A handbook of theory and recipes

R for Photobiology

Theory and recipes for common calculations

Pedro J. Aphalo

Andreas Albert

T. Matthew Robson

Titta K. Kotilainen

Helsinki, 27 December 2016

Available through Leanpub

Contents

Contents

Contents

List of Tables

List of Figures

List of Text Boxes

Preface

Status as of 2016-12-26. We have updated the manuscript to track changes to package 'ggspectra'. This required/made possible some changes to the code in Chapter 16. In addition changes in recent versions of 'ggtern' allowed to easily improve one of the colour plotting examples. Although 'photobiology' has also been updated after the last "printing" the changes introduced did not require any changes to the code examples in this handbook. The book has been built today using current versions of all packages.

Status as of 2016-10-28. We have updated the manuscript to track package updates since the previous version uploaded nearly three months ago, and added examples of the new functionality added to packages 'ggspectra', 'ggrepel', and 'ggplot2'. Now seven of the packages in the suite are in CRAN. Package 'photobiology' has gone through a major update of the astronomy-related functions. The user interface has changed a little. The values returned are slightly different as a different algorithm has been implemented. Package 'photobiologyInOut' has been expanded in its scope. Bugs have been fixed, but most of them only affected borderline cases.

Some errors in the text of the manuscript have been corrected. During the last couple of months more time was spent in trying to get all the packages in the suite ready for submission to CRAN than on expanding and revising the text of this book. However, quite many of the code examples in the book have been simplified or updated to make use of all the improvements to the packages. Many new plotting examples were added.

I expect the next significant update to be ready for uploading to Leanpub in one or two months time.

This handbook describes how to use R as a tool for doing calculations related to research in photobiology. Photobiology is the branch of science that studies the interactions of living organisms with visible and ultraviolet radiation. Many of the most frequently used calculations are either related to the characterization of radiation and of the responses of organisms to radiation. We emphasize the first of these aspects as in many cases the characterization of the responses of organisms to radiation differ little from the characterization of similar responses elicited through

other physical or chemical stimuli. Many of the examples in this handbook make use of the 'r4photobiology' suite of R packages, but the use of other packages is also described.

The biophysical theory needed to understand the purpose and use of the different calculations is presented in the chapters of Part I (p. 3 ss.). This first part is simple and concise, and full understanding of the subject matter will require previous experience or further reading (e.g. Aphalo, Albert, Björn, Ylianttila, et al. 2012; Björn 2015).

The software used in the examples including a suite of R packages developed by one of the authors is described in Part II (p. 49 ss.). However, as several up-to-date texts suitable for learning the R language are available, we assume that readers either already have experience with R, or will familiarize with R using other books (e.g. Aphalo 2016; Horton et al. 2015; Paradis 2005; Peng 2016).

A cookbook with recipes for different calculation tasks is included as Part III (p. 65 ss.). These tasks cover a wide range of subjects but emphasis is on spectral data and their manipulation and how on to obtain from them different summary quantities relevant to different organisms. These calculations include in many cases weighted spectral data. Calculations of day length, times of sunset and sunrise, twilight, and the position of the sun are also described. Finally calculations related to colour vision are briefly described.

Part IV (p. 283 ss.) includes chapters dedicated to reading and writing data in *foreign* formats, including direct acquisition from spectrometer and other measuring instruments. Part V (p. 329 ss.) gives a brief overview of the data sets included in the suite.

Although this handbook includes many different recipes, it is not comprehensive in covering all the functionality of the packages. The packages themselves include *User Guides* and help pages describing their functionality in detail. The documentation is available as a web site at `http://docs.r4photobiology.info`. A series of articles describing specific aspects of the use of the suite is being published in the UV4Plants Bulletin.

This handbook assumes that readers are already familiar with the R language and in Part I that they are familiar with Physics and Mathematics, including calculus and geometry.

1 Typographical conventions

Code examples are typeset in monospaced font and syntax highlighted in colour. References to R language elements—i.e. R 'code'—in the main text are also in a `monospaced` font but in black on a faint background. Package names are typeset between single quotes in a 'sans serif' font.

We use the icon exemplified in the page margin next to this paragraph to highlight contents that require special attention because they are frequent causes of errors and problems.

We use the icon exemplified in the page margin next to this paragraph to highlight contents that is advanced and will require the reader to linger on it to get a deep understanding—and which can, alternatively, be skipped on first reading by those readers which want a faster path to learning to do simpler calculations.

2 Acknowledgements

We thank Stefano Catola, Paula Salonen, David Israel, Neha Rai, Tendry Randriamanana, Saara Hartikainen, Christian Bianchi-Strømme, Fang Wang and …for very useful comments and suggestions on the draft manuscript and examples used in training schools. The friendly and generous R community also deserves a big 'Thank you!'.

Helsinki, August 2016.

The authors.

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	absorptance (%).
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	reflectance (%).
σ	Stefan-Boltzmann constant.
τ	transmittance (%).
χ	water vapour content in the air ($\mathrm{g\,m^{-3}}$).
A	absorbance (absorbance units).
ANCOVA	analysis of covariance.
ANOVA	analysis of variance.
BSWF	biological spectral weighting function.
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 ($W\,m^{-2}$).
$E(\lambda)$	spectral (energy) irradiance ($W\,m^{-2}\,nm^{-1}$).
E_0	fluence rate, also called scalar irradiance ($W\,m^{-2}$).
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 ($kJ\,m^{-2}\,d^{-1}$).
H^{BE}	biologically effective (energy) exposure ($kJ\,m^{-2}\,d^{-1}$).
H_p^{BE}	biologically effective photon exposure ($mol\,m^{-2}\,d^{-1}$).
HPS	high pressure sodium, a type of discharge lamp.
HSD	honestly signifcant difference.
k_B	Boltzmann constant.
L	radiance ($W\,sr^{-1}\,m^{-2}$).
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	photosynthetically active radiation, 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	proportional-integral-derivative (control algorithm).
PMMA	polymethylmethacrylate.
PPFD	photosynthetic photon flux density, 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 ($mol\,m^{-2}\,s^{-1}$ or $\mu mol\,m^{-2}\,s^{-1}$).
$Q(\lambda)$	spectral photon irradiance ($mol\,m^{-2}\,s^{-1}\,nm^{-1}$ or $\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}$).
r_0	distance from sun to earth.
RAF	radiation amplification factor (nondimensional).
RH	relative humidity (%).
s	energy effectiveness (relative units).
$s(\lambda)$	spectral energy effectiveness (relative units).
s^p	quantum effectiveness (relative units).
$s^p(\lambda)$	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 ($\lambda = 100$–400 nm).
UV-A	ultraviolet-A radiation ($\lambda = 315$–400 nm).
UV-B	ultraviolet-B radiation ($\lambda = 280$–315 nm).
UV-C	ultraviolet-C radiation ($\lambda = 100$–280 nm).
UV^{BE}	biologically effective UV radiation.
UTC	coordinated universal time, replaces GMT in technical use.
VIS	radiation visible to the human eye (≈ 400–700 nm).
WMO	World Meteorological Organization.
VPD	water vapour pressure deficit (Pa).
WOUDC	World Ozone and Ultraviolet Radiation Data Centre.

