,m^{-2}\,d^{-1}}$).
$H_{\mathrm{p}}^{\mathrm{BE}}$	biologically effective photon exposure ($\mathrm{mol\,m^{-2}\,d^{-1}}$).
HPS	high pressure sodium, a type of discharge lamp.
HSD	honestly signifcant difference.
k_{B}	Boltzmann constant.
L	radiance ($\mathrm{W\,sr^{-1}\,m^{-2}}$).
LAI	leaf area index, the ratio of projected leaf area to the ground area.
LED	light emitting diode.
LME	linear mixed effects (type of statistical model).
LSD	least significant difference.
n	number of replicates (number of experimental units per treatment).
N	total number of experimental units in an experiment.
N_{A}	Avogadro constant (also called Avogadro's number).
NIST	National Institute of Standards and Technology (U.S.A.).
NLME	non-linear mixed effects (statistical model).
OTC	open-top chamber.
PAR	, 400–700 nm.
	measured as energy or photon irradiance.
PC	polycarbonate, a plastic.
PG	UV action spectrum for plant growth.
PHIN	UV action spectrum for photoinhibition of isolated chloroplasts.
PID	(control algorithm).
PMMA	polymethylmethacrylate.
PPFD	, another name for
	PAR photon irradiance (Q_{PAR}).
PTFE	polytetrafluoroethylene.
PVC	polyvinylchloride.
q	energy in one photon ('energy of light').
q'	energy in one mole of photons.
Q	photon irradiance ($\mathrm{mol\,m^{-2}\,s^{-1}}$ or $\mathrm{\mu mol\,m^{-2}\,s^{-1}}$).
$Q(\lambda)$	spectral photon irradiance ($\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$ or $\mathrm{\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}}$).
r_0	distance from sun to earth.
RAF	(nondimensional).
RH	relative humidity (%).
s	energy effectiveness (relative units).

$s(\lambda)$	spectral energy effectiveness (relative units).
s^{p}	quantum effectiveness (relative units).
$s^{\mathrm{p}}(\lambda)$	spectral quantum effectiveness (relative units).
s.d.	standard deviation.
SDK	software development kit.
s.e.	standard error of the mean.
SR	spectroradiometer.
t	time.
T	temperature.
TUV	tropospheric UV.
U	electric potential difference or voltage (e.g. sensor output in V).
UV	ultraviolet radiation (λ = 100–400 nm).
UV-A	ultraviolet-A radiation (λ = 315–400 nm).
UV-B	ultraviolet-B radiation (λ = 280–315 nm).
UV-C	ultraviolet-C radiation (λ = 100–280 nm).
$\mathrm{UV}^{\mathrm{BE}}$	biologically effective UV radiation.
UTC	coordinated universal time, replaces GMT in technical use.
VIS	radiation visible to the human eye ($\approx$ 400–700 nm).
WMO	World Meteorological Organization.
VPD	water vapour pressure deficit (Pa).
WOUDC	World Ozone and Ultraviolet Radiation Data Centre.

Part I

Preliminaries

CHAPTER 1

Introduction

Abstract

In this chapter we explain the physical basis of optics and photochemistry.

1.1 Radiation and molecules

CHAPTER 2

Optics

Abstract

In this chapter we explain how to .

2.1 Task:

CHAPTER 3

Photochemistry

Abstract

In this chapter we explain how to .

3.1 Task:

CHAPTER 4

Software

Abstract

In this chapter we describe the software we used to run the code examples and typeset this handbook, and how to install it.

4.1 Task:

4.2 Introduction

The software used for typesetting this handbook and developing the r4photobiology suite is free and open source. All of it is available for the most common operating systems (Unix including OS X, Linux and its variants, and Windows). It is also possible to run everything described here on a Linux server running the server version of RStudio, and access the server through a web browser.

For just running the examples in the handbook, you would need only to have R installed. That would be enough as long as you also have a text editor available. This is possible, but does not give a very smooth workflow for data analyses which are beyond the very simple. The next stage is to use a text editor which integrates to some extent with R, but still this is not ideal, specially for writing packages or long scrips. Currently the best option is to use the integrated development environment (IDE) called 'RStudio'. This is an editor, but tightly integrated with R. Its advantages are especially noticeable in the case of errors and 'debugging'. We also use a LaTeX for typesetting. Is what we used for the first handbook (Aphalo, Albert, Björn, Ylianttila et al. 2012), and what we routinely for reporting data analyses, and that PJA also uses for all 'overhead' slides he writes for lectures. You, do not need to go this far to be able to profit from R and our suite, but the set up we will describe here, is what we currently use, it is by far the best one we have encountered in 18 years of using and teaching how to use R.

We will not give software installation instructions in this handbook, but will keep a web page with up-to-date instructions. In the following sections we briefly describe the different components of a full and comfortable working environment, but there many alternatives and the only piece that you cannot replace is R itself.

4.3 The different pieces

4.3.1 R

You will not be able to profit from this handbook's 'Cook Book' part, unless you have access to R. R (also called Gnu S) is both the name of a software system, and a dialect of the language S. The language S, although designed with data analysis and statistics in mind, is a computer language that is very powerful in its own way. It allows object oriented programming. Being based in a programming language, and being able to call and being called by programs and subroutine libraries written in several other programming languages, makes it easily extensible.

R has a well defined mechanism for "add-ons" called packages, that are kept in the computer where R is running, in disk folders that conform the library. There is a standard mechanism for installing packages, that works across operating systems (OSs) and computer architectures. There is also a Comprehensive R Archive Network (CRAN) where publicly released versions of packages are kept. Packages can be installed and updated from CRAN and similar repositories directly from within R.

If you are not familiar with R, please, go through the Appendixes **??**, **??**, **??**, and **??**, and/or learn from some of the books listed in Appendix **??**, before delving into our 'Cook Book'.

4.3.2 RStudio

RStudio exists in two versions with identical user interface: a desktop version and a server version. The server version can be used remotely through a web browser. It can be for example run in the 'cloud', for example, as an AWS instance (Amazon Web Services) quite easily and cheaply, or on one's own server hardware.

4.3.3 Version control: Git and Subversion

Version control systems help with keeping track of the history of software development, data analysis, or even manuscript writing. They make it possible for several programmers, data analysts, authors and or editors to work on the same files in parallel and then merge their edits. They also allow easy transfer of whole 'projects' between computers. Git is very popular, and Github and Bitbucket are popular hosts for repositories. Git itself is free software, and can be also run locally, or as one's own private server, either as an AWS instance or on other hosting service, or on your own hardware.

4.3.4 C++ compiler

Although R is an interpreted language, a few functions in our suite are written in C++ to achieve better performance. On OS X and Windows, the normal practice is to install binary packages, which are ready compiled. In other systems like Linux and Unix it is the normal practice to install source packages that are compiled at the time of installation.

4.3.5 LaTeX

LaTeX is built on top of TeX. TeX code and features were 'frozen' (only bugs are fixed) long ago. There are currently a few 'improved' derivatives: pdfTeX, XeTeX, and LuaTeX. Currently the most popular TeX in western countries is pdftex which can directly output PDF files. XeTeX can handle text both written from left to right and right to left, even in the same document, and is the most popular TeX engine in China and other Asian countries.

CHAPTER 5

Photobiology R packages

Abstract

In this chapter we describe the suite of R packages for photobiological calculations ‘r4photobiology’, and explain how to install them.

5.1 The design of the user interface

The design of the ‘high level’ interface is based on the idea of achieving simplicity of use by hiding the computational difficulties and exposing objects, functions and operators that map directly to physical concepts. Computations and plotting of spectral data centers on two types of objects: *spectra* and *wavebands*. Al spectra have in common that all observations are referenced to a wavelength value, there are different types spectral objects, e.g. for light sources and responses. Waveband objects include much more than information about a range of wavelengths, they can also include information about a transformation of the spectral data, like a biological spectral weighting function (BSWF). In addition to functions for calculating summary quantities like irradiance from spectral irradiance, the packages define operators for spectra and wavebands. The use of operators simplifies the syntax and makes the interface easier to use.

```
e_irrad(sun.spct * polyester.new.spc, CIE())
```

Is all what is needed to obtain the CIE98-weigthed energy irradiance simulating the effect of a polyester filter on the example solar spectrum, which of course, can be substituted by other spectral data.

When we say that we hide the computational difficulties what we mean, is that in the example above, the data for the two spectra do not need to be

Table 5.1: Packages in the r4photobiology suite. Packages not yet released are highlighted with a red bullet 🔴, and those at 'beta' stage with a yellow bullet 🟡, those relatively stable with a green bullet 🟢.

	Package	Type	Contents
🟢	photobiology	funs + classes	basic functions, class definitions, class methods and example data
🟢	photobiologyWavebands	definitions	quantification of radiation
🟢	photobiologySun	data	spectral data for solar radiation
🟢	photobiologyLamps	data	spectral data for lamps
🟢	photobiologyLEDs	data	spectral data for LEDs
🟢	photobiologyFilters	data	transmittance data for filters
🟢	photobiologySensors	data	response data for broadband sensors
🟡	photobiologyPlants	funs + data	photoreceptors and optical properties
🟢	photobiologygg	functions	extensions to package `ggplot2`
🔴	rTUV	funs + data	TUV model interface
🟢	rOmniDriver	functions	control of Ocean Optics spectrometers

available at the same wavelengths values, and the BSWF is defined as a function. Interpolation of the spectral data and calculation of weighting factors takes place automatically and invisibly. All functions and operators function without error with spectra with varying (even arbitrarily and randomly varying) wavelength steps. Integration is always used rather than summation for summarizing the spectral data.

There is lower layer of functions, used internally, but also exported, which allow improved performance at the expense of more complex scripts and commands. This user interface is not meant for the casual user, but for the user who has to analyse thousands of spectra and uses scripts for this. For such users performance is the main concern rather than easy of use and easy to remember syntax. Also these functions handle any wavelength mismatch by interpolation before applying operations or functions.

The suite also includes data for the users to try options and ideas, and helper functions for plotting spectra using other R packages available from CRAN, in particular `ggplot2`. There are some packages, not part of the suite itself, for data acquisition from Ocean Optics spectrometers, and application of special calibration and correction procedures to those data. A package will provide an interface to the TUV model to allow easy simulation of the solar spectrum.

5.2 The suite

The suite consists in several packages. The main package is `photobiology` which contains all the generally useful functions, including many used in the other, more specialized, packages (Table 5.1).

Spectral irradiance objects (class `source_spct`) and spectral response/action objects (class `response_spct`) can be constructed using energy- or photon-based data, but this does not affect their behaviour. The same flexibility applies to spectral transmittance vs. spectral absorbance for classes `filter_spct`, `reflector_spct` and `object_spct`.

Although by default low-level functions expect spectral data on energy units, this is just a default that can be changed by setting the parameter `unit.in = "photon"`. Across all data sets and functions wavelength vectors have name `w.length`, spectral (energy) irradiance `s.e.irrad`, photon spectral irradiance `s.q.irrad`[1], absorbance ($\log_{10}$-based) A, transmittance (fraction of one) `Tfr`, transmittance (%) `Tpc`, reflectance (fraction of one) `Rfr`, reflectance (%) `Rpc`, and absorptance (fraction of one) `Afr`.

Wavelengths should always be in nm, and when conversion between energy and photon based units takes place no scaling factor is used (an input in $\mathrm{W\,m^{-2}\,nm^{-1}}$ yields an output in $\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$ rather than $\mathrm{\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}}$).

The suite is still under active development. Even those packages marked as ‘stable’ are likely to acquire new functionality. By stability, we mean that we hope to be able to make most changes backwards compatible, in other words, we hope they will not break existing user code.

5.3 `r4photo` repository

I have created a small repository for the packages. This repository follows the CRAN folder structure, so now package installation can be done using just the normal R commands. This means that dependencies are installed automatically, and that automatic updates are possible. The build most suitable for the current system and R version is also picked automatically if available. It is normally recommended that you do installs and updates on a clean R session (just after starting R or RStudio).For easy installation and updates of packages, the r4photo repository can be added to the list of repositories that R knows about.

Whether you use RStudio or not it is possible to add the r4photo repository to the current session as follows, which will give you a menu of additional repositories to activate:

```r
setRepositories(graphics = getOption("menu.graphics"),
                ind = NULL,
                addURLs = c(r4photo =
                      "http://www.mv.helsinki.fi/aphalo/R"))
```

If you know the indexes in the menu you can use this code, where ‘1’ and ‘6’ are the entries in the menu in the command above.

```r
setRepositories(graphics = getOption("menu.graphics"),
                ind = c(1, 6),
                addURLs = c(r4photo =
                      "http://www.mv.helsinki.fi/aphalo/R"))
```

[1] q derives from ‘quantum’.

Be careful not to issue this command more than once per R session, otherwise the list of repositories gets corrupted by having two repositories with the same name.

Easiest is to create a text file and name it '`.Rprofile`'. The commands above (and any others you would like to run at R start up) should be included, but with the addition that the package names for the functions need to be prepended. The minimum needed is.

```
utils::setRepositories(graphics = getOption("menu.graphics"),
                       ind = c(1, 6),
                       addURLs = c(r4photo =
                             "http://www.mv.helsinki.fi/aphalo/R"))
```

The `.Rprofile` file located in the current folder is sourced at R start up. It is also possible to have such a file affecting all of the user's R sessions, but its location is operating system dependent, it is in most cases the what the OS considers the current user's *HOME* directory or folder (e.g. 'My Documents' in recent versions of MS-Windows). If you are using RStudio, after setting up this file, installation and updating of the packages in the suite can take place exactly as for any other package archived at CRAN.

The commands and examples below can be used at the R prompt and in scripts whether RStudio is used or not.

After adding the repository to the session, it will appear in the menu when executing this command:

```
setRepositories()
```

and can be enabled and disabled.

In RStudio, after adding the r4photo repository as shown above, the photobiology packages can be installed and uninstalled through the normal RStudio menus and dialogues, and will listed after typing the first few characters of their names. For example when you type 'photob' in the packages field, all the packages with names starting with 'photob' will be listed.

They can be also installed at the R command prompt with the following command:

```
install.packages(c("photobiologyAll", "photobiologygg"))
```

and updated with:

```
update.packages()
```

The added repository will persist only during the current R session. Adding it permanently requires editing the R configuration file, as discussed above. Take into consideration that .Rprofile is read by R itself, and will take effect whether you use RStudio or not. It is possible to have a user wide .Rprofile file, and a different one on those folders needing different settings. There many options that can be modified by means of commands in the .Rprofile file.

5.4 How to install the packages

The examples given in this page assume that ‘r4photo’ is not in the list of repositories known to the current R session. See the section 5.3 on the r4photo repository for a better alternative to the approach given here. We mention these other commands because they may be useful in cases when the user does not have write access to his/hers home directory, or just wants to try the packages.

To install the latest version of one package (photobiology used as example) you just need to indicate the repository. However this simple command will only install the dependencies between the different photobiology packages.

```
install.packages("photobiology",
                 repos = "http://www.mv.helsinki.fi/aphalo/R")
```

To update what is already installed, this command is enough (even if the packages have been installed manually before):

```
update.packages(repos = "http://www.mv.helsinki.fi/aphalo/R")
```

The best way to install the packages is to specify both the r4photo repository and a normal CRAN repository, then all dependencies will be automatically installed. The package photobiolgyAll just loads and imports all the packages in the suite, except for photobiolygg. Because of this dependency all the packages are installed unless already installed by issuing this command.

```
install.packages(c("photobiologyAll", "photobiologygg"),
         repos = c(r4photo =
                     "http://www.mv.helsinki.fi/aphalo/R",
                   CRAN =
                     "http://cran.rstudio.com"))
```

This example also shows how one can use an array of package names (in this example all currently available “photobiology” packages) in the call to the function install.packages, this is useful if you want to install only a subset of the files, or if you want to make sure that any older install of the packages is overwritten:

```
photobiology_packages <- c("photobiology",
    "photobiologyWavebands",
    "photobiologyCry", "photobiologyPhy",
    "photobiologyLamps", "photobiologyLEDs",
    "photobiologySun", "photobiologygg",
    "photobiologyFilters",  "photobiologySensors")

install.packages(photobiology_packages,
         repos = c(r4photo =
                     "http://www.mv.helsinki.fi/aphalo/R",
                   CRAN =
                     "http://cran.rstudio.com"))
```

The commands above install all packages in the suite and all their dependencies from CRAN if needed. The following command will update all the packages currently installed (if new versions are available) and install any new dependencies.

```
update.packages(repos =
                  c(r4photo =
                      "http://www.mv.helsinki.fi/aphalo/R",
                    CRAN =
                      "http://cran.rstudio.com"))
```

The instructions above should work under Windows as long as you have a supported version of R (3.0.0 or later) because I have built suitable binaries, under other OSs you may need to add type="source" unless this is already the default. We will try to build OS X binaries for Mac so that installation is easier. Meanwhile if installation fails try adding type="source" to the commands given above. For example the first one would become:

```
install.packages("photobiology",
                 repos = "http://www.mv.helsinki.fi/aphalo/R",
                 type="source")
```

When using type=”source” you may need to install some dependencies like the splus2R package beforehand from CRAN if building it from sources fails.

Part II

Cookbook of calculations

CHAPTER 6

Storing data

Abstract

In this chapter we describe the objects used to store data and functions and operators for basic operations. We also give some examples of operating on these objects and their components using normal R functions and operators.

6.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)

## Loading required package:  data.table

library(photobiologyFilters)
library(photobiologyWavebands)
```

6.2 Introduction

The suite uses object-oriented programming for its higher level ‘user-friendly’ syntax. Objects are implemented using “S3” classes. The two main distinct kinds of objects are different types of spectra, and wavebands. Spectral objects contain, as their name implies, spectral data. Wavebands contain the information needed to calculate irradiance, non-weighted or weighted (effective), and a name and a label to be used in output printing. Functions and operators are defined for operations on these objects, alone and in combination. We will first describe spectra, and then wavebands, in each case describing operators and functions.

Table 6.1: Classes for spectral data and *mandatory* variable and attribute names

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	

6.3 Spectra

6.3.1 How are spectra stored?

For spectra the classes are a specialization of `data.table` which are in turn a specialization of `data.frame`. This means that they are *mostly* compatible with functions that operate on these classes.

The suite defines a `generic_spct` class, from which other specialized classes, '`filter_spct`, `reflector_spct`, `object_spct`, `source_spct`, `response_spct`, `response_spct`, `chroma_spct` and `cps_spct` are derived. Having this class structure allows us to create special methods and operators, which use the same 'names' than the generic ones defined by R itself, but take into account the special properties of spectra.

Except for *special* cases each spectral object holds only spectral data from a single measurement. When spectral data from more than one measurement is contained in a single object, the data for the different measurements are stored *melted*, in other words, in the same variable(s), and distinguished by means of an index factor. When a single measurement consists in several different quantities being measured, then these are stored in different variables, or columns, in the same spectral object. The name used for variables containing spectral data for a given quantity have mandatory names, and are always stored using the same units. Spectral objects also carry additional information in attributes, such a text comment, sorting key, time unit used for expression, and additional attributes indicating properties such as whether reflectance is **specular** or **total**. These strict rules allow the functions in the package to handle unit conversions, and units in labels and plots automatically. It also allows the use of operators like ('+') with spectra, and some sanity checks on the supplied spectral data and restriction of *some* invalid operations. Table 6.1 lists the mandatory names of variables and attributes for each of the classes. In Table 6.2 for each mandatory variable name, plus the additional names recognized by constructors are listed together with the respective units. Additional columns are allowed in the spectral objects, and deleted or set to NA only when the meaning of an operation on the whole spectrum is for these columns ambiguous.

Table 6.2: 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	$n\,\mathrm{s}^{-1}$	
s.e.irrad	$\mathrm{W\,m^{-2}\,nm^{-1}}$	time.unit = "second"
s.e.irrad	$\mathrm{J\,m^{-2}\,d^{-1}\,nm^{-1}}$	time.unit = "day"
s.e.irrad	varies	time.unit = *duration*
s.q.irrad	$\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$	time.unit = "second"
s.q.irrad	$\mathrm{mol\,m^{-2}\,d^{-1}\,nm^{-1}}$	time.unit = "day"
s.q.irrad	$\mathrm{mol\,m^{-2}\,nm^{-1}}$	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	$x\,\mathrm{J^{-1}\,s^{-1}\,nm^{-1}}$	time.unit = "second"
s.e.response	$x\,\mathrm{J^{-1}\,d^{-1}\,nm^{-1}}$	time.unit = "day"
s.e.response	$x\,\mathrm{J^{-1}\,nm^{-1}}$	time.unit = "exposure"
s.e.response	varies	time.unit = *duration*
s.q.response	$x\,\mathrm{mol^{-1}\,s^{-1}\,nm^{-1}}$	time.unit = "second"
s.q.response	$x\,\mathrm{mol^{-1}\,d^{-1}\,nm^{-1}}$	time.unit = "day"
s.q.response	$x\,\mathrm{mol^{-1}\,nm^{-1}}$	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	$n\,\mathrm{s}^{-1}$	

6.3.2 Task: Create a spectral object from numeric vectors

'Traditional' constructor functions are available, and possibly easiest to use to those used R programming style. Constructor functions have the same name as the classes (e.g. `source_spct`). The constructor functions accept numeric vectors as arguments, and these can be "renamed" on the fly. The object is checked for consistency and within-range data, and missing required components are set to NA. We use `source_spct` in the examples but similar functions are defined for all the classes spectral objects.

We can create a new object of class `source_spct` from two `numeric` vectors, and as shown below, recycling applies.

```
source_spct(w.length = 300:500, s.e.irrad = 1)

##      w.length s.e.irrad
##   1:      300         1
##   2:      301         1
##  ---
## 200:      499         1
## 201:      500         1
```

The code above uses defaults for all attributes, and assumes that spectral energy irradiance is expressed in $\mathrm{W\,m^{-2}\,nm^{-1}}$. As elsewhere in the package, wavelengths should be expressed in nanometres. If our spectral data is in photon-based units with spectral photon irradiance expressed in $\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$ the code becomes:

```
source_spct(w.length = 300:500, s.q.irrad = 1)

##      w.length s.q.irrad
##   1:      300         1
##   2:      301         1
##  ---
## 200:      499         1
## 201:      500         1
```

Spectral objects have attributes, which store additional information needed for correct handling of units of expression, printing and plotting. The defaults need frequently to be changed, for example when spectral exposure is expressed as a daily integral, or other arbitrary exposure time. This length of time or `duration` should be set, whenever the unit of time used is different to second.

```
source_spct(w.length = 300:500, s.q.irrad = 1, time.unit = "day")

##      w.length s.q.irrad
##   1:      300         1
##   2:      301         1
##  ---
## 200:      499         1
## 201:      500         1
```

In addition to the character strings `"second"`, `"hour"`, and `"day"`, any object belonging to the class `duration` defined in package `lubridate` can be used. This means, that any arbitrary time duration can be used.

Please, see Tables 6.1 and 6.2 for the attributes defined for the different classes of spectral objects.

6.3.3 Task: Create a spectral object from a data frame

'Traditional' conversion functions with names given by names of classes preceded by `as.` (e.g. `as.source_spct`. These functions accept data frames, data tables, and lists with components of equal length as arguments. These functions are less flexible, as the component variables in the argument should be named using one of the names recognized. Table **??** lists the different 'names' understood by these constructor functions and the required and optional components of the different spectral object classes. The object is checked for consistency and within-range data, and missing required components are set to NA. We use `source_spct` in the examples but similar functions are defined for all the classes spectral objects.

We can create a new object of class `source_spct` by making a copy of a data frame or a data table and converting it into a source spectrum:

```
my.df <- data.frame(w.length = 300:500, s.e.irrad = 1)
my.spct <- as.source_spct(my.df)
```

In this case `sun.data` remains independent, and whatever change we make to my.spct does not affect `sun.data`:

```
my.spct <- my.spct * 2
my.spct

##      w.length s.e.irrad
##   1:      300         2
##   2:      301         2
##  ---
## 200:      499         2
## 201:      500         2

head(my.df)

##   w.length s.e.irrad
## 1      300         1
## 2      301         1
## 3      302         1
## 4      303         1
## 5      304         1
## 6      305         1
```

These functions, in the same way as constructors, accept attributes as arguments.

Using a technical term, we have used a copy constructor, which is the normal behaviour in R.

6.3.4 Task: Convert a data frame into a spectral object

The last possibility, is to use a syntax that is unusual for R, but which is some settings will lead to faster execution: convert an existing data frame or data table, *in situ* or by reference, into a source_spct object. These set functions have the same semantics as setDT and setDF from package data.table. Table ?? lists the different 'names' understood by these conversion functions and the required and optional components of the different spectral object classes. The object is checked for consistency and within-range data, and missing required components are set to NA. We use source_spct in the examples but similar functions are defined for all the classes spectral objects.

```
my.ref.spct <- setSourceSpct(my.df)
is.source_spct(my.df)

## [1] TRUE
```

If we combine the above code with an assignment, we end up having two names for the *same* object, which has been converted to a source_spct object. (At the moment we get a copy!)

```
my.ref.spct <- my.ref.spct * 2
my.ref.spct

##      w.length s.e.irrad
##   1:      300         2
##   2:      301         2
##  ---
## 200:      499         2
## 201:      500         2

my.df

##      w.length s.e.irrad
##   1:      300         1
##   2:      301         1
##  ---
## 200:      499         1
## 201:      500         1
```

In fact, the assignment is unnecessary, as the class of my.df is set:

```
setSourceSpct(my.df)
```

These functions, in the same way as constructors, accept attributes as arguments.

Using a technical term, we have converted an object by *reference*, which is *not* the normal behaviour in R.[1]

6.3.5 Task: trimming a spectrum

This is basically a subsetting operation, but our functions operate only based on wavelengths, while R subset is more general. On the other hand, our functions trim_spct and trim_tails add a few 'bells and whistles'. The

[1]Avoiding copying can improve performance for huge objects, but will rarely make a tangible difference for individual spectra of moderate size.

trimming is based on wavelengths and by default the cut points are inserted by interpolation, so that the spectrum returned includes the limits given as arguments. In addition, by default the trimming is done by deleting both spectral irradiance and wavelength values outside the range delimited by the limits (just like `subset` does), but through parameter `fill` the values outside the limits can be replaced by any value desired (most commonly NA or 0.) It is possible to supply a only one, or both of `low.limit` and `high.limit`, depending on the desired trimming, or use a `waveband` definition or a numeric vector as an argument for `range`. If the limits are outside the original data set, then the output spectrum is expanded and the tails filled with the value given as argument for `fill` unless `fill` is equal to NA, which is the default.

```
trim_spct(sun.spct, range = UV())

## Warning in trim_spct(sun.spct, range = UV()):  Not trimming
short end as low.limit is outside spectral data range.

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 107:      399 5.861190e-01 1.954901e-06
## 108:      400 6.081049e-01 2.033314e-06

trim_spct(sun.spct, range = UV(), fill = 0)

##      w.length s.e.irrad s.q.irrad
##   1:      100         0         0
##   2:      101         0         0
##  ---
## 701:      799         0         0
## 702:      800         0         0

trim_spct(sun.spct, low.limit = 400)

##      w.length s.e.irrad    s.q.irrad
##   1:      400 0.6081049 2.033314e-06
##   2:      401 0.6261742 2.098967e-06
##  ---
## 400:      799 0.4185850 2.795738e-06
## 401:      800 0.4069055 2.721132e-06

trim_spct(sun.spct, low.limit = 250, fill = 0.0)

##      w.length s.e.irrad    s.q.irrad
##   1:      250 0.0000000 0.000000e+00
##   2:      251 0.0000000 0.000000e+00
##  ---
## 550:      799 0.4185850 2.795738e-06
## 551:      800 0.4069055 2.721132e-06

trim_spct(sun.spct, range = c(300, 400, 500, 600))

##      w.length   s.e.irrad    s.q.irrad
##   1:      300 0.001264554 3.171207e-09
##   2:      301 0.002623718 6.601607e-09
##  ---
## 300:      599 0.624741526 3.128191e-06
## 301:      600 0.637276746 3.196284e-06
```

If the limits are outside the range of the input spectral data, and `fill` is set to a value other than NULL the output is expanded up to the limits and filled.

```
trim_spct(sun.spct, range=c(300, 1000))

## Warning in trim_spct(sun.spct, range = c(300, 1000)):  Not
trimming long end as high.limit is outside spectral data range.

##      w.length   s.e.irrad    s.q.irrad
##   1:      300 0.001264554 3.171207e-09
##   2:      301 0.002623718 6.601607e-09
##  ---
## 500:      799 0.418584959 2.795738e-06
## 501:      800 0.406905530 2.721132e-06

trim_spct(sun.spct, range=c(300, 1000), fill = NA)

##      w.length s.e.irrad s.q.irrad
##   1:      293        NA        NA
##   2:      294        NA        NA
##  ---
## 708:      999        NA        NA
## 709:     1000        NA        NA

trim_spct(sun.spct, range=c(300, 1000), fill = 0.0)

##      w.length s.e.irrad s.q.irrad
##   1:      293         0         0
##   2:      294         0         0
##  ---
## 708:      999         0         0
## 709:     1000         0         0
```

Function `trim_tails` can be used for trimming spectra when data is available as vectors.

6.3.6 Task: interpolating a spectrum

Functions `interpolate_spct` and `interpolate_spectrum` allow interpolation to different wavelength values. `interpolate_spectrum` is used internally, and accepts spectral data measured at arbitrary wavelengths. Raw data from array spectrometers is not available with a constant wavelength step. It is always best to do any interpolation as late as possible in the data analysis.

In this example we generate interpolated data for the range 280 nm to 300 nm at 1 nm steps, by default output values outside the wavelength range of the input are set to NAs unless a different argument is provided for parameter `fill`:

```
interpolate_spct(sun.spct, seq(290, 300, by=0.1))

##      w.length   s.e.irrad    s.q.irrad
##   1:    290.0          NA           NA
##   2:    290.1          NA           NA
##  ---
## 100:    299.9 0.001216553 3.050176e-09
## 101:    300.0 0.001264554 3.171207e-09

interpolate_spct(sun.spct, seq(290, 300, by=0.1), fill=0.0)

##      w.length   s.e.irrad    s.q.irrad
##   1:    290.0 0.000000000 0.000000e+00
##   2:    290.1 0.000000000 0.000000e+00
##  ---
## 100:    299.9 0.001216553 3.050176e-09
## 101:    300.0 0.001264554 3.171207e-09
```

`interpolate_spct` accepts any spectral object, and returns an object of the same type as its input.

```
interpolate_spct(polyester.new.spct, seq(290, 300, by=0.1))

##      w.length   Tfr
##   1:    290.0 0.004
##   2:    290.1 0.004
##  ---
## 100:    299.9 0.003
## 101:    300.0 0.003
```

Function `interpolate_spectrum` takes numeric vectors as arguments, but is otherwise functionally equivalent.

These functions, in their current implementation, always return interpolated values, even when the density of wavelengths in the output is less than that in the input. A future version of the package will include a `smooth_spectrum` function, and possibly a `remap_w.length` function that will automatically choose between interpolation and smoothing/averaging as needed.

6.4 Internal-use functions

The generic function `check` can be used on `generic_spct` objects (i.e. any spectral object), and depending on their class it checks that the required components are present, and in some cases whether they are within the expected range. If they are missing they are added. If it is possible to calculate the missing values from other optional components, they are calculated, otherwise they are filled with NA. It is used internally during the creation of spectral objects.

The function `check_spectrum` may need to be called by the user if he/she disables automatic sanity checking to increase calculation speed.

The function `insert_hinges` is used internally to insert individual interpolated values to the spectra when needed to reduce errors in calculations.

6.5 Wavebands

6.5.1 How are wavebands stored?

Wavebands are derived from R lists. All valid R operations for lists can be also used with `waveband` objects. However, there are `waveband`-specific specializations of some generic R methods as described in Chapter 7 and Chapter 9.

6.5.2 Task: Create waveband objects

Wavebands are created by means of function `waveband` which have in addition to the parameter(s) giving the wavelength range, additional arguments with default values.

The simplest `waveband` creation call is one supplying as argument just any R object for which the `range` function returns the wavelength limits of the desired band in nanometres. Such a call yields a `waveband` object defining an un-weighted range of wavelengths.

Any numeric vector of at least two elements, any spectral object or any existing `waveband` object for which a `range` method exists is valid input, as long as the values can be interpreted as wavelengths in nanometres.

```
waveband(c(300, 400))

## range.300.400
## low (nm) 300
## high (nm) 400
## weighted none

waveband(sun.spct)

## Total
## low (nm) 293
## high (nm) 800
## weighted none

waveband(c(400, 300))

## range.300.400
## low (nm) 300
## high (nm) 400
## weighted none
```

As you can see above, a name and label are created automatically for the new `waveband`. The user can also supply these as arguments, but must be careful not to duplicate existing names[2].

```
waveband(c(300, 400), wb.name="a.name")

## a.name
```

[2]It is preferable that `wb.name` complies with the requirements for R object names and file names, while labels have fewer restrictions as they are meant to be used only for output text labels.

```
## low (nm) 300
## high (nm) 400
## weighted none
```

```
waveband(c(300, 400), wb.name="a.name", wb.label="A nice name")

## a.name
## low (nm) 300
## high (nm) 400
## weighted none
```

See chapter 10 on page 63, in particular sections 10.4, 10.3, and 10.5 for further examples, and a more in-depth discussion of the creation and use of *un-weighted* waveband objects.

For both functions, even if we supply a *weighting function* (SWF), a lot of flexibility remains. One can supply either a function that takes energy irradiance as input or a function that takes photon irradiance as input. Unless both are supplied, the missing function will be automatically created. There are also arguments related to normalization, both of the output, and of the SWF supplied as argument. In the examples above, 'hinges' are created automatically for the range extremes. When using SWF with discontinuous derivatives, best results are obtained by explicitly supplying the hinges to be used as an argument to the waveband call. An example follows for the definition of a waveband for the CIE98 SWF—the function CIE_e_fun is defined in package photobiologyWavebands but any R function taking a numeric vector of wavelengths as input and returning a numeric vector of the same length containing weights can be used.

```
waveband(c(250, 400),
         weight="SWF", SWF.e.fun=CIE_e_fun, SWF.norm=298,
         norm=298, hinges=c(249.99, 250, 298, 328, 399.99, 400),
         wb.name="CIE98.298", wb.label="CIE98")

## CIE98.298
## low (nm) 250
## high (nm) 400
## weighted SWF
## normalized at 298 nm
```

See chapter 11 on page 83, in particular sections **??**, **??**, and **??** for further examples, and a more in-depth discussion of the creation and use of *weighted* waveband objects.

6.5.3 Task: trimming a waveband

This operation either changes the boundaries of waveband objects, or deletes waveband objects from a list of waveband. The first argument can be either a waveband object or a list of waveband objects. Those wavebands fully outside the limits are always discarded and those fully within the limits always kept. In the case of those wavebands crossing a limit, if the argument trim is set to FALSE, they are discarded, but if trim is set to TRUE their boundary is moved to be at the trimming limit. Trimming is based on wavelengths and by default the cut points are inserted. Trimming is done by shrinking the waveband, expansion is not possible. During trimming labels stored in the `waveband` object are 'edited' to reflect the altered boundaries. Trimming does not affect weighting functions stored within the waveband.

```
trim_waveband(UV(), range = UVB())

## [[1]]
## UV.ISO.tr.lo.hi
## low (nm) 280
## high (nm) 315
## weighted none

trim_waveband(VIS_bands(), low.limit = 400, trim = FALSE)

## [[1]]
## Blue.ISO
## low (nm) 450
## high (nm) 500
## weighted none
##
## [[2]]
## Green.ISO
## low (nm) 500
## high (nm) 570
## weighted none
##
## [[3]]
## Yellow.ISO
## low (nm) 570
## high (nm) 591
## weighted none
##
## [[4]]
## Orange.ISO
## low (nm) 591
## high (nm) 610
## weighted none
##
## [[5]]
## Red.ISO
## low (nm) 610
## high (nm) 760
## weighted none

trim_waveband(VIS_bands(), low.limit = 400, trim = TRUE)

## [[1]]
## Purple.ISO.tr.lo
## low (nm) 400
## high (nm) 450
## weighted none
##
## [[2]]
## Blue.ISO
## low (nm) 450
## high (nm) 500
## weighted none
```

```
##
## [[3]]
## Green.ISO
## low (nm) 500
## high (nm) 570
## weighted none
##
## [[4]]
## Yellow.ISO
## low (nm) 570
## high (nm) 591
## weighted none
##
## [[5]]
## Orange.ISO
## low (nm) 591
## high (nm) 610
## weighted none
##
## [[6]]
## Red.ISO
## low (nm) 610
## high (nm) 760
## weighted none

trim_waveband(VIS_bands(), range = c(500, 600))

## [[1]]
## Green.ISO
## low (nm) 500
## high (nm) 570
## weighted none
##
## [[2]]
## Yellow.ISO
## low (nm) 570
## high (nm) 591
## weighted none
##
## [[3]]
## Orange.ISO.tr.hi
## low (nm) 591
## high (nm) 600
## weighted none
```

```
try(detach(package:photobiologyWavebands))
try(detach(package:photobiologyFilters))
try(detach(package:photobiology))
```

CHAPTER 7

Math operators and functions

Abstract

In this chapter we describe math functions and operators for spectra and wavebands. Many of these are specializations of the generic operators and functions existing in R.

7.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)
library(photobiologyWavebands)
library(photobiologyFilters)
```

7.2 Introduction

The suite uses object-oriented programming for its higher level 'user-friendly' syntax. Objects are implemented using "S3" classes. The two main distinct kinds of objects are different types of spectra, and wavebands. Spectral objects contain, as their name implies, spectral data. Wavebands contain the information needed to calculate summaries integrating a range of wavelengths, or for convoluting spectral data with a weighting function. In this chapter we do not describe functions for calculating such summaries, but instead we describe the use of the usual math operators and functions with spectra and wavebands.

Table 7.1: 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

7.3 Operators and operations between two spectra

All operations with spectral objects affect only the required components listed in Table 7.1, redundant components are always deleted[1], while unrecognized components, including all factors and character variables, are preserved only when one of the operands is a numeric vector of any length. There will be seldom need to add numerical components to spectral objects, and the user should take into account that the paradigm of the suite is that data from each spectral measurement is stored as a separate object. However, it is allowed, and possibly useful to have factors as components with levels identifying different bands, or color vectors with RGB values. Such ancillary information is useful for presentation and plotting and can be added with functions described in Chapter ??. Exceptionally, objects can contain spectral data from several measurements and an additional factor indexing them. Such objects cannot be directly used with operators and summary functions, but can be a convenient format for storing related spectra.

[1]e.g. equivalent quantities expressed in different types of units, such as spectral energy irradiance and spectral photon irradiance

Binary maths operators (+, -, *, /), and unary math operators (+, -) are defined for spectral objects as well functions (`log`, `log10`,`sqrt`). Using operators is an easy and familiar way of doing calculations, but operators are rather inflexible (they can take at most two arguments, the operands) and performance is usually slower than with functions with additional parameters that allow optimizing the algorithm. Which operations are legal between different combinations of operands depends on the laws of Physics, but in cases in which exceptions might exist, they are allowed. This means that some mistakes can be prevented, but other may happen either with a warning or silently. So, although a class system provides a safer environment for calculations, it is not able to detect all possible 'nonsensical' calculations. The user must be aware that sanity checks and good understanding of the algorithms are still a prerequisite for reliable results.

Table **??** list the available operators and the operands accepted as legal, together with the class of the objects returned. Only in extreme cases errors will be triggered, in most cases when errors occur an operation between two `reflector_spct` yields a `reflector_spct` object, and operations between a `filter_spct` object and a `source_spct`, between a `reflector_spct` and a `source_spct`, or between two `source_spct` objects yield `source_spct` objects. The object returned contains data only for the overlapping region of wavelengths. The objects do NOT need to have values at the same wavelengths, as interpolation is handled transparently. All four basic maths operations are supported with any combination of spectra, and the user is responsible for deciding which calculations make sense and which not. Operations can be concatenated and combined. The unary negation operator is also implemented.

We can convolute the emission spectrum of a light source and the transmittance spectrum of a filter by simply multiplying them.

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

##      w.length    s.e.irrad
##   1:      293 7.828995e-09
##   2:      294 1.842720e-08
##  ---
## 507:      799 3.809123e-01
## 508:      800 3.706909e-01
```

7.4 Operators and operations between a spectrum and a numeric vector

The same four basic math operators plus power ('^') are defined for operations between a spectrum and a numeric vector, possibly of length one. Recycling rules apply for the numeric vector. Normal R type conversions also take place, so a logical vector can substitute for a numeric one´. These operations do not alter `w.length`, just the other *required* components such as spectral irradiance and transmittance. The optional components are deleted as they can be recalculated if needed. Unrecognized 'user' components are left unchanged.

For example we can divide a spectrum by a numeric value (a vector of length 1, which gets recycle). The value returned is a spectral object of the same type as the spectral argument.

```
sun.spct / 2

##      w.length    s.e.irrad
##   1:       293 1.304833e-06
##   2:       294 3.071200e-06
##  ---
## 507:       799 2.092925e-01
## 508:       800 2.034528e-01

2 * sun.spct

##      w.length    s.e.irrad
##   1:       293 5.219330e-06
##   2:       294 1.228480e-05
##  ---
## 507:       799 8.371699e-01
## 508:       800 8.138111e-01

sun.spct * 2

##      w.length    s.e.irrad
##   1:       293 5.219330e-06
##   2:       294 1.228480e-05
##  ---
## 507:       799 8.371699e-01
## 508:       800 8.138111e-01
```

7.5 Math functions taking a spectrum as argument

Logarithms (`log, log10`), square root (`sqrt`) and exponentiation (`exp`) are defined for spectra. These functions are not applied on `w.length`, but instead to the other mandatory component `s.e.irrad`, `Rfr` or `Tfr`. Any optional numeric components are discarded. Other user-supplied components remain unchanged.

```
log10(sun.spct)

##      w.length  s.e.irrad
##   1:       293 -5.5834152
##   2:       294 -5.2116619
##  ---
## 507:       799 -0.3782164
## 508:       800 -0.3905064
```

7.6 Task: Simulating spectral irradiance under a filter

Package `phobiologyFilters` makes available many different filter spectra, from which we choose Schott filter GG400. Package `photobiology` makes available one example solar spectrum. Using these data we will simulate the filtered solar spectrum.

```
filtered_sun.spct <- sun.spct * gg400.spct
filtered_sun.spct

##      w.length    s.e.irrad
##   1:      293 2.609665e-11
##   2:      294 6.142401e-11
##  ---
## 507:      799 4.060274e-01
## 508:      800 3.946984e-01
```

The GG440 data is for internal transmittance, consequently the results above would be close to the truth only for filters treated with an anti-reflexion multicoating. Let's assume a filter with 9% reflectance across all wavelengths (a coarse approximation for uncoated glass):

```
filtered_uncoated_sun.spct <- sun.spct * gg400.spct * (100 - 9) / 100
filtered_uncoated_sun.spct

##      w.length    s.e.irrad
##   1:      293 2.374795e-11
##   2:      294 5.589585e-11
##  ---
## 507:      799 3.694849e-01
## 508:      800 3.591755e-01
```

Calculations related to filters will be explained in detail in chapter **??**. This is just an example of how the operators work, even when, as in this example, the wavelength values do not coincide between the two spectra.

7.7 Task: Uniform scaling of a spectrum

As noted above operators are available for `generic_spct`, `source_spct`, `filter_spct` and `reflector_spct` objects, and 'recycling' takes place when needed:

```
sun.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06

sun.spct * 2

##      w.length    s.e.irrad
##   1:      293 5.219330e-06
##   2:      294 1.228480e-05
##  ---
## 507:      799 8.371699e-01
## 508:      800 8.138111e-01
```

All four basic binary operators (`+`, `-`, `*`, `/`) can be used in the same way. By default all calculations are done using energy based units, and only values in these units returned. If the operands need conversion, they are silently converted before applying the operator. The default behaviour can be switched into doing operations and returning values in photon-based units by setting an R option, using the normal R `options` mechanism.

7.7.1 Task: Arithmetic operations within one spectrum

As spectral objects behave in many respects as data frames it is possible to do calculations involving columns as usual, e.g. using `with` or explicit selectors. A non-nonsensical example follows using R syntax on a data frame, returning a vector.

Using data frame syntax on a data frame, data table or spectral object, returning a vector:

```
# not run
sun.spct$s.e.irrad^2 / sun.spct$w.length
```

```
# not run
with(sun.spct, s.e.irrad^2 / w.length)
```

As spectral objects are derived from `data.table`, it is also possible to use data table syntax, returning a vector:

```
# not run
sun.spct[ , s.e.irrad^2 / w.length]
```

or, adding the result to the spectral object—in this case we need to first copy `sun.spct`, because it is protected because it is part of a package:

```
# run
my_sun.spct <- copy(sun.spct)
my_sun.spct[ , result := s.e.irrad^2 / w.length]

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06
##            result
##   1: 2.324352e-14
##   2: 1.283302e-13
##  ---
## 507: 2.192908e-04
## 508: 2.069651e-04

my_sun.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:       799 4.185850e-01 2.795738e-06
## 508:       800 4.069055e-01 2.721132e-06
##             result
##   1: 2.324352e-14
##   2: 1.283302e-13
##  ---
## 507: 2.192908e-04
## 508: 2.069651e-04
```

7.7.2 Task: Using operators on underlying vectors

If data for two spectra are available for the same wavelength values, then we can simply use the built in R math operators on the component vectors. These operators are vectorized, which means that an addition between two vectors adds the elements at the same index position in the two vectors with data, in this case for two different spectra. However, there is no internal check that the wavelength values agree at each element. So, although such code may execute faster, it is not recommended in normal use.

Operations using built-in R operators cannot be used if the wavelengths in two spectral data sets are not matched. In this situation functions and operators defined in package `photobiology` come to the rescue by transparently making the two operand spectra compatible by interpolation. The result they return includes all the individual wavelength values (the set union of the wavelengths from the two spectra in the region where they overlap). The functions are `sum_spectra`, `subt_spectra`, `prod_spectra`, `div_spectra`, and `oper_spectra`. Here is a very simple hypothetical example:

```
# not run
out1.dt <- sum_spectra(spc1$w.length, spc2$w.length,
                       spc1$s.e.irrad, spc2$s.e.irrad)
```

However, we can achieve the same result, with simpler syntax, using spectral objects and the corresponding operators. The actual computations are done in both cases by the same code, but the example below adds some “syntactic sugar” to make the script code more readable.

```
out2.spct <- sun.spct + sun.spct
out2.spct

##      w.length    s.e.irrad
##   1:      293 5.219330e-06
##   2:      294 1.228480e-05
##  ---
## 507:      799 8.371699e-01
## 508:      800 8.138111e-01
```

```
out3.spct <- e2q(sun.spct + sun.spct)
out3.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 5.219330e-06 1.278346e-11
##   2:       294 1.228480e-05 3.019128e-11
##  ---
## 507:       799 8.371699e-01 5.591475e-06
## 508:       800 8.138111e-01 5.442264e-06
```

In both cases only spectral energy irradiance is calculated during the summing operation, while in the second example, it is simple to convert the returned spectral energy irradiance values into spectral photon irradiance. `out1.data` is a `data.table` while `out2.data` and `out3.data` are `source_spct`. The class of the returned spectrum depends on the classes of the operands.

The function `oper_spectra` takes the operator to use as an argument, and this abstraction both simplifies the package code, and also makes it easy for users to add other operators if needed:

```
# not run
out.data <- oper_spectra(spc1$w.length, spc2$w.length,
                         spc1$s.e.irrad, spc2$s.e.irrad,
                         bin.oper=`^`)
```

and yields one spectrum to a power of a second one. Such additional functions are not predefined, as I cannot think of any use for them. `oper_spectra` is used internally to define the functions for the four basic maths operators, and the corresponding operators.

7.7.3 Task: conversion from energy to photon base

The energy of a quantum of radiation in a vacuum, q, depends on the wavelength, λ, or frequency[2], ν,

$$q = h \cdot \nu = h \cdot \frac{c}{\lambda} \tag{7.1}$$

with the Planck constant $h = 6.626 \times 10^{-34}$ J s and speed of light in vacuum $c = 2.998 \times 10^{8}$ m s^{-1}. When dealing with numbers of photons, the equation (7.1) can be extended by using Avogadro's number $N_A = 6.022 \times 10^{23}$ mol^{-1}. Thus, the energy of one mole of photons, q', is

$$q' = h' \cdot \nu = h' \cdot \frac{c}{\lambda} \tag{7.2}$$

with $h' = h \cdot N_A = 3.990 \times 10^{-10}$ J s mol^{-1}.

`numeric` vectors

Function `as_quantum` converts W m^{-2} into *number of photons* per square meter per second, and `as_quantum_mol` does the same conversion but returns mol m^{-2} s^{-1}. Function `as_quantum` is based on the equation 7.1 while `as_quantum_mol` uses equation 7.2. To obtain µmol m^{-2} s^{-1} we multiply by 10^6:

[2]Wavelength and frequency are related to each other by the speed of light, according to $\nu = c/\lambda$ where c is speed of light in vacuum. Consequently there are two equivalent formulations for equation 7.1.

```
as_quantum_mol(550, 200) * 1e6

## [1] 919.5147
```

The calculation above is for monochromatic light (200 $\mathrm{W\,m^{-2}}$ at 550 nm).

The functions are vectorized, so they can be applied to whole spectra (when data are available as vectors), to convert $\mathrm{W\,m^{-2}\,nm^{-1}}$ to $\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$:

```
head(sun.spct$s.e.irrad, 10)

##  [1] 2.609665e-06 6.142401e-06 2.176175e-05
##  [4] 6.780119e-05 1.533491e-04 3.669677e-04
##  [7] 7.845430e-04 1.264554e-03 2.623718e-03
## [10] 3.922583e-03

s.q.irrad <- with(sun.spct,
                  as_quantum_mol(w.length, s.e.irrad))
head(s.q.irrad, 10)

##  [1] 6.391730e-12 1.509564e-11 5.366385e-11
##  [4] 1.677626e-10 3.807181e-10 9.141345e-10
##  [7] 1.960893e-09 3.171207e-09 6.601607e-09
## [10] 9.902505e-09
```

`source_spct` objects

Once again, easiest is to use spectral objects. The default is to add `s.q.irrad` to the source spectrum, unless it is already present in the object in which case values are not recalculated. It can also be used as a roundabout way of removing a `s.e.irrad` column, which could be useful in some cases.

```
sun.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06

my_sun.spct <- copy(sun.spct)
e2q(my_sun.spct)

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06
```

`e2q` has a parameter `action`, with default `"add"`. Another valid argument value is `"replace"`.

```
sun.spct
```

```
##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06

my_sun.spct <- copy(sun.spct)
e2q(my_sun.spct, "replace")

##      w.length    s.q.irrad
##   1:      293 6.391730e-12
##   2:      294 1.509564e-11
##  ---
## 507:      799 2.795738e-06
## 508:      800 2.721132e-06

my_sun.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06
```

`response_spct` objects

In the case of response spectra expressed per energy unit, as the energy unit is a divisor, the conversion is done with the inverse of the factor in equation 7.1. Although the method name is `e2q` as for `source_spct` objects, the appropriate conversion is applied.

7.7.4 Task: conversion from photon to energy base

`as_energy` is the inverse function of `as_quantum_mol`:

`numeric` vectors

In Aphalo, Albert, Björn, Ylianttila et al. 2012 it is written: “Example 1: red light at 600 nm has about 200 $\mathrm{kJ\,mol^{-1}}$, therefore, 1 µmol photons has 0.2 J. Example 2: UV-B radiation at 300 nm has about 400 $\mathrm{kJ\,mol^{-1}}$, therefore, 1 µmol photons has 0.4 J. Equations 7.1 and 7.2 are valid for all kinds of electromagnetic waves.” Let’s re-calculate the exact values—as the output from `as_energy` is expressed in $\mathrm{J\,mol^{-1}}$ we multiply the result by 10^{-3} to obtain $\mathrm{kJ\,mol^{-1}}$:

```
as_energy(600, 1) * 1e-3

## [1] 199.3805

as_energy(300, 1) * 1e-3

## [1] 398.7611
```

Because of vectorization we can also operate on a whole spectrum:

```
s.e.irrad <- with(sun.data, as_energy(w.length, s.q.irrad))
```

`source_spct` objects

Function `q2e` is the reverse of `e2q`, converting spectral energy irradiance in $\mathrm{W\,m^{-2}\,nm^{-1}}$ to spectral photon irradiance in $\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$. It can also be used as a roundabout way of removing a `s.e.irrad` column, which could be useful in some cases.

```
sun.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06

my_sun.spct <- copy(sun.spct)
q2e(my_sun.spct, "replace")

##      w.length    s.e.irrad
##   1:      293 2.609665e-06
##   2:      294 6.142401e-06
##  ---
## 507:      799 4.185850e-01
## 508:      800 4.069055e-01
```

As we have seen above by default `q2e` and `e2q` return a modified copy of the spectrum as a new object. This is safe, but inefficient in use of memory and computing resources. We first copy the data to a new object, and delete the `s.e.irrad` variable, so that we can test the use of the functions by reference. When parameter `byref` is given TRUE as argument the original spectrum is modified.

```
q2e(my_sun.spct, byref=TRUE)

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06

my_sun.spct

##      w.length    s.e.irrad    s.q.irrad
##   1:      293 2.609665e-06 6.391730e-12
##   2:      294 6.142401e-06 1.509564e-11
##  ---
## 507:      799 4.185850e-01 2.795738e-06
## 508:      800 4.069055e-01 2.721132e-06
```

`response_spct` objects

In the case of response spectra expressed per energy unit, as the energy unit is a divisor, the conversion is done with the inverse of the factor in equation 7.1. Although the method name is `q2e` as for `source_spct` objects, the appropriate conversion is applied.

7.8 Wavebands

7.8.1 How are wavebands stored?

Wavebands are derived from R lists. All valid R operations for lists can be also used with `waveband` objects. However, there are `waveband`-specific specializations of generic R methods.

7.8.2 Operators and functions

Multiplying any spectrum by an un-weighted waveband, is equivalent to trimming with `fill` set to NA (see section 6.5.3.

```
is_effective(UVA())

## [1] FALSE

sun.spct * UVA()

##     w.length s.e.irrad
##  1:  315.000 0.1127901
##  2:  316.000 0.1020587
## ---
## 85:  399.000 0.5861190
## 86:  399.999 0.6080829
```

Multiplying a `source_spct` object by a weighted waveband convolutes the spectrum with weights, yielding effective spectral irradiance.

```
is_effective(CIE())

## [1] TRUE

sun.spct * CIE()

##      w.length    s.e.irrad
##   1:  293.000 2.609665e-06
##   2:  294.000 6.142401e-06
##  ---
## 107:  399.000 7.378801e-05
## 108:  399.999 7.395674e-05
```

```
try(detach(package:photobiologyFilters))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

CHAPTER 8

Spectra: simple summaries and features

Abstract

In this chapter we explain how to obtain different summaries common to all types of spectral data. In addition we describe how to extract spectral features from spectral data.

8.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)
library(photobiologygg)

## Loading required package:  photobiologyWavebands
## Loading required package:  proto
## Loading required package:  ggplot2
## Loading required package:  methods
## Loading required package:  scales

library(photobiologyLamps)
library(photobiologyFilters)
library(photobiologyReflectors)
```

8.2 Task: Summaries related to object properties

In the case of the `summary` method, specializations for `source_spct` and ... are provided. But for other spectral objects, the `summary` method for `data.table` is called. For the `summary` specializations defined, the corresponding `print` method specializations are also defined.

```
summary(sun.spct)

## wavelength ranges from 293 to 800 nm
## largest wavelength step size is 1 nm
## spectral irradiance ranges from 2.61e-06 to 0.8205 W m-2 nm-1
## energy irradiance is 269.1 W m-2
## spectral photon irradiance ranges from 6.392e-06 to 3.375 umol s-1 m-2 nm-1
## photon irradiance is 1255 umol s-1 m-2
```

8.3 Task: Summaries related to wavelength

Functions `max`, `min`, `range`, `midpoint` when used with an object of class `generic_spct` (or a derived class) return the result of applying these functions to the `w.length` component of these objects, returning always values expressed in nanometres as long as the objects have been correctly created.

```
range(sun.spct)

## [1] 293 800

midpoint(sun.spct)

## [1] 546.5

max(sun.spct)

## [1] 800

min(sun.spct)

## [1] 293
```

Functions `spread` are `stepsize` are generics defined in package photobiology. `spread` returns maximum less minimum wavelengths values in nanometres, while `stepsize` returns a numeric vector of length two with the maximum and the minimum wavelength step between observations, also in nanometers.

```
spread(sun.spct)

## [1] 507

stepsize(sun.spct)

## [1] 1 1
```

8.4 Task: Finding the class of an object

R method `class` can be used with any R object, including spectra.

```
class(sun.spct)

## [1] "source_spct"  "generic_spct" "data.table"
## [4] "data.frame"

class(polyester.new.spct)

## [1] "filter_spct"  "generic_spct" "data.table"
## [4] "data.frame"
```

The method `class_spct` is a convenience wrapped on `class` which returns only class attributes corresponding to spectral classes defined in package photobiology.

```
class_spct(sun.spct)

## [1] "source_spct"  "generic_spct"

class_spct(polyester.new.spct)

## [1] "filter_spct"  "generic_spct"
```

The method `is.any_spct` is a synonym of `is.generic_spct` as `generic_spct` is the base class from which all spectral classes are derived.

```
is.any_spct(sun.spct)

## [1] TRUE

is.any_spct(polyester.new.spct)

## [1] TRUE
```

Equivalent methods exist for all the classes defined in package photobiology. We show two examples below, with a radiation source and a filter.

```
is.source_spct(sun.spct)

## [1] TRUE

is.source_spct(polyester.new.spct)

## [1] FALSE
```

```
is.filter_spct(sun.spct)

## [1] FALSE

is.filter_spct(polyester.new.spct)

## [1] TRUE
```

8.5 Task: Querying other attributes

Both `response_spct` and `source_spct` objects have an attribute `time.unit` that can be queried.

```
getTimeUnit(sun.spct)

## [1] "second"
```

```
is_effective(sun.spct * CIE())

## [1] TRUE

is_effective(sun.spct * UV())

## [1] FALSE
```

```
getBSWFUsed(sun.spct * CIE())

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

Normalization and scaling can be applied to different types of spectral objects.

```
sun.norm.spct <- normalize(sun.spct, norm = 600)

## Warning in setTimeUnit(new.spct, time.unit.spct):
Overrriding existing 'time.unit' 'second' with 'unknown' may
invalidate data!

is_normalized(sun.norm.spct)

## [1] TRUE

getNormalized(sun.norm.spct)

## [1] 600
```

```
sun.scaled.spct <- fscale(sun.spct, f = "mean")
is_scaled(sun.scaled.spct)

## [1] TRUE
```

We now consider `filter_spct` objects (see Chapter 12 for an explanation of the meaning of these attributes and how they affect calculations).

```
getTfrType(polyester.new.spct)

## [1] "total"
```

and `reflector_spct` objects.

```
getRfrType(gold.spct)

## [1] "total"
```

8.6 Task: Query how spectral data contained is expressed

We first consider the case of `source.spct` objects. If an object contains the same data expressed differently, it is possible, as in the example for both statement to return true.

```
head(sun.spct)

##    w.length    s.e.irrad    s.q.irrad
## 1:      293 2.609665e-06 6.391730e-12
## 2:      294 6.142401e-06 1.509564e-11
## 3:      295 2.176175e-05 5.366385e-11
## 4:      296 6.780119e-05 1.677626e-10
## 5:      297 1.533491e-04 3.807181e-10
## 6:      298 3.669677e-04 9.141345e-10

is_energy_based(sun.spct)

## [1] TRUE

is_photon_based(sun.spct)

## [1] TRUE
```

If we delete the energy based spectral data, the result of the test changes.

```
my.spct <- copy(sun.spct)
my.spct$s.e.irrad <- NULL
head(my.spct)

##    w.length    s.q.irrad
## 1:      293 6.391730e-12
## 2:      294 1.509564e-11
## 3:      295 5.366385e-11
## 4:      296 1.677626e-10
## 5:      297 3.807181e-10
## 6:      298 9.141345e-10

is_energy_based(my.spct)

## [1] FALSE

is_photon_based(my.spct)

## [1] TRUE
```

We now consider `filter_spct` objects.

```
is_transmittance_based(polyester.new.spct)

## [1] TRUE
```

```
is_absorbance_based(polyester.new.spct)

## [1] FALSE
```

8.7 Task: Querying about 'origin' of data

All spectral objects (`generic_spct` and derived types) can be queried whether they are the result of the normalization or re-scaling of another spectrum. In the case of normalization, the normalization wavelength in nanometres is returned, otherwise a logical value.

```
is_normalized(sun.spct)

## [1] FALSE

is_scaled(sun.spct)

## [1] FALSE
```

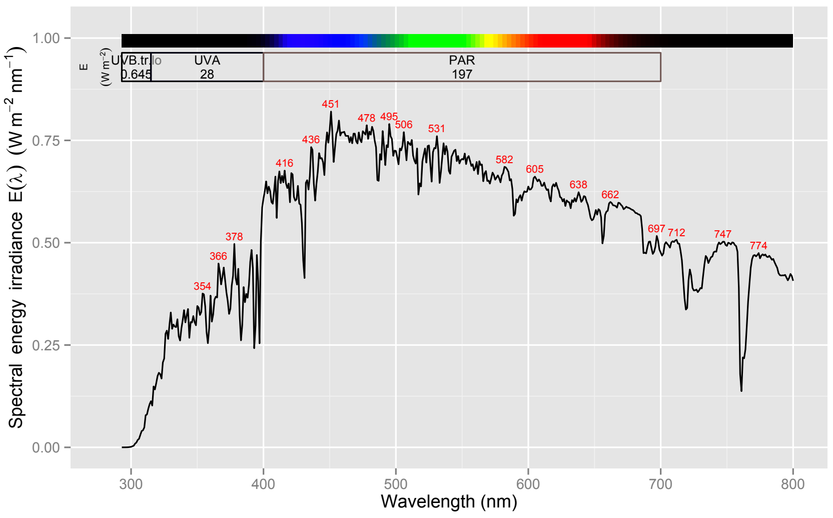
`source_spct` objects can be queried to learn if they are the result of a calculation involving a weighting function.

```
is_effective(sun.spct)

## [1] FALSE
```

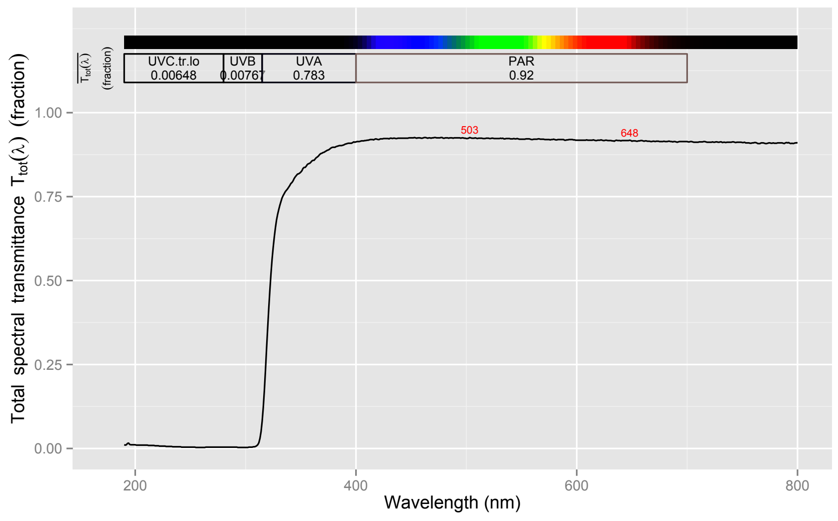
8.8 Task: Plotting a spectrum

Method `plot` is defined for `waveband` objects, and can be used to visually check their properties. Plotting is discussed in detail in chapter **??**.

```
plot(sun.spct)
```



```
plot(polyester.new.spct)
```



8.9 Task: Other R's methods

Methods `names` and `comment` should work as usual. In the case of the comment attribute, most operations on spectral objects preserve comments, sometimes with additions, or by merging of comments from operands. Comments are optional, so for some objects `comment` may return a NULL.

```
names(sun.spct)

## [1] "w.length"  "s.e.irrad" "s.q.irrad"

comment(sun.spct)

## NULL
```

8.10 Task: Find peaks and valleys

Methods `peaks` and `valleys` can be used on most spectral objects to find local maxima and local minima in spectral data. They return an object of the same class containing only the observations corresponding to these local extremes.

```
peaks(philips.tl12.spct)

##     w.length s.e.irrad
##  1:      313   0.29181
##  2:      365   0.06587
## ---
## 29:      691   0.00191
## 30:      730   0.00153
```

```
peaks(philips.tl12.spct, unit.out = "photon")

##      w.length    s.q.irrad
##  1:       313 7.635026e-07
##  2:       365 2.009771e-07
## ---
## 30:       691 1.103259e-08
## 31:       730 9.336418e-09

peaks(philips.tl12.spct, span = 50)

## span increased to next odd value:  51
##      w.length s.e.irrad
##  1:       313   0.29181
##  2:       365   0.06587
##  3:       404   0.15675
##  4:       435   0.38773
##  5:       492   0.00154
##  6:       545   0.16449
##  7:       578   0.03238
##  8:       612   0.00131
##  9:       691   0.00191
## 10:       730   0.00153
```

```
valleys(kg5.spct)

##    w.length     Tfr
## 1:      530 0.87000
## 2:     3000 0.00018

peaks(kg5.spct, filter.qty = "absorbance")

##    w.length          A
## 1:      530 0.06048075
## 2:     3000 3.74472749
```

8.10.1 Obtaining the location of peaks as an index into the spectral data

Function `find_peaks`, takes as argument a `numeric` vector, and returns a logical vector of the same length, with TRUE for local maxima and FALSE for all other observations. Infinite values are discarded.

```
head(find_peaks(sun.spct$s.e.irrad))

## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

To obtain the indexes, one can use R's function `which`

```
head(which(find_peaks(sun.spct$s.e.irrad)))

## [1] 23 25 29 35 38 40
```

8.10.2 Obtaining the location of peaks as a wavelength in nanometres

Function `get_peaks` takes two numeric vectors as as arguments, x is, for spectra assumed to be a vector of wavelengths, and y the spectral variable to search for local maxima.

```
with(sun.spct, get_peaks(w.length, s.e.irrad, span = 51))[["x"]]

## [1] 451 495 747
```

The returned value is a (shorter) data frame with two numeric vectors, x and y, and an optional chracter variables `label`, for each local maximum found in y, but we extract `x`.

8.11 Task: Refining the location of peaks and valleys

The functions described in the previous section locate the observation with the locally highest y-value. This is in most cases the true location of the peaks as they may fall in between two observations along the wavelength axis. By fitting a suitable model to describe the shape of the peak, which is the result of the true peak and the slit function of the spectrometer, the true location of a peak can be approximated more precisely. There is no universally useful model, so we show some examples of a possible method of peak-position refinement.

In this example, in the second statement we refine the location of the shortest-wavelength peak found by `get_peaks` in the first statement. For this approach to work, the peaks should be clearly visible, and not very close to each other. We use the spectral irradiance measured from a UV-B lamp as an example.

```
stepsize(germicidal.spct)

## [1] 0.43 0.48

peaks <-
   with(germicidal.spct,
        get_peaks(w.length, s.e.irrad, span = 5))
fit <- nls(s.e.irrad ~ d + a1*exp(-0.5*((w.length-c1)/b1)^2),
           start=list(a1=3.1, b1=1, c1=peaks[1, 1], d=0),
           data=germicidal.spct)
fit

## Nonlinear regression model
##   model: s.e.irrad ~ d + a1 * exp(-0.5 * ((w.length - c1)/b1)^2)
##    data: germicidal.spct
##        a1        b1        c1         d
## 2.996e+00 5.056e-01 2.539e+02 2.386e-03
##  residual sum-of-squares: 0.1309
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 6.219e-06

fit$m$getPars()[["c1"]]

## [1] 253.8703
```

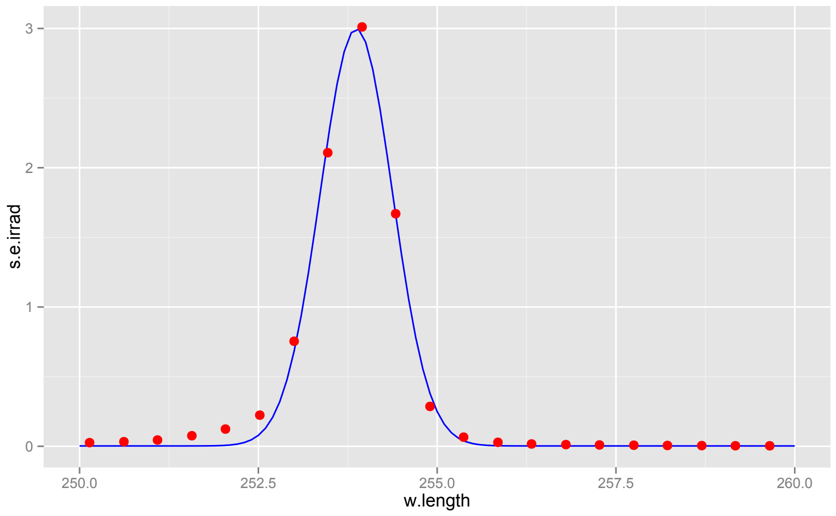
```
peaks[1, 1]

## [1] 253.95
```

In this case the change was rather small, and shows a small wavelength calibration error for the spectrometer that can be calculated as:

```
signif(fit$m$getPars()[["c1"]], 6) - 253.652

## [1] 0.218
```

```
predicted <-
  predict(fit,
          data.frame(w.length = seq(250, 260, length.out = 101)))
fitted_peak.spct <-
  source_spct(w.length  = seq(250, 260, length.out = 101),
              s.e.irrad = predicted)
```

```
ggplot(data = fitted_peak.spct, aes(w.length, s.e.irrad)) +
  geom_line(data = fitted_peak.spct, colour = "blue") +
  geom_point(data = germicidal.spct, colour = "red", size = 3) +
  xlim(250, 260)

## Warning:  Removed 1404 rows containing missing values
(geom_point).
```



```
try(detach(package:photobiologyReflectors))
try(detach(package:photobiologyFilters))
try(detach(package:photobiologyLamps))
try(detach(package:photobiologygg))
try(detach(package:photobiology))
```

CHAPTER 9

Wavebands: simple summaries and features

Abstract

In this chapter we explain how to obtain different summaries and query features from wavebands.

9.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)
library(photobiologyWavebands)
library(photobiologygg)
```

9.2 Task: Summaries related to object properties

In the case of the summary method, specializations for source_spct and . . . are provided. But for other spectral objects, the summary method for data.table is called. For the summary specializations defined, the corresponding print method specializations are also defined.

```
my.wb <- waveband(c(400,500))
summary(my.wb)

##           Length Class  Mode
## low       1      -none- numeric
## high      1      -none- numeric
## weight    1      -none- character
## SWF.e.fun 0      -none- NULL
## SWF.q.fun 0      -none- NULL
## SWF.norm  0       -none- NULL
## norm      0       -none- NULL
## hinges    4       -none- numeric
## name      1       -none- character
## label     1       -none- character
```

```
vis.wb <- VIS()
summary(vis.wb)

##             Length Class  Mode
## low         1      -none- numeric
## high        1      -none- numeric
## weight      1      -none- character
## SWF.e.fun 0      -none- NULL
## SWF.q.fun 0      -none- NULL
## SWF.norm  0      -none- NULL
## norm      0      -none- NULL
## hinges    4      -none- numeric
## name      1      -none- character
## label     1      -none- character
```

```
cie.wb <- CIE()
summary(cie.wb)

##             Length Class  Mode
## low         1      -none- numeric
## high        1      -none- numeric
## weight      1      -none- character
## SWF.e.fun 1      -none- function
## SWF.q.fun 1      -none- function
## SWF.norm  1      -none- numeric
## norm      1      -none- numeric
## hinges    6      -none- numeric
## name      1      -none- character
## label     1      -none- character
```

9.3 Task: Summaries related to wavelength

Functions `max`, `min`, `range`, `midpoint` when used with an object of class `waveband` return the result of applying these functions to the wavelength component boundaries of these objects, returning always values expressed in nanometres as long as the objects have been correctly created.

```
range(vis.wb)

## [1] 380 760

midpoint(vis.wb)

## [1] 570

max(vis.wb)

## [1] 760
```

```
min(vis.wb)

## [1] 380
```

Functions `spread` are `stepsize` are generics defined in package photobiology. `spread` returns maximum less minimum wavelengths values in nanometres, while `stepsize` returns a numeric vector of length two with the maximum and the minimum wavelength step between observations, also in nanometers.

```
spread(vis.wb)

## [1] 380
```

9.4 Task: Querying other properties

It is possible to query whether a `waveband` object includes a weighting function using function `is_effective`. Weighting functions are used for the calculation *effective irradiances* and *effecctive exposures*.

```
is_effective(vis.wb)

## [1] FALSE

is_effective(cie.wb)

## [1] TRUE
```

9.5 Task: R's methods

The "labels" can be retrieved with R's method `labels`. Waveband objects have two slots for names, normally used when wavebands are plotted or printed.

```
labels(my.wb)

## $label
## [1] "range.400.500"
##
## $name
## [1] "range.400.500"

labels(vis.wb)

## $label
## [1] "VIS"
##
## $name
## [1] "VIS.ISO"

labels(cie.wb)
```

```
## $label
## [1] "CIE98"
##
## $name
## [1] "CIE98.298"
```

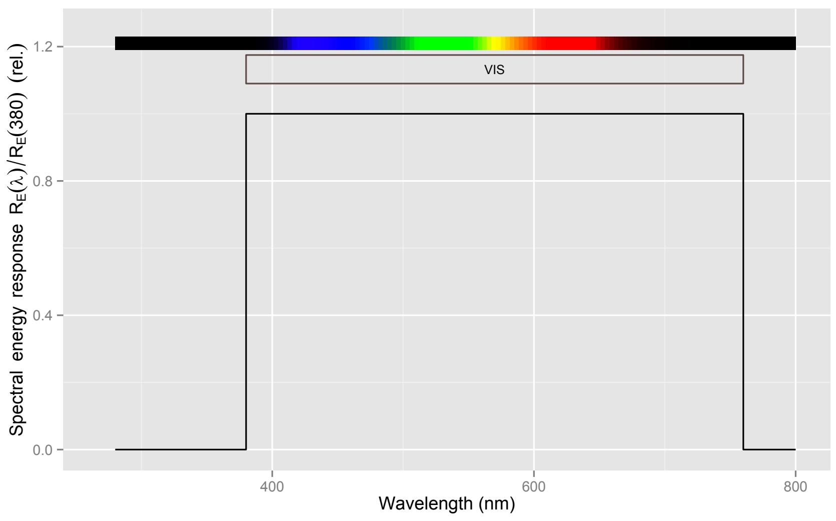
As with any R object, method `names` returns a vector of names of the object's components.

```
names(vis.wb)

##  [1] "low"       "high"      "weight"
##  [4] "SWF.e.fun" "SWF.q.fun" "SWF.norm"
##  [7] "norm"      "hinges"    "name"
## [10] "label"

names(cie.wb)

##  [1] "low"       "high"      "weight"
##  [4] "SWF.e.fun" "SWF.q.fun" "SWF.norm"
##  [7] "norm"      "hinges"    "name"
## [10] "label"
```

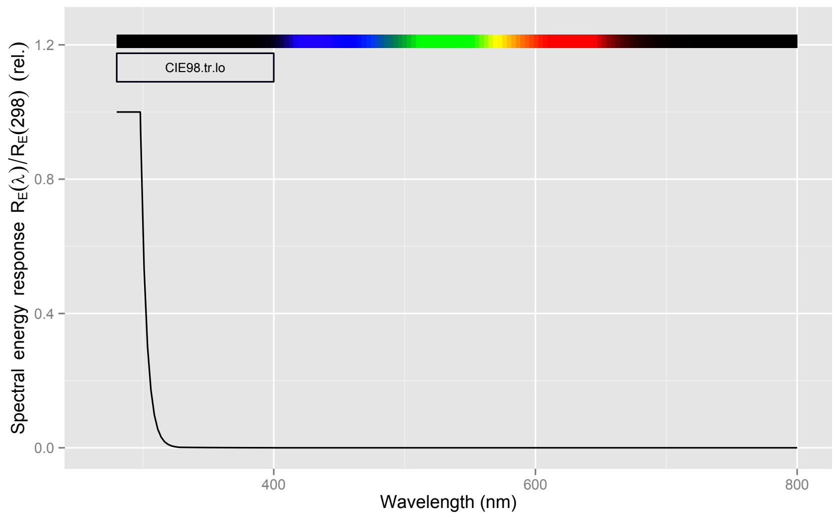
9.6 Task: Plotting a waveband

Method `plot` is defined for `waveband` objects, and can be used to visually check their properties. Plotting is discussed in detail in chapter **??**.

```
plot(vis.wb)
```



```
plot(cie.wb)
```



```r
try(detach(package:photobiologygg))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

CHAPTER 10

Unweighted irradiance

Abstract

In this chapter we explain how to calculate unweighted energy and photon irradiances from spectral irradiance.

10.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```r
library(photobiology)
library(photobiologyWavebands)
```

10.2 Introduction

Functions `e_irrad` and `q_irrad` return energy irradiance and photon (or quantum) irradiance, and both take as argument a `source_spct` object containing either spectral (energy) irradiance or spectral photon irradiance data. An additional parameter accepting a `waveband` object, or a list of `waveband` objects, can be used to set the range(s) of wavelengths and spectral weighting function(s) to use for integration(s). Two additional functions, `energy_irradiance` and `photon_irradiance`, are defined for equivalent calculations on spectral irradiance data stored as numeric vectors.

We start by describing how to use and define `waveband` objects, for which we need to use function `e_irrad` in some examples before a detailed explanation of its use (see section 10.6) on page 71 for details).

10.3 Task: use simple predefined wavebands

Please, consult the packages' documentation for a list of predefined functions for creating wavebands also called `waveband` *constructors*. Here we will present just a few examples of their use. We usually associate wavebands with colours, however, in many cases there are different definitions in use. For this reason, the functions provided accept an argument that can be used to select the definition to use. In general, the default, is to use the ISO standard whenever it is applicable. The case of the various definitions in use for the UV-B waveband are described on page 65

We can use a predefined function to create a new `waveband` object, which as any other R object can be assigned to a variable:

```
uvb <- UVB()
uvb

## UVB.ISO
## low (nm) 280
## high (nm) 315
## weighted none
```

As seen above, there is a specialized `print` method for `wavebands`. `waveband` methods returning wavelength values in nm are `min`, `max`, `range`, `midpoint`, and `spread`. Method `labels` returns the name and label stored in the waveband, and method `color` returns a color definition calculated from the range of wavelengths.

```
red <- Red()
red

## Red.ISO
## low (nm) 610
## high (nm) 760
## weighted none

min(red)

## [1] 610

max(red)

## [1] 760

range(red)

## [1] 610 760

midpoint(red)

## [1] 685

spread(red)

## [1] 150
```

```
labels(red)

## $label
## [1] "Red"
##
## $name
## [1] "Red.ISO"

color(red)

## $CMF
##   Red.CMF
## "#900000"
##
## $CC
##    Red.CC
## "#FF0000"
```

The argument `standard` can be used to choose a given alternative definition[1]:

```
UVB()

## UVB.ISO
## low (nm) 280
## high (nm) 315
## weighted none

UVB("ISO")

## UVB.ISO
## low (nm) 280
## high (nm) 315
## weighted none

UVB("CIE")

## UVB.CIE
## low (nm) 280
## high (nm) 315
## weighted none

UVB("medical")

## UVB.medical
## low (nm) 290
## high (nm) 320
## weighted none

UVB("none")

## UVB.none
## low (nm) 280
## high (nm) 320
## weighted none
```

[1]When available, the definition in the ISO standard is the default.

Here we demonstrate the importance of complying with standards, and how much photon irradiance can depend on the definition used in the calculation.

```
e_irrad(sun.spct, UVB("ISO"))

##  UVB.ISO.tr.lo
##      0.6445035
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

e_irrad(sun.spct, UVB("none"))

##  UVB.none.tr.lo
##        1.337169
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

e_irrad(sun.spct, UVB("ISO")) / e_irrad(sun.spct, UVB("none"))

##  UVB.ISO.tr.lo
##      0.4819911
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

10.4 Task: define simple wavebands

Here we briefly introduce `waveband` and `new_waveband`, and only in chapter **??** we describe their use in full detail, including the use of spectral weighting functions (SWFs). The examples in the present section only describe `waveband`s that define a wavelength range.

A `waveband` can be created based on any R object for which function `range` is defined, and returns numbers interpretable as wavelengths expressed in nanometres:

```
waveband(c(400,700))

## range.400.700
## low (nm) 400
## high (nm) 700
## weighted none

waveband(400:700)

## range.400.700
## low (nm) 400
## high (nm) 700
## weighted none

waveband(sun.spct)
```

```
## Total
## low (nm) 293
## high (nm) 800
## weighted none

wb_total <- waveband(sun.spct, wb.name="total")
```

```
e_irrad(sun.spct, wb_total)

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

A `waveband` can also be created based on extreme wavelengths expressed in nm.

```
wb1 <- new_waveband(500,600)
wb1

## range.500.600
## low (nm) 500
## high (nm) 600
## weighted none

e_irrad(sun.spct, wb1)

##  range.500.600
##       68.48951
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

wb2 <- new_waveband(500,600, wb.name="my.colour")
wb2

## my.colour
## low (nm) 500
## high (nm) 600
## weighted none

e_irrad(sun.spct, wb2)

##  my.colour
##   68.48951
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

10.5 Task: define lists of simple wavebands

Lists of wavebands can be created by grouping `waveband` objects using the R-defined constructor `list`,

```
UV.list <- list(UVC(), UVB(), UVA())
UV.list

## [[1]]
## UVC.ISO
## low (nm) 100
## high (nm) 280
## weighted none
##
## [[2]]
## UVB.ISO
## low (nm) 280
## high (nm) 315
## weighted none
##
## [[3]]
## UVA.ISO
## low (nm) 315
## high (nm) 400
## weighted none
```

in which case wavebands can be non-contiguous and/or overlapping.

In addition function `split_bands` can be used to create a list of contiguous wavebands by supplying a numeric vector of wavelength boundaries in nanometres,

```
split_bands(c(400,500,600))

## $wb1
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $wb2
## range.500.600
## low (nm) 500
## high (nm) 600
## weighted none
```

or with longer but more meaningful names,

```
split_bands(c(400,500,600), short.names=FALSE)

## $range.400.500
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $range.500.600
## range.500.600
## low (nm) 500
## high (nm) 600
## weighted none
```

It is also possible to also provide the limits of the region to be covered by the list of wavebands and the number of (equally spaced) wavebands desired:

```
split_bands(c(400,600), length.out=2)

## $wb1
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $wb2
## range.500.600
## low (nm) 500
## high (nm) 600
## weighted none
```

in all cases coderange is used to find the list boundaries, so we can also split the region defined by an existing `waveband` object into smaller wavebands,

```
split_bands(PAR(), length.out=3)

## $wb1
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $wb2
## range.500.600
## low (nm) 500
## high (nm) 600
## weighted none
##
## $wb3
## range.600.700
## low (nm) 600
## high (nm) 700
## weighted none
```

or split a whole spectrum[2] into equally sized regions,

```
split_bands(sun.spct, length.out=3)

## $wb1
## range.293.462
## low (nm) 293
## high (nm) 462
## weighted none
##
## $wb2
```

[2]This is not restricted to `source_spct` objects as all other classes of `____.spct` objects also have `range` methods defined.

```
## range.462.631
## low (nm) 462
## high (nm) 631
## weighted none
##
## $wb3
## range.631.800
## low (nm) 631
## high (nm) 800
## weighted none
```

It is also possible to supply a list of wavelength ranges[3], and, when present, names are copied from the input list to the output list:

```
split_bands(list(c(400,500), c(600,700)))

## $wb.a
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $wb.b
## range.600.700
## low (nm) 600
## high (nm) 700
## weighted none

split_bands(list(blue=c(400,500), PAR=c(400,700)))

## $blue
## range.400.500
## low (nm) 400
## high (nm) 500
## weighted none
##
## $PAR
## range.400.700
## low (nm) 400
## high (nm) 700
## weighted none
```

Package photobiologyWavebands also predefines some useful constructors of lists of wavebands, currently `VIS_bands`, `UV_bands` and `Plant_bands`.

```
UV_bands()

## [[1]]
## UVC.ISO
## low (nm) 100
## high (nm) 280
## weighted none
##
```

[3]When using a list argument, even overlapping and non-contiguous wavelength ranges are valid input

```
## [[2]]
## UVB.ISO
## low (nm) 280
## high (nm) 315
## weighted none
##
## [[3]]
## UVA.ISO
## low (nm) 315
## high (nm) 400
## weighted none
```

10.6 Task: (energy) irradiance from spectral irradiance

The task to be completed is to calculate the (energy) irradiance (E) in $\mathrm{W\,m^{-2}}$ from spectral (energy) irradiance ($E(\lambda)$) in $\mathrm{W\,m^{-2}\,nm^{-1}}$ and the corresponding wavelengths (λ) in nm.

$$E_{\lambda_1<\lambda<\lambda_2} = \int_{\lambda_1}^{\lambda_2} E(\lambda)\ \mathrm{d}\ \lambda \tag{10.1}$$

Let's assume that we want to calculate photosynthetically active radiation (PAR) energy irradiance, for which the most accepted limits are $\lambda_1 = 400$nm and $\lambda_1 = 700$nm. In this example we will use example data for sunlight to calculate $E_{400\,\mathrm{nm}<\lambda<700\,\mathrm{nm}}$. The function used for this task when working with spectral objects is `e_irrad` returning energy irradiance. The "names" of the returned valued is set according to the waveband used, and `sun.spct` is a `source_spct` object.

```
e_irrad(sun.spct, waveband(c(400,700)))

##  range.400.700
##       196.6343
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or using the PAR waveband constructor, defined in package photobiologyWavebands as a convenience function,

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

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

or if no waveband is supplied as argument, then irradiance is computed for the whole range of wavelengths in the spectral data, and the 'name' attribute is generated accordingly.

```
e_irrad(sun.spct)

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

If a waveband extends outside of the wavelength range of the spectral data, spectral irradiance for unavailable wavelengths is assumed to be zero:

```
e_irrad(sun.spct, waveband(c(100,400)))

##  range.100.400.tr.lo
##             28.62868
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

e_irrad(sun.spct, waveband(c(100,250)))

## out of range
##           NA
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

Both `e_irrad` and `q_irrad` accept, in addition to a waveband as second argument, a list of wavebands. In this case, the returned value is a numeric vector of the same length as the list.

```
e_irrad(sun.spct, list(UVB(), UVA()))

##  UVB.ISO.tr.lo        UVA.ISO
##      0.6445035     27.9841813
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

Storing emission spectral data in `source_spct` objects is recommended, as it allows better protection against mistakes, and allows automatic detection of input data base of expression and units. However, it may be sometimes more convenient or efficient to keep spectral data in individual numeric vectors, or data frames. In such cases function `energy_irradiance`, which accepts the spectral data as vectors can be used at the cost of less concise code and weaker error tests. In this case, the user must indicate whether spectral data is on energy or photon based units through parameter `unit.in`, which defaults to `"energy"`.

For example when using function PAR(), the code above becomes:

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

```
##       PAR
## 196.7004

with(sun.spct,
     energy_irradiance(w.length, s.e.irrad, PAR(), unit.in="energy"))

##       PAR
## 196.7004
```

where `sun.spct` is a data frame. However, the data can also be stored in separate numeric vectors of equal length.

The `sun.spct` data frame also contains spectral photon irradiance values:

```
names(sun.spct)

## [1] "w.length"  "s.e.irrad" "s.q.irrad"
```

which allows us to use:

```
with(sun.spct,
     energy_irradiance(w.length, s.q.irrad, PAR(), unit.in="photon"))

##       PAR
## 196.7004
```

The other examples above can be re-written with similar syntax.

10.7 Task: photon irradiance from spectral irradiance

The task to be completed is to calculate the photon irradiance (Q) in $\mathrm{mol\,m^{-2}\,s^{-1}}$ from spectral (energy) irradiance ($E(\lambda)$) in $\mathrm{W\,m^{-2}\,nm^{-1}}$ and the corresponding wavelengths (λ) in nm.

Combining equations 10.1 and 7.2 we obtain:

$$Q_{\lambda_1<\lambda<\lambda_2} = \int_{\lambda_1}^{\lambda_2} E(\lambda)\,\frac{h'\cdot c}{\lambda}\,\mathrm{d}\,\lambda \tag{10.2}$$

Let's assume that we want to calculate photosynthetically active radiation (PAR) photon irradiance (frequently called PPFD or photosynthetic photon flux density), for which the most accepted limits are $\lambda_1 = 400$nm and $\lambda_1 = 700$nm. In this example we will use example data for sunlight to calculate $E_{400\,\mathrm{nm}<\lambda<700\,\mathrm{nm}}$. The function used for this task when working with spectral objects is `q_irrad`, returning photon irradiance in $\mathrm{mol\,m^{-2}\,s^{-1}}$. The "names" of the returned valued is set according to the waveband used, and `sun.spct` is a `source_spct` object.

```
q_irrad(sun.spct, waveband(c(400,700)))

##  range.400.700
##   0.0008941352
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

to obtain the photon irradiance expressed in $\mu\text{mol}\,\text{m}^{-2}\,\text{s}^{-1}$ we multiply the returned value by 1×10^6:

```
q_irrad(sun.spct, waveband(c(400,700))) * 1e6

##  range.400.700
##       894.1352
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

or using the PAR waveband constructor, defined in package photobiologyWavebands as a convenience function,

```
q_irrad(sun.spct, PAR()) * 1e6

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

Examples given in section 10.6 can all be converted by replacing `e_irrad` function calls with `q_irrad` function calls.

Storing emission spectral data in `source_spct` objects is recommended (see section 10.6). However, it may be sometimes more convenient or efficient to keep spectral data in individual numeric vectors, or data frames. In such cases function `photon_irradiance`, which accepts the spectral data as vectors can be used at the cost of less concise code and weaker error tests. In this case, the user must indicate whether spectral data is on energy or photon based units through parameter `unit.in`, which defaults to `"energy"`.

For example when using function `PAR()`, the code above becomes:

```
with(sun.spct,
     photon_irradiance(w.length, s.e.irrad, PAR()), unit.in="energy")  * 1e6

##      PAR
## 893.7598

with(sun.spct,
     photon_irradiance(w.length, s.e.irrad, PAR()))  * 1e6

##      PAR
## 893.7598
```

where `sun.spct` is a data frame. However, the data can also be stored in separate numeric vectors of equal length.

10.8 Task: irradiances for more than one waveband

As discussed above, it is possible to calculate simultaneously the irradiances for several wavebands with a single function call by supplying a `list` of `wavebands` as argument:

```
q_irrad(sun.spct, list(Red(), Green(), Blue())) * 1e6

##     Red.ISO   Green.ISO    Blue.ISO
##    451.1084    220.1957    149.0288
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

Q.RGB <- q_irrad(sun.spct, list(Red(), Green(), Blue())) * 1e6
signif(Q.RGB, 3)

##     Red.ISO   Green.ISO    Blue.ISO
##         451         220         149
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

Q.RGB[1]

##   Red.ISO
## 451.1084

Q.RGB["Green.ISO"]

## <NA>
##   NA
```

as the value returned is in $\mathrm{mol\,m^{-2}\,s^{-1}}$ we multiply it by 1×10^6 to obtain $\mathrm{\mu mol\,m^{-2}\,s^{-1}}$.

A named list can be used to override the names used for the output:

```
q_irrad(sun.spct, list(R=Red(), G=Green(), B=Blue())) * 1e6

##     Red.ISO   Green.ISO    Blue.ISO
##    451.1084    220.1957    149.0288
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

Even when using a single waveband:

```
q_irrad(sun.spct, list('ultraviolet-B'=UVB())) * 1e6

##   UVB.ISO.tr.lo
##        1.675344
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

The examples above, can be easily rewritten using functions `e_irrad`, `energy_irradiance` or `photon_irradiance`.