Part I

Theory behind calculations

1

Radiation properties

1.1 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(ggplot2)
library(ggspectra)
library(photobiologyWavebands)
library(photobiologySun)
library(photobiologyLamps)
library(photobiologyLEDs)
```

1.2 Ultraviolet and visible radiation

From the viewpoint of Physics, ultraviolet (UV) and visible (VIS) radiation are both considered electromagnetic waves and are described by Maxwell's equations.[1] The wavelength ranges of UV and visible radiation and their usual names are listed in Table 1.1. The long wavelengths of solar radiation, called infrared (IR) radiation, are also listed. The colour ranges indicated in Table 1.1 are an approximation as different individual human observers will not perceive colours exactly in the same way. We follow the ISO definitions for wavelength boundaries for colours (ISO 2007). Other finer-grained colour name series are also in use (e.g. Aphalo, Albert, Björn, Ylianttila, et al. 2012, Table xx)). The electromagnetic spectrum is continuous with no clear boundaries between one colour and the next, the colours could be thought as artifacts produced by our sensory system, and are meaningful only from the perspective of an *average* human observer. Especially in the IR region the subdivision is somewhat arbitrary and the boundaries used in the literature vary.

Radiation can also be thought of as composed of quantum particles or photons. The energy of a quantum of radiation in a vacuum, q, depends on the wavelength, λ,

---

[1] These equations are a system of four partial differential equations describing classical electromagnetism.

or frequency[2], ν,

$$q = h \cdot \nu = h \cdot \frac{c}{\lambda} \tag{1.1}$$

with the Planck constant $h = 6.626 \times 10^{-34}$ Js and speed of light in vacuum $c = 2.998 \times 10^8$ m s^{-1}. When dealing with numbers of photons, the equation (1.1) 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

$$q' = h' \cdot \nu = h' \cdot \frac{c}{\lambda} \tag{1.2}$$

with $h' = h \cdot N_A = 3.990 \times 10^{-10}$ Js mol^{-1}. Example 1: red light at 600 nm has about 200 kJ mol^{-1}, therefore, 1 μmol photons has 0.2 J. Example 2: UV-B radiation at 300 nm has about 400 kJ mol^{-1}, therefore, 1 μmol photons has 0.4 J. Equations 1.1 and 1.2 are valid for all kinds of electromagnetic waves (see Sections 7.7.3 and 7.7.4 for worked-out calculation examples).

One way of understanding the relationship between the distance and positions of source and observer (or sensor) on the amount of radiation received is to use a geometric model. In the model we will use, a point source is located at the centre or origin of an imaginary sphere. As the distance from the origin increases, the surface area of the sphere at this distance increases. The relationship between the distance increase and area increase is, obviously, not linear. In addition, according to the well known cosine law, the amount of radiation received per unit area depends on the angle of incidence. After this informal introduction we will describe the model in more detail.

When a beam or the radiation passing into a space or sphere is analysed, two important parameters are necessary: the distance to the source and the measuring position—i.e. if the receiving surface is perpendicular to the beam or not. The geometry is illustrated in Figure 1.1 with a radiation source at the origin. The radiation is received at distance r by a surface of area dA, tilted by an angle α to the unit sphere's surface element, so called solid angle, $d\Omega$, which is a two-dimensional angle in a space. The relation between dA and $d\Omega$ in spherical coordinates is geometrically explained in Figure 1.1.

The solid angle is calculated from the zenith angle θ and azimuth angle ϕ, which denote the direction of the radiation beam

$$d\Omega = d\theta \cdot \sin\theta d\phi \tag{1.3}$$

The area of the receiving surface is calculated by a combination of the solid angle of the beam, the distance r from the radiation source and the angle α of the tilt:

$$dA = \frac{r d\theta}{\cos\alpha} \cdot r \sin\theta d\phi \tag{1.4}$$

---

[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 1.1.

Table 1.1: Regions of the electromagnetic radiation spectrum according to different authorities, standards or in common use. The use of what we have called *medical* and *common* definitions of the UV bands should be avoided, as it makes interpretation of experimental results and comparison of radiation quantities with studies using the accepted international standard very difficult. ISO 21348 (Technical Committee ISO/TC 20 et al. 2007), BTV (Aphalo, Albert, Björn, Ylianttila, et al. 2012), Smith (Smith 1981), Sellaro (Sellaro et al. 2010).

Waveband name	Wavelength range (nm)					
	ISO 21348	BTV	Smith	Sellaro	*medical*	*common*
UV	$100 \leq \lambda < 400$	$100 \leq \lambda < 400$				
UVC	$100 \leq \lambda < 280$	$100 \leq \lambda < 280$			$220 \leq \lambda < 290$	$200 \leq \lambda < 280$
UVB	$280 \leq \lambda < 315$	$280 \leq \lambda < 315$			$290 \leq \lambda < 320$	$280 \leq \lambda < 320$
UVA	$315 \leq \lambda < 400$	$315 \leq \lambda < 400$				$320 \leq \lambda < 400$
VIS	$380 \leq \lambda < 760$					
Purple (Violet)	$360 \leq \lambda < 450$	$400 \leq \lambda < 455$				
Blue	$450 \leq \lambda < 500$	$455 \leq \lambda < 492$		$420 \leq \lambda < 490$		
Green	$500 \leq \lambda < 570$	$492 \leq \lambda < 577$		$500 \leq \lambda < 570$		
Yellow	$570 \leq \lambda < 591$	$577 \leq \lambda < 597$				
Orange	$591 \leq \lambda < 610$	$597 \leq \lambda < 622$				
Red	$610 \leq \lambda < 760$	$622 \leq \lambda < 700$	$655 \leq \lambda < 665$	$620 \leq \lambda < 680$		
Far red		$(700 \leq \lambda < 770)$	$725 \leq \lambda < 735$	$700 \leq \lambda < 750$		
IR-A (near IR)	$760 \leq \lambda < 1400$	$770 \leq \lambda < 3000$				
IR-B (mid IR)	$1400 \leq \lambda < 3000$	$3000 \leq \lambda < 50000$				
IR-C (far IR)	$3000 \leq \lambda < 10^6$	$50000 \leq \lambda < 10^6$				

Figure 1.1: Definition of the solid angle $d\Omega$ and the geometry of areas in the space (redrawn after Eichler et al. 1993), where the given solid angle $d\Omega$ remains the same, regardless of distance r, while the exposed area exemplified by dA will change with distance r from the origin (light source) and the angle α, if the exposed area (or detector) is tilted. The angle denoted by ϕ is the azimuth angle and θ is the zenith angle.

which can be rearranged to

$$\Rightarrow dA = \frac{r^2}{\cos \alpha} \quad d\Omega \tag{1.5}$$

Thus, the solid angle is given by

$$\Omega = \int_A \frac{dA \cdot \cos \alpha}{r^2} \tag{1.6}$$

The unit of the solid angle is a steradian (sr). The solid angle of an entire sphere is calculated by integration of equation (1.3) over the zenith (θ) and azimuth (ϕ) angles, $0 \leq \theta \leq \pi(180°)$ and $0 \leq \phi \leq 2\pi(360°)$, and is 4π sr. For example, the sun or moon seen from the Earth's surface appear to have a diameter of about $0.5°$ which corresponds to a solid angle element of about 6.8×10^{-5} sr.

When radiation travels through a medium it can be absorbed (the energy 'taken up' by the material's atoms) or scattered (the direction of travel of the radiation randomly altered). Both of these phenomena affect the amount of radiation that reaches the 'other end of the path' where the observer or sensor is located, and their effect depends on the length of the path. Once again, this informal description, is stated formally below.

The processes responsible for the variation of the radiance $L(\lambda, \theta, \phi)$ as the radiation beam travels through any kind of material, are primarily absorption a and scattering b, which are called inherent optical properties, because they depend only on the characteristics of the material itself and are independent of the light field. Radiance is added to the directly transmitted beam, coming from different directions, due to elastic scattering, by which a photon changes direction but not wavelength or energy level. An example of this is Raleigh scattering in very small particles, which causes the scattering of light in a rainbow. A further gain of radiance into the direct path is due to inelastic processes like fluorescence, where a photon is absorbed by the material and reemitted as a photon with a longer wavelength and lower energy level, and Raman scattering. The elastic and inelastic scattered radiance is denoted as L^E and L^I, respectively. Internal sources of radiances, L^S, like bioluminescence of biological organisms or cells contribute also to the detected radiance. The path of the radiance through a thin horizontal layer with thickness $\mathrm{d}z = z_1 - z_0$ is shown schematically in Figure 1.2.
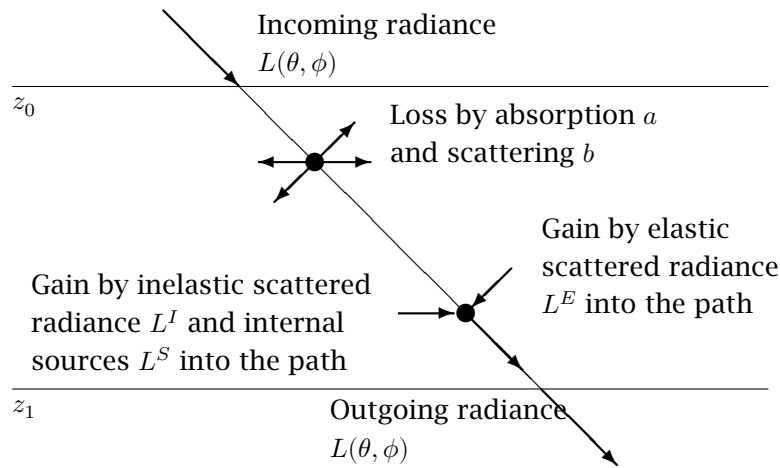
Figure 1.2: Path of the radiance and influences of absorbing and scattering particles in a thin homogeneous horizontal layer of air or water. The layer is separated from other layers of different characteristics by boundary lines at height z_0 and z_1.

Putting all this together, the radiative transfer equation is

$$\cos\theta \, \frac{\mathrm{d}L}{\mathrm{d}z} = -(a+b) \cdot L + L^E + L^I + L^S \tag{1.7}$$

The dependencies of L on λ, θ, and ϕ are omitted here for brevity. No exact analytical solution to the radiative transfer equation exists, hence it is necessary either to use numerical models or to make approximations and find an analytical parametrisation. A numerical model is for example the Monte Carlo method. The parameters of the light field can be simulated by modelling the paths of photons. For an infinite number

of photons the light field parameters reach their exact values asymptotically. The advantage of the Monte Carlo method is a relatively simple structure of the program, and that it simulates nature in a straightforward way, but its disadvantage is the time-consuming computation involved. Details of the Monte Carlo method are explained for example by Prahl et al. (1989), Wang et al. (1995)[3], or Mobley (1994).

Alternatively the radiative transfer equation be solved through analytical paramet-erisations making use of approximations for all the quantities needed. In this case, the result is not exact, but it has the advantage of fast computing and the analytical equations can be inverted just as fast. This leads to the idealised case of a source-free ($L^S = 0$) and non-scattering media, i.e. $b = 0$ and therefore $L^E = L^I = 0$. Then, equation 1.7 can be integrated easily and yields

$$L(z_1) = L(z_0) \cdot e^{-\frac{a \cdot (z_1 - z_0)}{\cos \theta}} \tag{1.8}$$

The boundary value $L(z_0)$ is presumed known. This result is known as Beer's law (or Lambert's law, Bouguer's law, Beer-Lambert law), denotes any instance of exponential attenuation of light and is exact only for purely absorbing media—i.e. media that do not scatter radiation. It is of direct application in analytical chemistry, as it describes the direct proportionality of absorbance (A) to the concentration of a coloured solute in a transparent solvent.

Different physical quantities are used to describe the "amount of radiation" and their definitions and abbreviations are listed in Table 1.2. Taking into account Equa-tion 1.6 and assuming a homogenous flux, the important correlation between irradi-ance E and intensity I is

$$E = \frac{I \cdot \cos \alpha}{r^2} \tag{1.9}$$

The irradiance decreases by the square of the distance to the source and depends on the tilt of the detecting surface area. This is valid only for point sources. For outdoor measurements the sun can be assumed to be a point source. For artificial light sources simple LEDs (light-emitting diodes) without optics on top are also effectively point sources. However, LEDs with optics—and other artificial light sources with optics or reflectors designed to give a more focused dispersal of the light—deviate to various extents from the rule of a decrease of irradiance proportional to the square of the distance from the light source.

Besides the physical quantities used for all electromagnetic radiation, there are also equivalent quantities to describe visible radiation, so called photometric quantities. The human eye as a detector led to these photometric units, and they are commonly used by lamp manufacturers to describe their artificial light sources. See the next section for a short description of these quantities and units.

---

[3]Their program is available from the website of Oregon Medical Laser Center at http://omlc.ogi.edu/software/mc/

Table 1.2: Physical quantities of light.

Symbol	Unit	Description
$\Phi = \frac{\partial q}{\partial t}$	$\text{W} = \text{J s}^{-1}$	Radiant flux: absorbed or emitted energy per time interval
$H = \frac{\partial q}{\partial A}$	J m^{-2}	Exposure: energy towards a surface area. (In plant research this is called usually *dose* (H), while in Physics *dose* refers to absorbed radiation.)
$E = \frac{\partial \Phi}{\partial A}$	W m^{-2}	Irradiance: flux or radiation towards a surface area, radiant flux density
$I = \frac{\partial \Phi}{\partial \Omega}$	W sr^{-1}	Radiant intensity: emitted radiant flux of a surface area per solid angle
$\epsilon = \frac{\partial \Phi}{\partial A}$	W m^{-2}	Emittance: emitted radiant flux per surface area
$L = \frac{\partial^2 \Phi}{\partial \Omega (\partial A \cdot \cos \alpha)} = \frac{\partial I}{\partial A \cdot \cos \alpha}$	$\text{W m}^{-2} \text{ sr}^{-1}$	Radiance: emitted radiant flux per solid angle and surface area depending on the angle between radiant flux and surface perpendicular