For example, the second example above becomes:

```
e_irrad(sun.spct, list(R=Red(), G=Green(), B=Blue()))

##     Red.ISO  Green.ISO   Blue.ISO
##    79.38161   49.26861   37.55208
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or

```
with(sun.spct,
     energy_irradiance(w.length, s.e.irrad,
                       list(R=Red(), G=Green(), B=Blue())))

##        R        G        B
## 79.61176 49.30478 37.57760
```

10.9 Task: photon ratios

In photobiology sometimes we are interested in calculation the photon ratio between two wavebands. It makes more sense to calculate such ratios if both numerator and denominator wavebands have the same 'width' or if the numerator waveband is fully nested in the denominator waveband. However, frequently used ratios like the UV-B to PAR photon ratio do not comply with this. For this reason, our functions do not enforce any such restrictions.

For example a ratio frequently used in plant photobiology is the red to far-red photon ratio (R:FR photon ratio or ζ). If we follow the wavelength ranges in the definition given by **Morgan1981a** using photon irradiance[4]:

$$\zeta = \frac{Q_{655\mathrm{nm}<\lambda<665\mathrm{nm}}}{Q_{725\mathrm{nm}<\lambda<735\mathrm{nm}}} \tag{10.3}$$

To calculate this for our example sunlight spectrum we can use the following code:

```
q_ratio(sun.spct, Red("Smith10"), Far_red("Smith10"))

##  Red.Smith10: FarRed.Smith10(q:q)
##                          1.266705
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

Function `q_ratio` also accepts lists of wavebands, for both denominator and numerator arguments, and recycling takes place when needed. Calculation of the contribution of different colors to visible light, using ISO-standard definitions.

```
q_ratio(sun.spct, UVB(), list(UV(), VIS()))
```

[4]In the original text photon fluence rate is used but it not clear whether photon irradiance was meant instead.

```
##  UVB.ISO.tr.lo: UV.ISO.tr.lo(q:q)
##                        0.01936927
##       UVB.ISO.tr.lo: VIS.ISO(q:q)
##                        0.00154142
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

```
q_ratio(sun.spct,
         list(Red(), Green(), Blue()), VIS())

##    Red.ISO: VIS.ISO(q:q)  Green.ISO: VIS.ISO(q:q)
##                0.4150476                0.2025936
##   Blue.ISO: VIS.ISO(q:q)
##                0.1371157
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

or using a predefined list of wavebands:

```
q_ratio(sun.spct, VIS_bands(), VIS())

##  Purple.ISO: VIS.ISO(q:q)
##                0.15087805
##    Blue.ISO: VIS.ISO(q:q)
##                0.13711570
##   Green.ISO: VIS.ISO(q:q)
##                0.20259363
##  Yellow.ISO: VIS.ISO(q:q)
##                0.06106050
##  Orange.ISO: VIS.ISO(q:q)
##                0.05545497
##     Red.ISO: VIS.ISO(q:q)
##                0.41504763
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

Using spectral data stored in numeric vectors:

```
with(sun.spct,
     photon_ratio(w.length, s.e.irrad,  Red("Smith10"), Far_red("Smith10")))

## [1] 1.275899
```

or using the predefined convenience function R_FR_ratio:

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

## [1] 1.275899
```

10.10 Task: energy ratios

An energy ratio, equivalent to ζ can be calculated as follows:

```
e_ratio(sun.spct, Red("Smith10"), Far_red("Smith10"))

##  Red.Smith10: FarRed.Smith10(e:e)
##                          1.401143
## attr(,"radiation.unit")
## [1] "e:e ratio"
```

other examples in section 10.9 above, can be easily edited to use `e_ratio` instead of `q_ratio`.

Using spectral data stored in vectors:

```
with(sun.spct,
     energy_ratio(w.length, s.e.irrad,
                  Red("Smith10"), Far_red("Smith10")))

## [1] 1.411385
```

For this infrequently used ratio, no pre-defined function is provided.

10.11 Task: calculate average number of photons per unit energy

When comparing photo-chemical and photo-biological responses under different light sources it is of interest to calculate the photons per energy in mol J^{-1}. In this case only one waveband definition is used to calculate the quotient:

$$\bar{q}' = \frac{Q_{\lambda_1<\lambda<\lambda_2}}{E_{\lambda_1<\lambda<\lambda_2}} \tag{10.4}$$

From this equation it follows that the value of the ratio will depend on the shape of the emission spectrum of the radiation source. For example, for PAR the R code is:

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

##    q:e( PAR)
## 4.547199e-06
## attr(,"radiation.unit")
## [1] "q:e ratio"
```

for obtaining the same quotient in $\mu\text{mol J}^{-1}$ we just need to multiply by 1×10^6,

```
qe_ratio(sun.spct, PAR()) * 1e6

## q:e( PAR)
##  4.547199
## attr(,"radiation.unit")
## [1] "q:e ratio"
```

The seldom needed inverse ratio in J mol^{-1} can be calculated with function `eq_ratio`.

Both functions accept lists of wavebands, so several ratios can be calculated with a single function call:

```
qe_ratio(sun.spct, VIS_bands())

## q:e( Purple.ISO)   q:e( Blue.ISO)
##     3.433901e-06     3.968590e-06
##  q:e( Green.ISO) q:e( Yellow.ISO)
##     4.469290e-06     4.851392e-06
## q:e( Orange.ISO)    q:e( Red.ISO)
##     5.020950e-06     5.682783e-06
## attr(,"radiation.unit")
## [1] "q:e ratio"
```

The same ratios can be calculated for data stored in numeric vectors using function `photons_energy_ratio`:

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

## [1] 4.543762e-06
```

For obtaining the same quotient in $\mu mol\,J^{-1}$ from spectral data in $W\,m^{-2}\,nm^{-1}$ we just need to multiply by 1×10^{6}:

```
with(sun.spct,
     photons_energy_ratio(w.length, s.e.irrad, PAR())) * 1e6

## [1] 4.543762
```

10.12 Task: calculate the contribution of different regions of a spectrum to energy irradiance

It can be of interest to split the total (energy) irradiance into adjacent regions delimited by arbitrary wavelengths. When working with `source_spct` objects, the best way to achieve this is to combine the use of the functions `e_irrad` and `split_bands` already described above, for example,

```
e_irrad(sun.spct, split_bands(c(400, 500, 600, 700)))

##  range.400.500  range.500.600  range.600.700
##       69.69042       68.48951       58.45435
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or

```
e_irrad(sun.spct, split_bands(PAR(), length.out=3))

##  range.400.500  range.500.600  range.600.700
##       69.69042       68.48951       58.45435
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or

```
my_bands <- split_bands(PAR(), length.out=3)
e_irrad(sun.spct, my_bands)

##  range.400.500  range.500.600  range.600.700
##       69.69042       68.48951       58.45435
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

For the example immediately above, we can calculate relative values as

```
e_irrad(sun.spct, my_bands) / e_irrad(sun.spct, PAR())

##  range.400.500  range.500.600  range.600.700
##      0.3544164      0.3483091      0.2972745
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or more efficiently as

```
irradiances <- e_irrad(sun.spct, my_bands)
irradiances / sum(irradiances)

##  range.400.500  range.500.600  range.600.700
##      0.3544164      0.3483091      0.2972745
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

The examples above use short names, the default, but longer names are also available,

```
e_irrad(sun.spct, split_bands(c(400, 500, 600, 700), short.names=FALSE))

##  range.400.500  range.500.600  range.600.700
##       69.69042       68.48951       58.45435
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

e_irrad(sun.spct, split_bands(PAR(), short.names=FALSE, length.out=3))

##  range.400.500  range.500.600  range.600.700
##       69.69042       68.48951       58.45435
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

With spectral data stored in numeric vectors, we can use function `energy_irradiance` together with function `split_bands` or we can use the convenience function `split_energy_irradiance` to obtain to energy of each of the regions delimited by the values in nm supplied in a numeric vector:

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400, 500, 600, 700)))

## range.400.500 range.500.600 range.600.700
##      69.63243      68.53291      58.53508
```

It possible to obtain the 'split' as a vector of fractions adding up to one,

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400, 500, 600, 700),
                             scale="relative"))

## range.400.500 range.500.600 range.600.700
##     0.3540024     0.3484126     0.2975849
```

or as percentages:

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400, 500, 600, 700),
                             scale="percent"))

## range.400.500 range.500.600 range.600.700
##      35.40024      34.84126      29.75849
```

If the 'limits' cover only a region of the spectral data, relative and percent values will be calculated with that region as a reference.

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400,500,600,700),
                             scale="percent"))

## range.400.500 range.500.600 range.600.700
##      35.40024      34.84126      29.75849
```

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400,500,600),
                             scale="percent"))

## range.400.500 range.500.600
##       50.3979       49.6021
```

A vector of two wavelengths is valid input, although not very useful for percentages:

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400, 700),
                             scale="percent"))
```

```
## range.400.700
##           100
```

In contrast, for `scale="absolute"`, the default, it can be used as a quick way of calculating an irradiance for a range of wavelengths without having to define a `waveband`:

```
with(sun.spct,
     split_energy_irradiance(w.length, s.e.irrad,
                             c(400, 700)))

## range.400.700
##      196.7004
```

```
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

CHAPTER 11

Weighted and effective irradiance

Abstract

In this chapter we explain how to calculate weighted energy and photon irradiances from spectral irradiance.

11.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```r
library(photobiology)
library(photobiologyWavebands)
```

11.2 Introduction

Weighted irradiance is usually reported in weighted energy units, but it is also possible to use weighted photon based units. In practice the R code to use is exactly the same as for unweighted irradiances, as all the information needed for applying weights is stored in the `waveband` object. An additional factor comes into play and it is the *normalization wavelength*, which is accepted as an argument by the predefined waveband creation functions that describe biological spectral weighting functions (BSWFs). The focus of this chapter is on the differences between calculations for weighted irradiances compared to those for un-weighted irradiances described in chapter 10. In particular it is important that you read sections **??**, 10.7, on the calculation of irradiances from spectral irradiances and sections 10.3, and 10.4 before reading the present chapter.

Most SWFs are defined using measured action spectra or spectra derived by combining different measured action spectra. As these spectra have been measured under different conditions, what is of interest is the shape of the curve as a function of wavelength, but not the absolute values. Because of this, SWFs are normalized to an action of one at an arbitrary wavelength. In many cases there is no consensus about the wavelength to use. Normalization is simple, it consists in dividing all action values along the curve by the action value at the selected normalization wavelengths.

Another complication is that it is not always clear if a given SWF definition is based on energy or photon units for the fluence rate or irradiances. In photobiology using photon units for expressing action spectra is the norm, but SWFs based on them have rather frequently been used as weights for spectral energy irradiance. The current package makes this difference explicit, and uses the correct weights depending on the spectral data, as long as the `waveband` objects have been correctly defined. In the case of the definitions in package photobiologyWavebands, we have used, whenever possible the correct interpretation when described in the literature, or the common practice when information has been unavailable.

11.3 Task: specifying the normalization wavelength

Several constructors for SWF-based `waveband` objects are supplied. Most of them have parameters, in most cases with default arguments, so that different common uses and misuses in the literature can be reproduced. For example, function `GEN.G()` is predefined in package photobiologyWavebands as a convenience function for Green's formulation of Caldwell's generalized plant action spectrum (GPAS) **Green198x**

```
e_irrad(sun.spct, GEN.G())

##  GEN.G.300.tr.lo
##        0.1028376
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

The code above uses the default normalization wavelength of 300 nm, which is almost universally used nowadays, but not the value used in the original publication (**Caldwell1973**). Any arbitrary wavelength (nm), within the range of the waveband is accepted as `norm` argument:

```
range(GEN.G())

## [1] 275.0 313.3

e_irrad(sun.spct, GEN.G(280))

##  GEN.G.280.tr.lo
##       0.02397115
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

11.4 Task: use of weighted wavebands

Please, consult the documentation of package photobiologyWavebands for a list of predefined constructor functions for weighted wavebands. Here we will present just a few examples of their use. We usually think of weighted irradiances as being defined only by the weighting function, however, as mentioned above, in many cases different normalization wavelengths are in use, and the result of calculations depends very strongly on which wavelength is used for normalization. In a few cases different mathematical formulations are available for the 'same' SWF, and the differences among them can be also important. In such cases separate functions are provided for each formulation (e.g. GEN.N and GEN.T for Green's and Thimijan's formulations of Caldwell's GPAS).

```
GEN.G()

## GEN.G.300
## low (nm) 275
## high (nm) 313
## weighted SWF
## normalized at 300 nm

GEN.T()

## GEN.T.300
## low (nm) 275
## high (nm) 345
## weighted SWF
## normalized at 300 nm
```

We can use one of the predefined functions to create a new waveband object, which as any other R object can be assigned to a variable:

```
cie <- CIE()
cie

## CIE98.298
## low (nm) 250
## high (nm) 400
## weighted SWF
## normalized at 298 nm
```

As described in section 10.3, there are several methods for querying and printing waveband objects. The same functions described for un-weighted waveband objects can be used with any waveband object, including those based on SWFs.

11.5 Task: define wavebands that use weighting functions

In sections ?? and 7.8 we briefly introduced functions waveband and new_waveband, and here we describe their use in full detail. Most users are unlikely to frequently need to define new waveband objects as common SWFs are already defined in package photobiologyWavebands.

Although the constructors are flexible, and can automatically handle both definitions based on action or response spectra in photon or energy units, some care is needed when performance is important.

When defining a new weighted waveband, we need to supply to the constructor more information than in the case on un-weighted wavebands. We start with a simple 'toy' example:

```
toy.wb <- waveband(c(400,700), weight="SWF",
                   SWF.e.fun=function(wl){(wl / 550)^2},
                   norm=550, SWF.norm=550,
                   wb.name="TOY")
toy.wb

## TOY
## low (nm) 400
## high (nm) 700
## weighted SWF
## normalized at 550 nm
```

where the first argument is the range of wavelengths included, `weight="SWF"` indicates that spectral weighting will be used, `SWF.e.fun=function(wl)wl * 2 / 550` supplies an 'anonymous' spectral weighting function based on energy units, `norm=550` indicates the default normalization wavelength to use in calculations, `SWF.norm=550` indicates the normalization wavelength of the output of the SWF, and `wb.name="TOY"` gives a name for the waveband.

In the example above the constructor generates automatically the SWF to use with spectral photon irradiance from the function supplied for spectral energy irradiance. The reverse is true if only an SWF for spectral photon irradiance is supplied. If both functions are supplied, they are used, but no test for their consistency is applied.

11.6 Task: calculate effective energy irradiance

We can use the waveband object defined above in calculations:

```
e_irrad(sun.spct, toy.wb)

##      TOY
## 196.9238
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

Just in the same way as we can use those created with the specific constructors, including using anonymous objects created on the fly:

```
e_irrad(sun.spct, CIE())

##  CIE98.298.tr.lo
##       0.08177754
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or lists of wavebands, such as

```
e_irrad(sun.spct, list(GEN.G(), GEN.T()))

##  GEN.G.300.tr.lo  GEN.T.300.tr.lo
##        0.1028376        0.1473586
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

or

```
e_irrad(sun.spct, list(GEN.G(280), GEN.G(300)))

##  GEN.G.280.tr.lo  GEN.G.300.tr.lo
##       0.02397115       0.10283765
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

Nothing prevents the user from defining his or her own `waveband` object constructors for new SWFs, and making this easy was an important goal in the design of the packages.

11.7 Task: calculate effective photon irradiance

All what is needed is to use function `q_irrad` instead of `e_irrad`. However, one should think carefully if such a calculation is what is needed, as in some research fields it is rarely used, even when from the theoretical point of view would be in most cases preferable.

```
q_irrad(sun.spct, GEN.G())

##  GEN.G.300.tr.lo
##     2.578929e-07
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

11.8 Task: calculate daily effective energy exposure

To calculate daily exposure values, we need to apply the same code as used above, but using spectral daily exposure instead of spectral irradiance as starting point:

```
e_irrad(sun.daily.spct, GEN.G())

##  GEN.G.300.tr.lo
##         2786.987
## attr(,"time.unit")
## [1] "day"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

the output from the code above is in units of $\mathrm{J\,m^{-2}\,d^{-1}}$, the code below returns the same result in the more common uints of $\mathrm{kJ\,m^{-2}\,d^{-1}}$:

```
e_irrad(sun.daily.spct, GEN.G()) * 1e-3

##  GEN.G.300.tr.lo
##         2.786987
## attr(,"time.unit")
## [1] "day"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

by comparing these result to those for effective irradiances above, it can be seen that the `time.unit` attribute of the spectral data is copied to the result, allowing us to distinguish irradiance values (`time.unit="second"`) from daily exposure values (`time.unit="day"`).

```
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

CHAPTER 12

Transmission and reflection

Abstract

In this chapter we explain how to do calculations related to the description of absortion and reflection of UV and VIS radiation.

12.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```r
library(photobiology)
library(photobiologyWavebands)
library(photobiologyFilters)
library(photobiologyLEDs)
```

12.2 Introduction

12.3 Task: absorbance and transmittance

Transmittance is defined as:

$$\tau(\lambda) = \frac{I}{I_0} = \frac{E(\lambda)}{E_0(\lambda)} = \frac{Q(\lambda)}{Q_0(\lambda)} \tag{12.1}$$

Given this simple relation $\tau(\lambda)$ can be calculated as a division between two "source_spct" objects. This gives the correct answer, but as an object of class "source.scpt".

```r
tau <- spc_above / spc_below
```

Absorptance is just $1 - \tau(\lambda)$, but should be distinguished from absorbance ($A(\lambda)$) which is measured on a logarithmic scale:

$$A(\lambda) = -\log_{10}\frac{I}{I_0} \tag{12.2}$$

In chemistry 10 is always used as the base of the logarithm, but in other contexts sometimes e is used as base.

Given the simple equation, $A(\lambda)$ can be also easily calculated using the operators for spectra. This gives the correct answer, but in an object of class "source.scpt".

The conversion between $\tau(\lambda)$ and $A(\lambda)$ is:

$$A(\lambda) = -\log_{10}\tau(\lambda) \tag{12.3}$$

which in S language is:

```
my_T2A <- function(x) {-log10(x)}
```

The conversion between $A(\lambda)$ and $\tau(\lambda)$ is:

$$\tau(\lambda) = 10^{-A(\lambda)} \tag{12.4}$$

which in S language is:

```
my_A2T <- function(x) {10^-x}
```

Instead of these functions, the package defines generic functions and specialized functions, that can be used on vectors and on `filter_spct` objects. Then functions defined above could be directly applied to vectors but doing this on a column in a `filter_spct` is more cumbersome. As the spectra objects are data.tables, one can add a new column, say with transmittances to a copy of the filter data as follows.

```
my_gg400.spct <- copy(gg400.spct)
my_gg400.spct[ , A := T2A(Tfr)]

##      w.length   Tfr A
##   1:      200 1e-05 5
##   2:      210 1e-05 5
##  ---
## 179:     5100 1e-05 5
## 180:     5150 1e-05 5

my_gg400.spct

##      w.length   Tfr A
##   1:      200 1e-05 5
##   2:      210 1e-05 5
##  ---
## 179:     5100 1e-05 5
## 180:     5150 1e-05 5
```

12.4 Task: spectral absorbance from spectral transmittance

Using `filter_spct` objects, the calculations become very simple.

```
my_gg400.spct <- copy(gg400.spct)
T2A(my_gg400.spct)

##      w.length   Tfr A
##   1:       200 1e-05 5
##   2:       210 1e-05 5
##  ---
## 179:      5100 1e-05 5
## 180:      5150 1e-05 5

a.gg400.spct <- T2A(my_gg400.spct, action="replace")
```

12.5 Task: spectral transmittance from spectral absorbance

```
A2T(a.gg400.spct)

##      w.length A   Tfr
##   1:       200 5 1e-05
##   2:       210 5 1e-05
##  ---
## 179:      5100 5 1e-05
## 180:      5150 5 1e-05

A2T(a.gg400.spct, action="replace")

##      w.length   Tfr
##   1:       200 1e-05
##   2:       210 1e-05
##  ---
## 179:      5100 1e-05
## 180:      5150 1e-05
```

12.6 Task: reflected or transmitted spectrum from spectral reflectance and spectral irradiance

When we multiply a `source_spct` by a `filter_spct` or by a `reflector_spct` we obtain as a result a new `source_spct`.

```
class(sun.spct)

## [1] "source_spct"  "generic_spct" "data.table"
## [4] "data.frame"

class(gg400.spct)

## [1] "filter_spct"  "generic_spct" "data.table"
## [4] "data.frame"
```

```
my_sun.spct <- copy(sun.spct)
my_gg400.spct <- copy(gg400.spct)
filtered_sun.spct <- sun.spct * gg400.spct
class(filtered_sun.spct)

## [1] "source_spct"  "generic_spct" "data.table"
## [4] "data.frame"

head(filtered_sun.spct)

##    w.length     s.e.irrad
## 1:      293  2.609665e-11
## 2:      294  6.142401e-11
## 3:      295  2.176175e-10
## 4:      296  6.780119e-10
## 5:      297  1.533491e-09
## 6:      298  3.669677e-09
```

The result of the calculation can be directly used as an argument, for example, when calulating irradiance.

```
q_irrad(sun.spct, UV()) * 1e6

##  UV.ISO.tr.lo
##      86.49495
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

q_irrad(my_sun.spct, UV()) * 1e6

##  UV.ISO.tr.lo
##      86.49495
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

q_irrad(filtered_sun.spct, UV()) * 1e6

##  UV.ISO.tr.lo
##      3.651128
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

q_irrad(sun.spct * gg400.spct, UV()) * 1e6

##  UV.ISO.tr.lo
##      3.651128
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

q_irrad(my_sun.spct * my_gg400.spct, UV()) * 1e6
```

```
##  UV.ISO.tr.lo
##      3.651128
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

```
q_irrad(my_sun.spct * my_gg400.spct) * 1e6

##    Total
## 1135.601
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"

q_irrad(my_sun.spct * my_gg400.spct,
             new_waveband(min(sun.spct), max(sun.spct))) * 1e6

##  range.293.800
##       1135.601
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

Remember, thet if we want to predict the output of a light source composed of different lamps or LEDs we can add the individual spectral irradiance, but using data measured from the target positions of each individaul light source. If we want then to add the effect of a filter we must multiply by the filter transmittance.

In the current version of package `photobiology` the operator is “chosen” based on the first operand. For this reason, when including a numeric operand, it should always be the second operand of binary operators for spectra.

```
# not working
my_luminaire <-
  (0.5 * Norlux_B.spct + Norlux_R.spct) *  PLX0A000_XT.spct
my_luminaire

##       w.length s.e.irrad
##    1:   200.00         0
##    2:   200.47         0
##   ---
## 2358:   937.00         0
## 2359:   937.34         0

# works fine
my_luminaire <-
  (Norlux_B.spct * 0.5 + Norlux_R.spct) *  PLX0A000_XT.spct
my_luminaire
```

```
##       w.length s.e.irrad
##    1:   200.00         0
##    2:   200.47         0
##   ---
## 2358:   937.00         0
## 2359:   937.34         0

q_ratio(my_luminaire,
             list(Red(), Blue(), Green()), PAR())

##    Red.ISO: PAR(q:q)   Blue.ISO: PAR(q:q)
##          0.816195602          0.146121825
##  Green.ISO: PAR(q:q)
##          0.003908976
## attr(,"radiation.unit")
## [1] "q:q ratio"

q_irrad(my_luminaire,
             list(PAR(), Red(), Blue(), Green())) *  1e6

##          PAR      Red.ISO     Blue.ISO
## 1.591314e-02 1.298824e-02 2.325257e-03
##    Green.ISO
## 6.220409e-05
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

12.7 Task: total spectral transmittance from internal spectral transmittance and spectral reflectance

12.8 Task: combined spectral transmittance of two or more filters

12.8.1 Ignoring reflectance

12.8.2 Considering reflectance

12.9 Task: light scattering media (natural waters, plant and animal tissues)

```
try(detach(package:photobiologyFilters))
try(detach(package:photobiologyLEDs))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

CHAPTER 13

Astronomy

Abstract

In this chapter we explain how to code some astronomical computations in R.

13.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)
library(lubridate)

##
## Attaching package:  'lubridate'
##
## The following objects are masked from 'package:data.table':
##
##     hour, mday, month, quarter, wday, week,
##     yday, year

library(ggplot2)
library(ggmap)
```

13.2 Introduction

This chapter deals with calculations that require times and/or dates as arguments. One could use R's built-in functions for POSIXct but package `lubridate` makes working with dates and times, much easier. Package `lubridate` defines functions for decoding dates represented as character strings, and for manipulating dates and doing calculations on dates. Each one of the different functions shown in the code chunk below can decode dates in different formats as long as the year, month and date order in the string agrees with the name of the function.

```
ymd("20140320")

## [1] "2014-03-20 UTC"

ymd("2014-03-20")

## [1] "2014-03-20 UTC"

ymd("14-03-20")

## [1] "2014-03-20 UTC"

ymd("2014-3-20")

## [1] "2014-03-20 UTC"

ymd("2014/3/20")

## [1] "2014-03-20 UTC"

dmy("20.03.2014")

## [1] "2014-03-20 UTC"

dmy("20032014")

## [1] "2014-03-20 UTC"

mdy("03202014")

## [1] "2014-03-20 UTC"
```

Similar functions including hours, minutes and seconds are defined by `lubridate` as well as functions for manipulating dates, and calculating durations with all the necessary and non-trivial corrections needed for leap years, summer time, and other idiosyncracies of the calendar system.

For astronomical calculations we also need as argument the geographical coordinates. It is, of course, possible to enter latitude and longitude values recorded with a GPS instrument or manually obtained from a map. However, when the location is searchable through Google Maps, it is also possible to obtain the coordinates by means of a query from within R using packages `RgoogleMaps`, or package `ggmap`, as done here. When inputing coordinate values manually, they should in degrees as numeric values (in other words the fractional part is given as part of floating point number in degrees, and not as separate integers representing minutes and seconds of degree).

```
geocode("Helsinki")

##        lon      lat
## 1 24.94102 60.17332
```

```
geocode("Viikinkaari 1, 00790 Helsinki, Finland")

##        lon     lat
## 1 25.01673 60.2253
```

13.3 Task: calculating the length of the photoperiod

Functions `day_length` and `night_length` have same parameter signature. They are vectorized for the `date` parameter.

Northern hemisphere latitudes are given as positive numbers and Southern hemisphere latitudes are given as negative numbers, in degrees, possibly with decimal fractions. The default date is `today`.

```
day_length(lat =  60, lon = 0)

## [1] 18.4526

day_length(lat = -60, lon = 0)

## [1] 5.550798
```

Longitudes can be given similarly, with East of Greenwich being negative and West of Greenwhich positive.

Function `geocode` from package `ggmap` returns suitable values in a `data.frame` based on search term(s). It uses Google to do the search, so some use restrictions apply.

```
my.city <- geocode('helsinki')
my.city

##        lon      lat
## 1 24.94102 60.17332
```

We can calculate the photoperiod for the current day as

```
day_length(lon = my.city$lon, lat = my.city$lat)

## [1] 18.51221
```

Or also give a date explicitly using functions from package `lubridate`.

```
day_length(ymd("2015-06-09", tz = "EET"),
           lon = my.city$lon, lat = my.city$lat)

## [1] 18.33841

day_length(dmy("9.6.2015", tz = "EET"),
           lon = my.city$lon, lat = my.city$lat)

## [1] 18.33841
```

The complementary function `night_length` gives

```
night_length(ymd("2015-06-09", tz = "EET"),
             lon = my.city$lon, lat = my.city$lat)

## [1] 5.661595
```

It is also possible to use a vector of dates, for example created as a sequence in the next chunk using functions from package `lubridate`.

Default time zone of `ymd` is UTC or GMT, but one should set the same time zone as will be used for further calculations.

```
dates <- seq(from = ymd("2015-01-01", tz = "EET"),
             to = ymd("2015-12-31", tz = "EET"),
             by = "7 day")
photoperiods.df <-
  data.frame(date = dates,
             photoperiod = day_length(dates,
                                      lon = my.city$lon,
                                      lat = my.city$lat))
```

The 10 lines at the top of the output are

```
head(photoperiods.df, 10)

##          date photoperiod
## 1  2015-01-01    5.630232
## 2  2015-01-08    5.930886
## 3  2015-01-15    6.347161
## 4  2015-01-22    6.849658
## 5  2015-01-29    7.412069
## 6  2015-02-05    8.013514
## 7  2015-02-12    8.638686
## 8  2015-02-19    9.277017
## 9  2015-02-26    9.921603
## 10 2015-03-05   10.568204
```

A further option described in section 13.4 allow setting the twilight angle to be used for the day length calculations.

13.4 Task: Calculating times of sunrise, solar noon and sunset

Functions `sunrise_time`, `sunset_time`, and `noon_time` have all the same parameter signature.

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 accordingly. In the examples below this can be recognized for example, by the time zone being reported as EEST instead of EET for Eastern Europe.

The default for `date` is the current day in time zone UTC.

```
sunrise_time(lat = 60)

## [1] "2015-06-16 02:47:13 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(tz = "UTC"), lat = 60, lon = 0, tz = "UTC")

## [1] "2015-06-15 02:47:32 UTC"

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

## [1] "2015-06-16 04:07:15 EEST"
```

Finally 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"), lat = 60, lon = 25, tz = "EET",
             twilight = "civil")

## [1] "2015-06-16 02:12:26 EEST"

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

## [1] NA

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

## [1] "2015-06-16 06:12:07 EEST"
```

We can reuse the array of dates from section 13.3, and the coordinates of Joensuu, to calculate the time at sunrise through the year.

```
time_at_sunrise.df <-
  data.frame(date = dates,
             sunrise_at =
               sunrise_time(dates,
                            lon = my.city$lon, lat =  my.city$lat,
                            tz = "EET", unit.out = "hour"))
```

The 10 lines at the top of the output are

```
head(time_at_sunrise.df, 10)

##          date sunrise_at
## 1  2015-01-01   9.580854
## 2  2015-01-08   9.484315
## 3  2015-01-15   9.322946
## 4  2015-01-22   9.109335
## 5  2015-01-29   8.855265
## 6  2015-02-05   8.570562
## 7  2015-02-12   8.262970
## 8  2015-02-19   7.938459
## 9  2015-02-26   7.601642
## 10 2015-03-05   7.256146
```

Functions `day_night` from our `photobiology` package uses function `sun_angles`, which is a modified version of function `sunAngle` from package `ode`, to calculate the elevation of the sun. We first find local solar noon by finding the maximal solar elevation, and then search for sunrise in the first half of the day and for sunset in the second half, defined based on the local solar noon. Sunset and sunrise are by default based on a solar elevation angle equal to zero. The argument `twilight` can be used to set the angle according to different conventions.

In the examples we use `geocode` to get the latitude and longitude of cities. `geocode` accepts any valid Google Maps search terms, including street addresses, and postal codes within cities. `day_length` returns a numeric vector. This first example is for Buenos Aires on two different dates, by use of the optional argument `tz` we request the results to be expressed in local time for Buenos Aires.

```
geo_code_BA <- geocode("Buenos Aires")
day_night(ymd("2013-12-21"),
          lon = geo_code_BA[["lon"]],
          lat = geo_code_BA[["lat"]],
          tz = "America/Argentina/Buenos_Aires")

## $day
## [1] "2013-12-21"
##
## $sunrise
## [1] "2013-12-21 05:42:00 ART"
##
## $noon
## [1] "2013-12-21 12:51:46 ART"
##
## $sunset
## [1] "2013-12-21 20:01:32 ART"
##
## $daylength
## [1] 14.32535
##
## $nightlength
## [1] 9.674652
```

And with `unit.out` set to `"hour"`

```
day_night(ymd("2013-12-21"),
          lon = geo_code_BA[["lon"]],
          lat = geo_code_BA[["lat"]],
          tz = "America/Argentina/Buenos_Aires",
          unit.out = "hour")

## $day
## [1] "2013-12-21"
##
## $sunrise
## [1] 5.700227
##
## $noon
## [1] 12.86291
##
```

```
## $sunset
## [1] 20.02557
##
## $daylength
## [1] 14.32535
##
## $nightlength
## [1] 9.674652
```

Next, we calculate day length based on different definitions of twilight for Helsinki, at the equinox:

```
geo_code_He <- geocode("Helsinki")
geo_code_He

##        lon      lat
## 1 24.94102 60.17332

day_length(ymd("2013-09-21"),
          lon = geo_code_He[["lon"]], lat = geo_code_He[["lat"]])

## [1] 12.12728

day_length(ymd("2013-09-21"),
          lon = geo_code_He[["lon"]], lat = geo_code_He[["lat"]],
          twilight = "civil")

## [1] 13.74776

day_length(ymd("2013-09-21"),
          lon = geo_code_He[["lon"]], lat = geo_code_He[["lat"]],
          twilight = "nautical")

## [1] 15.43503

day_length(ymd("2013-09-21"),
          lon = geo_code_He[["lon"]], lat = geo_code_He[["lat"]],
          twilight = "astronomical")

## [1] 17.28531
```

Or for a given angle in degrees, which for example can be positive in the case of an obstacle like a building or mountain, instead of negative as used for twilight definitions. In the case of obstacles the angle will be different for morning and afternoon, and can be entered as a numeric vector of length two.

```
day_length(ymd("2013-09-21"),
          lon = geo_code_He[["lon"]], lat = geo_code_He[["lat"]],
          twilight = c(20, 0))

## [1] 9.251774
```

In addition, function `day_night` returns a list containing all the quantities returned by the other functions. As other functions described in this chapter, `day_night` is vectorized for the `date` parameter.

```
day_night(ymd("2013-12-21"),
          lon = geo_code_BA[["lon"]],
          lat = geo_code_BA[["lat"]],
          tz = "America/Argentina/Buenos_Aires")

## $day
## [1] "2013-12-21"
##
## $sunrise
## [1] "2013-12-21 05:42:00 ART"
##
## $noon
## [1] "2013-12-21 12:51:46 ART"
##
## $sunset
## [1] "2013-12-21 20:01:32 ART"
##
## $daylength
## [1] 14.32535
##
## $nightlength
## [1] 9.674652
```

And with `unit.out` set to `"hour"`

```
day_night(ymd("2013-12-21"),
          lon = geo_code_BA[["lon"]],
          lat = geo_code_BA[["lat"]],
          tz = "America/Argentina/Buenos_Aires",
          unit.out = "hour")

## $day
## [1] "2013-12-21"
##
## $sunrise
## [1] 5.700227
##
## $noon
## [1] 12.86291
##
## $sunset
## [1] 20.02557
##
## $daylength
## [1] 14.32535
##
## $nightlength
## [1] 9.674652
```

13.5 Task: calculating the position of the sun

`sun_angles` not only returns solar elevation, but all the angles defining the position of the sun. The time argument to `sun_angles` is internally converted to UTC (universal time coordinates, which is equal to GMT) time zone, so time defined for any time zone is valid input. The time zone used for the output is by default that currently in use in the computer on which R is running, but we can easily specify the time coordinates used for the output with parameter `tz`, using any string accepted by package `lubridate`.

```
geo_code_Jo <- geocode("Joensuu")
geo_code_Jo

##          lon      lat
## 1 29.76353 62.60109

my_time <- ymd_hms("2014-05-29 18:00:00", tz="EET")
sun_angles(my_time,
         lon = geo_code_Jo[["lon"]], lat = geo_code_Jo[["lat"]])

## $time
## [1] "2014-05-29 18:00:00 EEST"
##
## $azimuth
## [1] 267.585
##
## $elevation
## [1] 25.81887
##
## $diameter
## [1] 0.5260482
##
## $distance
## [1] 1.013595
```

We can calculate the current position of the sun, in this case giving the position of the sun in the sky of Joensuu when this .PDF file was generated.

```
sun_angles(now(),
         lon = geo_code_Jo[["lon"]], lat = geo_code_Jo[["lat"]])

## $time
## [1] "2015-06-16 00:28:08 EEST"
##
## $azimuth
## [1] 352.3402
##
## $elevation
## [1] -3.822049
##
## $diameter
## [1] 0.5249084
##
## $distance
## [1] 1.015796
```

13.6 Task: plotting sun elevation through a day

Function `sun_angles` described above is vectorized, so it is very easy to calculate the position of the sun throughout a day at a given location on Earth. The example here uses sun only elevation, plotted for Helsinki through the course of 23 June 2014. We first a vector of times, using `seq` which can not only be used with numbers, but also with dates. Note that `by` is specified as a string.

```
opts_chunk$set(opts_fig_wide)
```

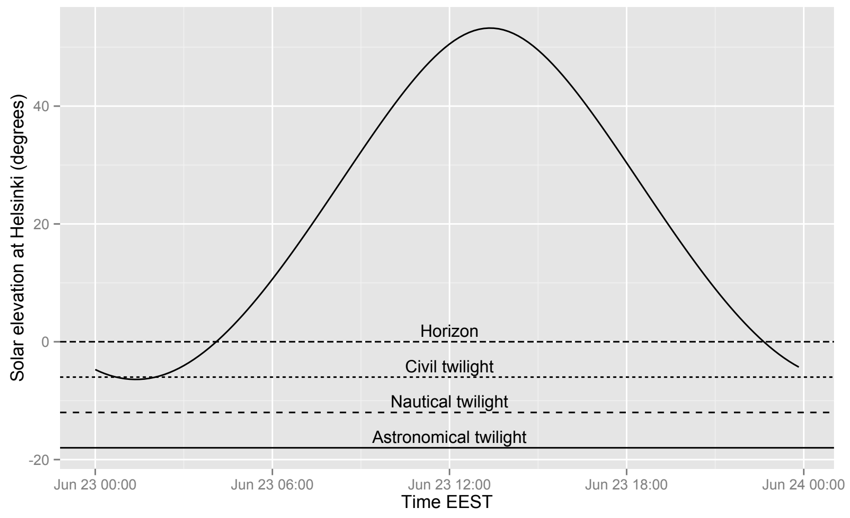
```
hours <- seq(from=ymd("2014-06-23", tz="EET"),
             by="10 min",
             length=24 * 6)
sun_elev_hel <- data.frame(time_eet = hours,
                           elevation =
                             sun_angles(hours,
                                        lon = geo_code_He[["lon"]],
                                        lat = geo_code_He[["lat"]])$elevation,
                           location = "Helsinki",
                           lon = geo_code_He[["lon"]],
                           lat = geo_code_He[["lat"]])
```

We also create a small data frame with data for plotting and labeling the different twilight conventions.

```
twilight <-
  data.frame(angle = c(0, -6, -12, -18),
             label = c("Horizon", "Civil twilight",
                       "Nautical twilight",
                       "Astronomical twilight"),
             time = rep(ymd_hms("2014-06-23 12:00:00",
                                tz="EET"),
                        4) )
```

We draw a plot using the data frames created above.

```
ggplot(sun_elev_hel,
       aes(x = time_eet, y = elevation)) +
  geom_line() +
  geom_hline(data=twilight,
             aes(yintercept = angle, linetype=factor(label))) +
  annotate(geom="text",
           x=twilight$time, y=twilight$angle,
           label=twilight$label, vjust=-0.4, size=4) +
  labs(y = "Solar elevation at Helsinki (degrees)",
       x = "Time EEST")
```



13.7 Task: plotting day length through the year

For this we first need to generate a sequence of dates. We use `seq` as in the previous section, but instead of supplying a length as argument we supply an ending time. Instead of giving `by` in minutes as above, we now use days:

```r
days <- seq(from=ymd("2014-01-01"), to=ymd("2014-12-31"),
            by="3 day")
```

To calculate the length of each day, we need to use an explicit loop as function `day_night` is not vectorized. We repeat the calculations for three locations at different latitudes, then row bind the data frames into a single data frame. Each individual data frame contains information to identify the sites:

```r
geo_code_He <- geocode("Helsinki")
daylengths_hel <-
  data.frame(day = days,
             daylength = day_length(days,
                                    lon = geo_code_He[["lon"]],
                                    lat = geo_code_He[["lat"]],
                                    tz = "EET"),
             location = "Helsinki",
             lon = geo_code_He[["lon"]],
             lat = geo_code_He[["lat"]])
```

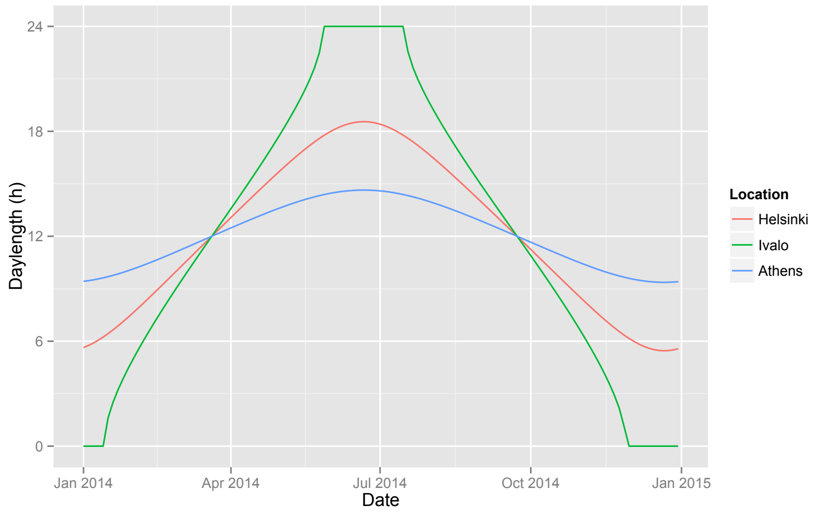
```r
geo_code_Iv <- geocode("Ivalo")
daylengths_ivalo <-
  data.frame(day = days,
             daylength = day_length(days,
                                    lon = geo_code_Iv[["lon"]],
                                    lat = geo_code_Iv[["lat"]],
                                    tz = "EET"),
             location = "Ivalo",
             lon = geo_code_Iv[["lon"]],
                lat = geo_code_Iv[["lat"]])
```

```
geo_code_At <- geocode("Athens, Greece")
daylengths_athens <-
  data.frame(day = days,
             daylength = day_length(days,
                              lon = geo_code_At[["lon"]],
                              lat = geo_code_At[["lat"]],
                              tz = "EET"),
             location = "Athens",
             lon = geo_code_At[["lon"]],
             lat = geo_code_At[["lat"]])
```

```
daylengths <- rbind(daylengths_hel,
                    daylengths_ivalo,
                    daylengths_athens)
```

Once we have the data available, plotting is simple:

```
ggplot(daylengths,
       aes(x = day, y = daylength, colour=factor(location))) +
  geom_line() +
  scale_y_continuous(breaks=c(0,6,12,18,24), limits=c(0,24)) +
  labs(x = "Date", y = "Daylength (h)", colour="Location")
```



13.8 Task: plotting local time at sunrise

For this we reuse `days` from the previous sections. We repeat the calculations for three locations at different latitudes, then row bind the data frames into a single data frame. Data frames contain information to identify the sites:

```
geo_code_He <- geocode("Helsinki")
sunrise_hel <-
  data.frame(day = days,
             sunrise = sunrise_time(days,
                                    lon = geo_code_He[["lon"]],
                                    lat = geo_code_He[["lat"]],
                                    tz = "EET", unit.out = "hour"),
             location = "Helsinki",
             lon = geo_code_He[["lon"]],
             lat = geo_code_He[["lat"]])
```

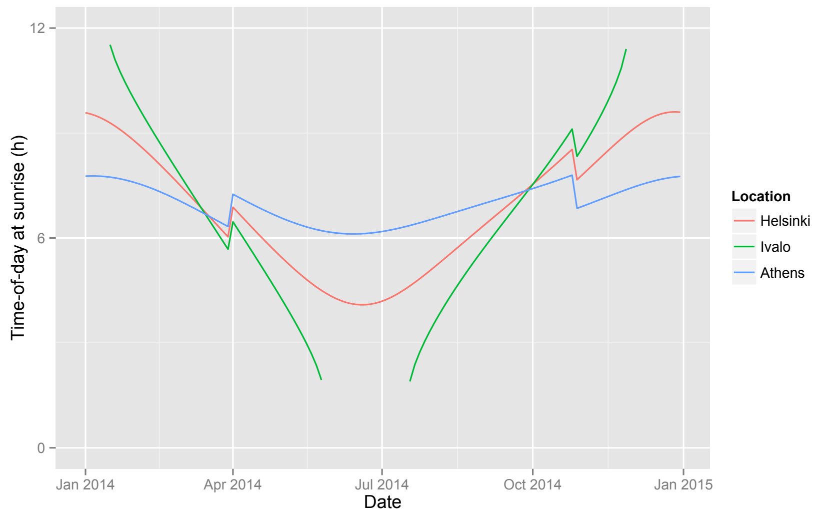
```
geo_code_Iv <- geocode("Ivalo")
sunrise_ivalo <-
  data.frame(day = days,
             sunrise = sunrise_time(days,
                                    lon = geo_code_Iv[["lon"]],
                                    lat = geo_code_Iv[["lat"]],
                                    tz = "EET", unit.out = "hour"),
             location = "Ivalo",
             lon = geo_code_Iv[["lon"]],
             lat = geo_code_Iv[["lat"]])
```

```
geo_code_At <- geocode("Athens, Greece")
sunrise_athens <-
  data.frame(day = days,
             sunrise = sunrise_time(days,
                                lon = geo_code_At[["lon"]],
                                lat = geo_code_At[["lat"]],
                                tz = "EET", unit.out = "hour"),
             location = "Athens",
             lon = geo_code_At[["lon"]],
             lat = geo_code_At[["lat"]])
```

```
sunrises <- rbind(sunrise_hel,
                  sunrise_ivalo,
                  sunrise_athens)
```

Once we have the data available, plotting is simple:

```
ggplot(sunrises,
       aes(x = day, y = sunrise, colour=factor(location))) +
  geom_line() +
  scale_y_continuous(breaks=c(0,6,12), limits=c(0,12)) +
  labs(x = "Date", y = "Time-of-day at sunrise (h)", colour="Location")

## Warning:  Removed 16 rows containing missing values
(geom_path).
```



The breaks in the lines are the result of the changes between winter and summer time coordinates.

```
try(detach(package:photobiology))
try(detach(package:lubridate))
try(detach(package:ggmap))
try(detach(package:ggplot2))
```

CHAPTER 14

Colour

Abstract

In this chapter we explain how to use colours according to visual sensitivity. For example calculating red-green-blue (RGB) values for humans.

14.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```r
library(photobiology)
```

14.2 Introduction

The calculation of equivalent colours and colour spaces is based on the number of photoreceptors and their spectral sensitivities. For humans it is normally accepted that there are three photoreceptors in the eyes, with maximum sensitivities in the red, green, and blue regions of the spectrum.

When calculating colours we can take either only the colour or both colour and apparent luminance. In our functions, in the first case one needs to provide as input 'chromaticity coordinates' (CC) and in the second case 'colour matching functions' (CMF). The suite includes data for humans, but the current implementation of the functions should be able to handle also calculations for other organisms with tri-chromic vision.

The 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 wavelengths, based on the assumption of a flat energy irradiance across this 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. The range of wavelengths used in the calculations is that in the chromaticity data.

One use of these functions is to generate realistic colour for 'key' on plots of spectral data. Other uses are also possible, like simulating how different, different objects would look to a certain organism.

This package is very 'young' so may be to some extent buggy, and/or have rough edges. We plan to add at least visual data for honey bees.