Photometric quantities

In contrast to (spectro-)radiometry, where the energy of any electromagnetic radiation is measured in terms of absolute power ($\text{J s} = W$), photometry measures light as perceived by the human eye. Therefore, radiation is weighted by a luminosity function or visual sensitivity function describing the wavelength dependent response of the human eye. Due to the physiology of the eye, having rods and cones as light receptors, different sensitivity functions exist for the day (photopic vision) and night (scotopic vision), $V(\lambda)$ and $V'(\lambda)$, respectively. The maximum response during the day is at $\lambda = 555$ nm and during night at $\lambda = 507$ nm. Both response functions (normalised to their maximum) are shown in Figure 1.3 as established by the Commission Internationale

de l'Ã‰clairage (CIE, International Commission on Illumination, Vienna, Austria) in 1924 for photopic vision and 1951 for scotopic vision (Schwiegerling 2004). The data are available from the Colour and Vision Research Laboratory at `http://www.cvrl.org`. Until now, $V(\lambda)$ is the basis of all photometric measurements.
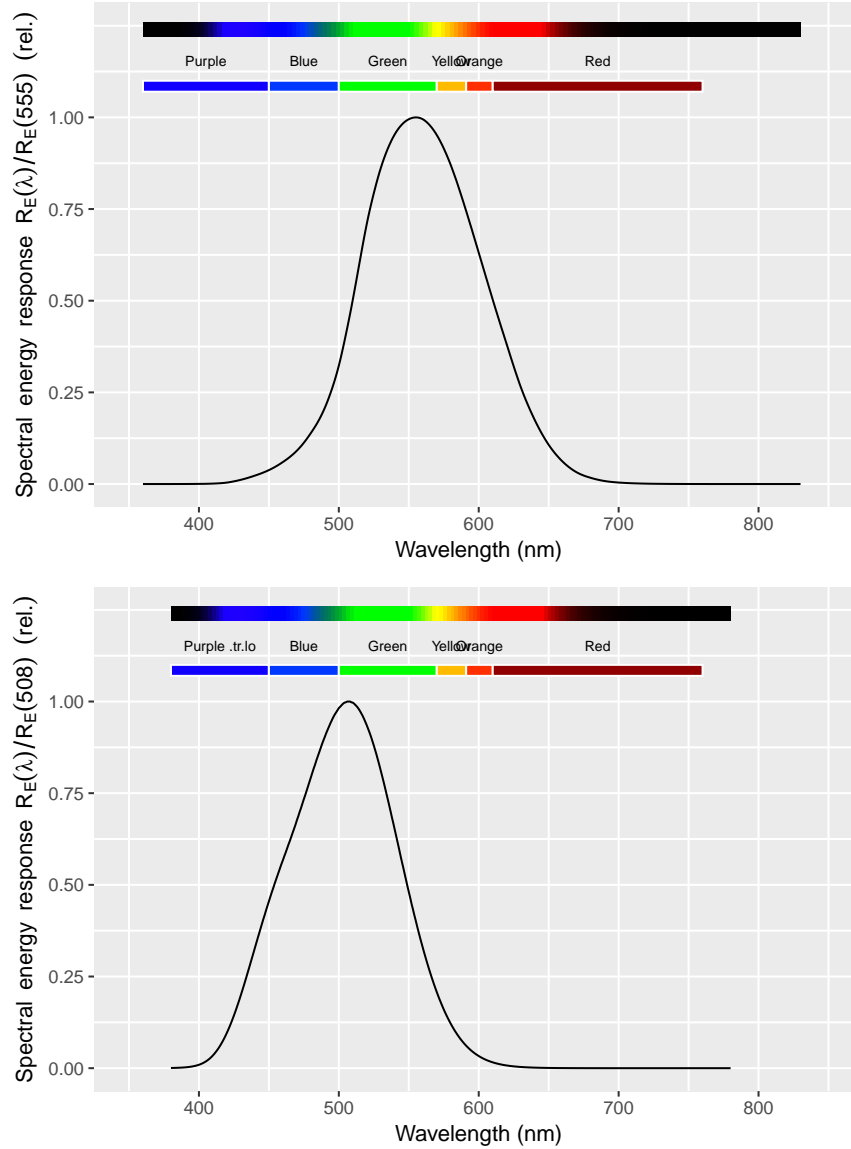
Figure 1.3: Relative spectral intensity of human colour sensation during day (solid line) and night (dashed line), $V(\lambda)$ and $V'(\lambda)$ respectively.

Corresponding to the physical quantities of radiation summarized in the table 1.2,

the equivalent photometric quantities are listed in Table 1.3 and have the subscript v. The ratio between the (physiological) luminous flux Φ_v and the (physical) radiant flux Φ is the (photopic) photometric equivalent $K(\lambda) = V(\lambda) \cdot K_m$ with $K_m = 683$ lm W^{-1} (lumen per watt) at 555 nm. The dark-adapted sensitivity of the eye (scotopic vision) has its maximum at 507 nm with 1700 lm W^{-1}. The base unit of luminous intensity is candela (cd). One candela is defined as the monochromatic intensity at 555 nm (540 THz) with $I = \frac{1}{683}$ W sr^{-1}. The luminous flux of a normal candle is around 12 lm. Assuming a homogeneous emission into all directions, the luminous intensity is about $I_v = \frac{12\,\text{lm}}{4\pi\,\text{sr}} \approx 1$ cd.

Table 1.3: Photometric quantities of light.

Symbol	Unit	Description
q_v	lm s	Luminous energy or quantity of light
$\Phi_v = \frac{\partial q_v}{\partial t}$	lm	Luminous flux: absorbed or emitted luminous energy per time interval
$I_v = \frac{\partial \Phi_v}{\partial \Omega}$	cd = lm sr^{-1}	Luminous intensity: emitted luminous flux of a surface area per solid angle
$E_v = \frac{\partial \Phi_v}{\partial A}$	lux = lm m^{-2}	Illuminance: luminous flux towards a surface area
$\epsilon_v = \frac{\partial \Phi_v}{\partial A}$	lux	Luminous emittance: luminous flux per surface area
$H_v = \frac{\partial q_v}{\partial A}$	lux s	Light exposure: quantity of light towards a surface area
$L_v = \frac{\partial^2 \Phi_v}{\partial \Omega(\partial A \cdot \cos \alpha)} = \frac{\partial I_v}{\partial A \cdot \cos \alpha}$	cd m^{-2}	Luminance: luminous flux per solid angle and surface area depending on the angle between luminous flux and surface perpendicular

Photon or quantum quantities of radiation.

When we are interested in photochemical reactions, the most relevant radiation quantities are those expressed in photons. The reason for this is that, as discussed in Chapter 3 on page 33, molecules are excited by the absorption of certain fixed amounts

of energy or quanta. The surplus energy "decays" by non-photochemical processes. When studying photosynthesis, where many photons of different wavelengths are simultaneously important, we normally use photon irradiance to describe amount of PAR. The name photosynthetic photon flux density, or PPFD, is also frequently used when referring to PAR photon irradiance. When dealing with energy balance of an object instead of photochemistry, we use (energy) irradiance. In meteorology both UV and visible radiation, are quantified using energy-based quantities. When dealing with UV photochemistry as in responses mediated by UVR8, an UV-B photoreceptor, the use of quantum quantities is preferred. According to the physical energetic quantities in the table 1.2, the equivalent photon related quantities are listed in the table below and have the subscript p.

Table 1.4: Photon quantities of light.

Symbol	Unit	Description
Φ_p	s^{-1}	Photon flux: number of photons per time interval
$Q = \frac{\partial \Phi_p}{\partial A}$	$m^{-2}\ s^{-1}$	Photon irradiance: photon flux towards a surface area, photon flux density (sometimes also symbolised by E_p)
$H_p = \int_t Q\ dt$	m^{-2}	Photon exposure: number of photons towards a surface area during a time interval, photon fluence

These quantities can be also used based on a 'chemical' amount of moles by dividing the quantities by Avogadro's number $N_A = 6.022 \times 10^{23}$ mol^{-1}. To determine a quantity in terms of photons, an energetic quantity has to be weighted by the number of photons, i.e. divided by the energy of a single photon at each wavelength as defined in equation 1.1. This yields for example

$$\Phi_p = \frac{\lambda}{h\,c} \cdot \frac{\partial q}{\partial t} \qquad \text{and} \qquad Q(\lambda) = \frac{\lambda}{h\,c} \cdot E(\lambda)$$

Conversion between energy and photon quantities of radiation

When dealing with bands of wavelengths, for example an integrated value like PAR from 400 to 700 nm, it is necessary to repeat these calculations at each wavelength and then integrate over the wavelengths. For example, the PAR photon irradiance or

PPFD in moles of photons is obtained by

$$\text{PPFD} = \frac{1}{N_A} \int_{400\,\text{nm}}^{700\,\text{nm}} \frac{\lambda}{hc}\, E(\lambda)\, \text{d}\lambda$$

For integrated values of UV-B or UV-A radiation the calculation is done analogously by integrating from 280 to 315 nm or 315 to 400 nm, respectively.

If we have measured (energy) irradiance, and want to convert this value to photon irradiance, the exact conversion will be possible only if we have information about the spectral composition of the measured radiation. Conversion factors at different wavelengths are given in the table below. For PAR, 1 $W\,m^{-2}$ of "average daylight" is approximately 4.6 $\mu mol\,m^{-2}\,s^{-1}$. This is exact only if the radiation is equal from 400 to 700 nm, because the factor is the value at the central wavelength at 550 nm. Further details are discussed in section 7.7.3 on page 76. Examples of conversion constants between energy-based and photo-based irradiance at different wavelengths are given in Table 1.5.

Table 1.5: Conversion factors of photon and energy quantities at different wavelengths.

	$W\,m^{-2}$ to $\mu mol\,m^{-2}\,s^{-1}$	λ (nm)
	2.34	280
UV-B	2.49	298
	2.63	315
UV-A	2.99	358
	3.34	400
PAR	4.60	550
	5.85	700

There are, in principle, two possible approaches to measuring radiation. The first is to observe light from one specific direction or viewing angle, which is the radiance L. The second is to use a detector, which senses radiation from more than one direction and measures the so-called irradiance E of the entire sphere or hemisphere. The correlation between irradiance E and radiance L of the wavelength λ is given by integrating over all directions of incoming photons.

$$E_0(\lambda) = \int_\Omega L(\lambda, \Omega)\text{d}\Omega \tag{1.10}$$

$$E(\lambda) = \int_\Omega L(\lambda, \Omega)|\cos\alpha|\text{d}\Omega \tag{1.11}$$

Depending on the shape of a detector (which may be either planar or spherical) the irradiance is called (plane) irradiance E or fluence rate (also called scalar irradiance) E_0. A planar sensor detects incoming photons depending on the incident angle and

a spherical sensor detects all photons equally weighted for all directions. See section Aphalo, Albert, Björn, McLeod, et al. (2012) for a more detailed discussion.

Here we have discussed the properties of light based on energy quantities. In photobiology and photobiology there are good reasons to quantify radiation based on photons. See Chapter 3 on page 33.

1.3 Solar radiation

When dealing with solar radiation, we frequently need to describe the position of the sun. The azimuth angle (ϕ) is measured clockwise from the North on a horizontal plane. The position on the vertical plane is measured either as the zenith angle (θ) downwards from the zenith, or as an elevation angle (h) upwards from the horizon. Consequently $h + \theta = 90° = \frac{\pi}{2}$ radians. See Figure 1.4 for a diagram. In contrast to Figure 1.1 and the discussion in section 1.2 where the point radiation source is located at the origin of the system of coordinates, when describing the position of the sun as in Figure 1.4 the observer is situated at the origin.
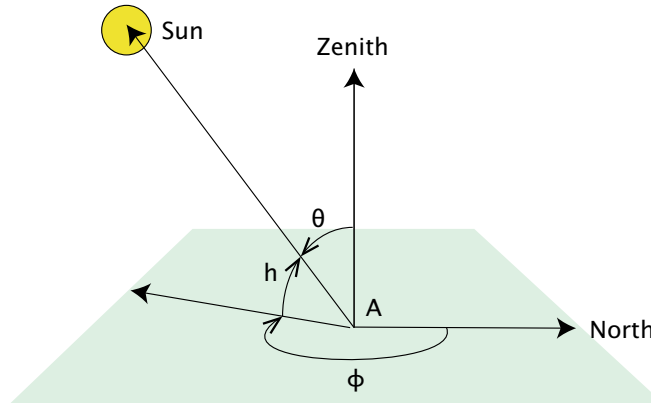
Figure 1.4: Position of the sun in the sky and the different angles used for its description by an observer located at point A. The azimuth angle is ϕ, the elevation angle is h and the zenith angle is θ. These angles are measured on two perpendicular planes, one horizontal and one vertical.

Ultraviolet and visible radiation are part of solar radiation, which reaches the Earth's surface in about eight minutes (t = time, r_0 = distance sun to earth, c = velocity of light in vacuum):

$$t = \frac{r_0}{c} \approx \frac{150 \times 10^9 \text{ m}}{3 \times 10^8 \frac{\text{m}}{\text{s}}} = 500 \text{ s} = 8.3 \text{ min}$$

The basis of all passive measurements is the incoming solar radiation, which can be estimated from the known activity of the sun ('productivity of photons'), that can be

approximated by the emitted spectral radiance (L_s) described by Planck's law of black body radiation at temperature T, measured in degrees Kelvin (K):

$$L_s(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{e^{(hc/k_B T\lambda)} - 1} \tag{1.12}$$

with Boltzmann's constant $k_B = 1.381 \times 10^{-23}$ JK^{-1}. The brightness temperature of the sun can be determined by Wien's displacement law, which gives the peak wavelength of the radiation emitted by a blackbody as a function of its absolute temperature

$$\lambda_{max} \cdot T = 2.898 \times 10^6 \, \text{nm K} \tag{1.13}$$

This means that for a maximum emission of the sun at about 500 nm the temperature of the sun surface is about 5800 K. The spectral irradiance of the sun $E_s(\lambda)$ can be estimated assuming a homogeneous flux and using the correlation of intensity I and radiance L from their definitions in table 1.2. The intensity of the sun $I_s(\lambda)$ is given by the radiance $L_s(\lambda)$ multiplied by the apparent sun surface (a non-tilted disk of radius $r_s = 7 \times 10^5$ km). To calculate the decreased solar irradiance at the moment of reaching the Earth's atmosphere, the distance of the sun to the Earth ($r_0 = 150 \times 10^6$ km) has to be taken into account due to the inverse square law of irradiance of equation (1.9). Thus, the extraterrestrial solar irradiance is

$$E_s(\lambda) = L_s(\lambda) \cdot \frac{\pi r_s^2}{r_0^2} \tag{1.14}$$

Remembering the solid angle of equation (1.6), the right multiplication factor represents the solid angle of the sun's disk as seen from the Earth's surface ($\approx 6.8 \times 10^{-5}$ sr). Figure 1.5 shows the spectrum of the measured extraterrestrial solar radiation (Wehrli, 1985)[4] and the spectrum calculated by equation 1.14 using Planck's law of equation 1.12 at a black body temperature of 5800 K. Integrated over all wavelengths, E_s is about 1361 to 1362 W m^{-2} at top of the atmosphere (Kopp and Lean 2011). This value is called the 'solar constant'. In former times, depending on different measurements, E_s varies by a few percent (Iqbal 1983). For example, the irradiance at the top of the atmosphere (the integrated value) changes by ± 50 W m^{-2} (3.7 %) during the year due to distance variation caused by orbit excentricity (Mobley 1994). More accurate measurements during the last 25 years by spaceborne radiometers show a variability of the solar radiation of a few tenth of a percent. A detailed analysis is given by (Fröhlich and Lean 2004). E_s can also be calculated by the Stefan-Boltzmann Law: the total energy emitted from the surface of a black body is proportional to the fourth power of its temperature. For an isotropically emitting source (Lambertian emitter), this means

$$L = \frac{\sigma}{\pi} \cdot T^4 \tag{1.15}$$

---

[4]Available as ASCII file at PMODWRC, ftp://ftp.pmodwrc.ch/pub/publications/pmod615.asc

with the Stefan-Boltzmann constant $\sigma = 5.6705 \times 10^{-8}$ W m^{-2} K^{-4}. With $T = 5800$ K equation 1.15 gives the radiance of the solar disc. From this value, we can obtain an approximation of the solar constant, by taking into account the distance from the Earth to the Sun and the apparent size of the solar disc (see equations 1.6 and 1.9).
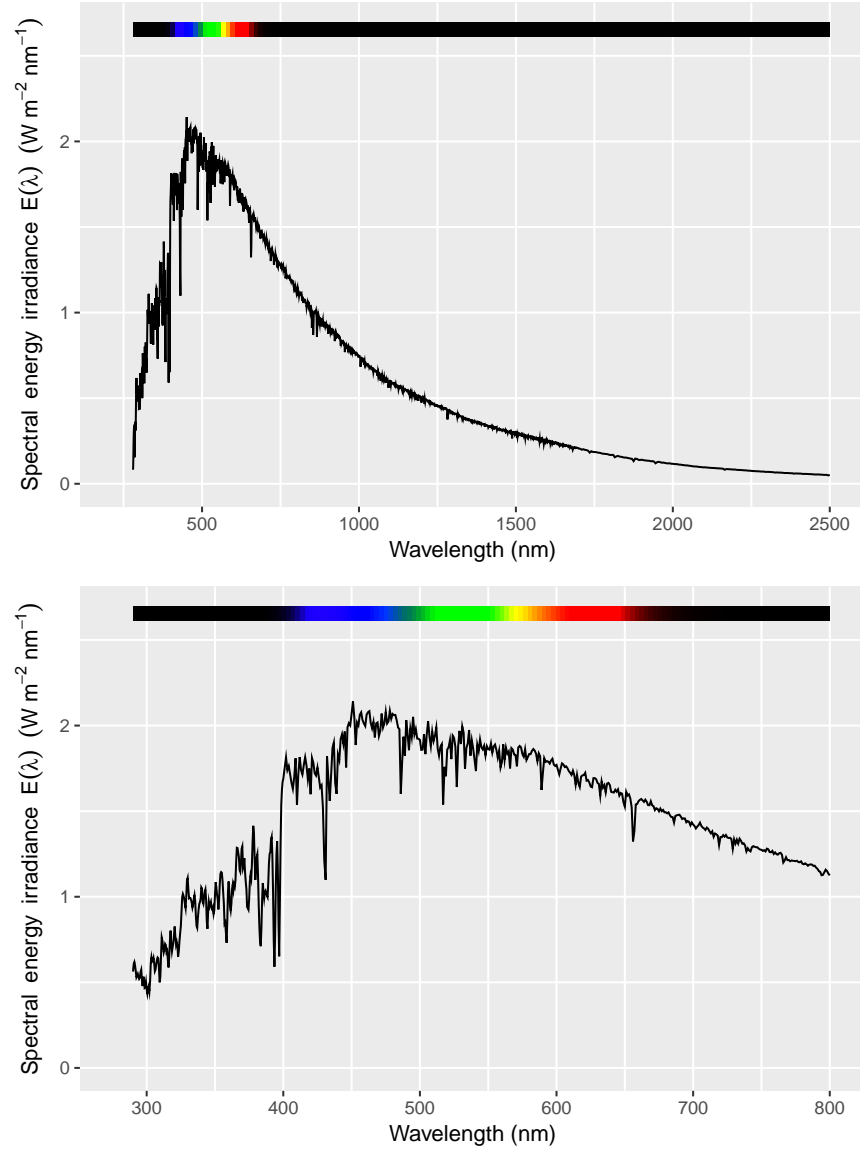
Figure 1.5: Extraterrestrial solar spectrum after (Gueymard 2004). Contrast with Fig. 1.6.

The total solar irradiance covers a wide range of wavelengths. Using some of the 'colours' introduced in table 1.1, table 1.6 lists the irradiance and fraction of E_s of

different wavelength intervals.

Table 1.6: Partition of the extraterrestrial solar irradiance E_s constant in different wavelength intervals calculated using the data of (Wehrli 1985) shown in Figure 1.5.