14.3 Task: calculating an RGB colour from a single wavelength

Function `w_length2rgb` must be used in this case. If a vector of wavelengths is supplied as argument, then a vector of `colors`, of the same length, is returned. Here are some examples of calculation of R color definitions for monochromatic light:

```
w_length2rgb(550) # green

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

w_length2rgb(630) # red

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

w_length2rgb(380) # UVA

## wl.380.nm
## "#000000"

w_length2rgb(750) # far red

## wl.750.nm
## "#000000"

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

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

14.4 Task: calculating an RGB colour for a range of wavelengths

Function `w_length_range2rgb` must be used in this case. This function expects as input a vector of two number, as returned by the function `range`. If a longer vector is supplied as argument, its range is used, with a warning. If a vector of lengths one is given as argument, then the same output as from function `w_length2rgb` is returned. This function assumes a flat energy spectral irradiance curve within the range. Some examples: Examples for wavelength ranges:

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

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

w_length_range2rgb(400:700)

## Using only extreme wavelength values.

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

w_length_range2rgb(sun.spct$w.length)

## Using only extreme wavelength values.

## 293-800 nm
##  "#554340"

w_length_range2rgb(550)

## Calculating RGB values for monochromatic light.

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

14.5 Task: calculating an RGB colour for spectrum

Function `s_e_irrad2rgb` in contrast to those described above, when calculating the color takes into account the spectral irradiance.

Examples for spectra, in this case 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"

with(sun.spct,
     s_e_irrad2rgb(w.length, s.e.irrad, sens=ciexyzCMF10.spct))
```

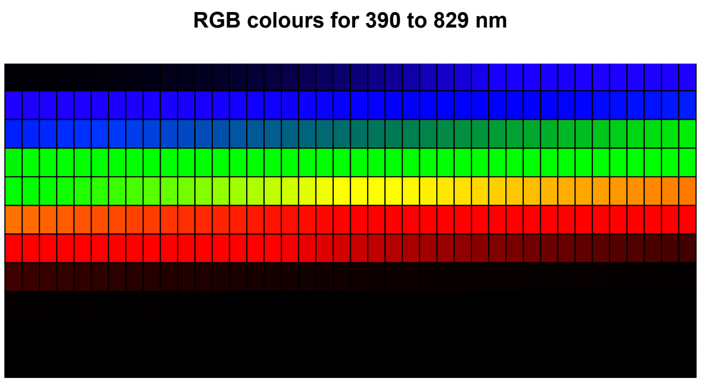
```
## [1] "#59534F"

with(sun.spct,
     s_e_irrad2rgb(w.length, s.e.irrad, sens=ciexyzCC2.spct))

## [1] "#B63C37"

with(sun.spct,
     s_e_irrad2rgb(w.length, s.e.irrad, sens=ciexyzCC10.spct))

## [1] "#BD3C33"
```

Except for the first example, we specificity the visual sensitivity data to use.

14.6 A sample of colours

Here we plot the RGB colours for the range covered by the CIE 2006 proposed standard calculated at each 1 nm step:

```
wl <- c(390, 829)

my.colors <- w_length2rgb(wl[1]:wl[2])

colCount <- 40 # number per row
rowCount <- trunc(length(my.colors) / colCount)

plot( c(1,colCount), c(0,rowCount), type="n",
      ylab="", xlab="",
      axes=FALSE, ylim=c(rowCount,0))
title(paste("RGB colours for",
            as.character(wl[1]), "to",
            as.character(wl[2]), "nm"))

for (j in 0:(rowCount-1))
{
  base <- j*colCount
  remaining <- length(my.colors) - base
  RowSize <-
    ifelse(remaining < colCount, remaining, colCount)
  rect((1:RowSize)-0.5, j-0.5, (1:RowSize)+0.5, j+0.5,
       border="black",
       col=my.colors[base + (1:RowSize)])
}
```



```
try(detach(package:photobiology))
```

CHAPTER 15

Plotting spectra and colours

Abstract

In this chapter we explain how to plot spectra and colours, using packages ggplot2, ggtern, and the functions in our package photobiologygg. Both ggtern for ternary plots and photobiologygg for annotating spectra build new functionality on top of the ggplot2 package. We also use several functions and data from package photobiology in the examples.

15.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(ggplot2)
library(scales)
library(ggtern)

##
## Attaching package:  'ggtern'
##
## The following objects are masked from 'package:ggplot2':
##
##    %+%, %+replace%, aes, calc_element,
##    geom_density2d, geom_segment,
##    geom_smooth, ggplot_build,
##    ggplot_gtable, ggsave, opts,
##    stat_density2d, stat_smooth, theme,
##    theme_bw, theme_classic, theme_get,
##    theme_gray, theme_grey, theme_minimal,
##    theme_set, theme_update

library(gridExtra)
```

```
## Loading required package:  grid

library(photobiology)
library(photobiologyFilters)
library(photobiologyWavebands)
library(photobiologygg)
```

15.2 Introduction to plotting spectra

We show in this chapter examples of how spectral data can be plotted. All the examples are done with package `ggplot2`, sometimes using in addition other packages. `ggplot2` provides the most recent, but stable, type of plotting functionality in R, and is what we use here for most examples. Both `base` graphic functions, part of R itself and 'trellis' graphics provided by package `lattice` are other popular alternatives. The new package `ggvis` uses similar grammar as `ggplot2` but drastically improves on functionality for interactive plots. Several of the functions used in this chapter are extensions to package `ggplot2`[1]

How to depict a spectrum in a figure has to be thought in relation to what aspect of the information we want to highlight. A line plot of a spectrum with peaks and/or valleys labelled highlights the shape of the spectrum, while a spectrum plotted with the area below the curve filled highlights the total energy irradiance (or photon irradiance) for a given region of the spectrum. Adding a bar with the colours corresponding to the different wavelengths, facilitates the reading of the plot for people not familiar with the interpretation on wavelengths expressed in nanometres. Labeling regions of the spectrum with waveband names also facilitates the understanding of plotted spectral data. A basic line plot of spectral data can be easily done with `ggplot2` or any of the other plotting functions in R. In this chapter we focus on how to add to basic line and dot plots all the 'fancy decorations' that can so much facilitate their reading and interpretation.

Towards the end of the chapter we give examples of plotting of RGB (red-green-blue) colours for human vision on a ternary plot, and show how to do a ternary plot for GBU (green-blue-ultraviolet) flower colours for honeybee vision using as reference the reflectance of a background.

If you are not familiar with `ggplot2` and `ggtern` plotting, please read Appendix **??** on page **??** before continuing reading the present chapter.

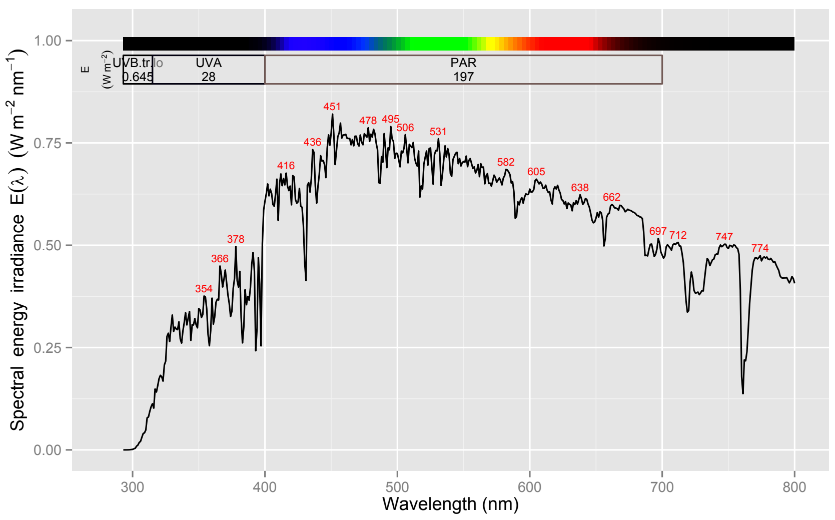
15.3 Task: simple plotting of spectra

Pakage `photobiologygg` defines specializations of the generic `plot` function of R. These functions are available for spectral objects. They return a

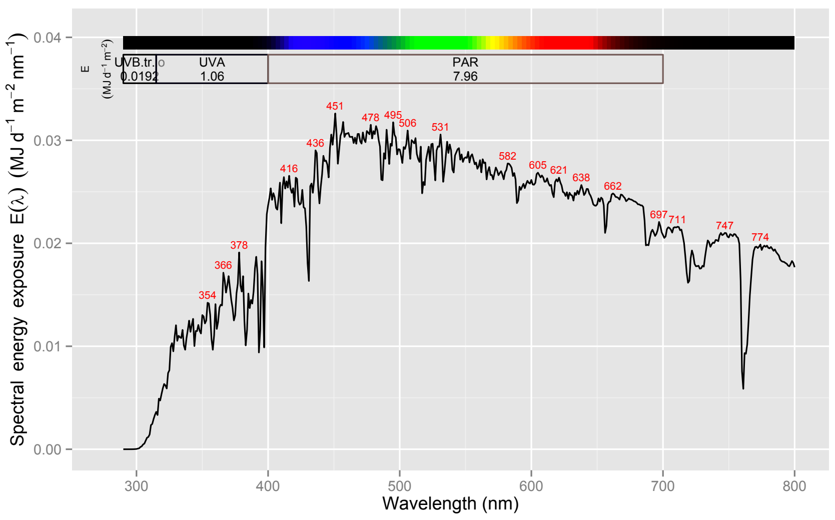
[1] `ggplot2` is feature-frozen, in other words the user interface defined by the functions and their arguments will not change in future versions. Consequently it is a good basis for adding application-specific functionality through separate packages. `ggplot2` uses the *grammar of graphics* for describing the plots. This grammar, because it is consistent, tends to be easier to understand, and makes it easier to design new functionality that uses extensions based on the same 'language grammar' as used by the original package.

ggplot object, to which additional layers can be added if desired. An example of it simplest use follows. As the spectral objects have spectral irradiance expressed in known energy or photon units, and an attribute indicating the time unit, the axis labels are produced automatically. The two plots that follow show spectral irradiance, and spectral daily exposure, respectively.

```
plot(sun.spct)
```

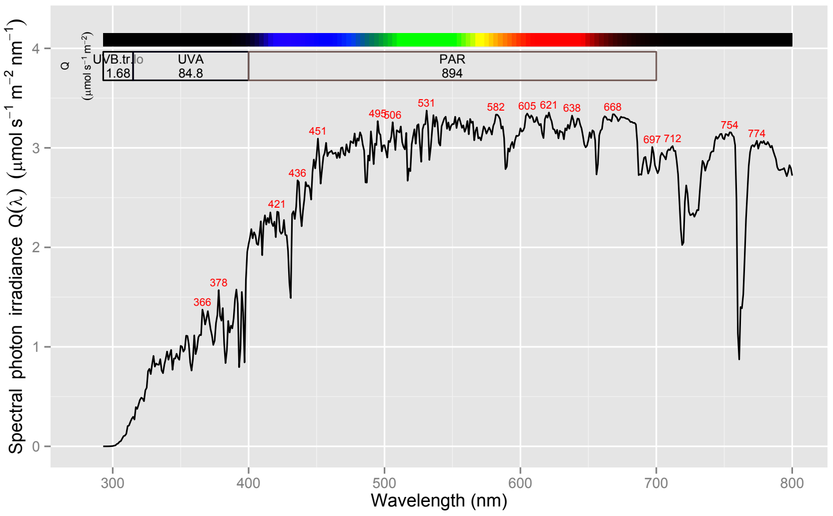


```
plot(sun.daily.spct)
```



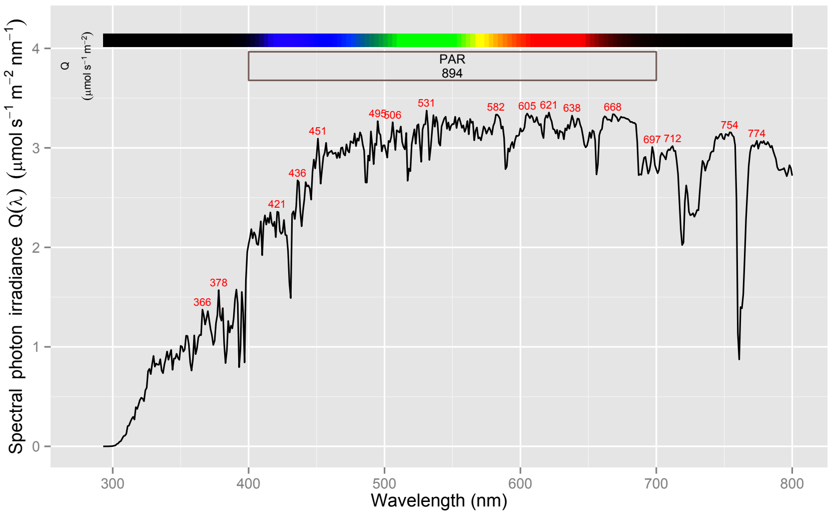
The parameter `unit` can be set to `"photon"` to obtain a plot depicting spectral photon irradiance. This works irrespective of whether the `source_spct` object contains the spectral data in photon or energy units.

```
plot(sun.spct, unit.out = "photon")
```

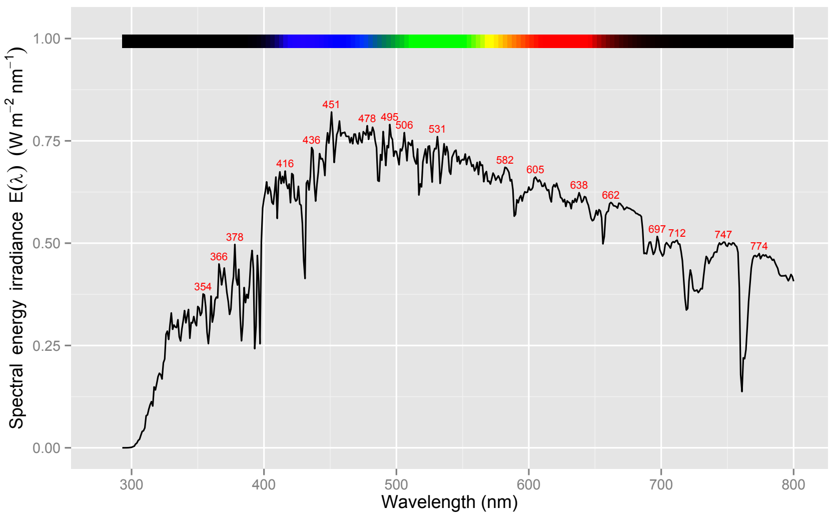


A list of wave bands, or a single wave band, to be used for annotation can be supplied through the `bands` parameter. A NULL waveband results in no waveband labels, while the next example shows how to obtain the total irradiance.

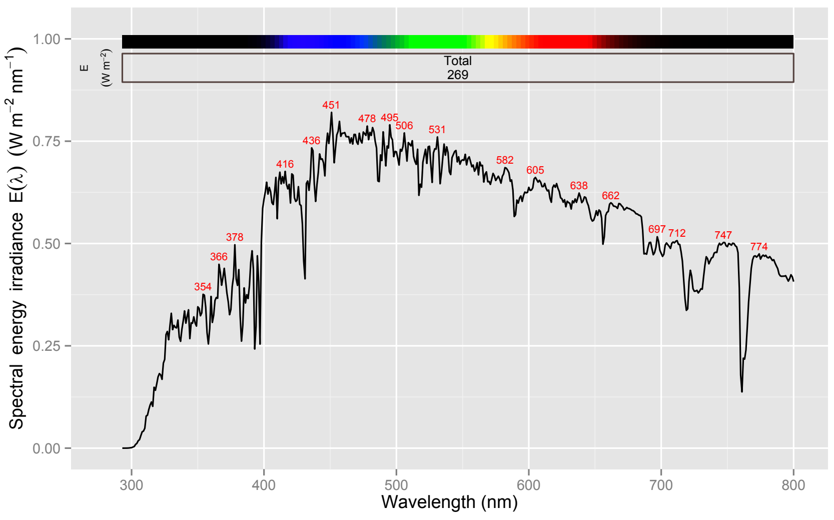
```
plot(sun.spct, w.band = PAR(), unit.out = "photon")
```



```
plot(sun.spct, w.band = NULL)
```

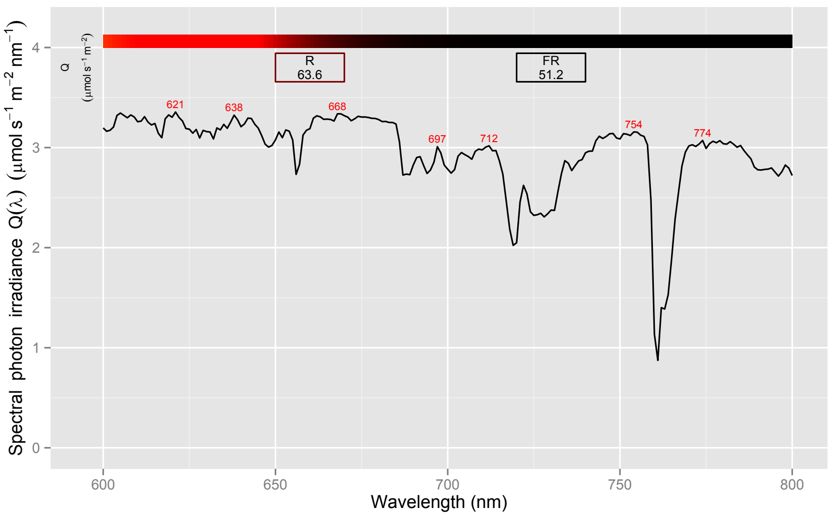


```
plot(sun.spct, w.band = waveband(sun.spct))
```



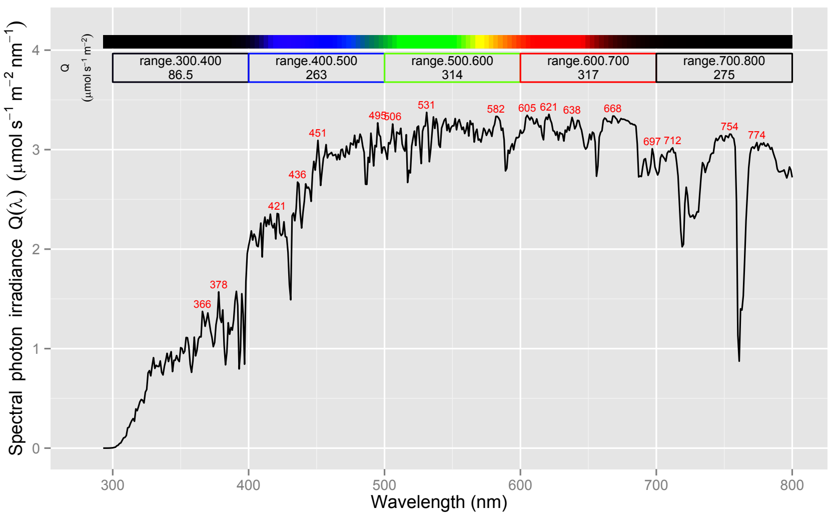
Of course the arguments to these parameters can be supplied in different combinations, and combined with other functions as need. This last example shows how to plot using photon-based units, selecting only a specific region of the spectrum, annotated with the red and far-red photon irradiances, using Prof. Harry Smith's definitions for these two wavebands.

```
plot(trim_spct(sun.spct, waveband(c(600,800))),
     w.band = list(Red("Smith20"), Far_red("Smith20")), unit.out = "photon")
```

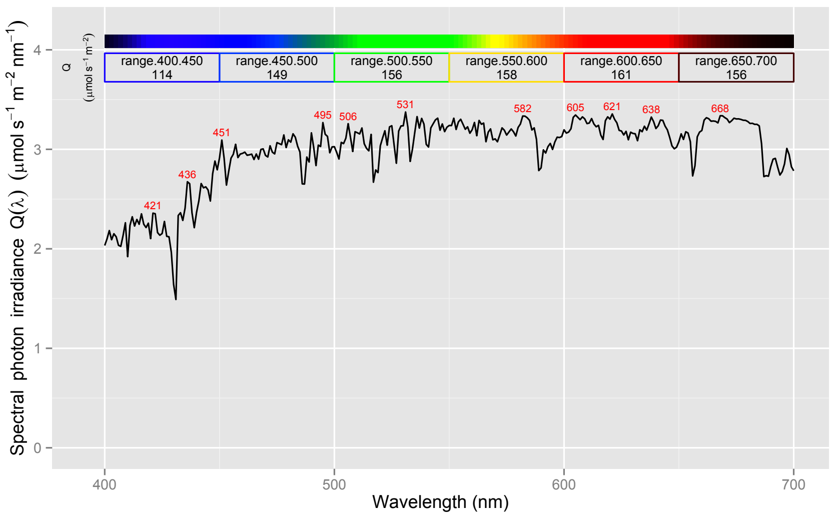


Two final examples show how to annotate a spectrum plot by equal sized wavebands.

```
plot(sun.spct,
     w.band = split_bands(c(300,800), length.out = 5), unit.out = "photon")
```



```
plot(trim_spct(sun.spct, PAR()),
     w.band=split_bands(PAR(), length.out = 6), unit.out = "photon")
```



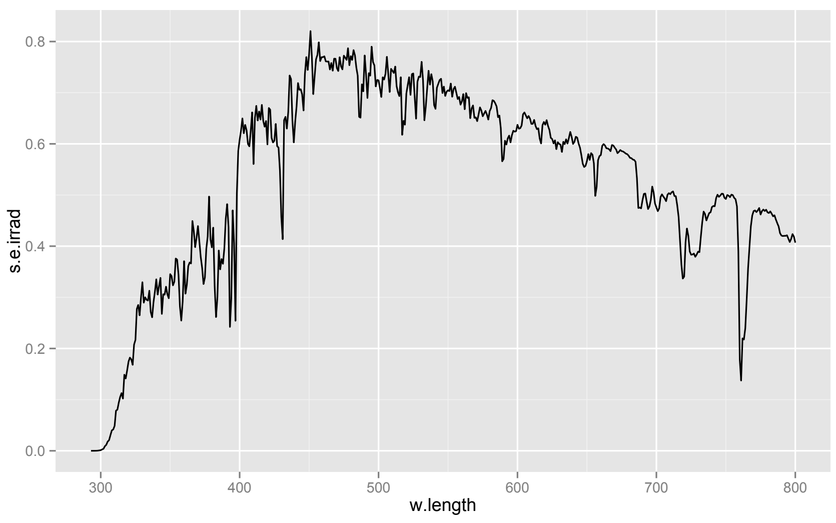
As the current implementation uses annotations rather than a `ggplot` 'statistic', waveband irradiance annotations ignore global aesthetics and facets. If used for simultaneous plotting of several spectra (stored in a single R object), then parameter `w.band` should given NULL as argument.

15.4 Task: plotting spectra with `ggplot2`

We create a simple line plot, assign it to a variable called `fig_sun.e0` and then on the next line `print` it[2]. We obtain a plot with the axis labeled with the names of the variables, which is enough to check the data, but not good enough for publication.

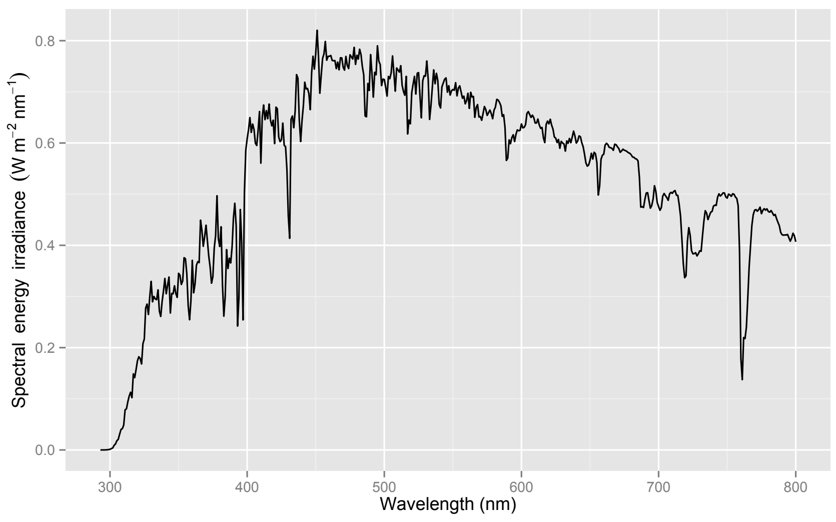
```
fig_sun.e0 <-
  ggplot(data=sun.spct, aes(x=w.length, y=s.e.irrad)) +
  geom_line()

fig_sun.e0
```

[2]we could have used `print(fig_sun.e0)` explicitly, but this is needed only in scripts because printing takes places automatically when working at the R console.



Next we add `labs` to obtain nicer axis labels, instead of assigning the result to a variable for reuse, we print it on-the-fly. As we need superscripts for the y-label we have to use `expression` instead of a character string as we use for the x-label. The syntax of expressions is complex, so please look at `help(plotmath)` and appendix **??** for more details.

```
fig_sun.e0  +
  labs(
    y = expression(Spectral~~energy~~irradiance~~(W~m^{-2}~nm^{-1})),
    x = "Wavelength (nm)")
```



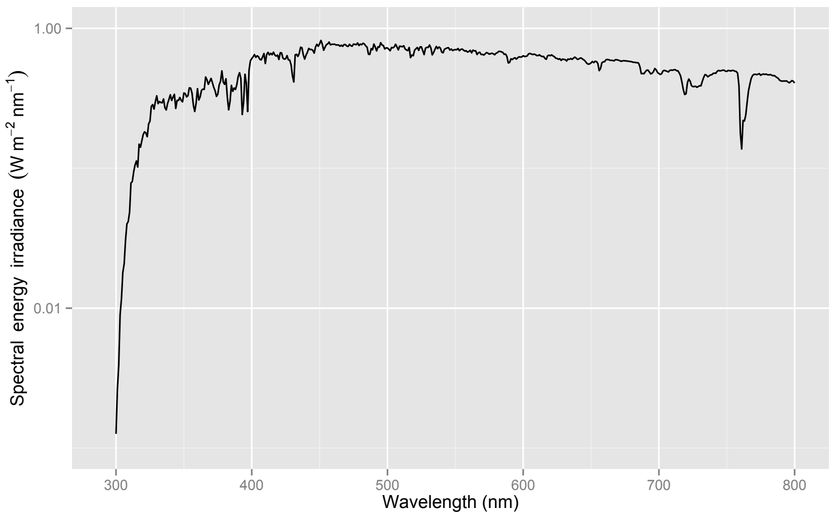
As we are going to re-use the same axis-labels in later plots, it is handy to save their definitions to variables. These definitions will be used in many of this chapter's plots. We also add `atop` to two of the expressions to making shorter versions by setting the spectral irradiance units on a second line in the axis labels.

```
ylab_watt <-
  expression(Spectral~~energy~~irradiance~~(W~m^{-2}~nm^{-1}))
ylab_watt_atop <-
  expression(atop(Spectral~~energy~~irradiance,
                  (W~m^{-2}~nm^{-1})))
ylab_umol <-
  expression(Spectral~~photon~~irradiance~~(mu*mol~m^{-2}~s^{-1}~nm^{-1}))
ylab_umol_atop <-
  expression(atop(Spectral~~photon~~irradiance,
                  (mu*mol~m^{-2}~s^{-1}~nm^{-1})))
xlab_nm <- "Wavelength (nm)"
```

15.5 Task: using a log scale

Here without need to recreate the figure, we add a logarithmic scale for the y-axis and print on the fly the result, and two of the just saved axis-labels. In this case we override the automatic limits of the scale. We do not give further examples of this, but could be also used with later examples, just by adjusting the values used as scale limits.

```
fig_sun.e0 +
  scale_y_log10(limits=c(1e-3, 1e0)) +
  labs(x = xlab_nm, y = ylab_watt)

## Warning:  Removed 7 rows containing missing values
(geom_path).
```



The code above generates some harmless warnings, which are due some y values not being valid input for `log10`, the function used for the re-scaling, or because they fall outside the scale limits.

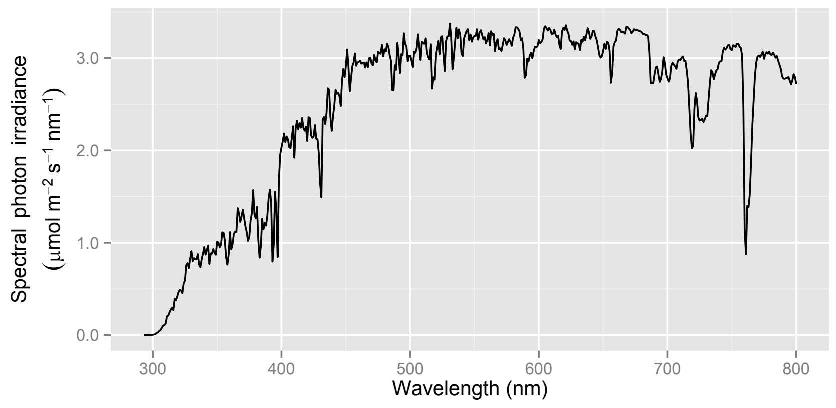
15.6 Task: compare energy and photon spectral units

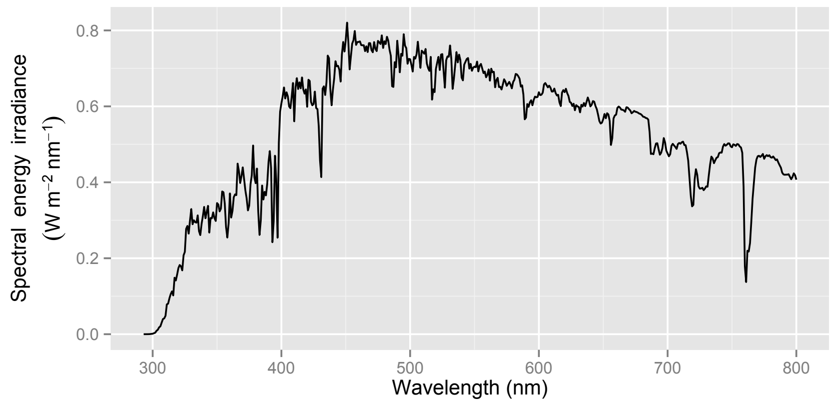
We use once more the axis-labels saved above, but this time use the two-line label for the y-axis. To make sure that the width of the plotting area of both plots is the same, we need to have tick labels of the same width and format in both plots. For this we define a formatting function `num_one_dec` and then use it in the scale definition.

```r
num_one_dec <- function(x, ...) {
  format(x, nsmall=1, trim=FALSE, width=4, ...)
  }

fig_sun.q <-
  ggplot(data=sun.spct, aes(x=w.length, y=s.q.irrad * 1e6)) +
  geom_line() +
  scale_y_continuous(labels = num_one_dec) +
  labs(x = xlab_nm)

fig_sun.e1 <-
  ggplot(data=sun.spct, aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  scale_y_continuous(labels = num_one_dec) +
  labs(x = xlab_nm)
```

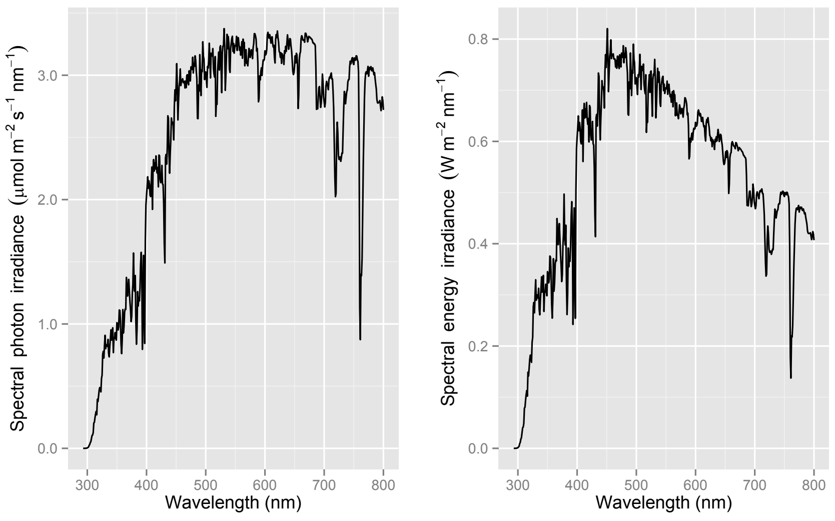
We can use function `multiplot` to make a single plot from two separate ggplots, and put them side by or on top of each other. We use different y-axis labels in the two cases to make better use of the available space.

```r
multiplot(fig_sun.q  + labs(y = ylab_umol_atop),
          fig_sun.e1 + labs(y = ylab_watt_atop),
          cols = 1)
```





```
multiplot(fig_sun.q  + labs(y = ylab_umol),
          fig_sun.e1 + labs(y = ylab_watt),
          cols = 2)
```



15.7 Task: finding peaks and valleys in spectra

We first show the use of function `get_peaks` that returns the wavelengths at which peaks are located. The parameter `span` determines the number of values used to find a local maximum (the higher the value used, the fewer maxima are detected), and the parameter `ignore_threshold` the fraction of the total span along the irradiance that is taken into account (a value of 0.75, requests only peaks in the upper 25% of the y-range to be returned; a value of -0.75 works similarly but for the lower half of the y-range)[3]. It is good to mention that `head` returns the first six rows of its argument, and we use it here just to reduce the length of the output, if you run these examples yourself, you can remove `head` from the code. In the output, x corresponds to wavelength, and y to spectral irradiance, while `label` is a character string with the wavelength, possibly formatted.

```
head(with(sun.spct,
          get_peaks(w.length, s.e.irrad, span=31)))

##     x         y label
## 1 378 0.4969714   378
## 2 416 0.6761818   416
## 3 451 0.8204633   451
## 4 478 0.7869773   478
## 5 495 0.7899872   495
## 6 531 0.7603297   531

head(with(sun.spct,
          get_peaks(w.length, s.e.irrad, span=31,
                    ignore_threshold=0.75)))

##     x         y label
## 1 416 0.6761818   416
## 2 451 0.8204633   451
## 3 478 0.7869773   478
## 4 495 0.7899872   495
## 5 531 0.7603297   531
## 6 582 0.6853736   582
```

The parameter span, indicates the size in number of observations (e.g. number of discrete wavelength values) included in the window used to find local maxima (peaks) or minima (valleys). By providing different values for this argument we can 'adjust' how *fine* or *coarse* is the structure described by the peaks returned by the function. The window is always defined using an odd number of observations, if an even number is provided as argument, it is increased by one, with a warning.

```
head(with(sun.spct,
          get_peaks(w.length, s.e.irrad, span=21)))

##     x         y label
```

[3]In the current example setting `ignore_threshold` equal to 0.75 given that the range of the spectral irradiance data goes from 0.00 $\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}$ to 0.82 $\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}$, causes any peaks having a spectral irradiance of less than 0.62 $\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}$ to be ignored.

```
## 1 354 0.3758625   354
## 2 366 0.4491898   366
## 3 378 0.4969714   378
## 4 416 0.6761818   416
## 5 436 0.7336607   436
## 6 451 0.8204633   451

head(with(sun.spct,
          get_peaks(w.length, s.e.irrad, span=51)))

##     x         y label
## 1 451 0.8204633   451
## 2 495 0.7899872   495
## 3 747 0.5025733   747
```

The equivalent function for finding valleys is `get_valleys` taking the same parameters as `get_peaks` but returning the wavelengths at which the valleys are located.

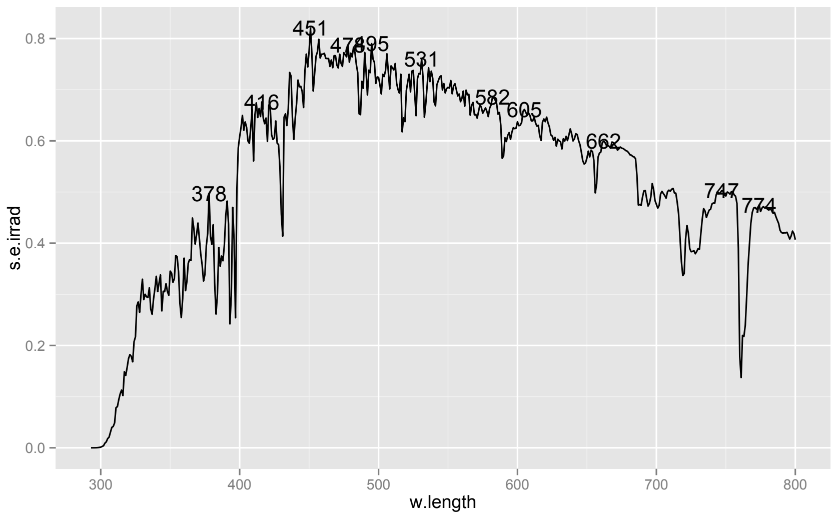
```
head(with(sun.spct,
          get_valleys(w.length, s.e.irrad, span=51)))

##     x         y label
## 1 358 0.2544907   358
## 2 393 0.2422023   393
## 3 431 0.4136900   431
## 4 487 0.6511654   487
## 5 517 0.6176652   517
## 6 589 0.5658760   589

head(with(sun.spct,
          get_valleys(w.length, s.e.irrad, span=51,
                      ignore_threshold=0.5)))

##     x         y label
## 1 431 0.4136900   431
## 2 487 0.6511654   487
## 3 517 0.6176652   517
## 4 589 0.5658760   589
## 5 656 0.4982959   656
```

In the next section, we plot spectra and annotate them with peaks and valleys. If you find the meaning of the parameters `span` and `ignore_threshold` difficult to grasp from the explanation given above, please, study the code and plots in section 15.8.

15.8 Task: annotating peaks and valleys in spectra

Here we show an example of the use the new `ggplot` 'statistics' `stat_peaks` from our package `photobiologygg`. It uses the same parameter names and take the same arguments as the `get_peaks` function described in section 15.7. We reuse once more `fig_sun.e` saved in section 15.4.

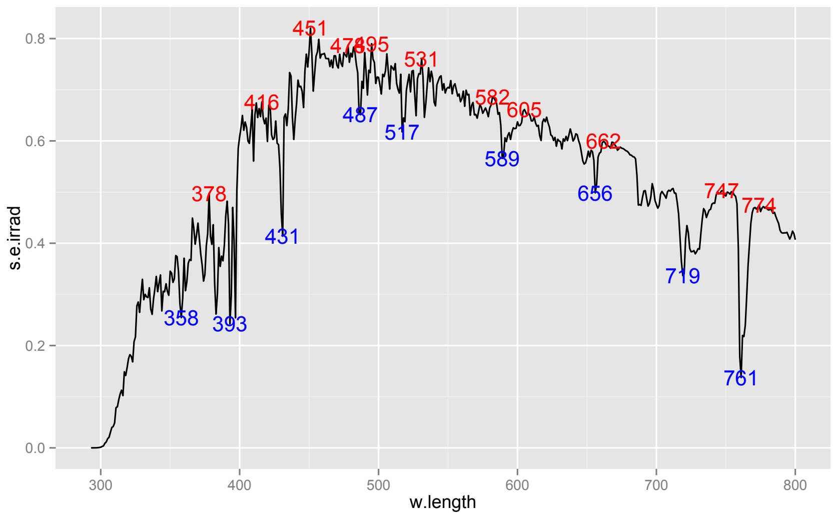
```
fig_sun.e0 + stat_peaks(span=31)
```



Now we play with ggplot2 to show different ways of plotting the peaks and valleys. It behaves as a ggplot2 stat_xxxx function accepting a geom argument and all the aesthetics valid for the chosen geom. By default geom_text is used.

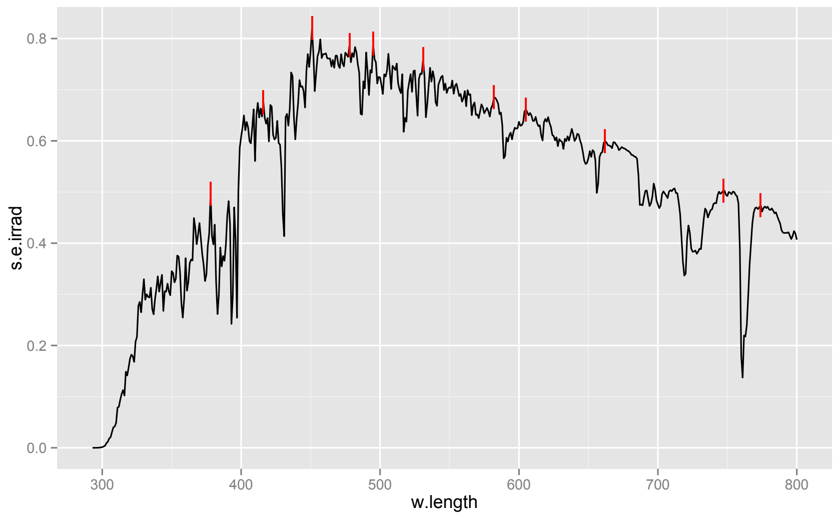
We can change aesthetics, for example the colour:

```
fig_sun.e0 + stat_peaks(colour="red", span=31) +
             stat_valleys(colour="blue", span=51)
```



We can also use a different geom, in this case geom_point, however, be aware that the geom parameter takes as argument a character string giving the name of the geom, in this case "point". We change a few additional aesthetics of the points: we set shape to a character, and set its size to 6.

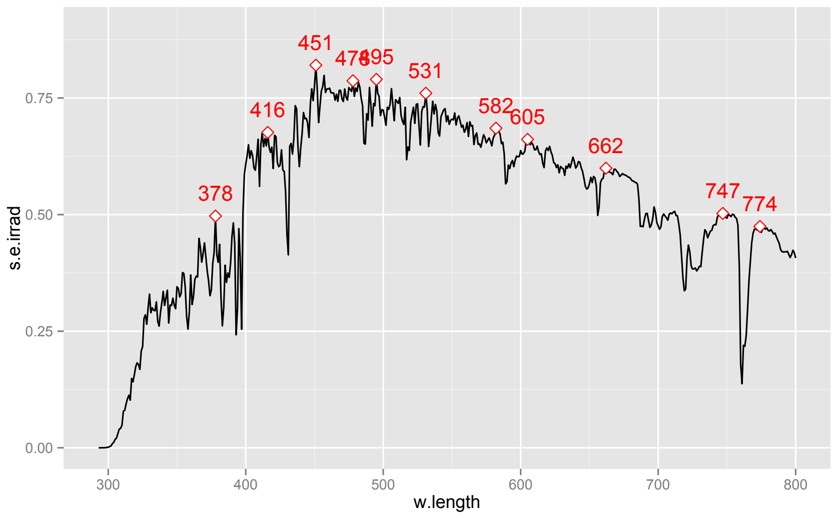
```
fig_sun.e0 +
  stat_peaks(colour="red", geom="point",
             shape="|", size=6, span=31)
```



We can add the same `stat` two or more times to a ggplot, in this example, each time with a different `geom`. First we add points to mark the peaks, and afterwards add labels showing the wavelengths at which they are located using geom `"text"`. For the `shape`, or type of symbol, we use one that supports 'fill', and set the `fill` to `"white"` but keep the border of the symbol `"red"` by setting `colour`, we also change the `size`. With the labels we use `vjust` to 'justify' the text moving the labels vertically, so that they do not overlap the line depicting the spectrum[4] In addition we expand the y-axis scale so that all labels fall within the plotting area.

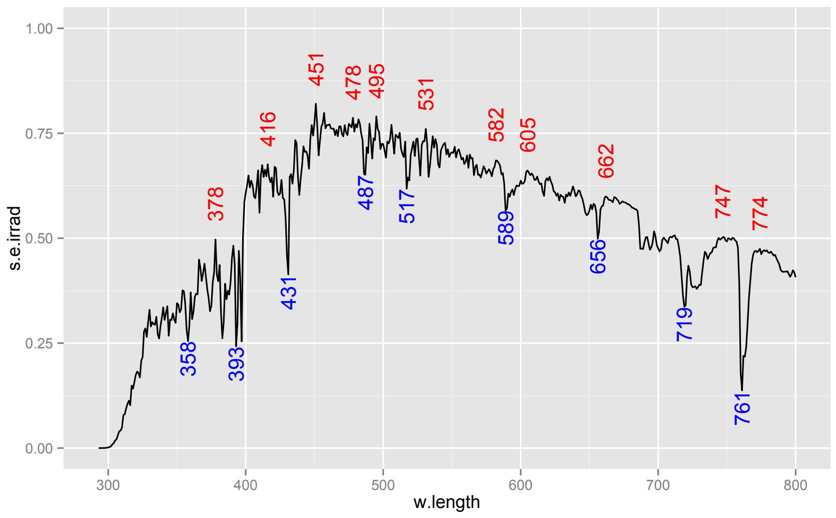
```
fig_sun.e0 +
  stat_peaks(colour="red", geom="point", shape=23,
             fill="white", size=3, span=31) +
  stat_peaks(colour="red", vjust=-1, span=31) +
  expand_limits(y=0.9)
```

[4]The default position of labels is to have them centred on the coordinates of the peak or valley. Unless we rotate the label, `vjust` can be used to shift the label along the y-axis, however, justification is a property of the text, not the plot, so the vertical direction is referenced to the position of the text of the label. A value of 0.5 indicates centering, a negative value 'up' and a positive value 'down'. For example a value of -1 puts the x, y coordinates of the peak or valley at the lower edge of the 'bounding box' of the text. For `hjust` values of -1 and 1 right and left justify the label with respect to the x, y coordinates supplied. Values other than -1, 0.5, and 1, are valid input, but are rather tricky to use for `hjust` as the displacement is computed relative to the width of the bounding box of the label, the displacement being different for the same numerical value depending on the length of the label text.



Finally an example with rotated labels, using different colours for peaks and valleys. Be aware that the 'justification' direction, as discussed in the footnote, is referenced to the position of the text, and for this reason to move the rotated labels upwards we need to use `hjust` as the desired displacement is horizontal with respect to the orientation of the text of the label. As we put peak labels above the spectrum and valleys bellow it, we need to use `hjust` values of opposite sign, but the exact values used were simply adjusted by trial and error until the figure looked as desired.

```
fig_sun.e0 +
  stat_peaks(angle=90, hjust=-0.5, colour="red", span=31) +
  stat_valleys(angle=90, hjust=1, color="blue", span=51) +
  expand_limits(y=1.0)
```



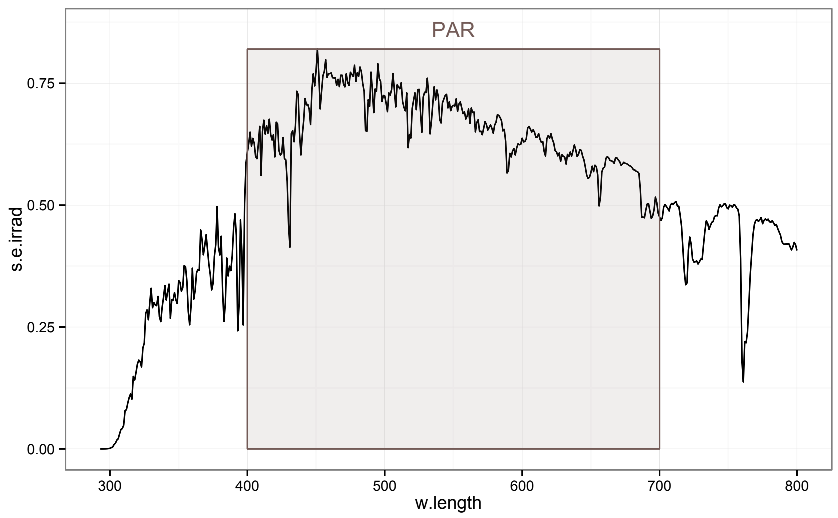
See section ?? in chapter ?? for an example these stats together with facets.

15.9 Task: annotating wavebands

The function `annotate_waveband` can be used to highlight a waveband in a plot of spectral data. Its first argument should be a `waveband` object, and the second argument a `geom` as a character string. The positions on the x-axis are calculated automatically by default, but they can be overridden by explicit arguments. The vertical positions have no default, except for `ymin` which is equal to zero by default. The colour has a default value calculated from waveband definition, in addition x is by default set to the midpoint of the waveband along the wavelength limits. The default value of the labels is the 'name' of the waveband as returned by `labels.waveband`.

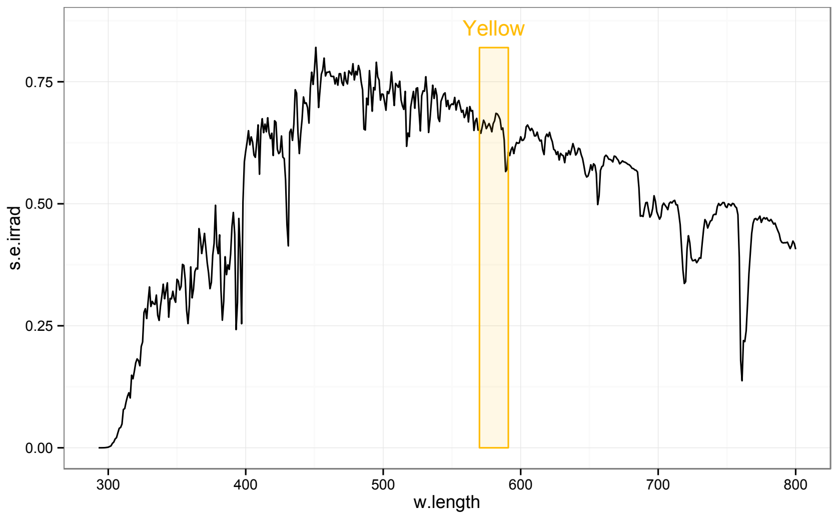
Here is an example for PAR using defaults, and with arguments supplied only for parameters with no defaults. The example does the annotation using two different 'geoms', `"rect"` for marking the region, and `"text"` for the labels.