Colour	Wavelength (nm)	Irradiance ($\mathrm{W\,m^{-2}}$)	Fraction of E_s (%)
UV-C	100 – 280	7	0.5
UV-B	280 – 315	17	1.2
UV-A	315 – 400	84	6.1
VIS	400 – 700	531	38.9
near IR	700 – 1 000	309	22.6
mid and far IR	> 1 000	419	30.7
total		1 367	100.0

The extraterrestrial solar spectrum differs from that at ground level due to the absorption of radiation by the atmosphere, because the absorption peaks of water, CO_2 and other components of the atmosphere, cause corresponding valleys to appear in the solar spectrum at ground level. For example, estimates from measurements of the total global irradiance at Helmholtz Zentrum München (11.60° E, 48.22° N, 490 m above sea level) on two sunny days (17th April 1996, sun zenith angle of 38° and 27th May 2005, 27°) result in about 5% for wavelengths below 400 nm, about 45% from 400 to 700 nm, and about 50% above 700 nm. In relation to plant research, only the coarse structure of peaks and valleys is relevant, because absorption spectra of pigments *in vivo* have broad peaks and valleys. However, the solar spectrum has a much finer structure, due to emission and absorption lines of elements, which is not observable with the spectroradiometers normally used in plant research.

At the Earth's surface, the incident radiation or *global radiation* has two components, direct radiation and scattered or 'diffuse' radiation. Direct radiation is radiation travelling directly from the sun, while diffuse radiation is that scattered by the atmosphere. Diffuse radiation is what gives the blue colour to the sky and white colour to clouds. The relative contribution of direct and diffuse radiation to global radiation varies with wavelength and weather conditions (Fig. 1.7). The contribution of diffuse radiation is larger in the UV region, and in the presence of clouds as can be visualized in photographs obtained at different wavebands (Lindfors and Ylianttila 2015).

Not only total irradiance, but also the wavelength distribution of the solar spectrum changes with the seasons of the year and time of day. The spectral wavelength distribution is also changed by the amount of UV-absorbing ozone in the atmosphere, known as the ozone column. Figure 1.8 shows how spectral irradiance changes throughout one day. When the whole spectrum is plotted using a linear scale the effect of
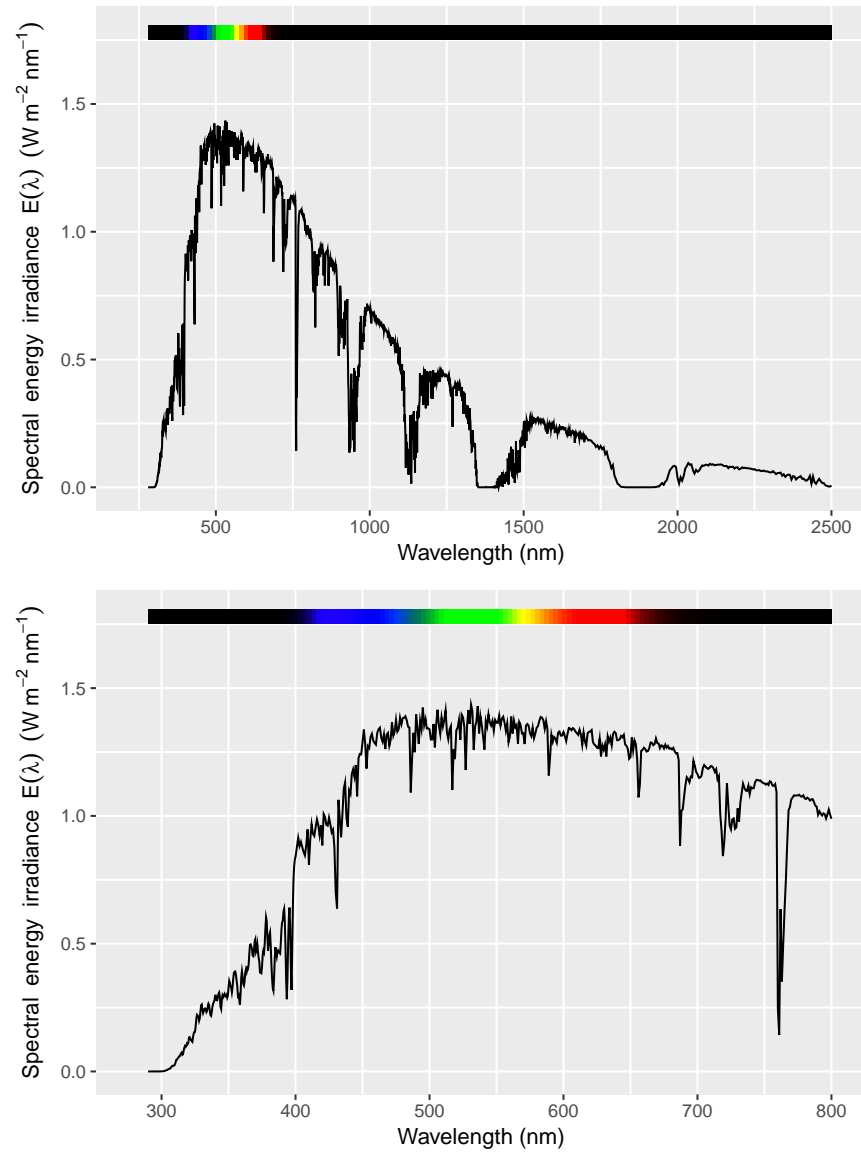
Figure 1.6: Ground level solar spectrum AM1.5 (1.5 air mass, solar zenith angle 48° 19′) according to ASTM G173-03 (ASTM 2012). Contrast with Fig. 1.5.
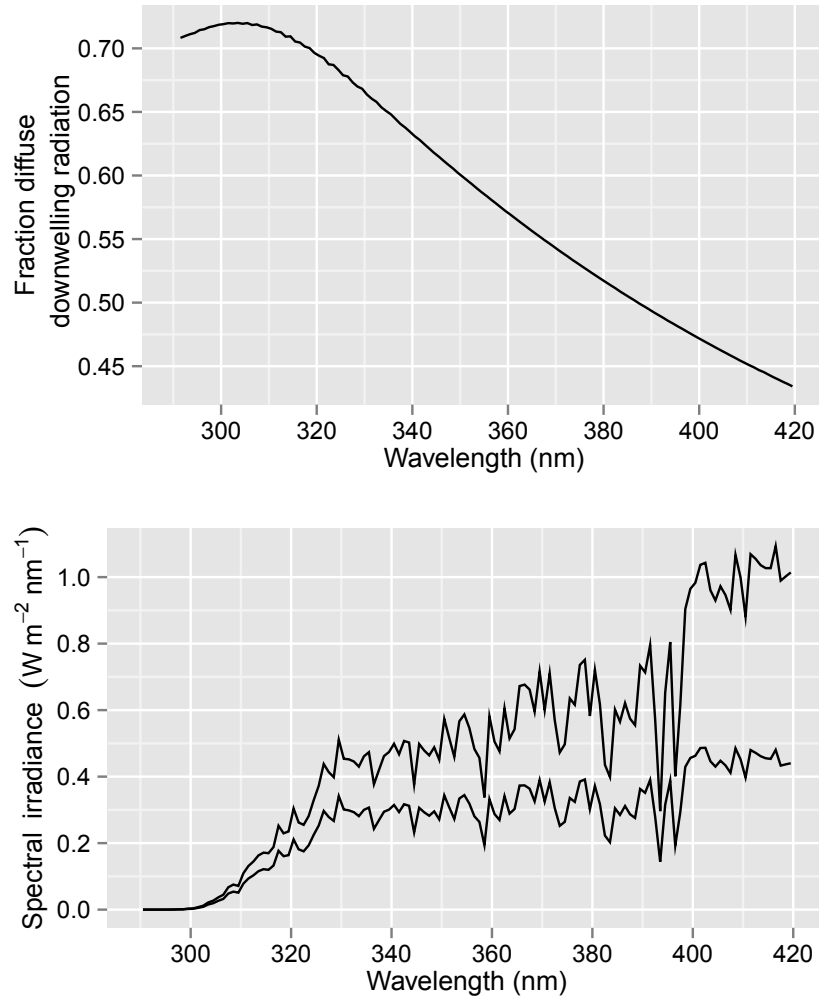
Figure 1.7: Diffuse component in solar UV. Spectral irradiance of total downwelling radiation (lower panel, solid line), diffuse downwelling radiation (lower panel, long dashes), and ratio of diffuse downwelling to total downwelling spectral irradiance (upper panel, dashed line) are shown. Data from TUV model (version 4.1) for solar zenith angle = $40°00'$, cloud-free conditions, 300 Dobson units. Simulations done with the Quick TUV calculator at http://cprm.acd.ucar.edu/Models/TUV/Interactive_TUV/.
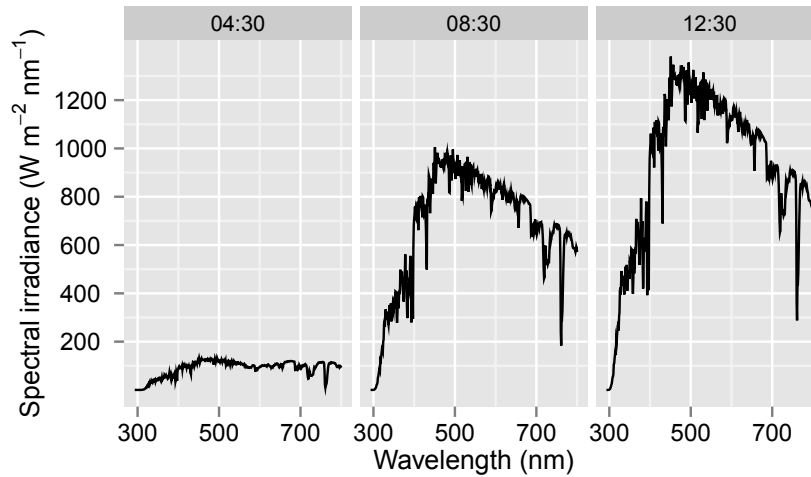
Figure 1.8: The solar spectrum through half a day. Simulations of global radiation (direct plus diffuse radiation) spectral irradiance on a horizontal surface at ground level) for a hypothetical 21 May with cloudless sky at Jokioinen (60°49'N, 23°30'E), under normal ozone column conditions. Effect of depletion is so small on the solar spectrum as a whole, that it would not visible in this figure. See (Kotilainen et al. 2011) for details about the simulations.

ozone depletion is not visible, however, if we plot only the UV region (Figure 1.9) or use a logarithmic scale (Figure 1.10), the effect becomes clearly visible. In addition, on a log scale, it is clear that the relative effect of ozone depletion on the spectral irradiance at a given wavelength increases with decreasing wavelength.

Seasonal variation in UV-B irradiance has a larger relative amplitude than variation in PAR . This causes a seasonal variation in the UV-B: PAR ratio. In addition to the regular seasonal variation, there is random variation as a result of changes in clouds. Normal seasonal and spatial variation in UV can be sensed by plants, and could play a role in their adaptation and acclimation to seasons and/or their position in the canopy.

UV-B irradiance increases with elevation in mountains and with decreasing latitude (Figure 1.11) and is particularly high on high mountains in equatorial regions.

An increase in the UV-B irradiance is caused by depletion of the ozone layer in the stratosphere, mainly as a consequence of the release of chlorofluorocarbons (CFCs), used in cooling devices such as refrigerators and air conditioners, and in some spray cans (see Graedel and Crutzen 1993). The most dramatic manifestation of this has been the seasonal formation of an "ozone hole" over Antarctica. It is controversial whether a true ozone hole has already formed in the Arctic, but strong depletion has occurred in year 2011 (Manney et al. 2011) and atmospheric conditions needed for
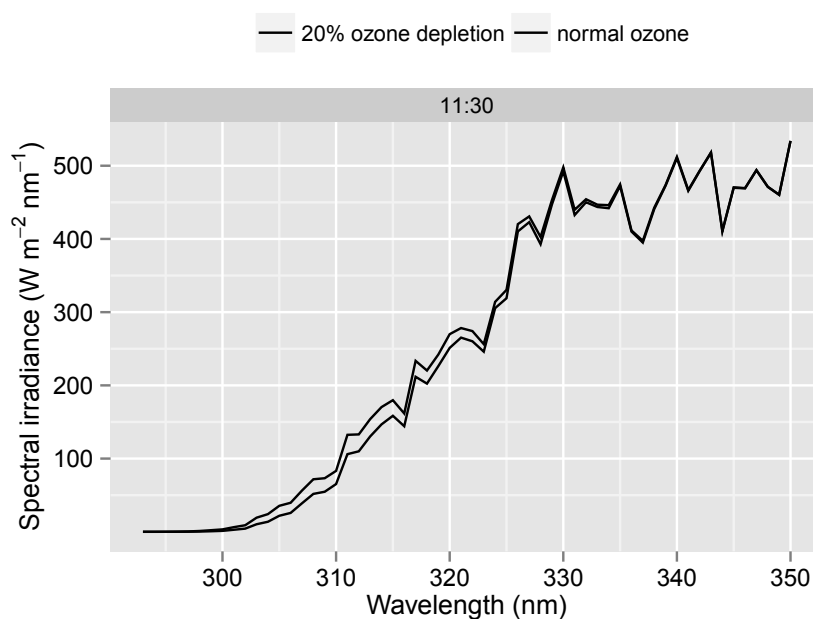
Figure 1.9: The effect of ozone depletion on the UV spectrum of global (direct plus diffuse) solar radiation at noon. See fig. 1.8 for details.
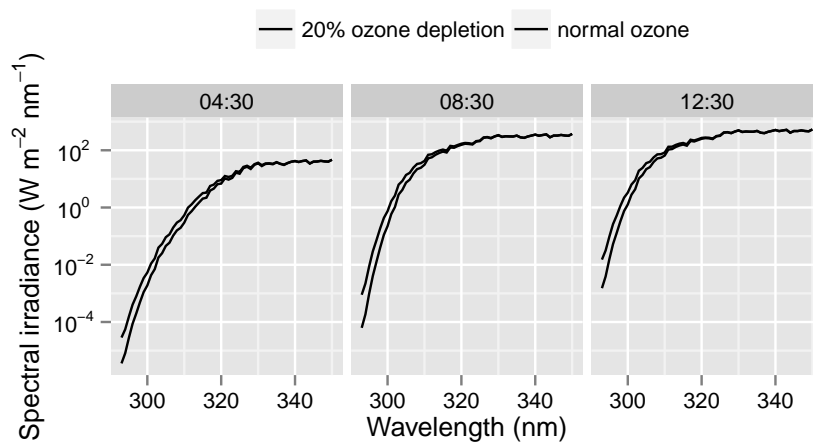
Figure 1.10: The solar UV spectrum through half a day. The effect of ozone depletion on global (direct plus diffuse) radiation. A logarithmic scale is used for spectral irradiance. See fig. 1.8 for details.
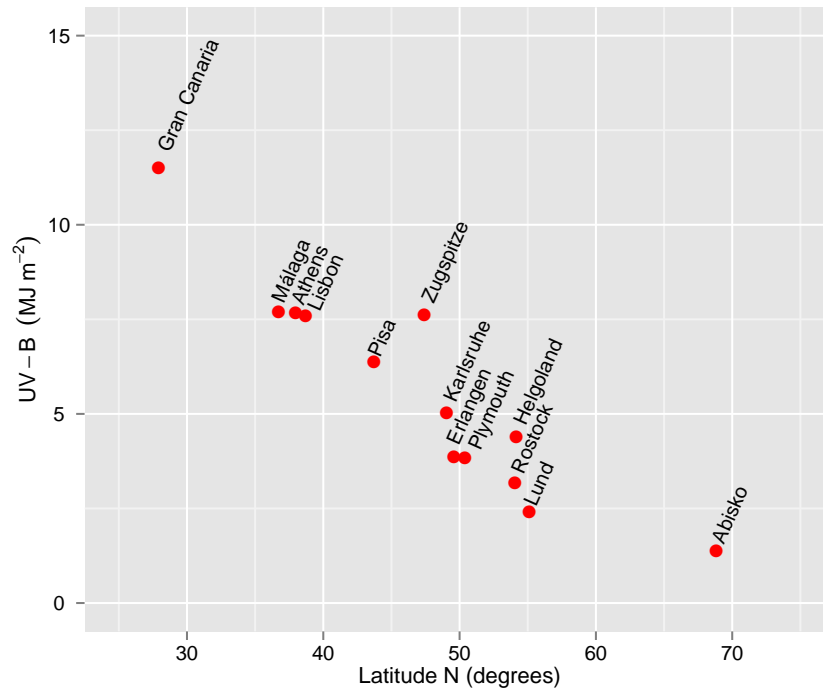
Figure 1.11: Latitudinal variation in UV-B radiation in the Northern hemisphere. UV-B annual exposure, measured with ELDONET instruments (see Häder et al. 2007, for details).

the formation of a "deep" ozone hole are not very different from those prevalent in recent years. Not so dramatic, but consistent, depletion has also been observed at mid-latitudes in both hemispheres. CFCs and some other halocarbons have been phased out following the Montreal agreement and later updates. However, as CFCs have a long half life in the atmosphere, of the order of 100 years, their effect on the ozone layer will persist for many years, even after their use has been drastically reduced. Model-based predictions of changes in atmospheric circulation due to global climate change have been used to derive future trends in UV index and ozone column thickness (Hegglin and Shepherd 2009). In addition, increased cloudiness and pollution, could lead to decreased UV and PAR, sometimes called 'global dimming' (e.g. Stanhill and Cohen 2001). It should be noted that, through reflection, broken clouds can locally increase UV irradiance to values above those under clear-sky conditions (Díaz et al. 1996; Frederick et al. 1993).
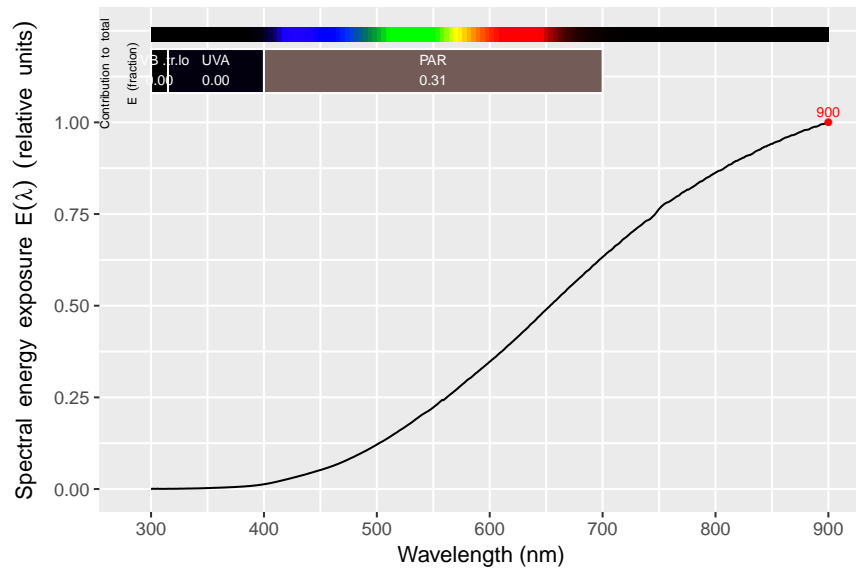
Figure 1.12: Spectral irradiance for a 60 W incandescent lamp

1.4 Artificial radiation

Different types of man-made VIS and UV radiation sources exist, based on exploiting different physical phenomena.

Incandescent light sources are "near-black bodies" heated at a very high temperature. Normal incandescent lamps are made from a Tungsten (also called Wolfram) wire heated at between 2500 and 3500 K by passing an electric current through it). The glass bulb enclosing it helps maintain the temperature and the low pressure inert gases filling it help slow down the evaporation of the metal (which can be seen in old lamps as a blackish deposit on the inside of glass bulb surface). These lamps produce a continuous spectrum (without well defined emission peaks), close to that from a true black body at the same temperature. Lamps with certain types of built-in reflectors may display a somewhat distorted spectrum as a result of interference or because of wavelength-selective properties (e.g. it is not unusual for lamps to have a reflector with high reflectivity for visible radiation but relatively high transmissivity for infra-red radiation).

Carbon arc lamps emit light by means of an electric "arc" between two carbon electrodes, the arc heats the electrodes and carbon evaporates and because of its high temperature in-between the electrodes inducing light emission. The spectrum is broad but rather different to sunlight. Carbon arc lamps can be very bright and used in cinema projectors. They were invented before the incandescent tungsten lamp.
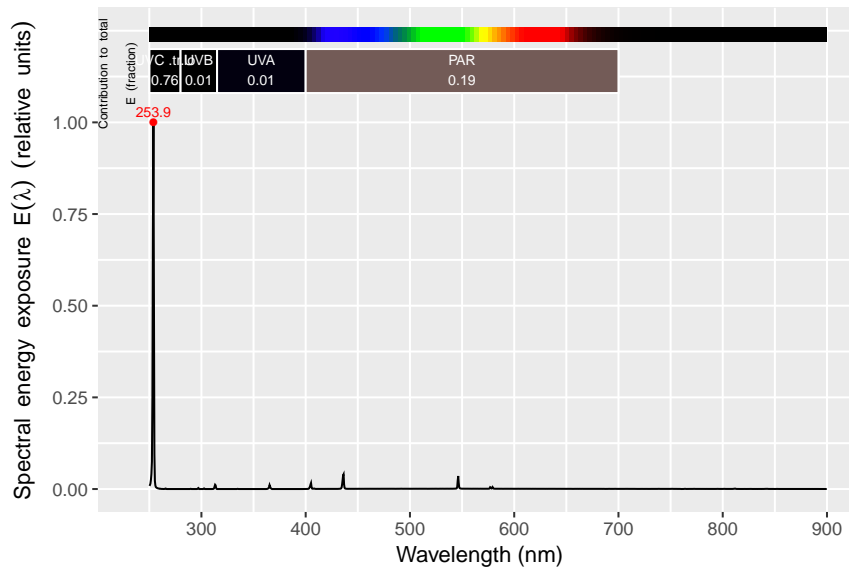
Figure 1.13: Spectral irradiance for a 'germicidal' low pressure mercury lamp.

Xenon arc lamps use Xenon gas enclosed in a special glass bulb. Xenon arc lamps have an emission spectrum rather similar to that of solar radiation, and together with UV-absorbing filters are frequently used in solar simulators. Some Xenon lamps do not emit continuously, such as modern "electronic" flashes used in photographic cameras. In such lamps the flash is produced by slowly charging a capacitor at a high voltage, and subsequently using this electrical charge to generate a short-lived arc in the lamp. Flash duration varies, but can be as short 0.1 ms in flashes used by photographers.

Other gas-discharge lamps use different gases or "vapours" or mixes of them. In these lamps, the elemental emission lines (corresponding to transitions between allowed energy states) are very well defined as long as the glass ampoule is not coated with special fluorescent compounds, and in many cases can be used as wavelength standards for calibration of spectrometers. The low pressure sodium lamps, easily recognizable by the orange light they emit, emit the same orange colour as that emitted by the flame of a gas ring in a cooker when water containing salt boils over from a pot. Low pressure mercury "vapour" lamps, such as germicidal ones made with an un-coated quartz-glass tube (technically called envelope) emit clearly at the know emission lines of mercury (Fig. 1.13). Being the container UV and VIS transmitting the strong line at 253.xx nm is very active as a germicidal agent. The "normal" fluorescent tubes used for illumination are enclosed in a tube coated with a so-called "phosphor" which absorbs UV radiation and re-emits it as visible radiation (Fig. 1.14). The spec-
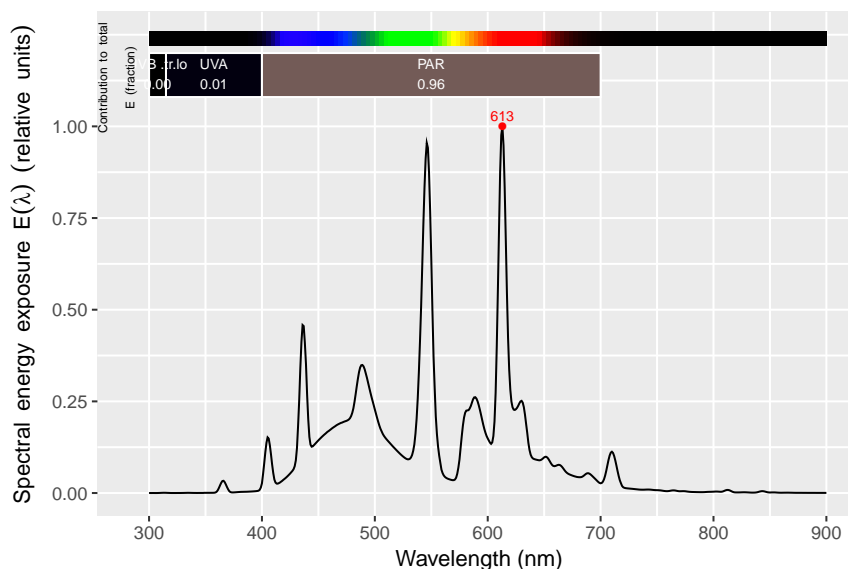
Figure 1.14: Spectral irradiance for a 'daylight', approx. 5200 K, fluorescent tube (Philips 36W 950).

trum of the emitted radiation is a combination of radiation emitted by the gaseous mercury, in particular those lines in the VIS region (e.g. 435.xx nm) and to some extent in the UVA region, together with visible radiation re-emitted by the coating.

A more recent development is light generated by solid-state semiconductor devices or light-emitting diodes, once again light emission is the result of a transition between energy states of matter, but although emission takes place as a single peak, the peak is not as narrow or well defined as for elemental emission lines in discharge lamps (Fig. 1.15). Emission peaks have usually HFW of between 10 and 30 nm and their central wavelength may slightly shift depending on temperature and electrical current flowing through them. True LEDs always have a single peak of emission. White LEDs are based on a similar principle to that of fluorescent lamps: blue (or in some cases UVA) emitting LEDs are combined with a "phosphor" which absorbs in part the emitted radiation and re-emits the energy as radiation at longer wavelengths (Figs. 1.16 and 1.17). In some designs the phosphor is coated close to the semiconductor die, but in other cases, especially some arrays, the phosphor is in or on the encapsulating polymer. This coating in white LEDs looks yellow or orange when they are switched off.

Finally lasers, are different in that a laser is not another type of primary source of radiation. Lasing is a phenomenon which allows the generation of coherent radiation from a beam of incoherent radiation by means of a "cavity". Different primary sources
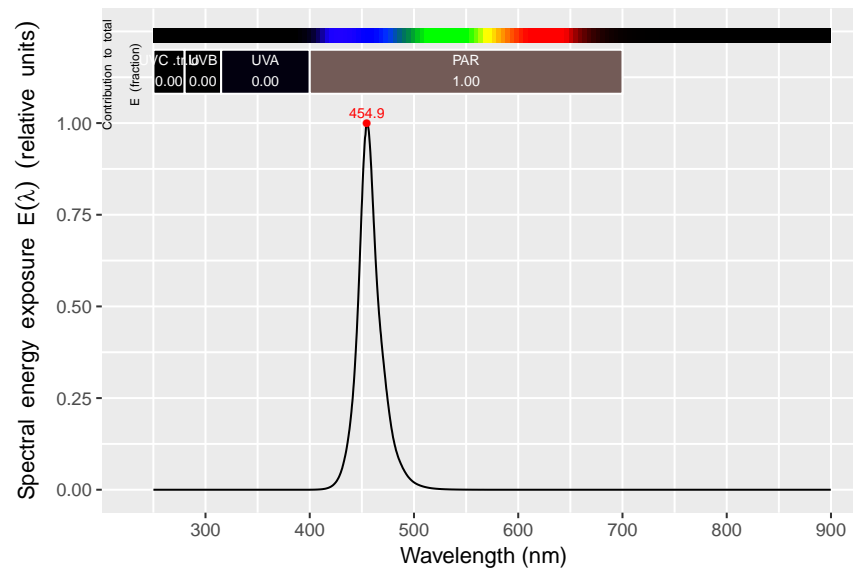
Figure 1.15: Spectral irradiance for a blue LED array (Huey Jann, 50 W).
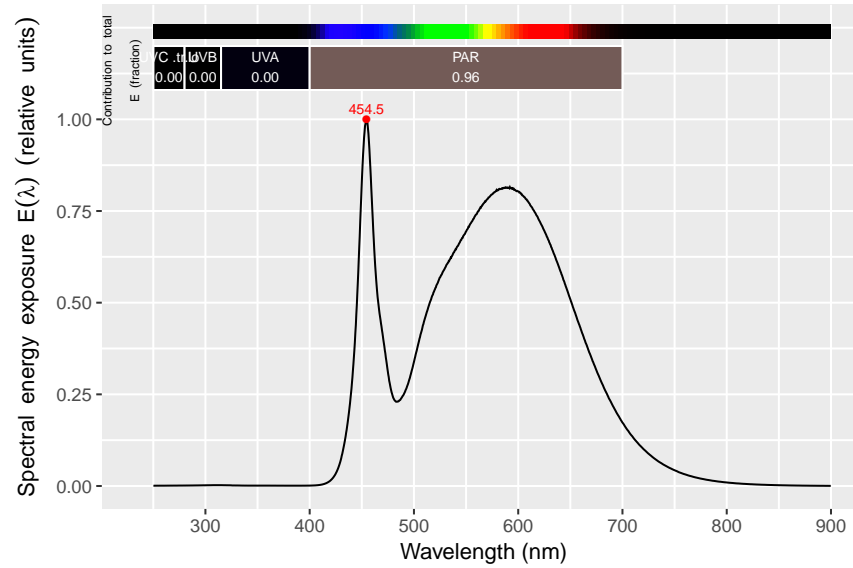
Figure 1.16: Spectral irradiance for 'neutral white' LED, 4000 K, array (Lumitronix SmartArray Q36 LED-Module, 39W, using Nichia 757 LEDs).
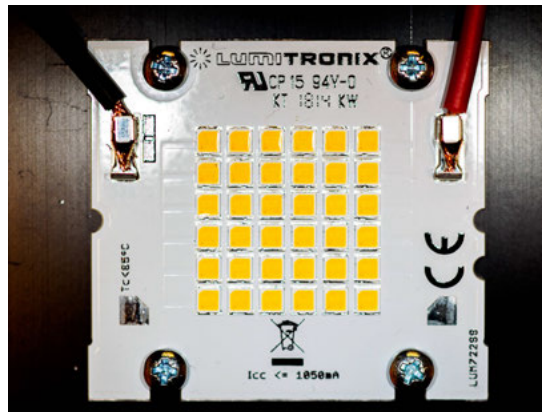
Figure 1.17: Photgraph of 'neutral white LED' array showing the yellow "phosphor" in the coating of the LEDs (Lumitronix SmartArray Q36 LED-Module, 39W, using Nichia 757 LEDs).

of radiation can be used in lasers. In the case of laser diodes, an LED is the primary source of radiation. There are different possible types of cavities, for example gas-filled or solid state. Lasers are pulsed light sources, they do not emit continuously, although in many cases the frequency at which pulses are produced can be high. Even though laser pointers and other readily available lasers seem to us as being a continuous source of radiation, they are not. In fact, they are pulsed, and the duty cycle is low (pulses are brief compared to 'gaps') but each pulse has high energy. As radiation is in addition in a very narrow beam and almost of a single wavelength, a laser delivers spatially and temporarily concentrated energy pulses, that can make the beam from a 1 mW laser pointer easily visible at a distance of tens of meters in a lecture hall illuminated with lamps emitting in total hundreds of watts of visible radiation. For the same reason, even lasers emitting as little 1 mW if pointed directly into the eyes can cause permanent eye-sight damage.