```r
figvl <- fig_sun.e0 + annotate_waveband(PAR(), "rect", ymax=0.82) +
                     annotate_waveband(PAR(), "text", y=0.86)

figvl + theme_bw()
```



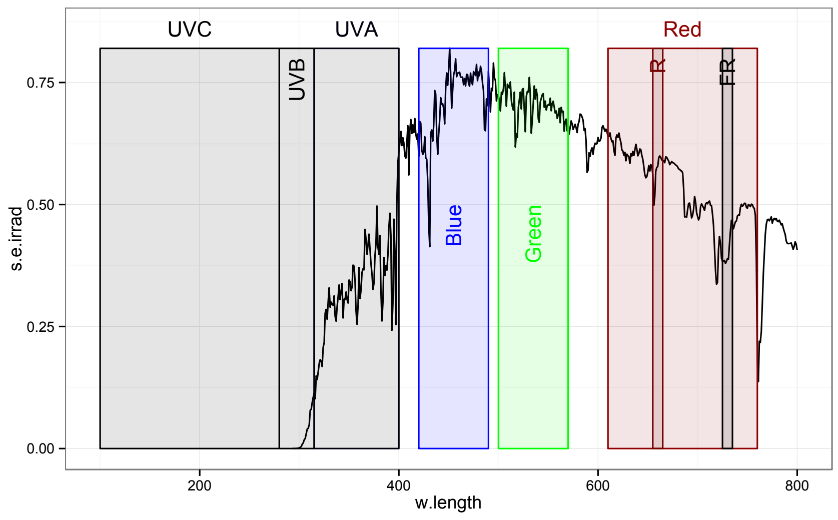
This example annotates a narrow waveband.

```r
figvl <- fig_sun.e0 + annotate_waveband(Yellow(), "rect", ymax=0.82) +
                     annotate_waveband(Yellow(), "text", y=0.86)

figvl + theme_bw()
```



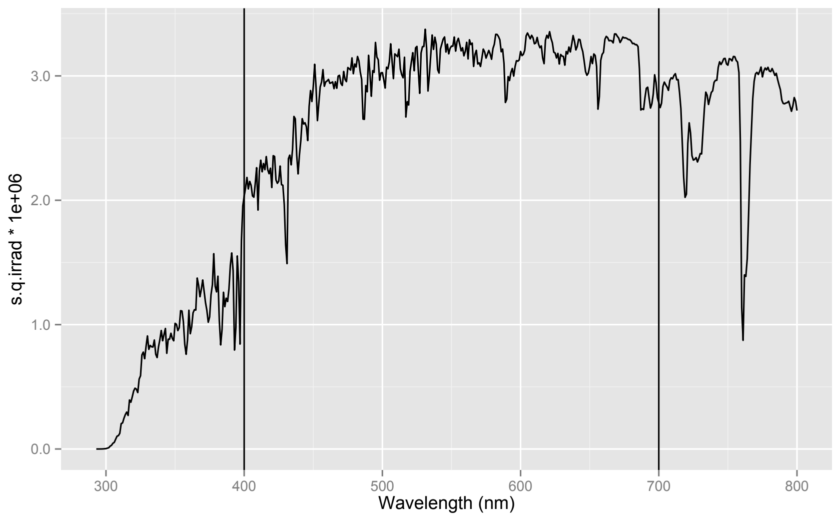
Now an example that is more complex, and demonstrates the flexibility of plots produced with `ggplot2`. We add annotations for eight different wavebands, some of them overlapping. For each one we use two 'geoms' and some labels are rotated and justified. We can also see in this example that the annotations look nicer on a white background, which can be obtained with `theme_bw`. A much simpler, but less flexible approach for adding annotations for several wavebands is described on page 160.

```
figv2 <- fig_sun.e0 +
  annotate_waveband(UVC(), "rect",
                    ymax=0.82) +
  annotate_waveband(UVC(), "text",
                    y=0.86) +
  annotate_waveband(UVB(), "rect",
                    ymax=0.82) +
  annotate_waveband(UVB(), "text",
                    y=0.80, angle=90, hjust=1) +
  annotate_waveband(UVA(), "rect",
                    ymax=0.82) +
  annotate_waveband(UVA(), "text",
                    y=0.86) +
  annotate_waveband(Blue("Sellaro"), "rect",
                    ymax=0.82) +
  annotate_waveband(Blue("Sellaro"), "text",
                    y=0.5, angle=90, hjust=1) +
  annotate_waveband(Green("Sellaro"), "rect",
                    ymax=0.82) +
  annotate_waveband(Green("Sellaro"), "text",
                    y=0.50, angle=90, hjust=1) +
  annotate_waveband(Red(), "rect",
                    ymax=0.82) +
  annotate_waveband(Red(), "text",
                    y=0.86) +
  annotate_waveband(Red("Smith10"), "rect",
                    ymax=0.82) +
  annotate_waveband(Red("Smith10"), "text",
                    y=0.80, angle=90, hjust=1) +
  annotate_waveband(Far_red("Smith10"), "rect",
                    ymax=0.82) +
  annotate_waveband(Far_red("Smith10"), "text",
                    y=0.80, angle=90, hjust=1)

figv2 + theme_bw()
```



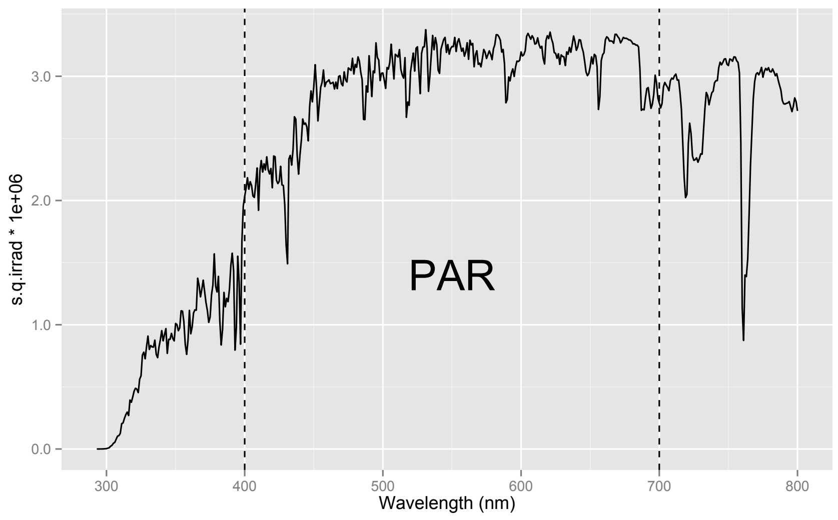
A simple example using `geom_vline`:

```
figvl3 <- fig_sun.q +
  geom_vline(xintercept=range(PAR()))

figvl3
```



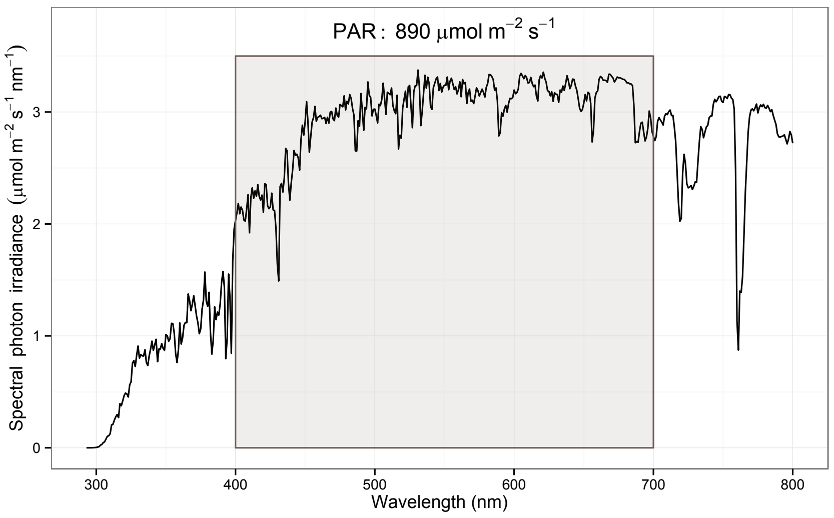
And one where we change some of the aesthetics, and add a label:

```
figvl4 <- fig_sun.q +
  geom_vline(xintercept=range(PAR()), linetype="dashed") +
  annotate_waveband(PAR(), "text", y=1.4, size=10, colour="black")

figvl4
```



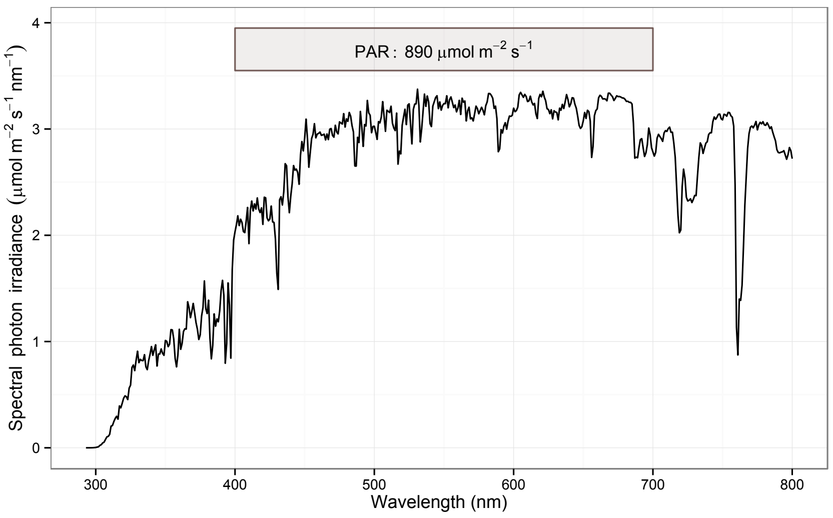
Now including calculated values in the label, first with a simple example with only PAR. Because of using expressions to obtain superscripts we need to add `parse=TRUE` to the call. In addition as we are expressing the integral in photon based units, we also change the type of units used for plotting the spectral irradiance (multiplying by $1 \cdot 10^6$ to because of the unit multiplier used).

```
fig_sun <- ggplot(data=sun.spct,
                  aes(x=w.length, y=s.q.irrad * 1e6)) +
  geom_line() +
  labs(y = ylab_umol,
       x = "Wavelength (nm)")

par <- q_irrad(sun.spct, PAR()) * 1e6

fig_sun2 <- fig_sun +
  annotate_waveband(PAR(), "rect", ymax=3.5) +
  annotate_waveband(PAR(), "text",
                    label=paste("PAR:~", signif(par,digits=2),
                                "*~mu*mol~m^{-2}~s^{-1}", sep=""),
                    y=3.75, colour="black", parse=TRUE)

fig_sun2 + theme_bw()
```



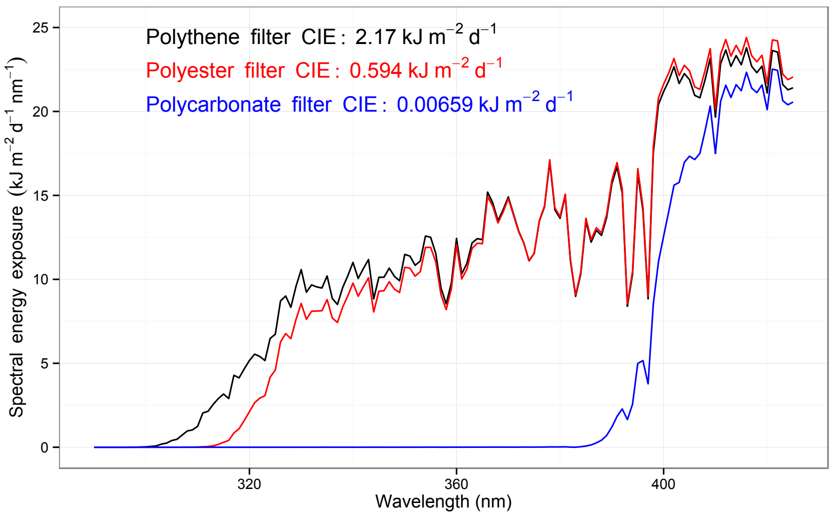
A variation of the previous figure shows how to use smaller rectangles for annotation, which yields plots where the spectrum itself is easier to see than when the rectangle overlaps the spectrum. We achieve this by supplying as argument both `ymax` and `ymin`, and slightly reducing the size of the text with `size = 4`.

```
fig_sun <- ggplot(data=sun.spct,
                  aes(x=w.length, y=s.q.irrad * 1e6)) +
  geom_line() +
  labs(y = ylab_umol,
       x = "Wavelength (nm)")

par <- q_irrad(sun.spct, PAR()) * 1e6

fig_sun2 <- fig_sun +
  annotate_waveband(PAR(), "rect", ymax=3.95, ymin=3.55) +
  annotate_waveband(PAR(), "text", size=4,
                    label=paste("PAR:~", signif(par,digits=2),
                                "*~mu*mol~m^{-2}~s^{-1}", sep=""),
                    y=3.75, colour="black", parse=TRUE)

fig_sun2 + theme_bw()
```



This type of annotations can be also easily done for effective exposures or doses, but in this example as we position the annotations manually, we can use ggplot2's 'normal' `annotate` function. We use `xlim` to restrict the plotted region of the spectrum to the range of wavelengths of interest.

```
fig_dsun <-
  ggplot(data=sun.daily.spct * polythene.new.spct,
         aes(x=w.length, y=s.e.irrad * 1e-3)) + geom_line() +
  geom_line(data=sun.daily.spct * polyester.new.spct,
            colour="red") +
  geom_line(data=sun.daily.spct * PC.spct,
            colour="blue") +
  labs(y =
   expression(Spectral~~energy~~exposure~~(kJ~m^{-2}~d^{-1}~nm^{-1})),
       x = "Wavelength (nm)") + xlim(290, 425) + ylim(0, 25)

cie.pe <-
  e_irrad(sun.daily.spct * polythene.new.spct, CIE()) * 1e-3
cie.ps <-
  e_irrad(sun.daily.spct * polyester.new.spct, CIE()) * 1e-3
cie.pc <-
  e_irrad(sun.daily.spct * PC.spct, CIE()) * 1e-3
y.pos <- 22.5

fig_dsun2 <- fig_dsun +
  annotate("text",
           label=paste("Polythene~~filter~~CIE:~",
                       signif(cie.pe, digits=3),
                       "*~kJ~m^{-2}~d^{-1}", sep=""),
           y=y.pos+2, x=300, hjust=0, colour="black",
           parse=TRUE) +
  annotate("text", label=paste("Polyester~~filter~~CIE:~",
                               signif(cie.ps, digits=3),
                               "*~kJ~m^{-2}~d^{-1}", sep=""),
           y=y.pos, x=300,  hjust=0,  colour="red",
           parse=TRUE) +
  annotate("text", label=paste("Polycarbonate~~filter~~CIE:~",
                               signif(cie.pc, digits=3),
                                        "*~kJ~m^{-2}~d^{-1}", sep=""),
           y=y.pos-2, x=300, hjust=0,  colour="blue",
           parse=TRUE)

fig_dsun2 + theme_bw()
```



15.10 Task: using colour as data in plots

The examples in this section use a single spectrum, sun.spct, but all functions used are methods for generiic.spct objects, so are equally applicable to the plotting of other spectra like transmittance, reflectance or response ones.

When we want to colour-label individual spectral values, for example, by plotting the individual data points with the colour corresponding to their wavelengths, or fill the area below a plotted spectral curve with colours, we need to first tag the spectral data set using a waveband definition or a list of waveband definitions. If we just want to add a guide or labels to the plot, we can create new data instead of tagging the spectral data to be plotted. In section 15.10.2 we show code based on tagging spectral data, and in section 15.10.3 the case of using different data for plotting the guide or key is described.

15.10.1 Scale definitions

First we define some new scales for use for plotting with ggplot when plotting wavelength derived colours. In the future something equivalent may be included in package photobiologygg as predefined scales. We define two very similar scales, one for colour, and one for fill aesthetics.

```
scale_colour_tgspct <-
  function(...,
           tg.spct,
           labels = NULL,
           guide = NULL,
           na.value=NA) {
    spct.tags <- attr(tg.spct, "spct.tags", exact=TRUE)
    if (is.null(guide)){
      if (spct.tags$wb.num > 12) {
        guide = "none"
      } else {
        guide = guide_legend(title=NULL)
      }
    }
    values <- as.character(spct.tags$wb.colors)
    if (is.null(labels)) {
      labels <- spct.tags$wb.names
    }
    ggplot2:::manual_scale("colour",
                           values = values,
                           labels = labels,
                           guide = guide,
                           na.value = na.value,
                           ...)
}
```

```
scale_fill_tgspct <-
  function(...,
           tg.spct,
           labels = NULL,
           guide = NULL,
           na.value=NA) {
    spct.tags <- attr(tg.spct, "spct.tags", exact=TRUE)
    if (is.null(guide)){
      if (spct.tags$wb.num > 12) {
        guide = "none"
      } else {
        guide = guide_legend(title=NULL)
      }
    }
    values <- as.character(spct.tags$wb.colors)
    if (is.null(labels)) {
      labels <- spct.tags$wb.names
    }

    ggplot2:::manual_scale("fill",
                           values = values,
                           labels = labels,
                           guide = guide,
                           na.value = na.value,
                           ...)
}
```

15.10.2 Plots using colour for the spectral data

We start by describing how to tag a spectrum, and then show how to use tagged spectra for plotting data. Tagging consist in adding wavelength-derived colour data and waveband-related data to a spectral object. We start with a very simple example.

```
cp.sun.spct <- copy(sun.spct)
tag(cp.sun.spct)

##      w.length    s.e.irrad    s.q.irrad wl.color
##    1:       293 2.609665e-06 6.391730e-12  #000000
##    2:       294 6.142401e-06 1.509564e-11  #000000
##   ---
## 507:       799 4.185850e-01 2.795738e-06  #000000
## 508:       800 4.069055e-01 2.721132e-06  #000000
##      wb.f
##    1:   NA
##    2:   NA
##   ---
## 507:   NA
## 508:   NA
```

As no waveband information was supplied as input, only wavelength-dependent colour information is added to the spectrum plus a factor `wb.f` with only NA level.

If we instead provide a waveband as input then both wavelength-dependent colour and waveband information are added to the spectral data object.

```
uvb.sun.spct <- copy(sun.spct)
tag(uvb.sun.spct, UVB())

##      w.length    s.e.irrad    s.q.irrad wl.color
##    1:       293 2.609665e-06 6.391730e-12  #000000
##    2:       294 6.142401e-06 1.509564e-11  #000000
##   ---
## 508:       799 4.185850e-01 2.795738e-06  #000000
## 509:       800 4.069055e-01 2.721132e-06  #000000
##           wb.f
##    1: UVB.tr.lo
##    2: UVB.tr.lo
##   ---
## 508:         NA
## 509:         NA

levels(uvb.sun.spct[["wb.f"]])

## [1] "UVB.tr.lo"
```

The output contains the same variables (columns) but now the factor `wb.f` has a level based on the name of the waveband, and a value of NA outside it.

We can alter the name used for the `wb.f` factor levels by using a named list as argument.

```
tag(uvb.sun.spct, list('ultraviolet-B' = UVB()))

## Warning in tag.generic_spct(uvb.sun.spct,
list(‘ultraviolet-B‘ = UVB())):  Overwriting old tags in
spectrum

##      w.length    s.e.irrad    s.q.irrad wl.color
##    1:       293 2.609665e-06 6.391730e-12  #000000
##    2:       294 6.142401e-06 1.509564e-11  #000000
##   ---
## 508:       799 4.185850e-01 2.795738e-06  #000000
## 509:       800 4.069055e-01 2.721132e-06  #000000
##            wb.f
##   1: UVB.tr.lo
##   2: UVB.tr.lo
##  ---
## 508:         NA
## 509:         NA

levels(uvb.sun.spct[["wb.f"]])

## [1] "UVB.tr.lo"
```

This example also shows, that re-tagging a spectrum replaces the old tagging data with the new one.

If we use a list of wavebands then the tagging is based on all of them, but be aware that the wavelength ranges of the wavebands overlap, the result is undefined.

```
plant.sun.spct <- copy(sun.spct)
tag(plant.sun.spct, Plant_bands())

##      w.length    s.e.irrad    s.q.irrad wl.color
##   1:      293 2.609665e-06 6.391730e-12  #000000
##   2:      294 6.142401e-06 1.509564e-11  #000000
##  ---
## 517:      799 4.185850e-01 2.795738e-06  #000000
## 518:      800 4.069055e-01 2.721132e-06  #000000
##            wb.f
##   1: UVB.tr.lo
##   2: UVB.tr.lo
##  ---
## 517:         NA
## 518:         NA

levels(plant.sun.spct[["wb.f"]])

## [1] "UVB.tr.lo" "UVA"        "Blue"
## [4] "Green"     "R"          "FR"
```

Tagging also adds some additional data as an attribute to the spectrum. This data can be retrieved with the base R function `attr`.

```
attr(cp.sun.spct, "spct.tag")

## $time.unit
## [1] "second"
##
## $wb.key.name
## [1] "Bands"
##
## $wl.color
## [1] TRUE
##
## $wb.color
## [1] TRUE
##
```

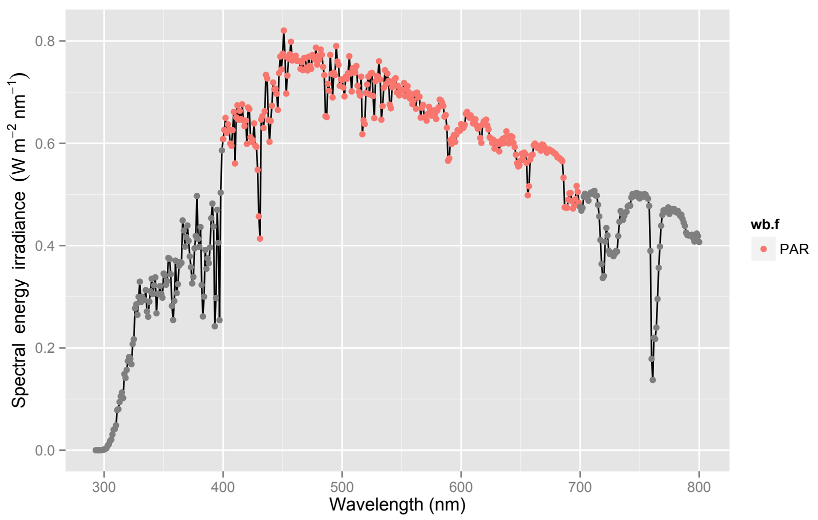
```
## $wb.num
## [1] 0
##
## $wb.colors
## [1] NA
##
## $wb.names
## [1] NA
##
## $wb.list
## NULL

attr(uvb.sun.spct, "spct.tag")

## $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]]
## [1] "black"
##
##
## $wb.names
## [1] "UVB.tr.lo"
##
## $wb.list
## $wb.list[[1]]
## UVB.ISO.tr.lo
## low (nm) 293
## high (nm) 315
## weighted none
```

We now tag a spectrum for use in our first plot example.

```
par.sun.spct <- copy(sun.spct)
tag(par.sun.spct, PAR())

##      w.length     s.e.irrad     s.q.irrad wl.color
##   1:      293 2.609665e-06 6.391730e-12  #000000
##   2:      294 6.142401e-06 1.509564e-11  #000000
##  ---
## 509:      799 4.185850e-01 2.795738e-06  #000000
## 510:      800 4.069055e-01 2.721132e-06  #000000
##      wb.f
##   1:   NA
##   2:   NA
##  ---
## 509:    NA
## 510:    NA
```

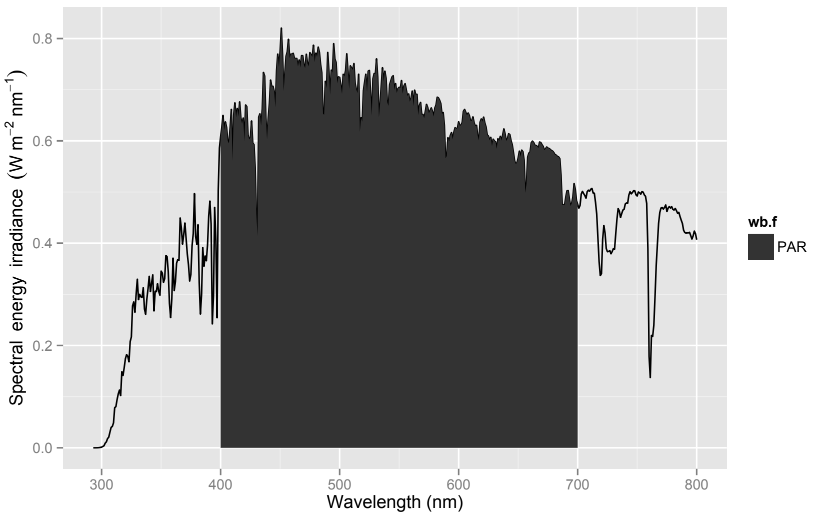
Here we simply use the `wb.f` factor that was added as part of the tagging, with the default colour scale of `ggplot2`, which results in a palette unrelated to the real colour of the different wavelengths.

```
fig_sun.t00 <-
  ggplot(data=par.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  geom_point(aes(color=wb.f)) +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.t00
```



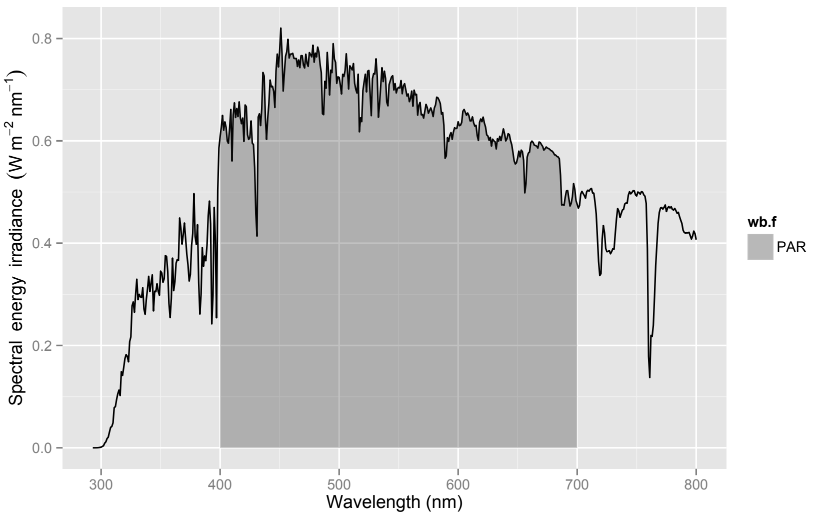
We can also use other geoms like `geom_area` in the next chunk, together with, as an example, a grey fill scale from `ggplot2`.

```
fig_sun.t01 <-
  ggplot(data=par.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  geom_area(color=NA, aes(fill=wb.f)) +
  scale_fill_grey(na.value=NA) +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.t01
```



The default fill looks too dark and bold, so we change the transparency of the fill by setting `fill = 0.3`. The grid in the background becomes slightly visible also in the filled region, facilitating 'reading' of the plot and avoiding a to stark contrast between regions, which tends to be disturbing. In later plots we frequently use `alpha` to improve how plots look, but we exemplify the effect of changing this aesthetic only here.

```
fig_sun.t01 <-
  ggplot(data=par.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  geom_area(color=NA, alpha=0.3, aes(fill=wb.f)) +
  scale_fill_grey(na.value=NA) +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.t01
```



As part of the tagging colour information was also added to the spectral data object[5]. We tag each observation in the solar spectrum with human vision colours as defined by ISO.

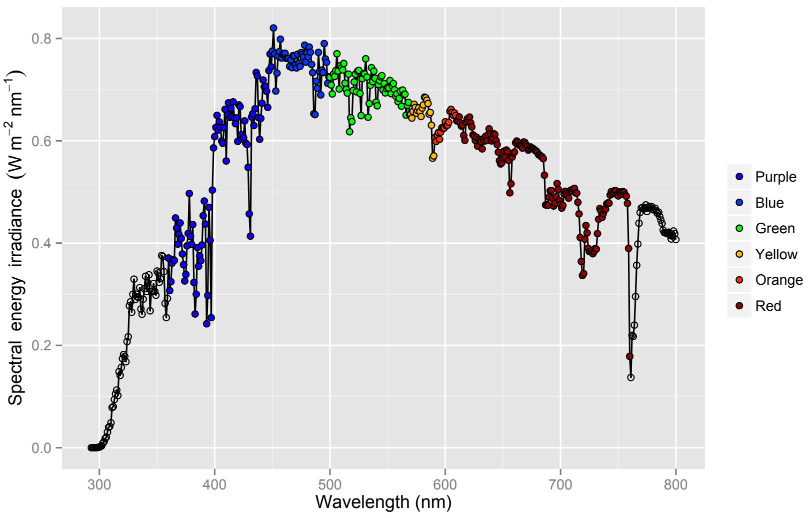
```
tg.sun.spct <- copy(sun.spct)
tag(tg.sun.spct, VIS_bands())

##      w.length    s.e.irrad    s.q.irrad wl.color
##   1:      293 2.609665e-06 6.391730e-12  #000000
##   2:      294 6.142401e-06 1.509564e-11  #000000
##  ---
## 514:      799 4.185850e-01 2.795738e-06  #000000
## 515:      800 4.069055e-01 2.721132e-06  #000000
##      wb.f
##   1:   NA
##   2:   NA
##  ---
## 514:   NA
## 515:   NA
```

See section 15.10.1 on page 137 for the definition of the colour and fill scales used for tagged spectra. These definitions are needed for most of the plots in the remaining of the present and next sections. These scales retrieve information about the wavebands both from the data itself and from the attribute described above.

Here we plot using colours by waveband—using the colour definitions by ISO—, with symbols filled with colours. The colour data outside the wavebands is set to NA so those points are not filled. One can play with the `size` of points until ones get the result wanted. The default 'shape' used by `ggplot2` do not accept a `fill` aesthetic, while shape '21' gives circles that can be 'filled'.

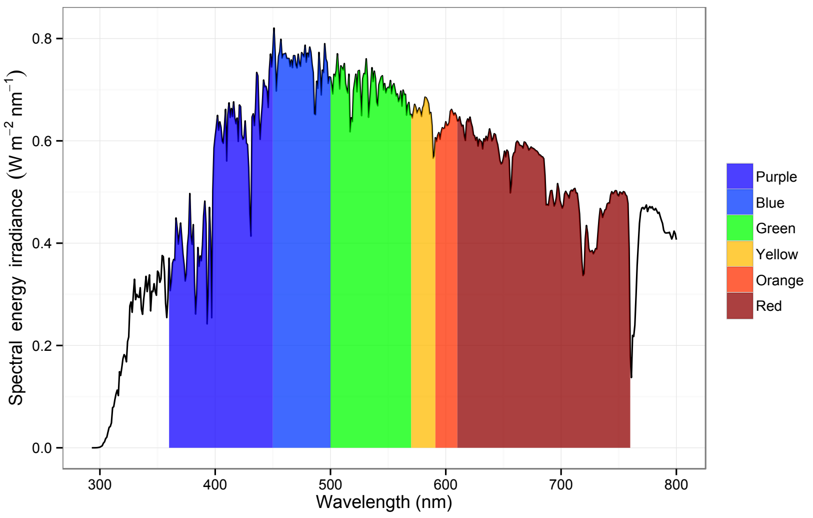
[5]We may want to increase the number of 'observations' in the spectrum by interpolation if there are too few observations for a smooth colour gradient.

```
fig_sun.t02 <-
  ggplot(data=tg.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  scale_fill_tgspct(tg.spct=tg.sun.spct) +
  geom_point(aes(fill=wb.f), shape=21)  +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.t02
```



Using `geom_area` we can fill the area under the curve according to the colour of different wavebands, we set the fill only for this geom, so that the NAs do not affect other plotting. To get a single black curve for the spectrum we use `geom_line`. This approach works as long as wavebands do not share the same value for the color, which means that it is not suitable either when more than one band is outside the visible range, or when using many narrow wavebands.

```
fig_sun.t03 <-
  ggplot(tg.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  scale_fill_tgspct(tg.spct=tg.sun.spct) +
  geom_line() +
  geom_area(aes(fill=wb.f), alpha=0.75) +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.t03 + theme_bw()
```

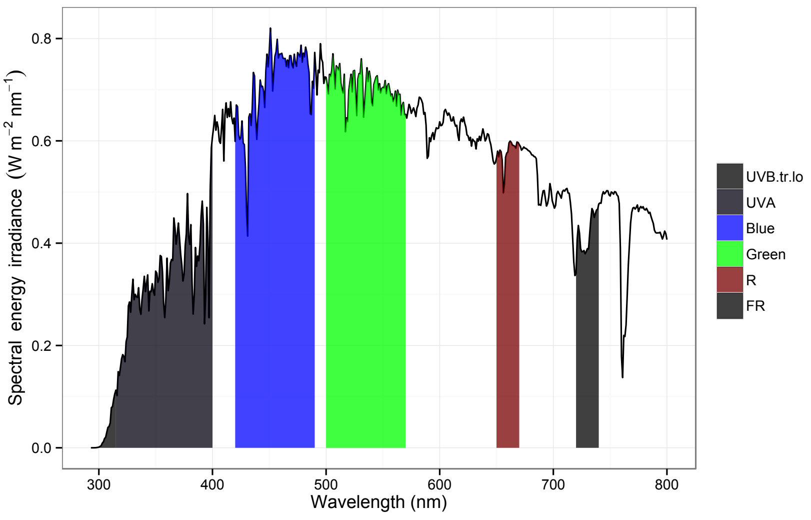


In the next example we tag the solar spectrum with colours using the definitions of plant sensory 'colours'.

```
pl.sun.spct <- copy(sun.spct)
tag(pl.sun.spct, Plant_bands())

##      w.length    s.e.irrad    s.q.irrad wl.color
##   1:      293 2.609665e-06 6.391730e-12  #000000
##   2:      294 6.142401e-06 1.509564e-11  #000000
##  ---
## 517:      799 4.185850e-01 2.795738e-06  #000000
## 518:      800 4.069055e-01 2.721132e-06  #000000
##           wb.f
##   1: UVB.tr.lo
##   2: UVB.tr.lo
##  ---
## 517:        NA
## 518:        NA
```

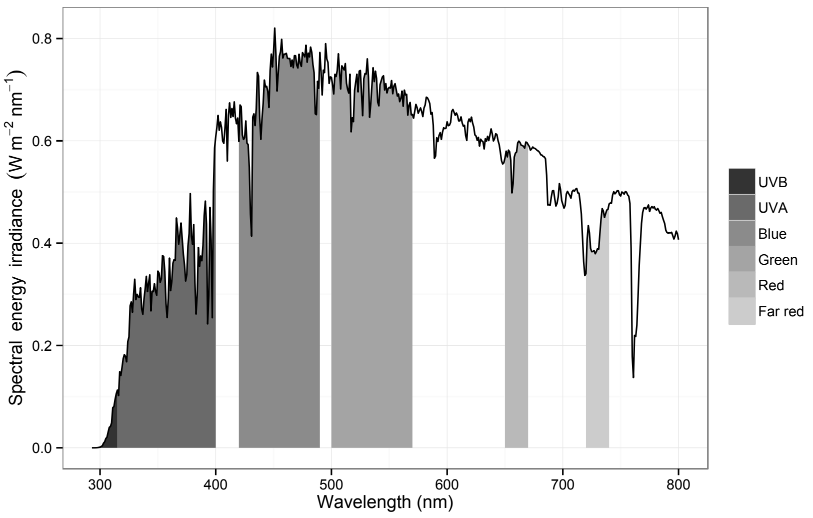
Here we plot the wavebands corresponding to plant sensory 'colours', using the spectrum we tagged in the previous code chunk.

```
fig_sun.pl0 <-
  ggplot(pl.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  scale_fill_tgspct(tg.spct=pl.sun.spct) +
  geom_line() +
  geom_area(aes(fill=wb.f), alpha=0.75) +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.pl0 + theme_bw()
```



We can also use the factor `wb.f` which has value NA outside the wavebands, changing the colour used for NA to NA which renders it invisible. We can change the labels used for the wavebands in two different way, when plotting by supplying a labels argument to the scale used, or when tagging the spectrum. The second approach is simpler when producing several different plots from the same spectral object, or when wanting to have consistent labels and names used also in derived results such as irradiance.

```
fig_sun.pl1 <-
  ggplot(pl.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_area(aes(fill=wb.f)) +
  scale_fill_grey(na.value=NA, name="",
                  labels=c("UVB", "UVA", "Blue",
                           "Green", "Red", "Far red")) +
  geom_line() +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.pl1 + theme_bw()
```



When using a factor we can play with the scale definitions and represent the wavebands in any way we may want. For example we can use `split_bands` to split a waveband or spectrum into many adjacent narrow bands and get an almost continuous gradient, but we need to get around the problem of repeated colours by using the factor and redefining the scale.

When an spectrum has very few observations we can ‘fake’ a longer spectrum by interpolation as a way of getting a more even fill. The example below is not run, in later examples we just use the example spectral data as is.

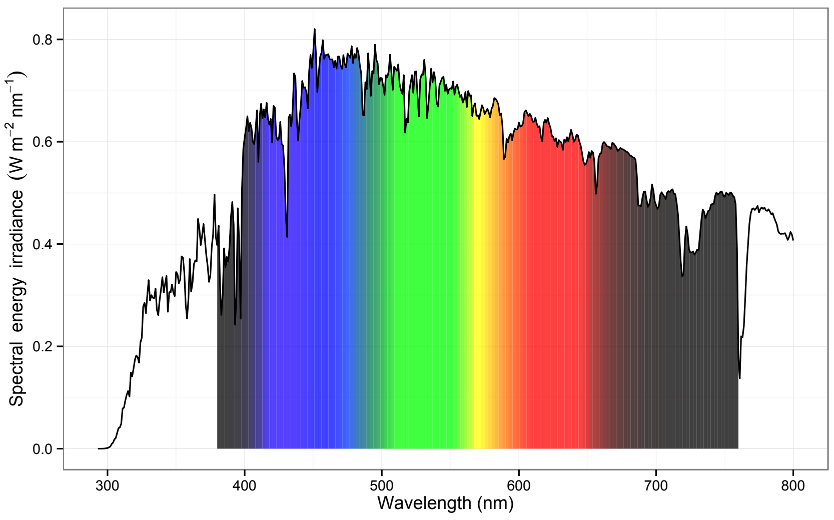
```
interpolate_spct(sun.spct, length.out=800)
```

We tag the VIS region of the spectrum with 150 narrow wavebands. As ‘hinges’ are inserted, there is no gap, and usually there is no need to increase the length of the spectrum by interpolation. If needed one could try something like. However, the longer spectrum should not be used for statistical calculations, not even plotting using `geom_smooth`.

```
splt.sun.spct <- copy(sun.spct)
tag(splt.sun.spct, split_bands(VIS(), length.out=150))

##      w.length     s.e.irrad    s.q.irrad wl.color
##   1:      293 2.609665e-06 6.391730e-12  #000000
##   2:      294 6.142401e-06 1.509564e-11  #000000
##  ---
## 798:      799 4.185850e-01 2.795738e-06  #000000
## 799:      800 4.069055e-01 2.721132e-06  #000000
##      wb.f
##   1:   NA
##   2:   NA
##  ---
## 798:   NA
## 799:   NA
```

In the code above, we made a copy of `sun.spct` because being part of the package, it is write protected, and `tag` works by modifying its argument.

```
fig_sun.splt0 <-
  ggplot(splt.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  scale_fill_tgspct(tg.spct=splt.sun.spct) +
  geom_area(aes(fill=wb.f), alpha=0.75) +
  geom_line() +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.splt0 + theme_bw()
```

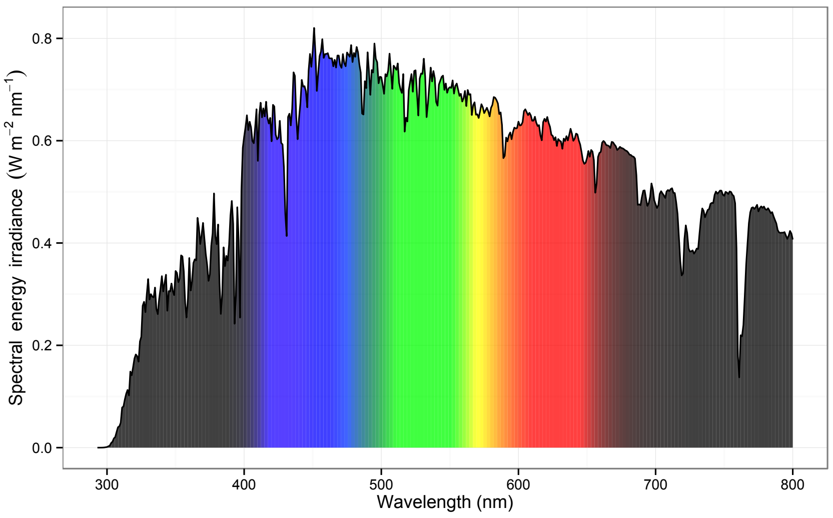


In this other example we tag the whole spectrum, dividing it into 200 wavebands.

```
splt1.sun.spct <- copy(sun.spct)
# splt1.sun.spct <- interpolate_spct(splt1.sun.spct, length.out=1000)
tag(splt1.sun.spct, split_bands(sun.spct, length.out=200))

##       w.length    s.e.irrad    s.q.irrad wl.color
##   1: 293.0000 2.609665e-06 6.391730e-12  #000000
##   2: 294.0000 6.142401e-06 1.509564e-11  #000000
##  ---
## 906: 799.9999 4.069067e-01 2.721139e-06  #000000
## 907: 800.0000 4.069055e-01 2.721132e-06  #000000
##        wb.f
##   1:   wb1
##   2:   wb1
##  ---
## 906: wb200
## 907:    NA
```

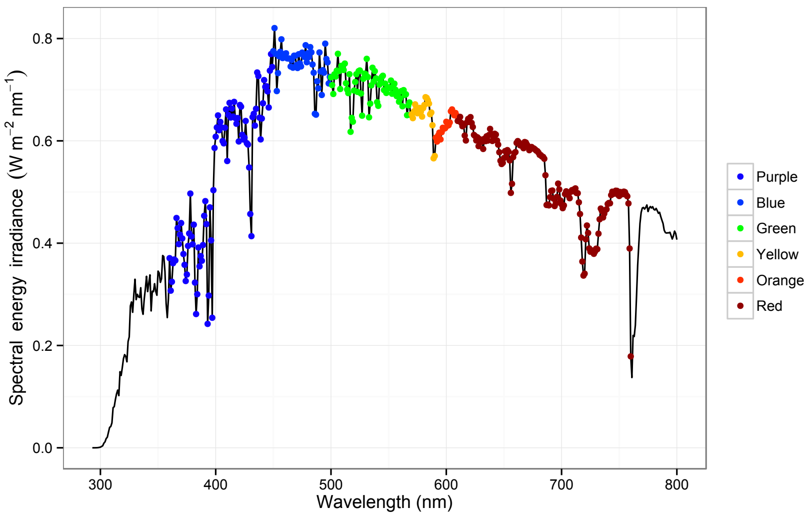
We use `geom_area` and `fill`, and colour the area under the curve. This does not work with `geom_line` because there would not be anything to fill, here we use `geom_area` instead.

```
fig_sun.splt1 <-
  ggplot(splt1.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  scale_fill_tgspct(tg.spct=splt1.sun.spct) +
  geom_area(aes(fill=wb.f), alpha=0.75) +
  geom_line() +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.splt1 + theme_bw()
```



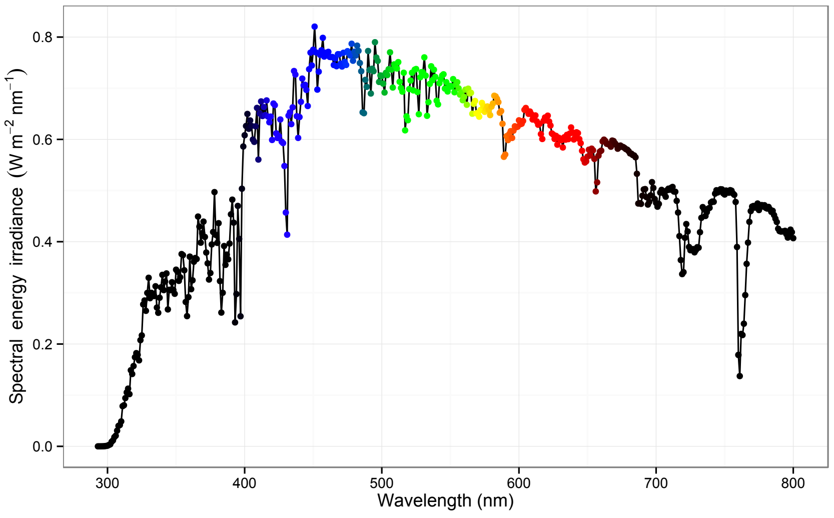
The next example uses `geom_point` and `colour` to color the data points according the waveband they are included in.

```
fig_sun.tg1 <-
  ggplot(tg.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  scale_colour_tgspct(tg.spct=tg.sun.spct) +
  geom_line() +
  geom_point(aes(colour=wb.f)) +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.tg1 + theme_bw()
```



When plotting points, rather than an area we may, instead of using colours from wavebands, want to plot the colour calculated for each individual wavelength value, which `tag` adds to the spectrum, whether a waveband definition is supplied or not. In this case we need to use `scale_color_identity`.

```
fig_sun.tg2 <-
  ggplot(data=tg.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  scale_color_identity() +
  geom_point(aes(color=wl.color))  +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.tg2  + theme_bw()
```



Other possibilities are for example, using one of the symbols that can be filled, and then for example for symbols with a black border and a colour matching its wavelength as a fill aesthetic. It is also possible to use `alpha` with points.

15.10.3 Plots using waveband definitions

In the previous section we showed how tagging spectral data can be used to add colour information that can be used when plotting. In contrast, in the present section we create new 'fake' spectral data starting from waveband definitions that then we plot as 'annotations'. We show different types of annotations based on plotting with different geoms. We show the use of `geom_rect`, `geom_text`, `geom_vline`, and `geom_segment`, that we consider the most useful geometries in this context.

We use three different functions from package `photobiology` to generate the data to be plotted from lists of waveband definitions. We use mainly pre-defined wavebands, but user defined wavebands can be used as well. We start by showing the output of these functions, starting with `wb2spct` the simplest one.

```
wb2spct(PAR())

##    w.length s.e.irrad s.q.irrad Tfr Rfl
## 1: 399.9999         0         0   0   0
## 2: 400.0000         0         0   0   0
## 3: 699.9999         0         0   0   0
## 4: 700.0000         0         0   0   0
##    s.e.response
## 1:            0
## 2:            0
## 3:            0
## 4:            0

wb2spct(Plant_bands())

##     w.length s.e.irrad s.q.irrad Tfr Rfl
##  1: 279.9999         0         0   0   0
##  2: 280.0000         0         0   0   0
## ---
## 21: 739.9999         0         0   0   0
## 22: 740.0000         0         0   0   0
##     s.e.response
##  1:            0
##  2:            0
## ---
## 21:            0
## 22:            0
```

Function `wb2tagged_spct` returns the same 'spectrum', but tagged with the same wavebands as used to create the spectral data, and you will also notice that a 'hinge' has been added, which is redundant in the case of a single waveband, but needed in the case of wavebands sharing a limit.

```
wb2tagged_spct(PAR())

##    w.length s.e.irrad s.q.irrad Tfr Rfl
## 1: 399.9999         0         0   0   0
## 2: 400.0000         0         0   0   0
## 3: 699.9999         0         0   0   0
## 4: 700.0000         0         0   0   0
##    s.e.response wl.color wb.f y
## 1:            0  #03001E   NA 0
## 2:            0  #03001E  PAR 0
## 3:            0  #070000  PAR 0
## 4:            0  #070000   NA 0

wb2tagged_spct(Plant_bands())

##     w.length s.e.irrad s.q.irrad Tfr Rfl
##  1: 279.9999         0         0   0   0
##  2: 280.0000         0         0   0   0
## ---
## 21: 739.9999         0         0   0   0
## 22: 740.0000         0         0   0   0
##     s.e.response wl.color wb.f y
##  1:            0  #000000   NA 0
##  2:            0  #000000  UVB 0
## ---
## 21:            0  #000000   FR 0
## 22:            0  #000000   NA 0
```

The third function, wb2rect_spct is what we use in most examples. It generates data that make it easier to plot rectangles with geom_rect as we will see in later examples.

```
wb2rect_spct(PAR())

##    w.length s.e.irrad s.q.irrad Tfr Rfl
## 1:      550         0         0   0   0
##    s.e.response wl.color wb.f wl.high wl.low y
## 1:            0  #00FF00  PAR     700    400 0

wb2rect_spct(Plant_bands())

##    w.length s.e.irrad s.q.irrad Tfr Rfl
## 1:    297.5         0         0   0   0
## 2:    357.5         0         0   0   0
## 3:    455.0         0         0   0   0
## 4:    535.0         0         0   0   0
## 5:    660.0         0         0   0   0
## 6:    730.0         0         0   0   0
##    s.e.response wl.color  wb.f wl.high wl.low y
## 1:            0  #000000   UVB     315    280 0
## 2:            0  #000000   UVA     400    315 0
## 3:            0  #0000FF  Blue     490    420 0
## 4:            0  #00FF00 Green     570    500 0
## 5:            0  #730000     R     670    650 0
## 6:            0  #010000    FR     740    720 0
```

In this case instead of two rows per waveband, we obtain only one row per waveband, with a `w.length` value corresponding to its midpoint but with two additional columns giving the low and high wavelength limits.

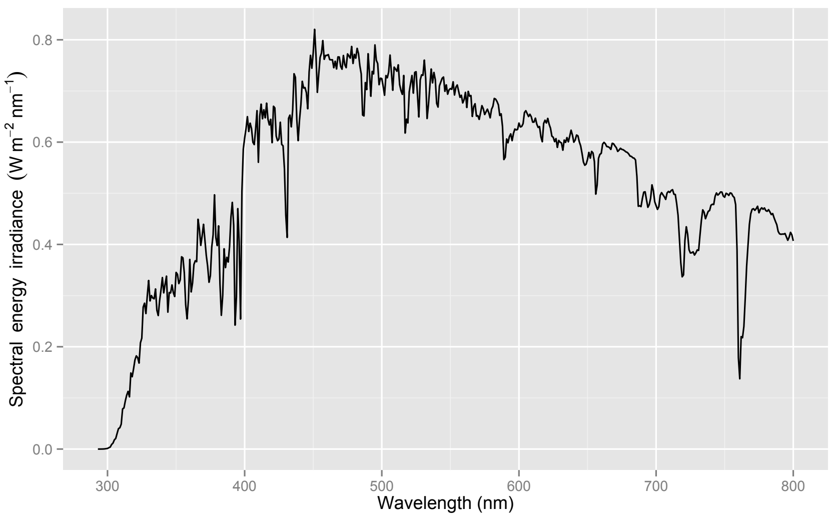
As we saw earlier for tagged spectra, additional data is stored in an attribute.

```
attr(wb2rect_spct(PAR()), "spct.tags")

## $time.unit
## [1] "none"
##
## $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
```

The first plot examples show how to add a colour bar as key. We create new data for use in what is closer to the concept of annotation that to plotting. In most of the examples below we use waveband definitions to create tagged spectral data for use in plotting the guide using `geom_rect`. We present three cases: an almost continuous colour reference guide, a reference guide for colours perceived by plants and one for ISO colour definitions. We also add labels to the bar with `geom_text` and show some examples of how to change the color of the line enclosing the rectangles and of text labels. Finally we show how to use `fill` and `alpha` to adjust how the guides look. Later on we show some examples using other geoms and also examples combining the use of tagged spectra as described in the previous section with the 'annotations' described here.

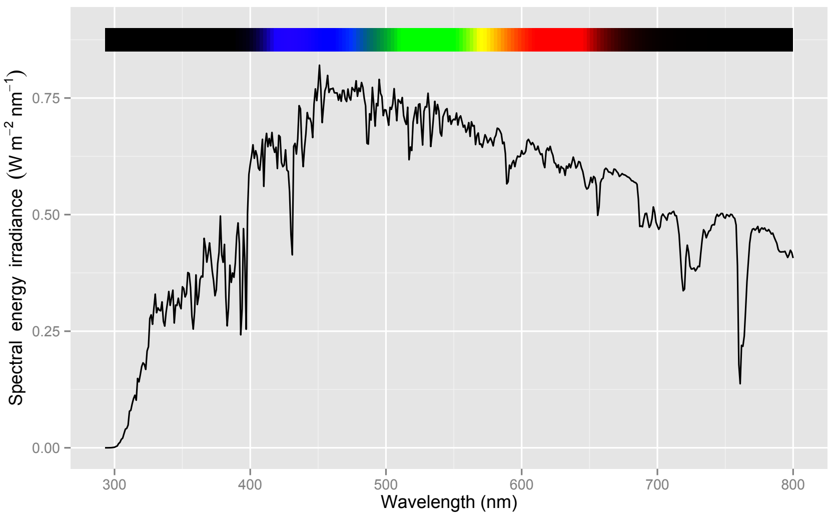
First we create a simple line plot of the solar spectrum, that we will use as a basis for most of the examples below.

```
fig_sun.z0 <-
  ggplot(data=sun.spct,
          aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.z0
```



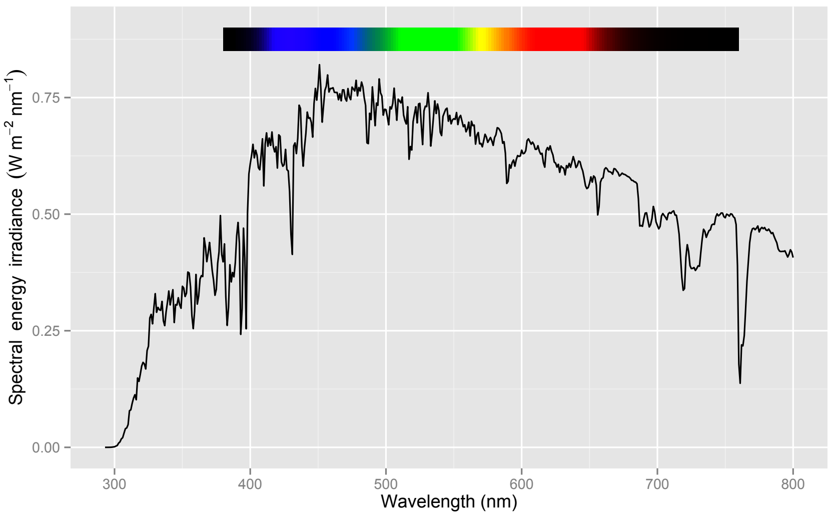
We now add to the plot created above a nearly continuous colour bar for the whole spectrum. To obtain an almost continuous colour scale we use a list of 200 wavebands. We need to specify `color = NA` to prevent the line enclosing each of the 200 rectangles from being plotted. We position the bar at the top because we think that it looks best, but by changing the values supplied to `ymax` and `ymin` move the bar vertically and also change its width.

```
wl.guide.spct <-
  wb2rect_spct(split_bands(sun.spct,
                           length.out=200))

fig_sun.z2 <- fig_sun.z0 +
  geom_rect(data=wl.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                y = 0, fill=wb.f),
            color = NA) +
  scale_fill_tgspct(tg.spct=wl.guide.spct)

fig_sun.z2
```



This second example differs very little from the previous one, but by using a waveband definition instead of a spectrum as argument to `split_bands`, we restrict the region covered by the colour fill to that of the waveband. In fax a vector of length two, or any object for which a `range` method is available can be used as input to this function.

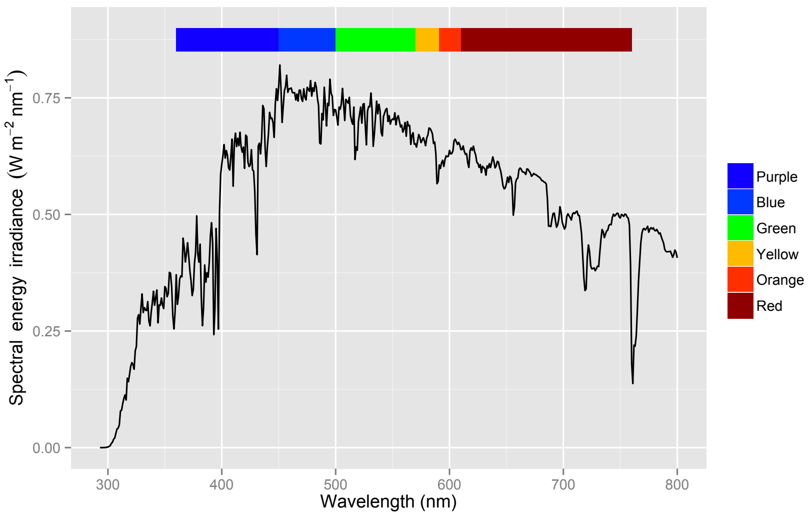
```
wl.guide.spct <- wb2rect_spct(split_bands(VIS(), length.out=200))

fig_sun.z1 <- fig_sun.z0 +
  geom_rect(data=wl.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                y = 0, fill=wb.f),
            color = NA) +
  scale_fill_tgspct(tg.spct=wl.guide.spct)

fig_sun.z1
```



In the examples above we have used a list of 200 waveband definitions created with `split_bands`. If we instead use a shorter list of definitions, we get a plot where the wavebands are clearly distinguished. By default if the list of wavebands is short, a key or 'guide' is also added to the plot.

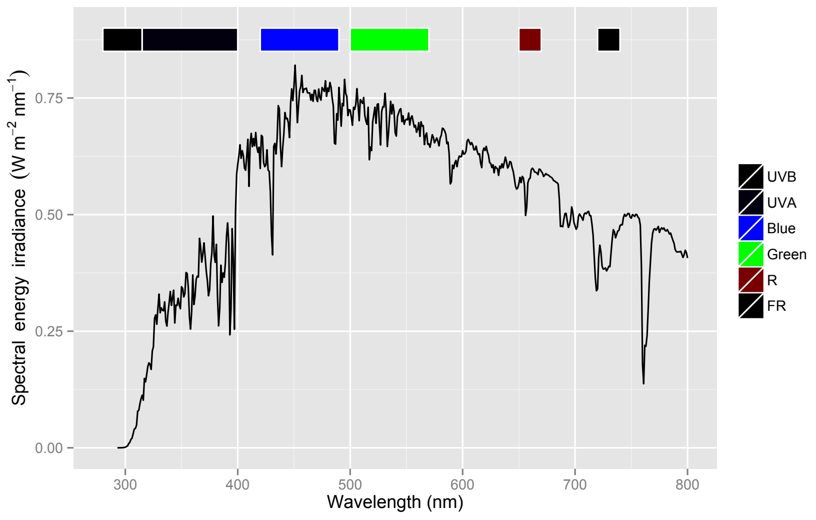
To demonstrate this we replace in the previous example, the previous tagged spectrum with one based on ISO colours. We need to do this replacement in the calls to both `geom_rect` and `scale_fill_tgspct`.

```
iso.guide.spct <- wb2rect_spct(VIS_bands())

fig_sun.z3 <- fig_sun.z0 +
  geom_rect(data=iso.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                y = 0, fill=wb.f),
            color = NA) +
  scale_fill_tgspct(tg.spct=iso.guide.spct)

fig_sun.z3
```



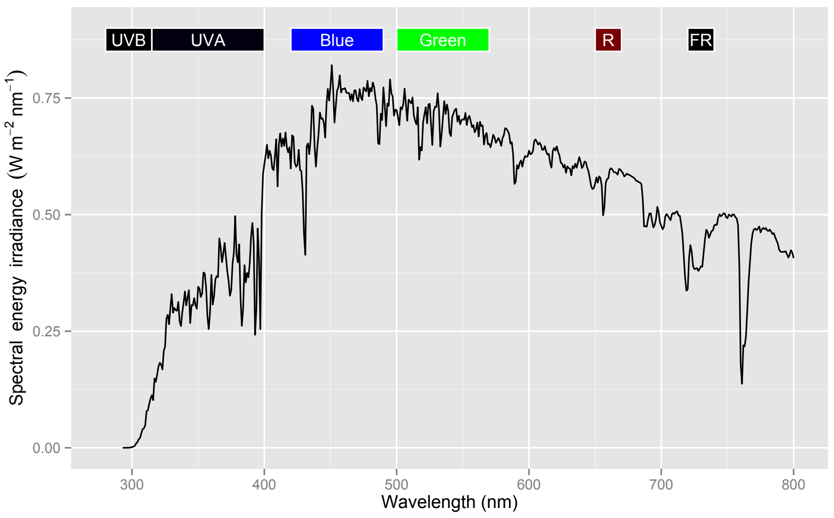
We use as an example plant's sensory colours, to show the case when the wavebands in the list are not contiguous.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z4 <- fig_sun.z0 +
  geom_rect(data=plant.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                y = 0, fill=wb.f),
            color = "white") +
  scale_fill_tgspct(tg.spct=plant.guide.spct)

fig_sun.z4
```



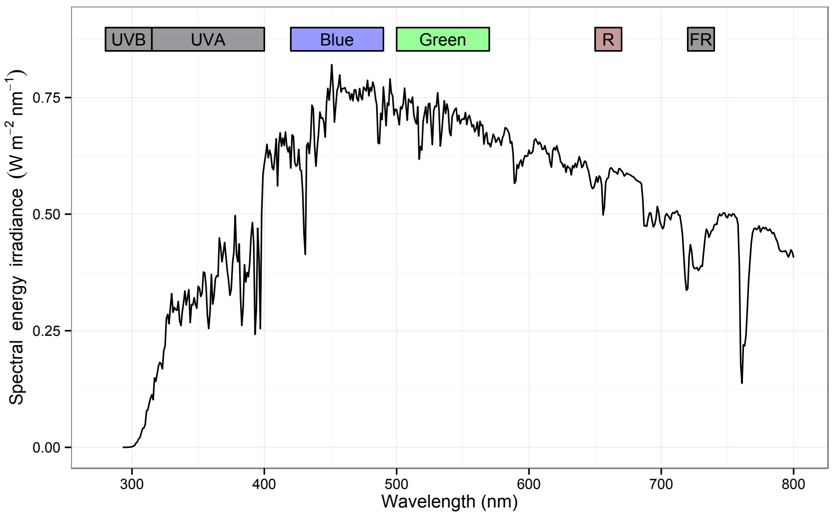
We add text labels on top of the guide, and make the rectangle borders and text white to make the separation between the different ‘invisible’ wavebands clear. As we are adding labels, the ‘guide’ or key becomes redundant and we remove it by adding `guide="none"` to the fill scale.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z5 <- fig_sun.z0 +
  geom_rect(data=plant.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                y = 0, fill=wb.f),
            color = "white") +
  geom_text(data=plant.guide.spct,
            aes(y = y + 0.875, label = as.character(wb.f)),
            color = "white", size=4) +
  scale_fill_tgspct(tg.spct=plant.guide.spct, guide="none")

fig_sun.z5
```



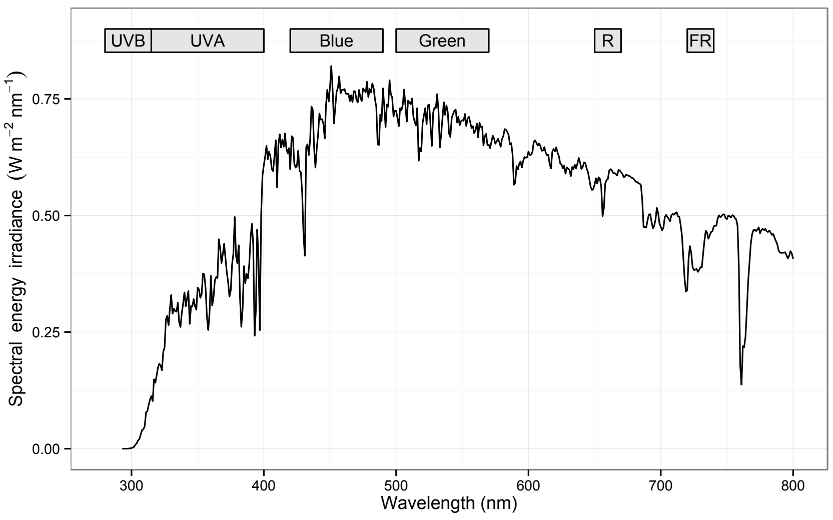
Here we add `alpha` or transparency to make the colours paler, and use black text and lines.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z6 <- fig_sun.z0 +
  geom_rect(data=plant.guide.spct,
        aes(xmin = wl.low, xmax = wl.high,
            ymin = y + 0.85, ymax = y + 0.9,
            y = 0, fill=wb.f),
        color = "black", alpha=0.4) +
  geom_text(data=plant.guide.spct,
        aes(y = y + 0.875, label = as.character(wb.f)),
        color = "black", size=4) +
  scale_fill_tgspct(tg.spct=plant.guide.spct, guide="none")

fig_sun.z6 + theme_bw()
```



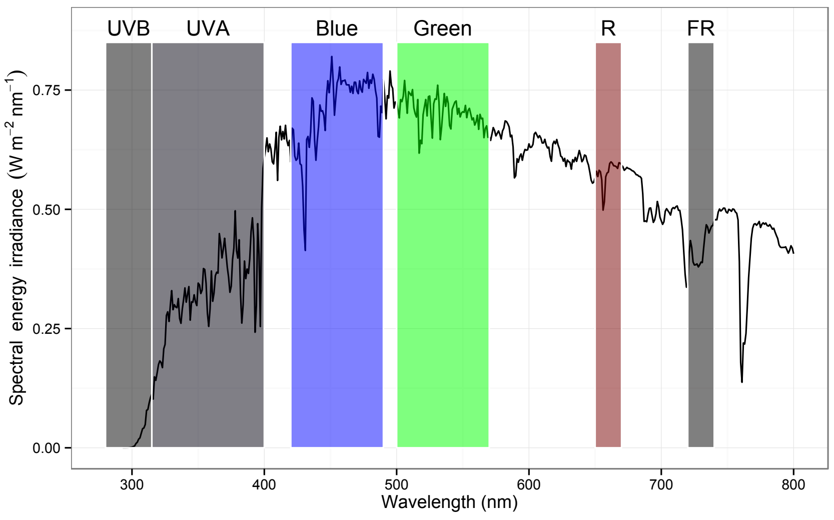
We change the guide so that all rectangles are filled with the same shade of grey by moving `fill` out of `aes` and setting it to a constant.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z7 <- fig_sun.z0 +
  geom_rect(data=plant.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                y = 0),
            color = "black", fill="grey90") +
  geom_text(data=plant.guide.spct,
            aes(y = y + 0.875, label = as.character(wb.f)),
            color = "black", size=4)

fig_sun.z7 + theme_bw()
```



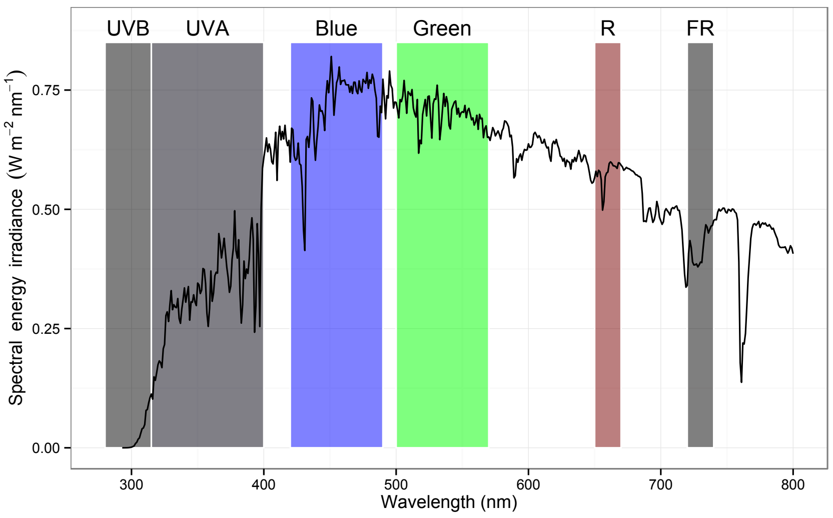
We can obtain annotations similar to those in **??** in page **??** created with `annotate_waveband` using geoms.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z8 <- fig_sun.z0 +
  geom_rect(data=plant.guide.spct,
       aes(xmin = wl.low, xmax = wl.high,
           ymin = y, ymax = y + 0.85,
           y = 0, fill=wb.f),
       color = "white", alpha=0.5) +
  geom_text(data=plant.guide.spct,
       aes(y = y + 0.88, label = as.character(wb.f)),
       color = "black") +
  scale_fill_tgspct(tg.spct=plant.guide.spct, guide="none")

fig_sun.z8 + theme_bw()
```



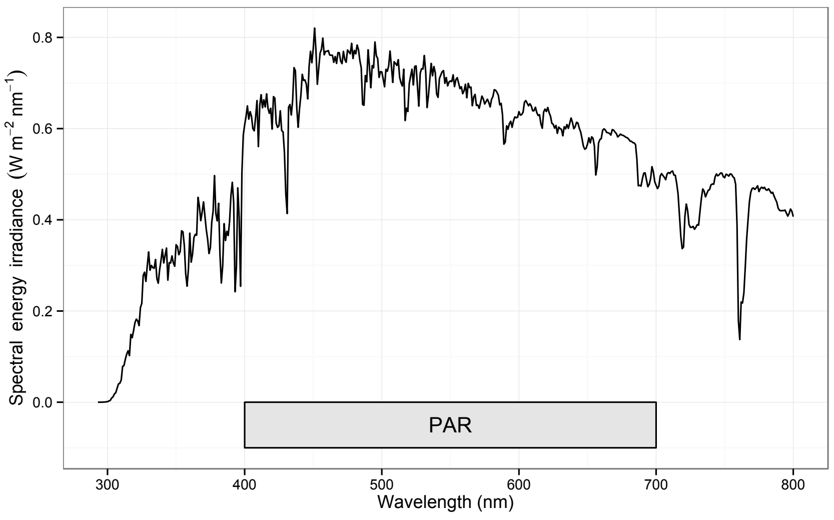
The example above can be improved by changing the order in which the geoms are added. In the plot above we can see that the rectangles are plotted on top of the line for the spectral irradiance. By changing the order we obtain a better plot.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z8a <-
  ggplot(data=sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_rect(data=plant.guide.spct,
       aes(xmin = wl.low, xmax = wl.high,
           ymin = y, ymax = y + 0.85,
           y = 0, fill=wb.f),
       color = "white", alpha=0.5) +
  geom_text(data=plant.guide.spct,
       aes(y = y + 0.88, label = as.character(wb.f)),
       color = "black") +
  geom_line() +
  scale_fill_tgspct(tg.spct=plant.guide.spct, guide="none") +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)")

fig_sun.z8a + theme_bw()
```



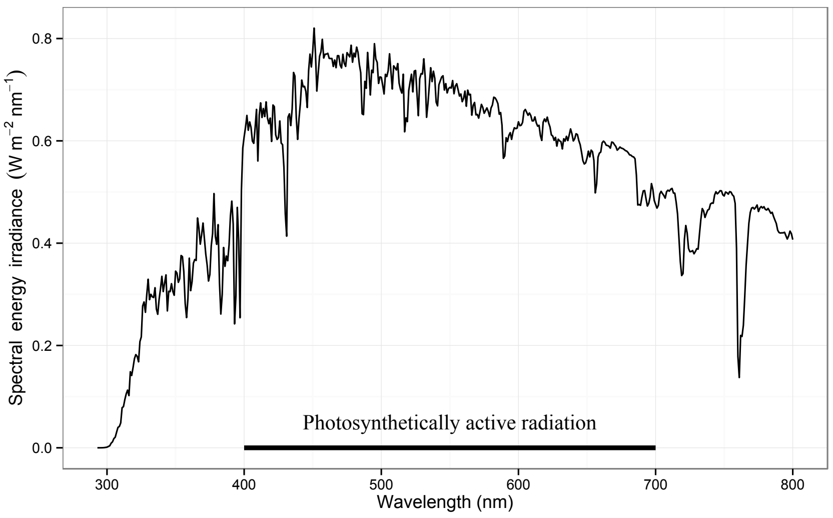
In the examples above we used predefined lists of wavebands, but one can, of course, use any list of waveband definitions, for example explicitly created with `list` and `new_waveband`, or `list` and any combination of user-defined and predefined wavebands. Even single waveband definitions are allowed.

```
par.guide.spct <- wb2rect_spct(PAR())

fig_sun.z9 <- fig_sun.z0 +
  geom_rect(data=par.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y - 0.1, ymax = y,
                y = 0),
            color = "black", fill="grey90") +
  geom_text(data=par.guide.spct,
            aes(y = y - 0.05, label = as.character(wb.f)),
            color = "black")

fig_sun.z9 + theme_bw()
```



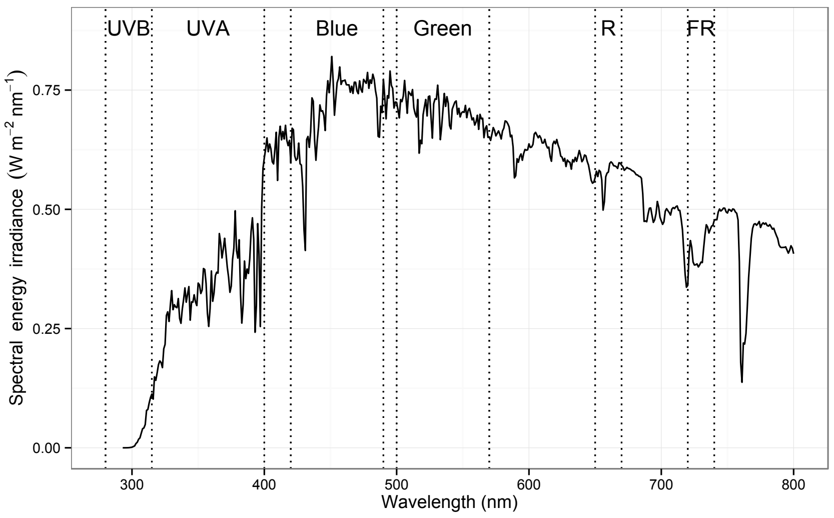
We can also use `geom_segment` to draw lines, including arrows. In this example we also set a different font `family` and label text. We can replace the label text which is by default obtained from the waveband definition by assigning a name to the waveband as member of the list. We use single quotes so that the long name containing space characters is accepted by `list`.

```
par.guide1.spct <-
  wb2rect_spct(list('Photosynthetically active radiation' = PAR()))

fig_sun.z10 <- fig_sun.z0 +
  geom_segment(data=par.guide1.spct,
       aes(x = wl.low, xend = wl.high,
           y = y, yend = y),
       size = 1.5, color = "black") +
  geom_text(data=par.guide1.spct,
       aes(y = y + 0.05, label = as.character(wb.f)),
       color = "black", family="serif")

fig_sun.z10 + theme_bw()
```



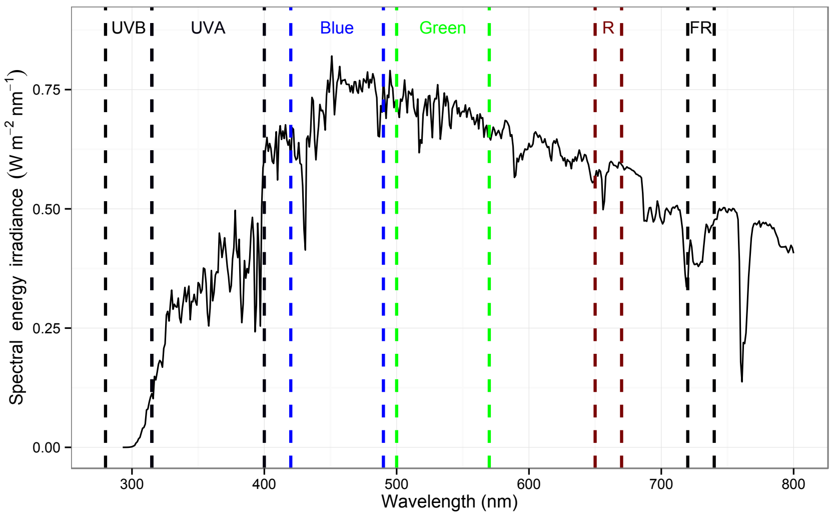
In this section we have used until now function `wb2rect_spct` to create 'spectral' annotation data from waveband definitions. Two other functions are available, that are needed or easier to use in some cases. One such case is when we have a list of wavebands and we would like to mark their boundaries with vertical lines. How to do this with `annotate` and `range` was show earlier in this chapter, but this can become tedious when we have several wavebands. Here we show an alternative approach.

```
plant.boundaries.spct <- wb2spct(Plant_bands())
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z11 <- fig_sun.z0 +
  geom_vline(data=plant.boundaries.spct,
             aes(xintercept = w.length),
             linetype = "dotted") +
  geom_text(data=plant.guide.spct,
        aes(y = y + 0.88, label = as.character(wb.f)),
        color = "black")

fig_sun.z11 + theme_bw()
```



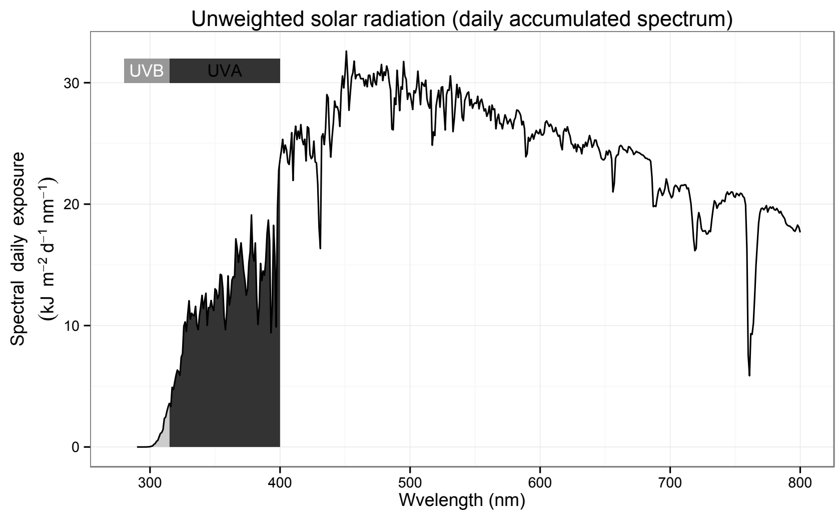
Function `wb2tagged_spct` returns the same data as `wb2spct` but 'tagged'. As shown in the next code chunk, tagging allows us to use waveband-dependent colours to the vertical lines.

```
plant.boundaries.spct <- wb2tagged_spct(Plant_bands())
plant.guide.spct <- wb2rect_spct(Plant_bands())

fig_sun.z12 <- fig_sun.z0 +
  geom_vline(data=plant.boundaries.spct,
             aes(xintercept = w.length, color=wb.f),
             size=1, linetype="dashed") +
  geom_text(data=plant.guide.spct,
        aes(y = y + 0.88, label = as.character(wb.f), colour=wb.f),
        size=4) +
  scale_colour_tgspct(tg.spct=plant.guide.spct, guide="none")

fig_sun.z12 + theme_bw()
```



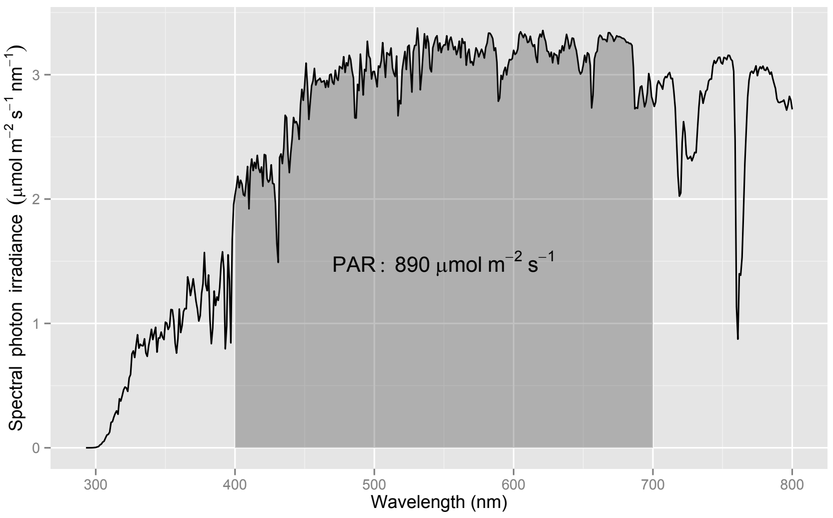
Of course it is possible to combine tagged data spectra and tagged spectra created from wavebands. The tagging is consistent, so, as demonstrated in the next figure, the same aesthetic 'link' works for both spectra. In this case the fill scale and the setting of fill to `wb.f` work accross different 'data' and yield a consistent look. This figure also shows that when assigning a constant to an aesthetic, it is possible to use a vector, which in the present example, saves us some work compared to adding a column to the data and using an identity scale. Contrary to earleir examples where we have added layers to a previously saved plot, here we show the whole code needed to build the figure.

```
my.sun.spct <- sun.daily.spct
tag(my.sun.spct, list(UVB(), UVA()))

##      w.length    s.e.irrad    s.q.irrad wl.color
##   1:      290 0.000000e+00 0.000000e+00  #000000
##   2:      291 1.788149e-04 4.349734e-10  #000000
##  ---
## 512:      799 1.808827e+04 1.208119e-01  #000000
## 513:      800 1.767467e+04 1.181973e-01  #000000
##           wb.f
##   1: UVB.tr.lo
##   2: UVB.tr.lo
##  ---
## 512:        NA
## 513:        NA

annotation.spct <- wb2rect_spct(list(UVB(), UVA()))
fig_sun.uv1 <- ggplot(my.sun.spct,
                      aes(x=w.length,
                          y=s.e.irrad * 1e-3,
                          fill=wb.f)) +
  scale_fill_grey(na.value=NA, guide="none") +
  geom_area() + geom_line() +
  labs(x = "Wvelength (nm)",
       y =  expression(atop(Spectral~~daily~~exposure,
                            (kJ~~m^{-2}~d^{-1}~nm^{-1}))),
       fill = "",
       title =
   "Unweighted solar radiation (daily accumulated spectrum)") +
  geom_rect(data=annotation.spct,
            aes(xmin=wl.low, xmax=wl.high, ymin=30, ymax=32)) +
  geom_text(data=annotation.spct,
            aes(label=as.character(wb.f), y=31),
            color=c("white","black"), size=4) +
  theme_bw()

fig_sun.uv1
```

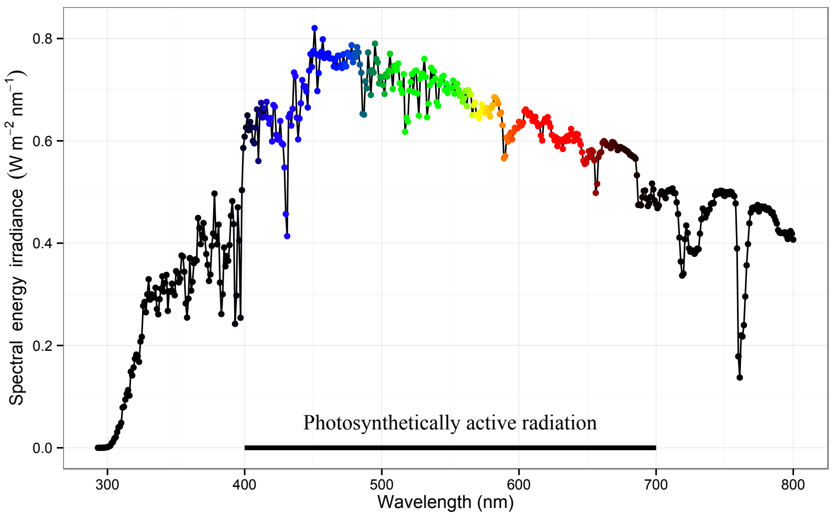


Possible variations are almost endless, so we invite the reader to continue exploring how the functions from package `photobiology` can be used together with `ggplot`, to obtain beautiful plots of spectra. As an example here we show new versions of two plots from the previous section, one using a filled area to label the PAR region, and another one using symbols with colours according to their wavelength, to which we add a guide for PAR.

```
par <- q_irrad(sun.spct, PAR()) * 1e6

fig_sun.tgrect1 <-
  ggplot(data=par.sun.spct,
         aes(x=w.length, y=s.q.irrad * 1e6)) +
  geom_line() +
  geom_area(color=NA, alpha=0.3, aes(fill=wb.f))  +
  scale_fill_grey(na.value=NA, guide="none") +
  labs(
    y = ylab_umol,
    x = "Wavelength (nm)") +
  annotate_waveband(PAR(), "text",
                    label=paste("PAR:~", signif(par,digits=2),
                                " * ~mu * mol~m^{-2}~s^{-1}", sep=""),
                    y=1.5, colour="black", size=5, parse=TRUE)

fig_sun.tgrect1
```



```
par.guide.spct <-
  wb2rect_spct(list('Photosynthetically active radiation' = PAR()))

fig_sun.tgrect2 <-
  ggplot(data=tg.sun.spct,
         aes(x=w.length, y=s.e.irrad)) +
  geom_line() +
  scale_color_identity() +
  geom_point(aes(color=wl.color))  +
  labs(
    y = ylab_watt,
    x = "Wavelength (nm)") +
  geom_segment(data=par.guide.spct,
       aes(x = wl.low, xend = wl.high, y = y, yend = y),
       size = 1.5, color = "black") +
  geom_text(data=par.guide.spct,
       aes(y = y + 0.05, label = as.character(wb.f)),
       color = "black", family="serif")

fig_sun.tgrect2 + theme_bw()
```

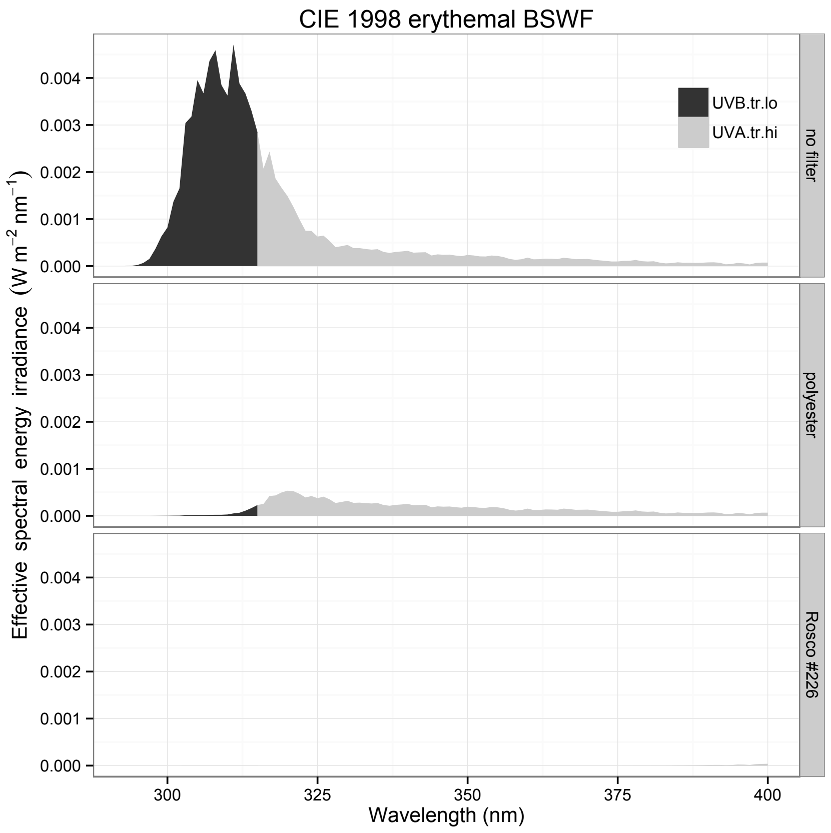


15.11 Task: plotting effective spectral irradiance

This task is here simply to show that there is nothing special about plotting spectra based on calculations, and that one can combine different functions to get the job done. We also show how to 'row bind' spectra for plotting, in this case to make it easy to use facets.

```
sun.eff.cie.nf.spct <- sun.spct * CIE()
sun.eff.cie.pe.spct <- sun.spct * polyester.new.spct * CIE()
sun.eff.cie.226.spct <- sun.spct * uv.226.new.spct * CIE()
tag(sun.eff.cie.nf.spct, UV_bands())

##      w.length    s.e.irrad wl.color      wb.f
##   1: 293.0000 2.609665e-06  #000000 UVB.tr.lo
##   2: 294.0000 6.142401e-06  #000000 UVB.tr.lo
##  ---
## 109: 399.9989 7.395673e-05  #03001E UVA.tr.hi
## 110: 399.9990 7.395674e-05  #03001E        NA

tag(sun.eff.cie.pe.spct, UV_bands())

##      w.length    s.e.irrad wl.color      wb.f
##   1: 293.0000 7.828995e-09  #000000 UVB.tr.lo
##   2: 294.0000 1.842720e-08  #000000 UVB.tr.lo
##  ---
## 109: 399.9989 6.752241e-05  #03001E UVA.tr.hi
## 110: 399.9990 6.752244e-05  #03001E        NA

tag(sun.eff.cie.226.spct, UV_bands())

##      w.length    s.e.irrad wl.color      wb.f
##   1: 293.0000 2.609665e-11  #000000 UVB.tr.lo
##   2: 294.0000 6.142401e-11  #000000 UVB.tr.lo
##  ---
## 109: 399.9989 3.863952e-05  #03001E UVA.tr.hi
## 110: 399.9990 3.863980e-05  #03001E        NA
```

```
invisible(sun.eff.cie.nf.spct[ , filter := 'no filter'])
invisible(sun.eff.cie.pe.spct[ , filter := 'polyester'])
invisible(sun.eff.cie.226.spct[ , filter := 'Rosco #226'])
sun.eff.cie.spct <- rbindspct(list(sun.eff.cie.nf.spct,
                                   sun.eff.cie.pe.spct,
                                   sun.eff.cie.226.spct))
invisible(sun.eff.cie.spct[ , filter := factor(filter)])

fig_sun.cie0 <-
  ggplot(data=sun.eff.cie.spct, aes(x=w.length, y=s.e.irrad, fill=wb.f)) +
  scale_fill_grey() +
  geom_area() +
  labs(x = xlab_nm,
       y = expression(Effective~~spectral~~energy~~irradiance~~(W~m^{-2}~nm^{-1})),
       title = "CIE 1998 erythemal BSWF") +
  facet_grid(filter~.) +
  labs(fill="") +
  xlim(NA, 400) +
  theme_bw() +
  theme(legend.position=c(0.90, 0.9))

fig_sun.cie0
```



There is one warning issued for each panel, as the use of `xlim` discards 400 observations for wavelengths longer than 400 (nm). One should be aware that these are estimated values and in practice stray light reduces the eficiency of the filters for blocking radiation, and the amount of stray light depends on many factors including the relative positions of plants, filter and sun.

A couple of details need to be remembered: the tagging has to be done before row-binding the spectra, as `tag` works only on spectra that have unique values for wavelengths and discards ‘repeated’ rows if they are present. We use `theme(legend.position=c(0.90, 0.9))` to change where the legend or guide is positioned. In this case, we move the legend to a place within the plotting region. As we are using also `theme_bw()` which resets the legend position to the default, the order in which they are added is significant.

15.12 Task: making a bar plot of effective irradiance

In this task we aim at creating bar plots depicting the contributions of the UVB and UVA bands to the total erythemal effective irradiance in sunlight filtered with different plastic films. First we calculate the effective energy irradiance using the waveband definition for erythemal BSWF (CIE98) separately for the estimated solar spectral irradiance under each filter type.

```
cie.nf.irrad <- e_irrad(sun.spct * CIE(),
                        list(UVB(), UVA()))
cie.pe.irrad <- e_irrad(sun.spct * polyester.new.spct * CIE(),
                        list(UVB(), UVA()))
cie.226.irrad <- e_irrad(sun.spct * uv.226.new.spct * CIE(),
                         list(UVB(), UVA()))
```

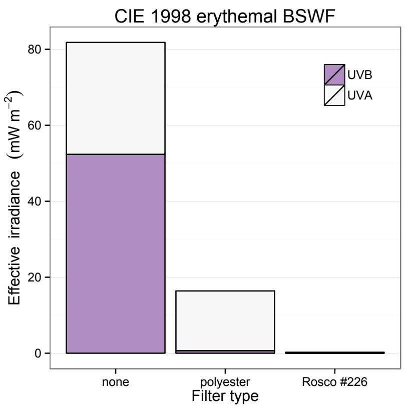
We assemble a data table by concatenating the irradiance and adding factors for filter type and wave bands. When defining the factors, we use `levels` to make sure that the levels are ordered as we would like to plot them.

```
cie.dt <- data.table(
  cie.irrad = c(cie.nf.irrad, cie.pe.irrad, cie.226.irrad),
  filter = factor(rep(c('none', 'polyester', 'Rosco #226'), c(2,2,2)),
                  levels=c('none', 'polyester', 'Rosco #226')),
  w.band = factor(rep(c('UVB', 'UVA'), 3),
                  levels=c('UVB', 'UVA')) )
```

Now we plot stacked bars using `geom_bar`, however as the default `stat` of this geom is not suitable for our data, we specify `stat="identity"` to have the data plotted as is. We set a specific palette for fill, and add a black border to the bars by means of `color="black"`, we remove the grid lines corresponding to the x-axis, and also position the legend within the plotting region.

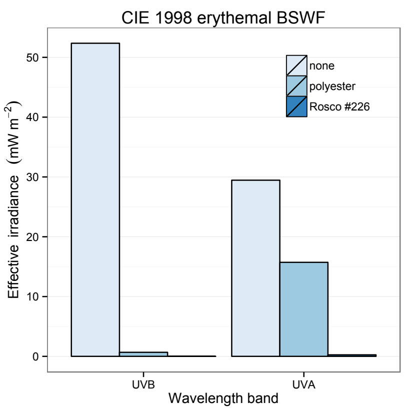
```
cie.dt

##       cie.irrad     filter w.band
## 1: 5.235708e-02       none    UVB
## 2: 2.945736e-02       none    UVA
## 3: 6.758170e-04  polyester    UVB
## 4: 1.572018e-02  polyester    UVA
## 5: 5.235708e-07 Rosco #226    UVB
## 6: 2.539816e-04 Rosco #226    UVA
```

```
fig_cie_bars0 <- ggplot(data=cie.dt,
                        aes(y = cie.irrad * 1e3,
                            x = filter,
                            fill = w.band)) +
  scale_fill_brewer(palette="PRGn") +
  geom_bar(stat="identity", colour="black") +
  labs(x = "Filter type",
       y = expression(Effective~~irradiance~~~(mW~m^{-2})),
       title = "CIE 1998 erythemal BSWF",
       fill = "") +
  theme_bw(13) +
  theme(legend.position=c(0.85, 0.85)) +
  theme(panel.grid.minor.x=element_blank(),
        panel.grid.major.x=element_blank())

fig_cie_bars0
```



The figure above is good for showing the relative contribution of UVB and UVA radiation to the total effect, and the size of the total effect. On the other hand if we would like to show how much the effective irradiance in the UVB and UVA decreases under each of the filters is better to avoid stacking of the bars, plotting them side by side using `position=position_dodge()`. In addition we swap the aesthetics to which the two factors are linked.

```
fig_cie_bars1 <- ggplot(data=cie.dt,
                        aes(y = cie.irrad * 1e3,
                            x = w.band,
                            fill=filter)) +
  geom_bar(stat="identity",
           position=position_dodge(),
           color="black") +
  scale_fill_brewer() +
  labs(x = "Wavelength band",
       y = expression(Effective~~irradiance~~~(mW~m^{-2})),
       title = "CIE 1998 erythemal BSWF",
       fill = "") +
  theme_bw() +
  theme(legend.position=c(0.80, 0.85)) +
  theme(panel.grid.minor.x=element_blank(),
        panel.grid.major.x=element_blank())

fig_cie_bars1
```



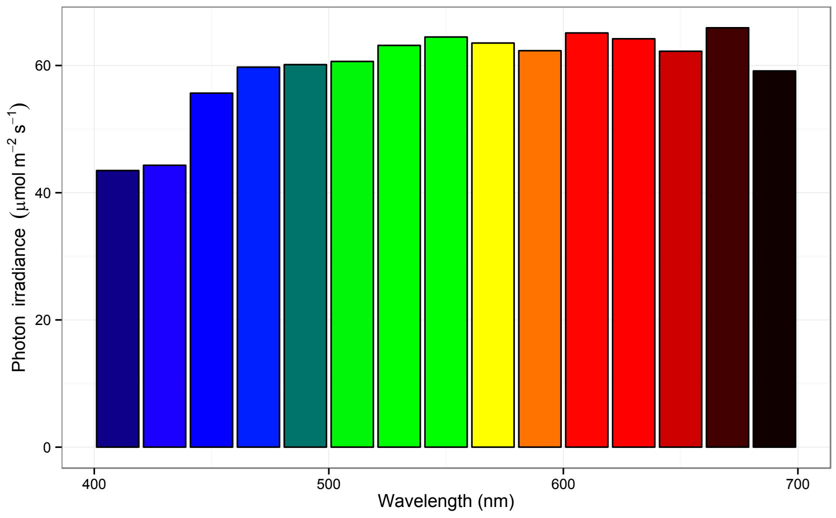
15.13 Task: plotting a spectrum using colour bars

We show now the last example, related to the ones above, but creating a bar plot with more bars. First we calculate photon irradiance for different equally spaced bands within PAR using function `split_bands`. The code is written so that by changing the first two lines you can adjust the output.

```
wl.range <- range(PAR())
num.bands <- 15
many.bands <- split_bands(wl.range, length.out=num.bands)
w.length <- numeric(num.bands)
wb.name <- wb.color <- character(num.bands)

for (i in 1:num.bands) {
  w.length[i] <- midpoint(many.bands[[i]])
  wb.color[i] <- color(many.bands[[i]], type="CMF")
  wb.name[i] <- labels(many.bands[[i]])[["name"]]
}

q.irrad.bands.sun <- q_irrad(sun.spct, many.bands)
q.irrad.sun.spct <- data.table(q.irrad = q.irrad.bands.sun,
                                   w.length = w.length,
                                   wb.color = wb.color,
                                   wb.name = wb.name)
```

Now we can plot the data as bars, filling each bar with the corresponding colour. In this case we plot the bars using a continuous variable, wavelength, for the x-axis.

```
fig_qirrad_bar <- ggplot(data=q.irrad.sun.spct,
                         aes(y = q.irrad * 1e6,
                             x = w.length,
                             fill=as.character(wb.color))) +
  geom_bar(stat="identity",
           color="black") +
  scale_fill_identity(guide="none") +
  labs(x = xlab_nm,
       y = expression(Photon~~irradiance~~(mu*mol~m^{-2}~s^{-1})),
        fill = "") +
  theme_bw()

fig_qirrad_bar
```



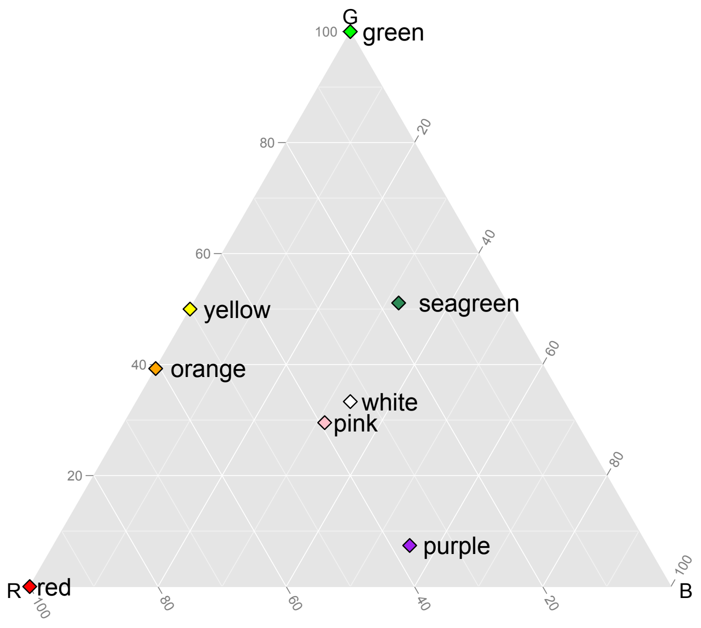
In the case of the example spectrum with equal wavelength steps, one could have directly summed the values, however, the approach shown here is valid for any type of spacing of the values along the wavelength axis, including variable one, like is the case for array spectrometers.

15.14 Task: plotting colours in Maxwell's triangle

15.14.1 Human vision: RGB

Given a color definition, we can convert it to RGB values by means of R's function `col2rgb`. We can obtain a color definition for monochromatic light from its wavelength with function `w_length2rgb` (see section **??**), from a waveband with function `color` (see section **??**), for a wavelength range with `w_length_range2rgb` (see section **??**), and from a spectrum with function `s_e_irrad2rgb` (see section **??**). The RGB values can be used to locate the position of any colour on Maxwell's triangle, given a set of chromaticity coordinates defining the triangle. In the first example we use some of R's predefined colors. We use the function `ggtern` from the package of the same name. It is based on `ggplot` and to produce a ternary diagram we need to use `ggtern` instead of `ggplot`. Geoms, aesthetics, stats and faceting function normally in most cases. Of course, being a ternary plot, the aesthetics `x`, `y`, and `z` should be all assigned to variables in the data.

```
colours <- c("red", "green", "yellow", "white",
             "orange", "purple", "seagreen", "pink")
rgb.values <- col2rgb(colours)
test.data <- data.frame(colour=colours,
                        R=rgb.values[1, ],
                                G=rgb.values[2, ],
                                B=rgb.values[3, ])
maxwell.tern <- ggtern(data=test.data,
                               aes(x=R, y=G, z=B, label=colour, fill=colour)) +
                               geom_point(shape=23, size=3) +
  geom_text(hjust=-0.2) +
  labs(x = "R", y="G", z="B") + scale_fill_identity()
maxwell.tern
```



15.15 Honey-bee vision: GBU

In this case we start with the spectral responsiveness of the photoreceptors present in the eyes of honey bees. Bees, as humans have three photoreceptors, but instead of red, green and blue (RGB), bees see green, blue and UV-A (GBU). To plot colours seen by bees one can still use a ternary plot, but the axes represent different photoreceptors than for humans, and the colour space is shifted towards shorter wavelengths.

The calculations we will demonstrate here, in addition are geared to compare a background to a foreground object (foliage vs. flower). We have followed xxxxx **chitka?** in this example, but be aware that calculations presented in this reference do not match the equations presented. In the original published example, the calculations have been simplified by leaving out $\delta\lambda$. Although not affecting the final result for their example, intermediate results are different (wrong?). We have further generalized the calculations and equations to make the calculations also valid for spectra measured using $\delta\lambda$ that itself varies along the wavelength axis. This is the usual situation with array spectrometers, nowadays frequently used when measuring reflectance.

The assessment of the perceived 'colour difference' between background and foreground objects requires taking into consideration several spectra: the incident 'light' spectrum, the reflectance spectra of the two objects, and the sensitivity spectra of three photoreceptors in the case of trichromic vision. In addition to these data, we need to take into consideration the shape of the dose response of the photoreceptors.

```
try(detach(package:photobiologygg))
try(detach(package:ggtern))
try(detach(package:ggplot2))
try(detach(package:gridExtra))
try(detach(package:photobiologyFilters))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

CHAPTER 16

Radiation physics

Abstract

In this chapter we explain how to code some optics and physics computations in R.

16.1 Packages used in this chapter

For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(ggplot2)
library(photobiologygg)

## Loading required package:  photobiology
## Loading required package:  photobiologyWavebands
##
## Attaching package:  'photobiologygg'
##
## The following objects are masked _by_ '.GlobalEnv':
##
##    scale_colour_tgspct, scale_fill_tgspct

library(photobiology)
library(photobiologyFilters)
```

16.2 Introduction

16.3 Task: black body emission

The emitted spectral radiance (L_S) is described by Planck's law of black body radiation at temperature T, measured in degrees Kelvin (K):

$$L_S(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{e^{(hc/k_B T\lambda)} - 1} \tag{16.1}$$

with Boltzmann's constant $k_B = 1.381 \times 10^{-23}$ J K^{-1}, Planck's constant $h = 6.626 \times 10^{-34}$ J s and speed of light in vacuum $c = 2.998 \times 10^8$ m s^{-1}.

We can easily define an R function based on the equation above, which returns W sr^{-1} m^{-3}:

```r
h <- 6.626e-34 # J s-1
c <- 2.998e8 # m s-1
kB <- 1.381e-23 # J K-1
black_body_spectrum <- function(w.length, Tabs) {
  w.length <- w.length * 1e-9 # nm -> m
  ((2 * h * c^2) / w.length^5) *
    1 / (exp((h * c / (kB * Tabs * w.length))) - 1)
}
```

We can use the function for calculating black body emission spectra for different temperatures:

```r
black_body_spectrum(500, 5000)

## [1] 1.212443e+13
```

The function is vectorized:

```r
black_body_spectrum(c(300,400,500), 5000)

## [1] 3.354907e+12 8.759028e+12 1.212443e+13
```

```r
black_body_spectrum(500, c(4500,5000))

## [1] 6.387979e+12 1.212443e+13
```

We aware that if two vectors are supplied, then the elements in each one are matched and recycled[1]:

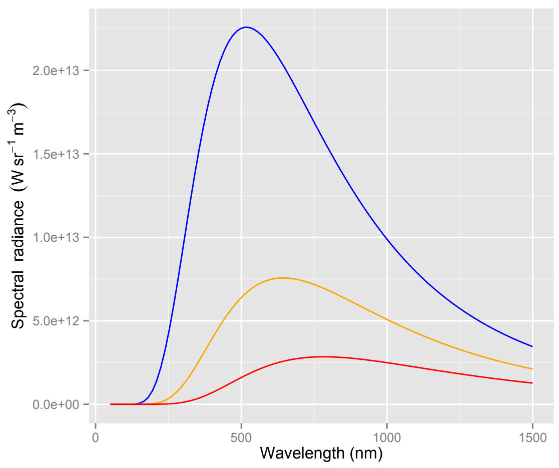
```r
black_body_spectrum(c(500, 500, 600, 600), c(4500,5000)) # tricky!

## [1] 6.387979e+12 1.212443e+13 7.474587e+12
## [4] 1.277769e+13
```

We can use the function defined above for plotting black body emission spectra for different temperatures. We use `ggplot2` and directly plot a function using `stat_function`, using `args` to pass the additional argument

[1]Exercise: calculate each of the four values individually to work out how the two vectors are being used.

giving the absolute temperature to be used. We plot three lines using three different temperatures (5600 K, 4500 K, and 3700 K):

```
ggplot(data=data.frame(x=c(50,1500)), aes(x)) +
  stat_function(fun=black_body_spectrum,
                args = list(Tabs=5600),
                colour="blue") +
  stat_function(fun=black_body_spectrum,
                args = list(Tabs=4500),
                colour="orange") +
  stat_function(fun=black_body_spectrum,
                args = list(Tabs=3700),
                colour="red") +
  labs(y=expression(Spectral~~radiance~~(W~sr^-1~m^-3)),
       x="Wavelength (nm)")
```



Wien's displacement law, gives the peak wavelength of the radiation emitted by a black body as a function of its absolute temperature.

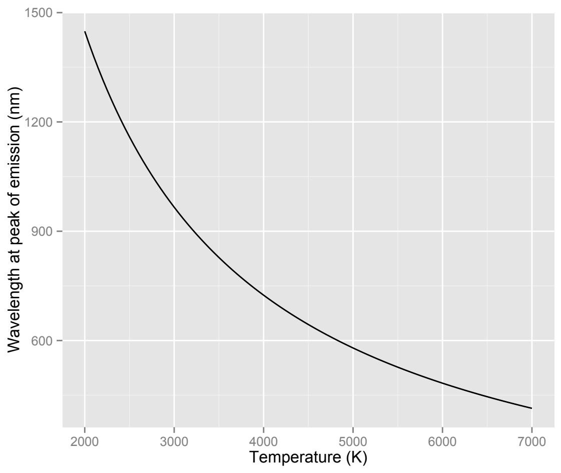
$$\lambda_{max} \cdot T = 2.898 \times 10^6 \,\mathrm{nm\,K} \qquad (16.2)$$

A function implementing this equation takes just a few lines of code:

```
k.wein <- 2.8977721e6 # nm K
black_body_peak_wl <- function(Tabs) {
  k.wein / Tabs
}
```

It can be used to plot the temperature dependence of the location of the wavelength at which radiance is at its maximum:

```
ggplot(data=data.frame(Tabs=c(2000,7000)), aes(x=Tabs)) +
  stat_function(fun=black_body_peak_wl) +
  labs(x="Temperature (K)",
       y="Wavelength at peak of emission (nm)")
```



```
try(detach(package:photobiologyFilters))
try(detach(package:photobiologygg))
try(detach(package:photobiology))
try(detach(package:ggplot2))
```

Part III

Catalogue of data sources

Part IV

Data acquisition and modelling

CHAPTER 17

Further reading about R

Bibliography

Allen, L. H., H. W. Gausman and W. A. Allen (1975). 'Solar Ultraviolet Radiation in Terrestrial Plant Communities'. In: *Journal of Environmental Quality* 4.3, pp. 285–294. DOI: 10.2134/jeq1975.00472425000400030001x.

Anderson, J., W. Chow and D. Goodchild (1988). 'Thylakoid Membrane Organisation in Sun/Shade Acclimation'. In: *Functional Plant Biology* 15, pp. 11–26. DOI: 10.1071/PP9880011.

Aphalo, P. J. (2003). 'Do current levels of UV-B radiation affect vegetation? The importance of long-term experiments'. In: *New Phytologist* 160. Invited commentary, pp. 273–276. DOI: 10.1046/j.1469-8137.2003.00905.x.

Aphalo, P. J., A. Albert, L. O. Björn, A. R. McLeod, T. M. Robson and E. Rosenqvist, eds. (2012). *Beyond the Visible: A handbook of best practice in plant UV photobiology*. 1st ed. COST Action FA0906 "UV4growth". Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology, pp. xxx + 174. ISBN: ISBN 978-952-10-8363-1 (PDF), 978-952-10-8362-4 (paperback). URL: http://hdl.handle.net/10138/37558.

Aphalo, P. J., A. Albert, L. O. Björn, L. Ylianttila, F. L. Figueroa and P. Huovinen (2012). 'Introduction'. In: *Beyond the Visible: A handbook of best practice in plant UV photobiology*. Ed. by P. J. Aphalo, A. Albert, L. O. Björn, A. R. McLeod, T. M. Robson and E. Rosenqvist. 1st ed. COST Action FA0906 "UV4growth". Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology. Chap. 1, pp. 1–33. ISBN: ISBN 978-952-10-8363-1 (PDF), 978-952-10-8362-4 (paperback). URL: http://hdl.handle.net/10138/37558 (cit. on pp. 9, 44).

Aphalo, P. J., A. Albert, A. R. McLeod, A. Heikkilä, I. Gómez, F. López Figueroa, T. M. Robson and Å. Strid (2012). 'Manipulating UV radiation'. In: *Beyond the Visible: A handbook of best practice in plant UV photobiology*. Ed. by P. J. Aphalo, A. Albert, L. O. Björn, A. R. McLeod, T. M. Robson and E. Rosenqvist. 1st ed. COST Action FA0906 "UV4growth". Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology. Chap. 2, pp. 35–70. ISBN: ISBN 978-952-10-8363-1 (PDF), 978-952-10-8362-4 (paperback). URL: http://hdl.handle.net/10138/37558.

Aphalo, P. J., T. M. Robson and H. Högmander (2012). 'Statistical design of UV experiments'. In: *Beyond the Visible: A handbook of best practice in plant UV photobiology*. Ed. by P. J. Aphalo, A. Albert, L. O. Björn, A. R. McLeod, T. M. Robson and E. Rosenqvist. 1st ed. COST Action FA0906 "UV4growth". Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology. Chap. 5, pp. 139–150. ISBN: ISBN 978-952-10-8363-1 (PDF), 978-

952-10-8362-4 (paperback). URL: http://hdl.handle.net/10138/37558.

Aphalo, P. J., R. Tegelberg and R. Julkunen-Tiitto (1999). 'The modulated UV-B irradiation system at the University of Joensuu'. In: *Biotronics* 28, pp. 109–120. URL: http://ci.nii.ac.jp/naid/110006175827/en.

Arends, G., R. K. A. M. Mallant, E. van Wensveen and J. M. Gouman (1988). *A fog chamber for the study of chemical reactions*. Tech. rep. Report - ECN - 210. Petten, Netherlands.

Austin, A. T. and C. L. Ballaré (2010). 'Dual role of lignin in plant litter decomposition in terrestrial ecosystems'. In: *Procceedings of the National Academy of Sciences of the U.S.A* 107, pp. 4618–4622. DOI: 10.1073/pnas.0909396107.

Bakker, J. C., G. P. A. Bot, H. Challa and N. J. Vand de Braak, eds. (1995). *Greenhouse climate control: An integrated approach*. Wageningen, The Netherlands: Wageningen Academic Publishers. 279 pp. ISBN: 978-90-74134-17-0. DOI: 10.3920/978-90-8686-501-7.

Ballaré, C. L., M. M. Caldwell, S. D. Flint, S. A. Robinson and J. F. Bornman (2011). 'Effects of solar ultraviolet radiation on terrestrial ecosystems. Patterns, mechanisms, and interactions with climate change'. In: *Photochemical and Photobiological Sciences* 10 (2), pp. 226–241. DOI: 10.1039/C0PP90035D.

Barnes, P. W., S. D. Flint, J. R. Slusser, W. Gao and R. J. Ryel (2008). 'Diurnal changes in epidermal UV transmittance of plants in naturally high UV environments'. In: *Physiologia Plantarum* 133, pp. 363–372. DOI: 10.1111/j.1399-3054.2008.01084.x.

Bazzaz, F. A. and R. W. Carlson (1982). 'Photosynthetic acclimation to variability in the light environment of early and late successional plants'. In: *Oecologia* 54, pp. 313–316. DOI: 10.1007/BF00379999.

Beckerman, A. P. and O. L. Petchey (2012). *Getting Started with R: An introduction for biologists*. Oxford: OUP Oxford, p. 128. ISBN: 0199601623. URL: http://www.amazon.co.uk/Getting-Started-introduction-biologists-Biology/dp/0199601623.

Bentham (1997). *A Guide to Spectroradiometry: Instruments & Applications for the Ultraviolet*. Tech. rep. Reading, U.K.: Bentham Instruments Ltd.

Bérces, A., A. Fekete, S. Gáspár, P. Gróf, P. Rettberg, G. Horneck and G. Rontó (1999). 'Biological UV dosimeters in the assessment of the biological hazard from environmental radiation'. In: *Journal of Photochemistry and Photobiology, B* 53.1-3, pp. 36–43. DOI: 10.1016/S1011-1344(99)00123-2.

Berger, D. S. (1976). 'The sunburning ultraviolet meter: design and performance'. In: *Photochemistry and Photobiology* 24, pp. 587–593. DOI: 10.1111/j.1751-1097.1976.tb06877.x.

Beytes, C., ed. (2003). *Ball Red Book: Greenhouses and equipment*. 17th ed. Batavia, IL, USA: Ball Publishing. 272 pp. ISBN: 1883052343.

Bickford, E. D. and S. Dunn (1972). *Lighting for plant growth*. Ohio, USA: Kent State University Press. x + 221. ISBN: 0873381165.

Bilger, W., T. Johnsen and U. Schreiber (2001). 'UV-excited chlorophyll fluorescence as a tool for the assessment of UV-protection by epidermins of plants'.

In: *Journal of Experimental Botany* 52, pp. 2007–2014. DOI: `10.1093/jexbot/52.363.2007`.

Bilger, W., M. Veit, L. Schreiber and U. Schreiber (1997). 'Measurement of leaf epidermal transmittance of UV radiation by chlorophyll fluorescence'. In: *Physiologia Plantarum* 101.4, pp. 754–763. DOI: `10.1111/j.1399-3054.1997.tb01060.x`.

Björn, L. O. (1995). 'Estimation of fluence rate from irradiance measurements with a cosine-corrected sensor'. In: *Journal of Photochemistry and Photobiology B Biology* 29, pp. 179–183. DOI: `10.1016/1011-1344(95)07135-O`.

Björn, L. O., ed. (2007). *Photobiology: The Science of Life and Light.* 2nd ed. Springer. 684 pp. ISBN: 0387726543.

Björn, L. O., A. R. McLeod, P. J. Aphalo, A. Albert, A. V. Lindfors, A. Heikkilä, P. Kolarz, L. Ylianttila, G. Zipoli, P. Grifoni D.and Huovinen et al. (2012). 'Quantifying UV radiation'. In: *Beyond the Visible: A handbook of best practice in plant UV photobiology.* Ed. by P. J. Aphalo, A. Albert, L. O. Björn, A. R. McLeod, T. M. Robson and E. Rosenqvist. 1st ed. COST Action FA0906 "UV4growth". Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology. Chap. 3, pp. 71–117. ISBN: ISBN 978-952-10-8363-1 (PDF), 978-952-10-8362-4 (paperback). URL: `http://hdl.handle.net/10138/37558`.

Björn, L. O. and A. H. Teramura (1993). 'Simulation of daylight ultraviolet radiation and effects of ozone depletion'. In: *Environmental UV Photobiology.* Ed. by A. R. Young, L. O. Björn, J. Moan and W. Nultsch. New York: Plenum Press, pp. 41–71. ISBN: 0-306-44443-7.

Björn, L. O. and T. C. Vogelmann (1996). 'Quantifying light and ultraviolet radiation in plant biology'. In: *Photochemistry and Photobiology* 64, pp. 403–406. DOI: `10.1111/j.1751-1097.1996.tb03084.x`.

Bloom, A. A., J. Lee-Taylor, S. Madronich, D. J. Messenger, P. I. Palmer, D. S. Reay and A. R. McLeod (2010). 'Global methane emission estimates from ultraviolet irradiation of terrestrial plant foliage'. In: *New Phytologist* 187.2, pp. 417–425. DOI: `10.1111/j.1469-8137.2010.03259.x`.

Bolker, B. M. (2008). *Ecological Models and Data in R.* 508th ed. Princeton University Press. ISBN: 0691125228.

Booker, F. L., E. L. Fiscus, R. B. Philbeck, A. S. Heagle, J. E. Miller and W. W. Heck (1992). 'A supplemental ultraviolet-B radiation system for open-top chambers'. In: *Journal of Environmental Quality* 21, pp. 56–61.

Borcard, D., F. Gillet and P. Legendre (2011). *Numerical Ecology with R.* Springer, p. 312. ISBN: 1441979751. URL: `http://www.amazon.com/Numerical-Ecology-R-Use/dp/1441979751`.

Bornman, J. F. and T. C. Vogelmann (1988). 'Penetration by blue and UV radiation measured by fiber optics in spruce and fir needles'. In: *Physiologia Plantarum* 72.4, pp. 699–705. DOI: `10.1111/j.1399-3054.1988.tb06368.x`.

Bowman, W. D. and J. N. Demas (1976). 'Ferrioxalate actinometry - Warning on its correct use'. In: *Journal of Physical Chemistry* 80.21, pp. 2434–2435. ISSN: 0022-3654. DOI: `10.1021/j100562a025`.

Braslavsky, S. E. (2007). 'Glossary of terms used in Photochemistry 3(rd) Edition (IUPAC Recommendations 2006)'. In: *Pure and Applied Chemistry* 79.3, pp. 293–465. DOI: `10.1351/pac200779030293`.

Brooms, A. C. (2010). *Data Manipulation with R*. DOI: `10.1080/02664760903075531`.

Brown, M. J., G. G. Parker and N. E. Posner (1994). 'A survey of ultraviolet-B radiation in forests'. In: *Journal of Ecology* 82.4, pp. 843–854. URL: `http://www.jstor.org/stable/2261448`.

Caldwell, M. M. (1971). 'Solar UV irradiation and the growth and development of higher plants'. In: *Photophysiology*. Ed. by A. C. Giese. Vol. 6. New York: Academic Press, pp. 131–177. ISBN: 012282606X (cit. on p. viii).

Caldwell, M. M. and S. D. Flint (1994). 'Stratospheric ozone reduction, solar UV-B radiation and terrestrial ecosystems'. In: *Climatic Change* 28.4, pp. 375–394. DOI: `10.1007/BF01104080`.

Caldwell, M. M. and S. D. Flint (2006). 'Use and Evaluation of Biological Spectral UV Weighting Functions for the Ozone Reduction Issue'. In: *Environmental UV Radiation: Impact on Ecosystems and Human Health and Predictive Models*. Ed. by F. Ghetti, G. Checcucci and J. F. Bornman. Vol. 57. NATO Science Series. Proceedings of the NATO Advanced Study Institute on Environmental UV Radiation: Impact on Ecosystems and Human Health and Predictive Models Pisa, Italy June 2001. Dordrecht: Springer, pp. 71–84. ISBN: 978-1-4020-3695-8. DOI: `10.1007/1-4020-3697-3`.

Caldwell, M. M., S. D. Flint and P. S. Searles (1994). 'Spectral balance and UV-B sensitivity of soybean: A field experiment'. In: *Plant, Cell and Environment* 17.3, pp. 267–276. DOI: `10.1111/j.1365-3040.1994.tb00292.x`.

Caldwell, M. M., W. G. Gold, G. Harris and C. W. Ashurst (1983). 'A modulated lamp system for solar UV-B (280-320 nm) supplementation studies in the field'. In: *Photochemistry and Photobiology* 37, pp. 479–485. DOI: `10.1111/j.1751-1097.1983.tb04503.x`.

Caldwell, M. M., A. H. Teramura and M. Tevini (1989). 'The Changing Solar Ultraviolet Climate and the Ecological Consequences for Higher Plants'. In: *Trends in Ecology & Evolution* 4.12, pp. 363–367. DOI: `10.1016/0169-5347(89)90100-6`.

Campbell, G. S. and J. M. Norman (1998). *An Introduction to Environmental Biophysics*. 2nd ed. New York: Springer. 286 pp. ISBN: 0-387-94937-2.

Cen, Y.-P. and J. F. Bornman (1993). 'The effect of exposure to enhanced UV-B radiation on the penetration of monochromatic and polychromatic UV-B radiation in leaves of Brassica napus'. In: *Physiologia Plantarum* 87.3, pp. 249–255. DOI: `10.1111/j.1399-3054.1993.tb01727.x`.

Chambers, J. (2009). *Software for Data Analysis: Programming with R (Statistics and Computing)*. Springer, p. 498. ISBN: 0387759352. URL: `http://www.amazon.com/Software-Data-Analysis-Programming-Statistics/dp/0387759352`.

Chang, W. (2013). *R Graphics Cookbook*. 1-2. Sebastopol: O'Reilly Media, p. 413. ISBN: 9781449316952. URL: `http://medcontent.metapress.com/index/A65RM03P4874243N.pdf`.

Christie, J. M. (2007). 'Phototropin Blue-Light Receptors'. In: *Annual Review of Plant Biology* 58, pp. 21–45. DOI: 10.1146/annurev.arplant.58.032806.103951.

Christie, J. M., A. S. Arvai, K. J. Baxter, M. Heilmann, A. J. Pratt, A. O'Hara, S. M. Kelly, M. Hothorn, B. O. Smith, K. Hitomi et al. (2012). 'Plant UVR8 Photoreceptor Senses UV-B by Tryptophan-Mediated Disruption of Cross-Dimer Salt Bridges'. In: *Science*. DOI: 10.1126/science.1218091.

Cochran, W. G. (1957). 'Analysis of covariance its nature and uses'. In: *Biometrics* 13, pp. 261–281. URL: http://www.jstor.org/stable/2527916.

Cohen, J. (1977). *Statistical Power Analysis for the Behavioral Sciences*. Revised edition. New York: Academic Press. 474 pp.

Coleman, A., R. Sarkany and S. Walker (2008). 'Clinical ultraviolet dosimetry with a CCD monochromator array spectroradiometer'. In: *Physics in Medicine and Biology* 53.18, pp. 5239–5255. DOI: 10.1088/0031-9155/53/18/026.

Cooley, N. M., H. M. F. Truscott, M. G. Holmes and T. H. Attridge (2000). 'Outdoor ultraviolet polychromatic action spectra for growth responses of *Bellis perennis* and *Cynosurus cristatus*'. In: *Journal of Photochemistry and Photobiology B: Biology* 59.1-3, pp. 64–71. DOI: 10.1016/S1011-1344(00)00141-X.

Cox, D. R. (1958). *Planning of Experiments*. New York: John Wiley & Sons. 308 pp.

Cox, D. R. and N. Reid (2000). *The Theory of the Design of Experiments*. 1st ed. Chapman and Hall/CRC. 314 pp. ISBN: 158488195X.

Crawley, M. J. (2002). *Statistical Computing: An Introduction to Data Analysis using {S}-Plus*. Chichester: Wiley, pp. x + 761. ISBN: 0-471-56040-5. URL: http://www.amazon.com/dp/0471560405.

— (2005). *Statistics: An Introduction using R*. Wiley, p. 342. ISBN: 0470022981. URL: http://www.amazon.com/Statistics-An-Introduction-using-R/dp/0470022981%20http://www.amazon.com/dp/0470022981.

— (2007). *The R Book*. John Wiley and Sons Ltd, p. 950. ISBN: 0470510242. URL: http://www.amazon.co.uk/dp/0470510242%20http://www.amazon.com/The-Book-Michael-J-Crawley/dp/0470973927%20http://www.amazon.com/The-Book-Michael-J-Crawley/dp/0470510242.

— (2012). *The R Book*. Wiley, p. 1076. ISBN: 0470973927. URL: http://www.amazon.com/The-Book-Michael-J-Crawley/dp/0470973927.

Cryer, J. D. and K.-S. Chan (2009). *Time Series Analysis: With Applications in R (Springer Texts in Statistics)*. Springer, p. 508. ISBN: 0387759581. URL: http://www.amazon.co.uk/Time-Series-Analysis-Applications-Statistics/dp/0387759581.

Dalgaard, P. (2002). *Introductory Statistics with R*. Statistics and Computing. New York: Springer, pp. xv + 267. ISBN: 0 387 95475 9.

Dalgaard, P. (2008). *Introductory Statistics with R*. Springer, p. 380. ISBN: 0387790543.

D'Antoni, H. L., L. J. Rothschild, C. Schultz, S. Burgess and J. W. Skiles (2007). 'Extreme environments in the forests of Ushuaia, Argentina'. In: *Geophysical Research Letters* 34.22. ISSN: 0094-8276. DOI: `10.1029/2007GL031096`.

D'Antoni, H. L., L. J. Rothschild and J. W. Skiles (2008). 'Reply to comment by Stephan D. Flint et al. on "Extreme environments in the forests of Ushuaia, Argentina"'. In: *Geophysical Research Letters* 35.13. ISSN: 0094-8276. DOI: `10.1029/2008GL033836`.

de la Rosa, T. M., R. Julkunen-Tiitto, T. Lehto and P. J. Aphalo (2001). 'Secondary metabolites and nutrient concentrations in silver birch seedlings under five levels of daily UV-B exposure and two relative nutrient addition rates'. In: *New Phytologist* 150, pp. 121–131. DOI: `10.1046/j.1469-8137.2001.00079.x`.

Deckmyn, G., E. Cayenberghs and R. Ceulemans (2001). 'UV-B and PAR in single and mixed canopies grown under different UV-B exclusions in the field'. In: *Plant Ecology* 154, pp. 125–133. DOI: `10.1023/A:1012920716047`.

DeLucia, E. H., T. A. Day and T. C. Vogelman (1992). 'Ultraviolet-B and visible light penetration into needles of two species of subalpine conifers during foliar development'. In: *Plant, Cell and Environment* 15.8, pp. 921–929. DOI: `10.1111/j.1365-3040.1992.tb01024.x`.

Demas, J. N., W. D. Bowman, E. F. Zalewski and R. A. Velapoldi (1981). 'Determination of the quantum yield of the ferrioxalate actinometer with electrically calibrated radiometers'. In: *Journal of Physical Chemistry* 85.19, pp. 2766–2771. ISSN: 0022-3654. DOI: `10.1021/j150619a015`.

Díaz, S., C. Camilión, J. Escobar, G. Deferrari, S. Roy, K. Lacoste, S. Demers, C. Belzile, G. Ferreyra, S. Gianesella et al. (2006). 'Simulation of ozone depletion using ambient irradiance supplemented with UV lamps.' In: *Photochemistry and Photobiology* 82.4, pp. 857–864. DOI: `10.1562/2005-09-28-RA-700`.

Díaz, S. B., J. E. Frederick, T. Lucas, C. R. Booth and I. Smolskaia (1996). 'Solar ultraviolet irradiance at Tierra del Fuego: Comparison of measurements and calculations over a full annual cycle'. In: *Geophysical Research Letters* 23.4, pp. 355–358. DOI: `10.1029/96GL00253`.

Diffey, B. L. (1987). 'A comparison of dosimeters used for solar ultraviolet radiometry'. In: *Photochem Photobiol* 46, pp. 55–60. DOI: `10.1111/j.1751-1097.1987.tb04735.x`.

— (1989). *Radiation Measurement in Photobiology.* London: Academic Press. 230 pp. ISBN: 0122158407.

Dunne, R. P. (1999). 'Polysulphone film as an underwater dosimeter for solar ultraviolet-B radiation in tropical latitudes'. In: *Marine Ecology Progress Series* 189, pp. 53–63. DOI: `10.3354/meps189053`.

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp.* Springer, p. 248. ISBN: 1461468671. URL: `http://www.amazon.co.uk/Seamless-Integration-Rcpp-Dirk-Eddelbuettel/dp/1461468671`.

Eisinger, W., T. E. Swartz, R. A. Bogomolni and L. Taiz (2000). 'The ultraviolet action spectrum for stomatal opening in broad bean'. In: *Plant Physiology* 122, pp. 99–105. DOI: `http://dx.doi.org/10.1104/pp.122.1.99`.

Everitt, B. and T. Hothorn (2011). *An Introduction to Applied Multivariate Analysis with R*. Springer, p. 288. ISBN: 1441996494. URL: http://www.amazon.co.uk/Introduction-Applied-Multivariate-Analysis-Use/dp/1441996494.

Everitt, B. S. and T. Hothorn (2009). *A Handbook of Statistical Analyses Using R*. 2nd ed. Chapman & Hall, p. 376. ISBN: 1420079336. URL: http://www.amazon.com/Handbook-Statistical-Analyses-Second-Edition/dp/1420079336.

Faraway, J. J. (2004). *Linear Models with R*. Boca Raton, FL: Chapman & Hall/CRC, p. 240. URL: http://www.maths.bath.ac.uk/~jjf23/LMR/.

Faraway, J. J. (2006). *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. Chapman & Hall/CRC Taylor & Francis Group, p. 345. ISBN: 158488424X.

Fernández, E., J. M. Figuera and A. Tobar (1979). 'Use of the potassium ferrioxalate actinometer below 254-nm'. In: *Journal of Photochemistry* 11.1, pp. 69–71. ISSN: 0047-2670. DOI: 10.1016/0047-2670(79)85008-X.

Flenley, J. R. (1992). 'Ultraviolet-B insolation and the altitudinal forest limit'. In: *Nature and dynamics of forest savanna boundaries*. Ed. by P. A. Furley, J. Proctor and J. A. Ratter. London: Chapman & Hall, pp. 273–282.

Flint, S. D., C. L. Ballare, M. M. Caldwell and R. L. McKenzie (2008). 'Comment on "Extreme environments in the forests of Ushuaia, Argentina" by Hector D'Antoni et al.' In: *Geophysical Research Letters* 35.13. ISSN: 0094-8276. DOI: 10.1029/2008GL033570.

Flint, S. D. and M. M. Caldwell (1996). 'Scaling plant ultraviolet spectral responses from laboratory action spectra to field spectral weighting factors'. In: *Journal of Plant Physiology* 148, pp. 107–114. DOI: 10.1016/S0176-1617(96)80301-4.

Flint, S. D. and M. M. Caldwell (1998). 'Solar UV-B and visible radiation in tropical forest gaps: measurements partitioning direct and diffuse radiation'. In: *Global Change Biology* 4.8, pp. 863–870. DOI: 10.1046/j.1365-2486.1998.00191.x.

Flint, S. D. and M. M. Caldwell (2003). 'A biological spectral weighting function for ozone depletion research with higher plants'. In: *Physiologia Plantarum* 117, pp. 137–144. DOI: 10.1034/j.1399-3054.2003.1170117.x.

Flint, S. D., R. J. Ryel, T. J. Hudelson and M. M. Caldwell (2009). 'Serious complications in experiments in which UV doses are effected by using different lamp heights'. In: *Journal of Photochemistry and Photobiology, B* 97.1, pp. 48–53. DOI: 10.1016/j.jphotobiol.2009.07.010.

Fox, J. (2002). *An {R} and {S-Plus} Companion to Applied Regression*. Thousand Oaks, CA, USA: Sage Publications. URL: http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/index.html.

Fox, J. and H. S. Weisberg (2010). *An R Companion to Applied Regression*. SAGE Publications, Inc, p. 472. ISBN: 141297514X. URL: http://www.amazon.com/An-R-Companion-Applied-Regression/dp/141297514X.

Frederick, J. E., P. F. Soulen, S. B. Diaz, I. Smolskaia, C. R. Booth, T. Lucas and D. Neuschuler (1993). 'Solar Ultraviolet Irradiance Observed From Southern Argentina: September 1990 to March 1991'. In: *Journal of Geophysical Research* 98, pp. 8891–8897. DOI: 10.1029/93JD00030.

Furness, N. H., P. A. Jolliffe and M. K. Upadhyaya (2005). 'Ultraviolet-B Radiation and Plant Competition: Experimental Approaches and Underlying Mechanisms'. In: *Photochemistry and Photobiology* 81, pp. 1026–1037. DOI: 10.1562/2005-08-18-RA-482.

Furusawa, Y., L. E. Quintern, H. Holtschmidt, P. Koepke and M. Saito (1998). 'Determination of erythema-effective solar radiation in Japan and Germany with a spore monolayer film optimized for the detection of UVB and UVA–results of a field campaign'. In: *Appl Microbiol Biotechnol* 50, pp. 597–603. DOI: 10.1007/s002530051341.

Gandrud, C. (2013). *Reproducible Research with R and RStudio*. Chapman and Hall/CRC, p. 294. ISBN: 1466572841. URL: http://www.amazon.com/Reproducible-Research-RStudio-Chapman-Series/dp/1466572841.

García-Pichel, F. (1995). 'A scalar irradiance fiber-optic microprobe for the measurement of ultraviolet radiation at high spatial resolution'. In: *Photochemistry and Photobiology* 61, pp. 248–254. DOI: 10.1111/j.1751-1097.1995.tb03967.x.

Geiss, O. (2003). *Manual for polysulphone dosimeter*. Tech. rep. EUR 20981 EN. European Union. URL: http://publications.jrc.ec.europa.eu/repository/bitstream/111111111/1227/1/EUR%2020981%20EN.pdf.

Gentleman, R. (2008). *R Programming for Bioinformatics*. Chapman and Hall/CRC, p. 328. ISBN: 1420063677. URL: http://www.amazon.com/Programming-Bioinformatics-Chapman-Computer-Analysis/dp/1420063677.

Goldstein, S. and J. Rabani (2008). 'The ferrioxalate and iodide-iodate actinometers in the UV region'. In: *Journal of Photochemistry and Photobiology A-Chemistry* 193.1, pp. 50–55. ISSN: 1010-6030. DOI: 10.1016/j.jphotochem.2007.06.006.

Gorton, H. L. (2010). 'Biological action spectra'. In: *Photobiological Sciences Online*. Ed. by K. C. Smith. American Society for Photobiology. URL: http://www.photobiology.info/Gorton.html.

Goulas, Y., Z. G. Cerovic, A. Cartelat and I. Moya (2004). 'Dualex: a new instrument for field measurements of epidermal ultraviolet absorbance by chlorophyll fluorescence'. In: *Applied Optics* 43.23, pp. 4488–4496. DOI: 10.1364/AO.43.004488.

Gould, K. S., T. C. Vogelmann, T. Han and M. J. Clearwater (2002). 'Profiles of photosynthesis within red and green leaves of *Quintinia serrata*'. In: *Physiologia Plantarum* 116.1, pp. 127–133. DOI: 10.1034/j.1399-3054.2002.1160116.x.

Graedel, T. E. and P. J. Crutzen (1993). *Atmospheric Change: An Earth System Perspective*. New York: WH Freeman. 446 pp. ISBN: board 0-7167-2334-4, paper 0-7167-2332-8.

Grant, R. H. (1998). 'Ultraviolet irradiance of inclined planes at the top of plant canopies'. In: *Agricultural and Forest Meteorology* 89, pp. 281–293. DOI: 10.1016/S0168-1923(97)00067-1.

— (1999a). 'Potential effect of soybean heliotropism on ultraviolet-B irradiance and dose'. In: *Agronomy Journal* 91, pp. 1017–1023. DOI: doi:10.2134/agronj1999.9161017x.

— (1999b). 'Ultraviolet-B and photosynthetically active radiation environment of inclined leaf surfaces in a maize canopy and implications for modeling'. In: *Agricultural and Forest Meteorology* 95, pp. 187–201. DOI: 10.1016/S0168-1923(99)00023-4.

— (2004). 'UV Radiation Penetration in Plant Canopies'. In: *Encyclopedia of Plant and Crop Science*, pp. 1261–1264. DOI: 10.1081/E-EPCS-120010624.

Green, A. E. S. and J. H. Miller (1975). 'Measures of biologically active radiation in the 280-340 nm region. Impacts of climate change on the environment'. In: CIAP Monograph 5, Part 1. Chap. 2.2.4.

Green, A. E. S., T. Sawada and E. P. Shettle (1974). 'The middle ultraviolet reaching the ground'. In: *Photochemistry and Photobiology* 19, pp. 251–259. DOI: 10.1111/j.1751-1097.1974.tb06508.x (cit. on p. viii).

Grifoni, D., F. Sabatini, G. Zipoli and M. Viti (2009). 'Action spectra affect variability in the climatology of biologically effective UV radiation (UVBE)'. In: *Poster presentation at the Final Seminar of COST Action 726, 13-14 May 2009, Warsaw, Poland.*

Grifoni, D., G. Zipoli, M. Viti and F. Sabatini (2008). 'Latitudinal and seasonal distribution of biologically effective UV radiation affecting human health and plant growth'. In: *Proceedings of 18th International Congress of Biometeorology, 22-26 September 2008, Tokyo, Japan.*

Häder, D.-P., E. W. Helbling, C. E. Williamson and R. C. Worrest (2011). 'Effects of UV radiation on aquatic ecosystems and interactions with climate change'. In: *Photochemical and Photobiological Sciences* 10 (2), pp. 242–260. DOI: 10.1039/C0PP90036B.

Häder, D.-P., M. Lebert, M. Schuster, L. del Ciampo, E. W. Helbling and R. McKenzie (2007). 'ELDONET—a decade of monitoring solar radiation on five continents'. In: *Photochem Photobiol* 83, pp. 1348–1357. DOI: 10.1111/j.1751-1097.2007.00168.x.

Hahne, F., W. Huber, R. Gentleman and S. Falcon (2008). *Bioconductor Case Studies (Use R!)* Springer, p. 284. ISBN: 0387772391. URL: http://www.amazon.com/Bioconductor-Case-Studies-Use-R/dp/0387772391.

Hannay, J. W. and D. J. Millar (1986). 'Phytotoxicity of phthalate plasticisers. I. Diagnosis and commercial implications'. In: *Journal of Experimental Botany* 37, pp. 883–897. DOI: 10.1093/jxb/37.6.883.

Hardwick, R. C. and R. A. Cole (1987). 'Plastics that kill plants'. In: *Outlook on Agriculture* 16.13, pp. 100–104.

Hatchard, C. G. and C. A. Parker (1956). 'A new sensitive chemical actinometer .2. Potassium ferrioxalate as a standard chemical actinometer'. In: *Proceedings of the Royal Society of London Series A-Mathematical and Physical Sciences* 235.1203, pp. 518–536. DOI: 10.1098/rspa.1956.0102.

Hegglin, M. I. and T. G. Shepherd (2009). 'Large climate-induced changes in ultraviolet index and stratosphere-to-troposphere ozone flux'. In: *Nature Geoscience* advance online publication, pp. 687-691. DOI: 10.1038/ngeo604.

Heijde, M. and R. Ulm (2012). 'UV-B photoreceptor-mediated signalling in plants'. In: *Trends in Plant Science*. DOI: `10.1016/j.tplants.2012.01.007`.

Hirose, T. (2005). 'Development of the Monsi–Saeki Theory on Canopy Structure and Function'. In: *Annals of Botany* 95, pp. 483–494. DOI: `10.1093/aob/mci047`.

Hogewoning, S. W., P. Douwstra, G. Trouwborst, W. van Ieperen and J. Harbinson (2010). 'An artificial solar spectrum substantially alters plant development compared with usual climate room irradiance spectra'. In: *Journal of Experimental Botany* 61.5, pp. 1267–1276. DOI: `10.1093/jxb/erq005`.

Holmes, M. G. (1984). 'Light Sources'. In: *Techniques in Photomorphogenesis*. Ed. by H. Smith and M. G. Holmes. Academic press, pp. 43–79. ISBN: 0126529906.

Holmes, M. G. and D. R. Keiller (2002). 'Effects of pubescence and waxes on the reflectance of leaves in the ultraviolet and photosynthetic wavebands: a comparison of a range of species'. In: *Plant Cell and Environment* 25.1, pp. 85–93. DOI: `10.1046/j.1365-3040.2002.00779.x`.

Horneck, G., P. Rettberg, E. Rabbow, W. Strauch, G. Seckmeyer, R. Facius, G. Reitz, K. Strauch and J. U. Schott (1996). 'Biological dosimetry of solar radiation for different simulated ozone column thicknesses'. In: *Journal of Photochemistry and Photobiology B-biology* 32.3, pp. 189–196. ISSN: 1011-1344. DOI: `10.1016/1011-1344(95)07219-5`.

Hunt, J. E. (1997). 'Ultraviolet-B radiation and its effects on New Zealand trees'. Ph.D. Dissertation. Canterbury, New Zealand: Lincoln University, p. 106.

Hunt, J. E. and D. L. McNeil (1998). 'Nitrogen status affects UV-B sensitivity of cucumber'. In: *Australian Journal of Plant Physiology* 25.1, pp. 79–86. DOI: `10.1071/PP97102`.

Hurlbert, S. H. (1984). 'Pseudoreplication and the design of ecological field experiments'. In: *Ecological Monographs* 54.2, pp. 187–211. DOI: `10.2307/1942661`.

Hyndman, R., A. B. Koehler, J. K. Ord and R. D. Snyder (2008). *Forecasting with Exponential Smoothing: The State Space Approach*. Springer, p. 362. ISBN: 3540719164. URL: `http://www.amazon.co.uk/Forecasting-Exponential-Smoothing-Approach-Statistics/dp/3540719164`.

Ibdah, M., A. Krins, H. K. Seidlitz, W. Heller, D. Strack and T. Vogt (2002). 'Spectral dependence of flavonol and betacyanin accumulation in *Mesembryanthemum crystallinum* under enhanced ultraviolet radiation'. In: *Plant, Cell and Environment* 25.9, pp. 1145–1154. DOI: `doi:10.1046/j.1365-3040.2002.00895.x`.

Ihaka, R. and R. Gentleman (1996). 'R: A Language for Data Analysis and Graphics'. In: *J. Comput. Graph. Stat.* 5, pp. 299–314.

Jagger, J. (1967). *Introduction to research in ultraviolet photobiology*. Englewood Cliffs, NJ, USA: Prentice-Hall. 164 pp. ISBN: 0134955722.

Jansen, M. A. K. and J. F. Bornman (2012). 'UV-B radiation: from generic stressor to specific regulator'. In: *Physiologia Plantarum* 145.4, pp. 501–504. ISSN: 1399-3054. DOI: `10.1111/j.1399-3054.2012.01656.x`.

Jenkins, G. I. (2009). 'Signal transduction in responses to UV-B radiation'. In: *Annual Review of Plant Biology* 60, pp. 407–431. DOI: `10.1146/annurev.arplant.59.032607.092953`.

Jones, H. G. (1992). *Plants and Microclimate: A Quantitative Approach to Environmental Plant Physiology*. 2nd ed. Cambridge University Press. 456 pp. ISBN: 0521425247.

Jones, L. W. and B. Kok (1966). 'Photoinhibition of Chloroplast Reactions. II. Multiple Effects'. In: *Plant Physiology* 41, pp. 1044–1049. DOI: 10.1104/pp.41.6.1044.

Julkunen-Tiitto, R., H. Häggman, P. J. Aphalo, A. Lavola, R. Tegelberg and T. Veteli (2005). 'Growth and defense in deciduous trees and shrubs under UV-B'. In: *Environmental Pollution* 137, pp. 404–414. DOI: 10.1016/j.envpol.2005.01.050.

Kalbin, G., S. Li, H. Olsman, M. Pettersson, M. Engwall and Å. Strid (2005). 'Effects of UV-B in biological and chemical systems: equipment for wavelength dependence determination'. In: *Journal of Biochemical and Biophysical Methods* 65, pp. 1–12. DOI: 10.1016/j.jbbm.2005.09.001.

Kalbina, I., S. Li, G. Kalbin, L. Björn and Å. Strid (2008). 'Two separate UV-B radiation wavelength regions control expression of different molecular markers in *Arabidopsis thaliana*'. In: *Functional Plant Biology* 35.3, pp. 222–227. DOI: 10.1071/FP07197.

Karabourniotis, G. and J. F. Bornman (1999). 'Penetration of UV-A, UV-B and blue light through the leaf trichome layers of two xeromorphic plants, olive and oak, measured by optical fibre microprobes'. In: *Physiologia Plantarum* 105, pp. 655–661. DOI: 10.1034/j.1399-3054.1999.105409.x.

Keen, K. J. (2010). *Graphics for Statistics and Data Analysis with R*. Chapman and Hall/CRC, p. 489. ISBN: 1584880872. URL: http://www.amazon.com/Graphics-Statistics-Analysis-Chapman-Statistical/dp/1584880872.

Keiller, D. R., S. A. H. Mackerness and M. G. Holmes (2003). 'The action of a range of supplementary ultraviolet (UV) wavelengths on photosynthesis in Brassica napus L. in the natural environment: effects on PSII, CO2 assimilation and level of chloroplast proteins'. In: *Photosynthesis Research* 75.2, pp. 139–150. DOI: 10.1023/A:1022812229445.

Kirk, A. D. and C. Namasivayam (1983). 'Errors in ferrioxalate actinometry'. In: *Analytical Chemistry* 55.14, pp. 2428–2429. ISSN: 0003-2700. DOI: 10.1021/ac00264a053.

Kolb, C. A., U. Schreiber, R. Gademann and E. E. Pfündel (2005). 'UV-A screening in plants determined using a new portable fluorimeter'. In: *Photosynthetica* 43.3, pp. 371–377. DOI: 10.1007/s11099-005-0061-7.

Kopp, G. and J. L. Lean (2011). 'A new, lower value of total solar irradiance: Evidence and climate significance'. In: *Geophys. Res. Lett.* 38.1, pp. L01706–. DOI: 10.1029/2010GL045777.

Kotilainen, T., A. Lindfors, R. Tegelberg and P. J. Aphalo (2011). 'How realistically does outdoor UV-B supplementation with lamps reflect ozone depletion: An assessment of enhancement errors'. In: *Photochemistry and Photobiology* 87, pp. 174–183. DOI: 10.1111/j.1751-1097.2010.00843.x.

Kotilainen, T., R. Tegelberg, R. Julkunen-Tiitto, A. Lindfors and P. J. Aphalo (2008). 'Metabolite specific effects of solar UV-A and UV-B on alder and birch leaf phenolics'. In: *Global Change Biology* 14, pp. 1294–1304. DOI: 10.1111/j.1365-2486.2008.01569.x.

Kotilainen, T., T. Venäläinen, R. Tegelberg, A. Lindfors, R. Julkunen-Tiitto, S. Sutinen, R. B. O'Hara and P. J. Aphalo (2009). 'Assessment of UV Biological Spectral Weighting Functions for Phenolic Metabolites and Growth Responses in Silver Birch Seedlings'. In: *Photochemistry and Photobiology* 85, pp. 1346–1355. DOI: 10.1111/j.1751-1097.2009.00597.x.

Kreuter, A. and M. Blumthaler (2009). 'Stray light correction for solar measurements using array spectrometers'. In: *Review of Scientific Instruments* 80.9, 096108, p. 096108. DOI: 10.1063/1.3233897.

Krizek, D. T. and R. M. Mirecki (2004). 'Evidence for phytotoxic effects of cellulose acetate in UV exclusion studies'. In: *Environmental and Experimental Botany* 51, pp. 33–43. DOI: 10.1016/S0098-8472(03)00058-3.

Kuhn, H., S. Braslavsky and R. Schmidt (2004). 'Chemical actinometry'. In: *Pure and Applied Chemistry* 76.12, pp. 2105–2146. ISSN: 0033-4545. DOI: 10.1351/pac200476122105.

Kuhn, H. J., S. E. Braslavsky and R. Schmidt (1989). 'Chemical actinometry'. In: *Pure and Applied Chemistry* 61.2, pp. 187–210. ISSN: 0033-4545. DOI: 10.1351/pac198961020187.

Lee, J. and H. H. Seliger (1964). 'Quantum yield of ferrioxalate actinometer'. In: *Journal of Chemical Physics* 40.2, pp. 519–523. ISSN: 0021-9606. DOI: 10.1063/1.1725147.

Lester, R. A., A. V. Parisi, M. G. Kimlin and J. Sabburg (2003). 'Optical properties of poly(2,6-dimethyl-1,4-phenylene oxide) film and its potential for a long-term solar ultraviolet dosimeter'. In: *Physics in Medicine and Biology* 48.22, pp. 3685–3698. DOI: 10.1088/0031-9155/48/22/005.

Leszczynski, K. (2002). 'Advances in Traceability of Solar Ultraviolet Radiation Measurements'. PhD thesis. University of Helsinki.

Long, S. P. and J.-E. Hällgren (1987). 'Measurement of CO_2 assimilation by plants in the field and the laboratory'. In: *Techniques in bioproductivity and photosynthesis.* Ed. by J. Coombes, D. O. Hall, S. P. Long and J. M. O. Scurlock. Oxford: Pergamon Press Ltd.

Loo, M. V. der and E. de Jonge (2012). *Learning RStudio for R Statistical Computing.* 1st ed. Birmingham, Mumbai: Packt Publishing, p. 126. ISBN: 9781782160601. URL: http://books.google.com/books?hl=en%5C&lr=%5C&id=EE8M9HCJok4C%5C&oi=fnd%5C&pg=PT9%5C&dq=Learning+RStudio+for+R+Statistical+Computing%5C&ots=lzFw3BLTRO%5C&sig=0uCpbnhXK219UhIirR0vZYFtOqI.

Maindonald, J. and W. J. Braun (2010). *Data Analysis and Graphics Using R: An Example-Based Approach.* Cambridge University Press, p. 552. ISBN: 0521762936. URL: http://www.amazon.com/Data-Analysis-Graphics-Using-Example-Based/dp/0521762936.

Manney, G. L., M. L. Santee, M. Rex, N. J. Livesey, M. C. Pitts, P. Veefkind, E. R. Nash, I. Wohltmann, R. Lehmann, L. Froidevaux et al. (2011). 'Unprecedented Arctic ozone loss in 2011'. In: *Nature* 478, pp. 469–475. DOI: 10.1038/nature10556.

Marijnissen, J. P. A. and W. M. Star (1987). 'Quantitative light dosimetry in vitro and in vivo'. In: *Lasers in Medical Science* 2, pp. 235–242. DOI: 10.1007/BF02594166.

Markvart, J., E. Rosenqvist, J. M. Aaslyng and C. .-.-O. Ottosen (2010). 'How is Canopy Photosynthesis and Growth of Chrysanthemums Affected by Diffuse and Direct Light?' In: *European Journal of Horticultural Science* 75.6, pp. 253–258. ISSN: 1611-4426.

Massonnet, C., D. Vile, J. Fabre, M. A. Hannah, C. Caldana, J. Lisec, G. T. S. Beemster, R. C. Meyer, G. Messerli, J. T. Gronlund et al. (2010). 'Probing the reproducibility of leaf growth and molecular phenotypes: a comparison of three *Arabidopsis* accessions cultivated in ten laboratories'. In: *Plant Physiol* 152, pp. 2142–2157. DOI: 10.1104/pp.109.148338.

Matloff, N. (2011). *The Art of R Programming: A Tour of Statistical Software Design*. No Starch Press, p. 400. ISBN: 1593273843. URL: http://www.amazon.com/The-Art-Programming-Statistical-Software/dp/1593273843.

McKinlay, A. F. and B. L. Diffey (1987). 'A reference action spectrum for ultraviolet induced erythema in human skin'. In: *CIE Journal* 6, pp. 17–22.

McLeod, A. R. (1997). 'Outdoor supplementation systems for studies of the effects of increased uv-b radiation'. In: *Plant Ecology* 128, pp. 78–92. DOI: 10.1023/A:1009794427697.

McLeod, A. R., S. C. Fry, G. J. Loake, D. J. Messenger, D. S. Reay, K. A. Smith and B.-W. Yun (2008). 'Ultraviolet radiation drives methane emissions from terrestrial plant pectins'. In: *New Phytologist* 180, pp. 124–132. DOI: 10.1111/j.1469-8137.2008.02571.x.

Messenger, D. J., A. R. McLeod and S. C. Fry (2009). 'The role of ultraviolet radiation, photosensitizers, reactive oxygen species and ester groups in mechanisms of methane formation from pectin'. In: *Plant Cell and Environment* 32, pp. 1–9. DOI: 10.1111/j.1365-3040.2008.01892.x.

Millar, D. J. and J. W. Hannay (1986). 'Phytotoxicity of phthalate plasticisers. II. Site and mode of action'. In: *Journal of Experimental Botany* 37, pp. 883–897. DOI: 10.1093/jxb/37.6.898.

Möglich, A., X. Yang, R. A. Ayers and K. Moffat (2010). 'Structure and function of plant photoreceptors'. In: *Annu Rev Plant Biol* 61, pp. 21–47. DOI: 10.1146/annurev-arplant-042809-112259.

Monsi, M. and T. Saeki (1953). 'Über den Lichfaktor in den Pflanzengesellschaften und seine Bedeutung für die Stoffproduktion'. In: *Japanese Journal of Botany* 14, pp. 22–52.

Montalti, M., A. Credi, L. Prodi and M. T. Gandolfi (2006). *Handbook of Photochemistry*. 3rd ed. Boca Raton, FL, USA: CRC Press. 664 pp. ISBN: 0824723775.

Monteith, J. and M. Unsworth (2008). *Principles of Environmental Physics*. 3rd ed. Academic Press. 440 pp. ISBN: 0125051034.

Morales, L. O., R. Tegelberg, M. Brosché, M. Keinänen, A. Lindfors and P. J. Aphalo (2010). 'Effects of solar UV-A and UV-B radiation on gene expression and phenolic accumulation in Betula pendula leaves'. In: *Tree Physiol* 30, pp. 923–934. DOI: 10.1093/treephys/tpq051.

Morison, J. I. L. and R. M. Gifford (1984). 'Ethylene contamination of CO2 cylinders. Effects on plant growth in CO2 enrichment studies'. In: *Plant Physiology* 75, pp. 275–277. DOI: 10.1104/pp.75.1.275.

Murrell, P. (2005a). *R Graphics*. Boca Raton, FL: Chapman & Hall/CRC, p. 301. ISBN: 1-584-88486-X. URL: `http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html`.

— (2005b). *R Graphics (Chapman & Hall/CRC The R Series)*. Chapman and Hall/CRC, p. 328. ISBN: 158488486X. URL: `http://www.amazon.com/Graphics-Chapman-Hall-CRC-Series/dp/158488486X`.

— (2011). *R Graphics, Second Edition (Chapman & Hall/CRC The R Series)*. CRC Press, p. 546. ISBN: 1439831769. URL: `http://www.amazon.com/Graphics-Second-Edition-Chapman-Series/dp/1439831769`.

Musil, C. F. (1995). 'Differential effects of elevated ultraviolet-B radiation on the photochemical and reproductive performances of dicotyledonous and monocotyledonous arid-environment ephemerals'. In: *Plant, Cell and Environment* 18, pp. 844–854. DOI: `10.1111/j.1365-3040.1995.tb00593.x`.

Musil, C. F., L. O. Björn, M. W. J. Scourfield and G. E. Bodeker (2002). 'How substantial are ultraviolet-B supplementation inaccuracies in experimental square-wave delivery systems?' In: *Environmental and Experimental Botany* 47.1, pp. 25–38. DOI: `DOI:10.1016/S0098-8472(01)00108-3`.

Nevas, S., A. Teuber, A. Sperling and M. Lindemann (2012). 'Stability of array spectroradiometers and their suitability for absolute calibrations'. In: *Metrologia* 49, S48–S52. DOI: `10.1088/0026-1394/49/2/S48`.

Newsham, K. K., A. R. McLeod, P. D. Greenslade and B. A. Emmett (1996). 'Appropriate controls in outdoor UV-B supplementation experiments'. In: *Global Change Biology* 2, pp. 319–324. DOI: `10.1111/j.1365-2486.1996.tb00083.x`.

Newsham, K. K., A. R. McLeod, J. D. Roberts, P. D. Greenslade and B. A. Emmet (1997). 'Direct effects of elevated UV-B radiation on the decomposition of Quercus robur leaf litter'. In: *Oikos* 79, pp. 592–602. URL: `http://www.jstor.org/stable/3546903`.

Newsham, K. K., P. Splatt, P. A. Coward, P. D. Greenslade, A. R. McLeod and J. M. Anderson (2001). 'Negligible influence of elevated UV-B radiation on leaf litter quality of *Quercus robur*'. In: *Soil Biology and Biochemistry* 33, pp. 659–665. DOI: `10.1016/S0038-0717(00)00210-8`.

Nobel, P. S. (2009). *Physicochemical and Environmental Plant Physiology*. 4th. Academic Press. 600 pp. ISBN: 0123741432.

Oke, T. R. (1988). *Boundary Layer Climates*. 2nd. Routledge. 464 pp. ISBN: 0415043190.

Okerblom, P., T. Lahti and H. Smolander (1992). 'Photosynthesis of a Scots Pine Shoot - A Comparison of 2 Models of Shoot Photosynthesis in Direct and Diffuse Radiation Fields'. In: *Tree Physiology* 10.2, pp. 111–125. DOI: `10.1093/treephys/10.2.111`.

Parisi, A., P. Schouten and D. J. Turnbull (2010). 'UV dosimeter based on Polyphenylene Oxide for the measurement of UV exposures to plants and humans over extended periods'. In: *NIWA 2010 UV Workshop: UV Radiation and its Effects - an Update 2010, 7-9 May 2010*. Queenstown, New Zealand.

Parisi, A., D. J. Turnbull, P. Schouten, N. Downs and T. J. (2010). 'Techniques for solar dosimetry in different environments'. In: *UV radiation in global climate change: measurements, modeling and effects on ecosystems*. Ed. by

W. Gao, D. L. Schmoldt and J. R. Slusser. Springer / Shingua University Press, pp. 192–204. ISBN: 978-3-642-03312-4.

Parisi, A. V., V. J. Galea and C. Randall (2003). 'Dosimetric measurement of the visible and UV exposures on field grown soybean plants'. In: *Agricultural and Forest Meteorology* 120, pp. 153–160. DOI: 10.1016/j.agrformet.2003.08.012.

Parisi, A. V. and M. G. Kimlin (2004). 'Personal solar UV exposure measurements employing modified polysulphone with an extended dynamic range'. In: *Photochem Photobiol* 79, pp. 411–415. DOI: 10.1111/j.1751-1097.2004.tb00028.x.

Parisi, A. V. and J. C. F. Wong (1996). 'Plant canopy shape and the influences on UV exposures to the canopy'. In: *Photochemistry and Photobiology* 63.6, pp. 143–148. DOI: 10.1111/j.1751-1097.1996.tb02434.x.

Parisi, A. V., J. C. F. Wong and C. Randall (1998). 'Simultaneous assessment of photosynthetically active and ultraviolet solar radiation'. In: *Agricultural and Forest Meteorology* 92, pp. 97–103. DOI: 10.1016/S0168-1923(98)00094-X.

Parker, C. A. (1953). 'A new sensitive chemical actinometer. 1. Some trials with potassium ferrioxalate'. In: *Proc. Roy. Soc. London* 220A.1140, pp. 104–116. DOI: 10.1098/rspa.1953.0175.

Passioura, J. (2006). 'The perils of pot experiments'. In: *Functional Plant Biology* 33.12, pp. 1075–1079. DOI: 10.1071/FP06223.

Paul, N. (2001). 'Plant responses to UV-B: time to look beyond stratospheric ozone depletion?' In: *New Phytologist* 150, pp. 5–8. DOI: 10.1046/j.1469-8137.2001.00090.x.

Paul, N. D., R. J. Jacobson, A. Taylor, J. J. Wargent and J. P. Moore (2005). 'The use of wavelength-selective plastic cladding materials in horticulture: understanding of crop and fungal responses through the assessment of biological spectral weighting functions'. In: *Photochem Photobiol* 81.5, pp. 1052–1060. DOI: 10.1562/2004-12-06-RA-392.

Petris, G., S. Petrone and P. Campagnoli (2009). *Dynamic Linear Models with R (Use R!)* Springer, p. 268. ISBN: 0387772375. URL: http://www.amazon.co.uk/Dynamic-Linear-Models-Giovanni-Petris/dp/0387772375.

Phoenix, G. K., D. Gwynn-Jones, J. A. Lee and T. V. Callaghan (2003). 'Ecological importance of ambient solar ultraviolet radiation to a sub-arctic heath community'. In: *Plant Ecology* 165, pp. 263–273. DOI: 10.1023/A:1022276831900.

Pinheiro, J. C. and D. M. Bates (2000). *Mixed-Effects Models in S and S-Plus*. New York: Springer.

Poorter, H., J. Bühler, D. van Dusschoten, J. Climent and J. A. Postma (2012). 'Pot size matters: a meta-analysis of the effects of rooting volume on plant growth'. In: *Functional Plant Biology*, DOI: 10.1071/FP12049.

Poorter, H., F. Fiorani, M. Stitt, U. Schurr, A. Finck, Y. Gibon, B. Usadel, R. Munns, O. K. Atkin, F. Tardieu et al. (2012). 'The art of growing plants for experimental purposes: a practical guide for the plant biologist'. In: *Functional Plant Biology*. DOI: 10.1071/FP12028.

Quaite, F. E., B. M. Sutherland and J. C. Sutherland (1992). 'Action spectrum for DNA damage in alfalfa lowers predicted impact of ozone depletion'. In: *Nature* 358, pp. 576–578. DOI: 10.1038/358576a0.

Quinn, G. P. and M. J. Keough (2002). *Experimental Design and Data Analysis for Biologists*. Cambridge, U.K.: Cambridge University Press. xvii + 537. ISBN: 0-521-00976-6.

Quintern, L. E., Y. Furusawa, K. Fukutsu and H. Holtschmidt (1997). 'Characterization and application of UV detector spore films: the sensitivity curve of a new detector system provides good similarity to the action spectrum for UV-induced erythema in human skin'. In: *J Photochem Photobiol B* 37, pp. 158–166. DOI: 10.1016/S1011-1344(96)04414-4.

Quintern, L. E., G. Horneck, U. Eschweiler and H. Bücker (1992). 'A biofilm used as ultraviolet-dosimeter'. In: *Photochemistry and Photobiology* 55, pp. 389–395. DOI: 10.1111/j.1751-1097.1992.tb04252.x.

Quintern, L. E., M. Puskeppeleit, P. Rainer, S. Weber, S. el Naggar, U. Eschweiler and G. Horneck (1994). 'Continuous dosimetry of the biologically harmful UV-radiation in Antarctica with the biofilm technique'. In: *J Photochem Photobiol B* 22, pp. 59–66. DOI: 10.1016/1011-1344(93)06954-2.

Ritz, C. and J. C. Streibig (2009). *Nonlinear Regression with R*. Springer, p. 148. ISBN: 0387096159. URL: http://www.amazon.co.uk/Nonlinear-Regression-R-Use/dp/0387096159.

Rizzini, L., J.-J. Favory, C. Cloix, D. Faggionato, A. O'Hara, E. Kaiserli, R. Baumeister, E. Schäfer, F. Nagy, G. I. Jenkins et al. (2011). 'Perception of UV-B by the *Arabidopsis* UVR8 Protein'. In: *Science* 332.6025, pp. 103–106. DOI: 10.1126/science.1200660.

Robert, C. and G. Casella (2009). *Introducing Monte Carlo Methods with R*. Springer, p. 306. ISBN: 1441915753. URL: http://www.amazon.co.uk/Introducing-Monte-Carlo-Methods-Use/dp/1441915753.

Robertson, D. F. (1972). 'Solar ultraviolet radiation in relation to human sunburn and skin cancer'. PhD thesis. University of Queensland.

Robson, T. M., V. A. Pancotto, C. L. Ballaré, O. E. Sala, A. L. Scopel and M. M. Caldwell (2004). 'Reduction of solar UV-B mediates changes in the *Sphagnum capitulum* microenvironment and the peatland microfungal community'. In: *Oecologia* 140, pp. 480–490. DOI: 10.1007/s00442-004-1600-9.

Rockwell, N. C., Y.-S. Su and J. C. Lagarias (2006). 'Phytochrome structure and signaling mechanisms'. In: *Annu Rev Plant Biol* 57, pp. 837–858. DOI: 10.1146/annurev.arplant.56.032604.144208.

Rosenqvist, E., F. López Figueroa, I. Gómez and P. J. Aphalo (2012). 'Plant growing conditions'. In: *Beyond the Visible: A handbook of best practice in plant UV photobiology*. Ed. by P. J. Aphalo, A. Albert, L. O. Björn, A. R. McLeod, T. M. Robson and E. Rosenqvist. 1st ed. COST Action FA0906 "UV4growth". Helsinki: University of Helsinki, Department of Biosciences, Division of Plant Biology. Chap. 4, pp. 119–138. ISBN: ISBN 978-952-10-8363-1 (PDF), 978-952-10-8362-4 (paperback). URL: http://hdl.handle.net/10138/37558.

Rousseaux, M. C., S. D. Flint, P. S. Searles and M. M. Caldwell (2004). 'Plant responses to current solar ultraviolet-B radiation and to supplemented solar ultraviolet-B radiation simulating ozone depletion: an experimental comparison'. In: *Photochem Photobiol* 80, pp. 224–230. DOI: 10.1562/2004-03-30-RA-129.

Rousseaux, M. C., R. Julkunen-Tiitto, P. S. Searles, A. L. Scopel, P. J. Aphalo and C. L. Ballaré (2004). 'Solar UV-B radiation affects leaf quality and insect herbivory in the southern beech tree *Nothofagus antarctica*'. In: *Oecologia* 138, pp. 505–512. DOI: 10.1007/s00442-003-1471-5.

Rozema, J., J. Vandestaaij, L. O. Björn and M. Caldwell (1997). 'UV-B as an environmental factor in plant life—Stress and regulation'. In: *Trends in Ecology & Evolution* 12, pp. 22–28. DOI: 10.1016/S0169-5347(96)10062-8.

Ruggaber, A., R. Dlugi and T. Nakajima (1994). 'Modelling radiation quantities and photolysis frequencies in the troposphere'. In: *Journal of Atmospheric Chemistry* 18, pp. 171–210. DOI: 10.1007/BF00696813.

Rupert, C. S. (1974). 'Dosimetric concepts in photobiology'. In: *Photochemistry and Photobiology* 20, pp. 203–212. DOI: 10.1111/j.1751-1097.1974.tb06568.x.

Saitou, T., Y. Tachikawa, H. Kamada, M. Watanabe and H. Harada (1993). 'Action spectrum for light-induced formation of adventitious shoots in hairy roots of horseradish'. In: *Planta* 189, pp. 590–592. DOI: 10.1007/BF00198224.

Sampath-Wiley, P. and L. S. Jahnke (2011). 'A new filter that accurately mimics the solar UV-B spectrum using standard UV lamps: the photochemical properties, stabilization and use of the urate anion liquid filter'. In: *Plant Cell Environ* 34, pp. 261–269. DOI: 10.1111/j.1365-3040.2010.02240.x.

Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. 1st ed. Springer, p. 268. ISBN: 0387759689. URL: http://www.amazon.com/Lattice-Multivariate-Data-Visualization-Use/dp/0387759689.

Schouten, P. W., A. V. Parisi and D. J. Turnbull (2007). 'Evaluation of a high exposure solar UV dosimeter for underwater use'. In: *Photochemistry and Photobiology* 83, pp. 931–937. DOI: 10.1111/j.1751-1097.2007.00085.x.

— (2008). 'Field calibrations of a long-term UV dosimeter for aquatic UV-B exposures'. In: *Journal of Photochemistry and Photobiology, B* 91, pp. 108–116. DOI: 10.1016/j.jphotobiol.2008.02.004.

— (2010). 'Usage of the polyphenylene oxide dosimeter to measure annual solar erythemal exposures'. In: *Photochemistry and Photobiology* 86, pp. 706–710. DOI: 10.1111/j.1751-1097.2010.00720.x.

Schwander, H., P. Koepke, A. Ruggaber, T. Nakajima, A. Kaifel and A. Oppenrieder (2000). *System for transfer of atmospheric radiation STAR - version 2000*.

Seckmeyer, G., A. Bais, G. Bernhard, M. Blumthaler, C. R. Booth, P. Disterhoft, P. Eriksen, R. L. McKenzie, M. Miyauchi and C. Roy (2001). *Instruments to Measure Solar Ultraviolet Radiation - Part 1: Spectral Instruments*. Tech. rep. WMO/TD-No. 1066, GAW Report No. 125. Geneva: World Meteorological Organization.

Seckmeyer, G., A. Bais, G. Bernhard, M. Blumthaler, C. R. Booth, K. Lantz, R. L. McKenzie, P. Disterhoft and A. Webb (2005). *Instruments to Measure Solar*

Ultraviolet Radiation Part 2: Broadband Instruments Measuring Erythemally Weighted Solar Irradiance. WMO-GAW Report 164. Geneva, Switzerland: World Meteorological Organization (WMO).

Seckmeyer, G., A. Bais, G. Bernhard, M. Blumthaler, S. Drüke, P. Kiedron, K. Lantz, R. L. McKenzie, S. Riechelmann, N. Kouremeti et al. (2010). *Instruments to Measure Solar Ultraviolet Radiation - Part 4: Array Spectroradiometers*. GAW Report 191. Geneva: Global Atmosphere Watch, World Meteorological Organization. URL: http://www.wmo.int/pages/prog/arep/gaw/documents/GAW191_TD_No_1538_web.pdf.

Seckmeyer, G., A. Bais, G. Bernhard, M. Blumthaler, B. Johnsen, K. Lantz and R. McKenzie (2010). *Instruments to Measure Solar Ultraviolet Radiation - Part 3: Multi-channel filter instruments*. Tech. rep. WMO/TD-No. 1537, GAW Report No. 190. Geneva: World Meteorological Organization.

Seckmeyer, G. and H.-D. Payer (1993). 'A new sunlight simulator for ecological research on plants'. In: *Journal of Photochemistry and Photobiology B: Biology* 21.2–3, pp. 175–181. DOI: 10.1016/1011-1344(93)80180-H.

Seliger, H. H. and W. D. McElroy (1965). *Light: Physical and biological action*. New York and London: Academic Press. xi+417. ISBN: 0126358508.

Setlow, R. B. (1974). 'The wavelengths in sunlight effective in producing skin cancer: a theoretical analysis'. In: *Procceedings of the National Academy of Sciences of the U.S.A.* 71, pp. 3363–3366.

Shimazaki, K.-I., M. Doi, S. M. Assmann and T. Kinoshita (2007). 'Light Regulation of Stomatal Movement'. In: *Annual Review of Plant Biology* 58, pp. 219–247. DOI: 10.1146/annurev.arplant.57.032905.105434.

Shropshire, W. (1972). 'Action spectroscopy'. In: *Phytochrome*. Ed. by K. Mitrakos and W. Shropshire. London: Academic Press, pp. 161–181. ISBN: 0125005504.

Sliney, D. H. (2007). 'Radiometric quantities and units used in photobiology and photochemistry: recommendations of the Commission Internationale de L'Eclairage (International Commission on Illumination)'. In: *Photochemistry and Photobiology* 83, pp. 425–432. DOI: 10.1562/2006-11-14-RA-1081 (cit. on p. vii).

Smith, H. F. (1957). 'Interpretation of adjusted treatment means and regressions in analysis of covariance'. In: *Biometrics* 13, pp. 281–308. URL: http://www.jstor.org/stable/2527917.

Soetaert, K., J. Cash and F. Mazzia. *Solving Differential Equations in R*. Springer. ISBN: 3642280692. URL: http://www.amazon.com/Solving-Differential-Equations-Karline-Soetaert/dp/3642280692.

Stanghellini, C. (1987). *Transpiration of greenhouse crops—an aid to climate management*. Wageningen, NL: Intituut voor Mechanisatie, Arbeid en Gebouwen.

Stanhill, G. and S. Cohen (2001). 'Global dimming: a review of the evidence for a widespread and significant reduction in global radiation with discussion of its probable causes and possible agricultural consequences'. In: *Agricultural and Forest Meteorology* 107, pp. 255–278. DOI: 10.1016/S0168-1923(00)00241-0.

Stanhill, G. and H. Z. Enoch, eds. (1999). *Greenhouse Ecosystems, Ecosystems of the world*. Vol. 20. Amsterdam, NL: Elsevier. 434 pp. ISBN: 0444882677.

Stanton, J. (2013). *An Introduction to Data Science*. Version 3. Syracuse University, p. 196. URL: `http://jsresearch.net/wiki/projects/teachdatascience`.

Tattar, P. N. (2013). *R Statistical Application Development by Example Beginner's Guide*. 1st ed. Birmingham, Mumbai: Packt Publishing, p. 345. ISBN: 9781849519441.

Teetor, P. (2011). *R Cookbook*. 1st ed. Sebastopol: O'Reilly Media, p. 436. ISBN: 9780596809157.

Tevini, M. (1993). 'Effects of Enhanced UV-B Radiation on Terrestrial Plants'. In: *UV-B Radiation and Ozone Depletion: Effects on Humans, Animals, Plants, Microorganisms, and Materials*. Ed. by M. Tevini. Boca Raton: Lewis Publishers, pp. 125–153. ISBN: 0-87371-911-5.

Thiel, S., T. Döhring, M. Köfferlein, A. Kosak, P. Martin and H. K. Seidlitz (1996). 'A Phytotron for Plant Stress Research: How Far Can Artificial Lighting Compare to Natural Sunlight?' In: *Journal of Plant Physiology* 148.3–4, pp. 456–463. DOI: `10.1016/S0176-1617(96)80279-3`.

Thimijan, R. W., H. R. Carns and L. E. Campbell (1978). *Final Report (EPA-IAG-D6-0168): Radiation sources and related environmental control for biological and climatic effects UV research (BACER)*. Tech. rep. Washington, DC: Environmental Protection Agency (cit. on p. viii).

Tukey, J. W. (1991). 'The Philosophy of Multiple Comparisons'. In: *Statistical Science* 6.1, pp. 100–116. DOI: `10.1214/ss/1177011945`.

Turnbull, D. J. and P. W. Schouten (2008). 'Utilising polyphenylene oxide for high exposure solar UVA dosimetry'. In: *Atmospheric Chemistry and Physics* 8.10, pp. 2759–2762. DOI: `10.5194/acp-8-2759-2008`.

UNEP (2011). *2010 assessment report of the Environmental effects of ozone depletion and its interactions with climate change*. Photochemical and Photobiological Sciences 10(2), 165–320. Also published by UNEP.

Urban, O., D. Janous, M. Acosta, R. Czerny, I. Markova, M. Navratil, M. Pavelka, R. Pokorny, M. Sprtova, R. Zhang et al. (2007). 'Ecophysiological controls over the net ecosystem exchange of mountain spruce stand. Comparison of the response in direct vs. diffuse solar radiation'. In: *Global Change Biology* 13, pp. 157–168. DOI: `10.1111/j.1365-2486.2006.01265.x`.

Urban, O., K. Klem, A. Ac, K. Havránková, P. Holisová, M. Navrátil, M. Zitová, K. Kozlová, R. Pokorný, M. Sprtová et al. (2012). 'Impact of clear and cloudy sky conditions on the vertical distribution of photosynthetic CO_2 uptake within a spruce canopy'. In: *Functional Ecology* 26, pp. 46–55. DOI: `10.1111/j.1365-2435.2011.01934.x`.

Van den Boogaard, R., J. Harbinson, M. Mensink and J. Ruijsch (2001). 'Effects of quality and daily distribution of irradiance on photosynthetic electron transport and CO2 fixation in tomato'. In: *Proceedings of the 12th International Congress on Photosynthesis, Brisbane, Australia*. Vol. S28-030,

Veit, M., T. Bilger, T. Muhlbauer, W. Brummet and K. Winter (1996). 'Diurnal changes in flavonoids'. In: *Journal of Plant Physiology* 148.3-4, pp. 478–482. DOI: `10.1016/S0176-1617(96)80282-3`.

Venables, W. N. and B. D. Ripley (1999). *Modern Applied Statistics with {S-PLUS}*. 3rd. Statistics and Computing. New York: Springer, pp. x + 501. ISBN: 0 387 98825 4.

Venables, W. N. and B. D. Ripley (2000). *S Programming.* Statistics and Computing. New York: Springer, pp. x + 264. ISBN: 0 387 98966 8.

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with {S}.* 4th. New York: Springer. ISBN: 0-387-95457-0. URL: `http://www.stats.ox.ac.uk/pub/MASS4/`.

Verzani, J. (2004). *Using R for Introductory Statistics.* Chapman & Hall/CRC, p. 432. ISBN: 1584884509.

Visser, A. J., M. Tosserams, M. W. Groen, G. W. H. Magendans and J. Rozema (1997). 'The combined effects of CO_2 concentration and solar UV-B radiation on faba bean grown in open-top chambers'. In: *Plant, Cell and Environment* 20.2, pp. 189–199. DOI: `10.1046/j.1365-3040.1997.d01-64.x`.

Vogelmann, T. C. and L. O. Björn (1984). 'Measurement of light gradients and spectral regime in plant tissue with a fiber optic probe'. In: *Physiologia Plantarum* 60, pp. 361–368. DOI: `10.1111/j.1399-3054.1984.tb06076.x`.

Vogelmann, T. C. and J. R. Evans (2002). 'Profiles of light absorption and chlorophyll within spinach leaves from chlorophyll fluorescence'. In: *Plant Cell and Environment* 25, pp. 1313–1323. DOI: `10.1046/j.1365-3040.2002.00910.x`.

Vogelmann, T. C. and T. Han (2000). 'Measurements of gradients of absorbed light in spinach leaves from chlorophyll fluorescence profiles'. In: *Plant Cell and Environment* 23, pp. 1303–1311. DOI: `10.1046/j.1365-3040.2000.00649.x`.

Wargent, J. J., V. C. Gegas, G. I. Jenkins, J. H. Doonan and N. D. Paul (2009). 'UVR8 in *Arabidopsis thaliana* regulates multiple aspects of cellular differentiation during leaf development in response to ultraviolet B radiation'. In: *New Phytologist* 183.2, pp. 315–326. DOI: `10.1111/j.1469-8137.2009.02855.x`.

Watanabe, M., M. Furuya, Y. Miyoshi, Y. Inoue, I. Iwahashi and K. Matsumoto (1982). 'Design and Performance of The Okazaki Large Spectrograph for Photobiological Research'. In: *Photochemistry and Photobiology* 36, pp. 491–498. DOI: `10.1111/j.1751-1097.1982.tb04407.x`.

Webb, A., J. Gröbner and M. Blumthaler (2006). *A Practical Guide to Operating Broadband Instruments Measuring Erythemally Weighted Irradiance.* Tech. rep. Produced by the joint efforts of WMO SAG UV, Working Group 4 of COST-726 Action "Long Term Changes and Climatology of UV Radiation over Europe".

Webb, A. R., H. Slaper, P. Koepke and A. W. Schmalwieser (2011). 'Know your standard: clarifying the CIE erythema action spectrum'. In: *Photochemistry and Photobiology* 87, pp. 483–486. DOI: `10.1111/j.1751-1097.2010.00871.x`.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis.* 2nd Printi. Springer, p. 224. ISBN: 0387981403. URL: `http://www.amazon.com/ggplot2-Elegant-Graphics-Data-Analysis/dp/0387981403`.

WMO (2008). *Guide to Meteorological Instruments and Methods of Observation, WMO-No. 8.* Tech. rep. Seventh edition. World Meteorological Organization.

Wu, D., Q. Hu, Z. Yan, W. Chen, C. Yan, X. Huang, J. Zhang, P. Yang, H. Deng, J. Wang et al. (2012). 'Structural basis of ultraviolet-B perception by UVR8'. In: *Nature* 484, pp. 214–219. DOI: `10.1038/nature10931`.

Wu, M., E. Grahn, L. A. Eriksson and A. Strid (2011). 'Computational evidence for the role of *Arabidopsis thaliana* UVR8 as UV-B photoreceptor and identification of its chromophore amino acids'. In: *Journal of Chemical Information and Modeling* 51, pp. 1287–1295. DOI: `10.1021/ci200017f`.

Xie, Y. (2013). *Dynamic Documents with R and knitr (Chapman & Hall/CRC The R Series)*. Chapman and Hall/CRC, p. 216. ISBN: 1482203537. URL: `http://www.amazon.com/Dynamic-Documents-knitr-Chapman-Series/dp/1482203537`.

Xu, C. and J. H. Sullivan (2010). 'Reviewing the Technical Designs for Experiments with Ultraviolet-B Radiation and Impact on Photosynthesis, DNA and Secondary Metabolism'. In: *Journal of Integrative Plant Biology* 52, pp. 377–387. DOI: `10.1111/j.1744-7909.2010.00939.x`.

Ylianttila, L., R. Visuri, L. Huurto and K. Jokela (2005). 'Evaluation of a single-monochromator diode array spectroradiometer for sunbed UV-radiation measurements'. In: *Photochemistry and Photobiology* 81, pp. 333–341. DOI: `10.1562/2004-06-02-RA-184`.

Zuur, A., E. N. Ieno and G. M. Smith (2007). *Analysing Ecological Data (Statistics for Biology and Health)*. Springer, p. 672. ISBN: 0387459677. URL: `http://www.amazon.com/Analysing-Ecological-Statistics-Biology-Health/dp/0387459677`.

Zuur, A., E. N. Ieno, N. Walker, A. A. Saveliev and G. M. Smith (2009). *Mixed Effects Models and Extensions in Ecology with R*. New York: Springer, p. 574. ISBN: 978-0-387-87457-9. URL: `http://www.amazon.com/Effects-Extensions-Ecology-Statistics-Biology/dp/0387874577`.

Zuur, A. F., E. N. Ieno and E. Meesters (2009). *A Beginner's Guide to R*. 1st ed. Springer, p. 236. ISBN: 0387938362. URL: `http://www.amazon.com/Beginners-Guide-Use-Alain-Zuur/dp/0387938362`.

APPENDIX A

Build information

```
Sys.info()

##                       sysname
##                     "Windows"
##                       release
##                       "7 x64"
##                       version
## "build 7601, Service Pack 1"
##                      nodename
##                       "MUSTI"
##                       machine
##                      "x86-64"
##                         login
##                      "aphalo"
##                          user
##                      "aphalo"
##                effective_user
##                      "aphalo"
```

```
sessionInfo()

## R version 3.2.0 (2015-04-16)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252
## [2] LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] grid      methods   tools     stats
## [5] graphics  grDevices utils     datasets
## [9] base
##
## other attached packages:
## [1] photobiologyWavebands_0.3.1.9000
## [2] photobiology_0.6.8
## [3] scales_0.2.5
## [4] proto_0.3-10
## [5] data.table_1.9.4
## [6] stringr_1.0.0
## [7] knitr_1.10.5
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.11.6
##  [2] RColorBrewer_1.1-2
##  [3] formatR_1.2
##  [4] plyr_1.8.3
##  [5] highr_0.5
##  [6] bitops_1.0-6
##  [7] photobiologyFilters_0.3.0.9000
##  [8] photobiologyLEDs_0.3.0
##  [9] photobiologyLamps_0.3.0.90000
## [10] digest_0.6.8
## [11] lubridate_1.3.3
## [12] evaluate_0.7
## [13] memoise_0.2.1
## [14] gtable_0.1.2
## [15] lattice_0.20-31
## [16] photobiologygg_0.3.3
## [17] png_0.1-7
## [18] mapproj_1.2-2
## [19] ggtern_1.0.5.0
## [20] gridExtra_0.9.1
## [21] maps_2.3-9
## [22] RgoogleMaps_1.2.0.7
## [23] caTools_1.17.1
## [24] jpeg_0.1-8
## [25] RJSONIO_1.3-0
## [26] sp_1.1-1
## [27] ggmap_2.4
## [28] ggplot2_1.0.1
## [29] reshape2_1.4.1
## [30] splus2R_1.2-0
## [31] magrittr_1.5
## [32] photobiologyReflectors_0.3.0
## [33] MASS_7.3-40
## [34] colorspace_1.2-6
## [35] geosphere_1.3-13
## [36] labeling_0.3
## [37] stringi_0.4-1
## [38] munsell_0.4.2
## [39] rjson_0.2.15
## [40] chron_2.3-45
```