```
try(detach(package:photobiologyLEDs))
try(detach(package:photobiologyLamps))
try(detach(package:photobiologySun))
try(detach(package:photobiologyWavebands))
try(detach(package:ggspectra))
try(detach(package:ggplot2))
try(detach(package:photobiology))
```

2

Radiation interactions

2.1 Radiation and molecules

2.1.1 Absorption

2.1.2 Fluorescence

2.1.3 Phosphorescence

2.2 Radiation and simple objects

2.2.1 Angle of incidence

2.2.2 Refraction

2.2.3 Difraction

2.2.4 Scatering

2.3 Radiation in tissues and cells

2.4 Radiation interactions in plant canopies

The attenuation of visible and UV radiation by canopies is difficult to describe mathematically because it is a complex phenomenon. The spatial distribution of leaves is in most cases not uniform, the display angle of the leaves is not random, and may change with depth in the canopy, and even in some cases with time-of-day. Here we give only a description of the simplest approach, the use of an approximation based on Beer's law as modified by Monsi and Saeki (1953), reviewed by Hirose (2005). Beer's law (Equation 1.8) assumes a homogeneous light absorbing medium such as a solution. However, a canopy is heterogenous, with discrete light absorbing objects (the leaves and stems) distributed in a transparent medium (air).

$$I_z = I_0 \cdot \mathrm{e}^{-K L_z} \tag{2.1}$$

Equation 2.1 describes the radiation attenuated as a function of leaf area index (L or LAI) at a given canopy depth (z). The equation does not explicitly account for the effects of the statistical spatial distribution of leaves and the effects of changing incidence angle of the radiation. Consequently, the empirical extinction coefficient (K) obtained may vary depending on these factors. K is not only a function of plant species (through leaf optical properties, and how leaves are displayed), but also of time-of-day, and season-of-year—as a consequence of solar zenith angle—and degree of scattering of the incident radiation. As the degree of scattering depends on clouds, and also on wavelength, the extinction coefficient is different for UV and visible radiation. Radiation extinction in canopies has yet to be studied in detail with respect to UV radiation, mainly because of difficulties in the measurement of UV radiation compared to PAR, a spectral region which has been extensively studied.

Ultraviolet radiation is strongly absorbed by plant surfaces, although cuticular waxes and pubescence on leaves can sometimes increase UV reflectance. The diffuse component of UV radiation is proportionally larger than that of visible light (Figure 1.7). In sunlit patches in forest gaps the diffuse radiation percentage is lower than in open areas, because direct radiation is not attenuated but part of the sky is occluded by the surrounding forest. Attenuation with canopy depth is on average usually more gradual for UV than for PAR. The UV irradiance decreases with depth in tree canopies, but the UV:PAR ratio tends to increase (see Brown et al. 1994). In contrast, Deckmyn et al. (2001) observed a decrease in UV:PAR ratio in white clover canopies with planophyle leaves. Allen et al. (1975) modelled the UV-B penetration in plant canopies, under normal and depleted ozone conditions. Parisi and Wong (1996) measured UV-B doses within model plant canopies using dosimeters. The position of leaves affects UV-B exposure, and it has been observed that heliotropism can moderate exposure and could be a factor contributing to differences in tolerance among crop cultivars (Grant 1998, 1999a,b, 2004).

Detailed accounts of different models describing the interaction of radiation and plant canopies, taking into account the properties of foliage, are given by Campbell and Norman (1998) and Monteith and Unsworth (2008).

2.5 Radiation interactions in water bodies

Missing for now...

2.6 Physical quantities

2.6.1 Specular and total reflectance

2.6.2 Internal and total transmittance

2.6.3 Absorbance and absorptance

3

Photochemistry and photobiology

4

Algorithms

4.1 Integration

⚠️ **What is wrong with adding-up spectral quantities to obtain a total?** With most spectrometers one can optionally export the spectral data re-expressed with wavelengths at 1 nm intervals. With any other step-size for wavelength, simplistically summing the spectral values will produce erroneous results. When re-expressed to 1 nm-step, the differences would frequently be only minor. However, interpolation and/or moving averages can distort narrow peaks and result in lost information. Although in many cases it leads to small differences between the values obtained by integration of the original data and the integration data re-expressed at different wavelengths, if the data were measured at a much finer wavelength resolution than the one it is re-expressed at, the loss of information can be important.

Functions in the photobiology package take the mean between each pair of spectral irradiance values at contiguous discrete wavelength values, and multiplies them by the distance in nm between these two wavelengths. This needs to be done individually for each of the pixels. It can be programmed in e.g. Excel, but summing up the values and multiplying by the mean interval will not give the correct integral for instruments with array detectors. The values obtained will be biased. This is because the intervals do not vary randomly, they increase in width as a function of the wavelength. This results in giving more weight to readings at one end of the spectrum than at the other. So, the error will depend on the shape of the emission spectrum of the light source.

4.1.1 Area under a spectral curve

Summary quantities over a range of wavelengths, are geometrically represented by the area under the spectral curve for the region delimited by the two extreme wavelengths
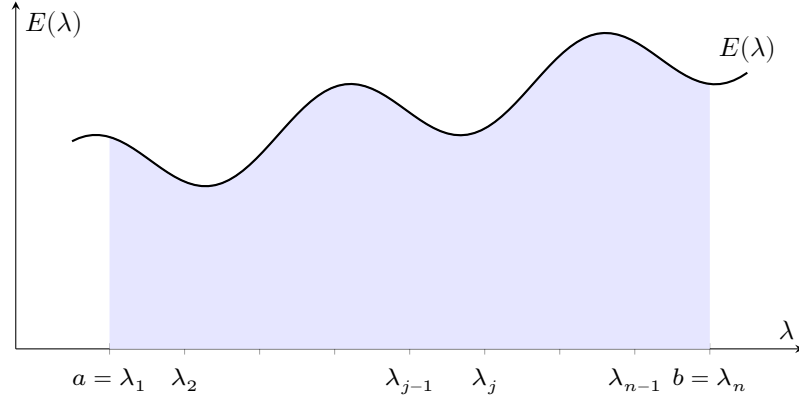
Figure 4.1: The actual integral without approximation.

in the range of interest (λ_1 and λ_n in Figure 4.1. Mathematically the operation is the computation of an integral over wavelengths.

Only if the wavelength step, $\Delta\lambda = 1$ nm, and uniform across the whole spectrum,

$$\int_{400}^{700} E(\lambda)\mathrm{d}\lambda \approx \sum_{i=400}^{699} E_i \tag{4.1}$$

if $\Delta\lambda \neq 1$ nm, but uniform, then this needs to be taken into account,

$$\int_{400}^{700} E(\lambda)\mathrm{d}\lambda \approx \sum_{i=1}^{n-1} E_i \cdot \Delta\lambda. \tag{4.2}$$

In equation 4.2 we multiply by the width of the wavelength step after summing the spectral irradiance values as this value is the same for all the observations included.

In equation 4.1 we sum 300 spectral irradiance values, if we would add both ends, then area would cover 301 nm instead of 300 nm.In both equations above we avoided this overestimation by excluding the observation at 700 nm. This approach can lead to bias. To avoid this bias we would need to take the average of the two extremes, in the case the wavelengths are measured (or interpolated) at the exact wavelengths without decimals. Only in the case when measurements are at wavelength values $\lambda_1 = 400.5; \lambda_2 = 401.5 \dots \lambda_{300} = 699.5$ nm, adding up these numbers would give an approximation of the correct irradiance. Figure 4.2 shows this case graphically, but with only eight and broad wavelength steps drawn.

If $\Delta\lambda < 1$ nm, using equation 4.1 instead of equation 4.2 would be represented in Figure 4.2 by overlapping rectangles, leading to overestimation by a factor equal to $1/\Delta\lambda$. In contrast, if $\Delta\lambda > 1$ nm, there would be gaps in between the columns, leading to underestimation.

In array spectrometers the actual measurements are not done at a constant pixel resolution. Instead, due to optical constraints, the wavelength step changes smoothly
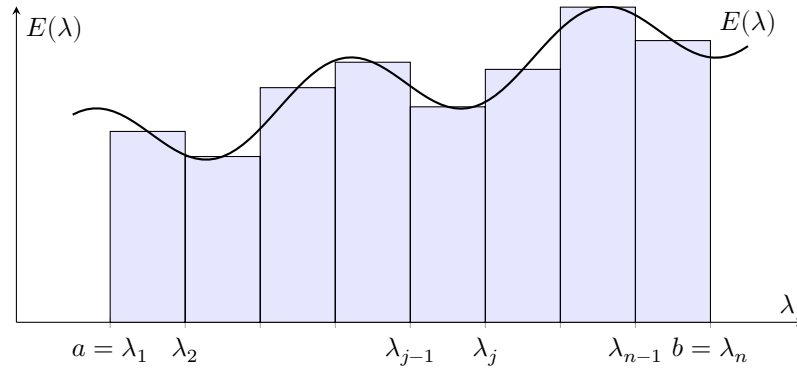
Figure 4.2: The sum of discrete observed values, is an approximation based on rectangles if their width is taken into account.

from pixel to pixel. Adding up observations will in such cases result in erroneous estimates of irradiance. Probably, the simplest method of numerical integration is the rule of the parallelogram. This method is an approximation, but not biased, and can be used with non-uniform $\Delta\lambda$. It is reliable when the number of observations in the range of wavelengths to be integrated is large enough to "track" the shape of the spectrum. Figure 4.3 shows the trapezium rule applied to a spectrum.

$$\int_{400}^{700} E(\lambda)\mathrm{d}\lambda \approx \sum_{i=1}^{n-1} \frac{E_i + E_{i+1}}{2} \cdot (\lambda_{i+1} - \lambda_i) \tag{4.3}$$

Equation 4.3 is directly applicable only if the data contains observations exactly at the boundaries of the range of wavelengths to be integrated. Otherwise the ends must be handled as special cases, calculating $\Delta\lambda$ using the extremes of the integration range instead of the wavelength corresponding to the observations immediately outside of the range.

4.2 Discontinuous functions

The integration algorithm described above is reliable only for continuous functions with a continuous derivative (slope). Spectra can be usually considered as continuous in relation to the resolution of spectrometers, as long as an instrument suitable for the measured light source has been used. Most commonly used biological spectral weighting functions (BSWFs) have discontinuities at their extremes wavelengths. They are not infinitely asymptotic but instead at a certain wavelength they abruptly end taking a value of zero past the boundary. A few BSWFs, of which the best known is CIE's erythemal action spectrum are defined as a series of linear segments with different
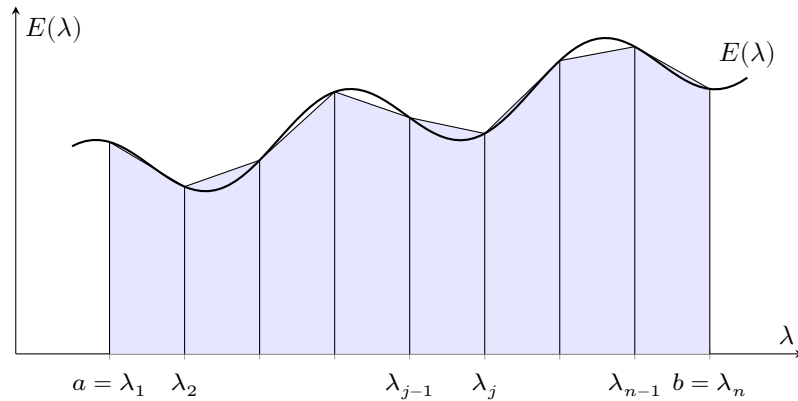
Figure 4.3: Trapezium rule as used by functions in package 'photobiology' except when the wavelength step is wide.

slopes. At the knots connecting a pair of segments, the derivative is discontinuous (the slope changes abruptly).

At their wavelength limits, the same problem affects the integration of wavelength ranges such color definitions/bands. This can introduce large errors when wavelength ranges are narrow, as is the case for the definitions of red and far-red used in plant biology, which are only 10 nm wide.

The way around this problem is to obtain estimates of spectral irradiance very near each side of the discontinuity *before* convolution of the spectral irradiance with the BSWF and/or numerical integration. This is still an approximation that requires additional computer processing, and is not always needed. In the functions in our packages, the user can override the default. The default is conservative, favouring accurate estimates rather than fast performance.

4.3 Scaling

Scaling consists in multiplying all individual values of the spectral variable by the same numeric constant. For example, if we have spectral irradiance, we could express scaling as:

$$E(\lambda) * k = E'(\lambda) \tag{4.4}$$

In many cases scaling is used when comparing spectra. For example, we may want to compare two spectra scaled so that they both share the same value for a summary quantity such as total irradiance. A simple case is re-scaling so that the area under two curves for the wavelength interval $400\text{nm} < \lambda < 700\text{nm}$ is the same, for example equal to one. We can then scale each spectrum as:
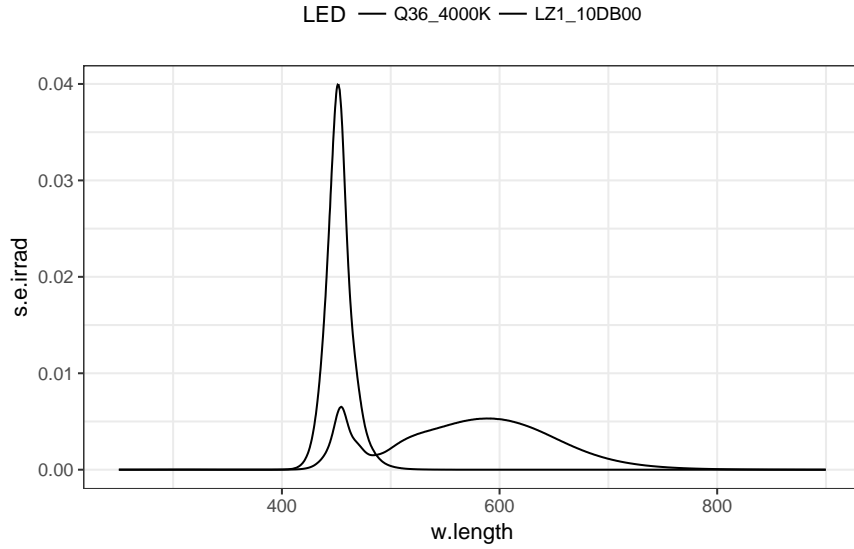
Figure 4.4: Scaling to equal PAR irradiance of the emission spectra of a blue and a white LED. The areas of the two exactly coincident faint rectangles represent the PAR irradiance integral under each of the two curves.

$$\frac{E(\lambda)}{\int_{400}^{700} E(\lambda)\mathrm{d}\lambda} = E_s(\lambda) \tag{4.5}$$

In Fig. 4.4 we demonstrate scaling according to equation 4.5 of the emission spectra of two LEDs, one with a narrow peak of emission, and one with a wide one. Other summary quantities can be used in addition to the integral.

4.4 Normalization

Normalization consists in scaling a spectrum so that the spectral quantity is equal to one at a certain wavelength. The most common normalization is to scale so that the maximum value of a spectral quantity is equal to one, in other words, we scale the spectrum by dividing it by the value at the highest peak. This case, more formally stated in equation 4.6, is exemplified in Fig. 4.5.

$$\frac{E(\lambda)}{\max E(\lambda)} = E_n(\lambda) \tag{4.6}$$

In some cases, normalization is done at a specific wavelength. A well known example is the normalization at $\lambda = 300$nm of BSWFs used in the quantification of ultraviolet radiation in biological studies.
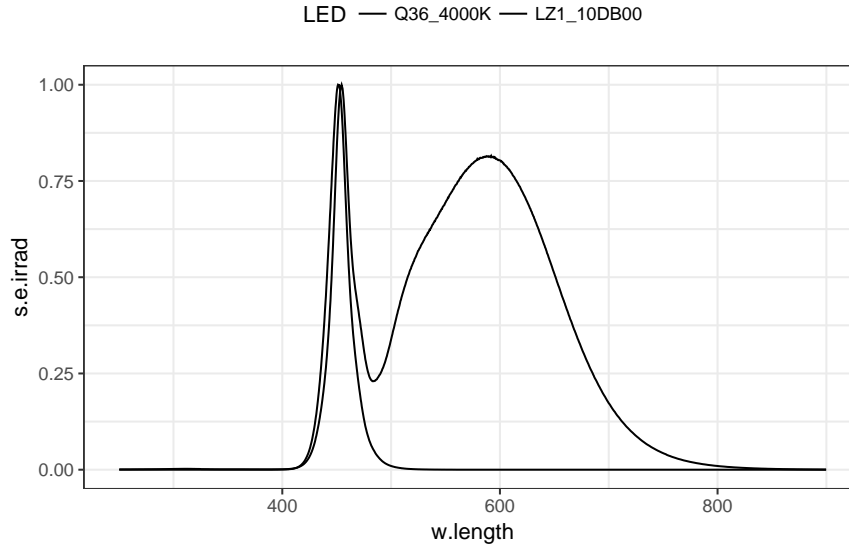
Figure 4.5: Normalization of the emission spectra of a blue and a white LED to one at peak of maximum emission. The areas of the faint rectangles represent the PAR irradiance integral under each of the two curves.

$$\frac{\mathcal{E}(\lambda)}{\mathcal{E}(\lambda = 300)} = \mathcal{E}_n(\lambda) \tag{4.7}$$

Where $\mathcal{E}(\lambda)$ is the spectral effectiveness.

4.5 Interpolation

4.6 Astronomy

Starting from version 0.9.12 of package 'photobiology' astronomical calculations are done according to the algorithms in the book *Astronomical Algorithms* (Meeus 1998). See http://www.esrl.noaa.gov/gmd/grad/solcalc/calcdetails.html for additional details and an on-line calculator.

4.6.1 Times to events

4.6.2 Position of the sun

4.7 Array-detector spectrometers

Contrary to detectors in digital photographic cameras which have square pixels arranged in a 2-dimensional grid, most array spectrometers have 1-dimensional arrays with rectangular pixels. Some exceptions exists with detectors with a very asymmetric but still 2-dimensional shape with a few pixels (e.g. 4) across the narrower dimension. A monochromator, usually with the aid of mirror optics projects the different wavelength components (colours) of the incoming light signal onto different pixels.

A detector pixel in a charge-coupled-device (CCD) can be thought of as a "bucket" or well collecting incoming photons. The integration time, is the time during which photons are collected by the pixels. At the end of this period the charge accumulated in each pixel is read. After the reading is taken, the "buckets" are emptied to get them ready for the next reading. Array detectors in miniature spectrometers can have detector arrays with from 128 to nearly 4000 pixels. Optical filters may be placed on the detector or at the spectrometer entrance.

Depending on the quantity to be measured, different entrance optics, either directly attached or connected through a optical fibre, are used. For irradiance, a cosine diffusor is required. For surface reflectance special probes are available with a relatively narrow angle of acceptance and integrating spheres allowing measurement of scattered plus directly transmitted radiation and specular plus scattered (total) reflectance. In the case of liquid samples, cuvette holders are used and also in this case, turbid suspensions will scatter light and will require special equipment.

Being array-detector spectrometers by necessity single monochromator instruments they present less than ideal signal:noise ratio (SNR) characteristics and are frequently afflicted by relatively high stray light levels. The electrical dynamic range of array detectors vary from instrument to instrument but depending on the range of wavelengths of interest may be limited by the dynamic range of the signal being measured and the different sensitivity of the detector to different wavelengths as the whole spectrum is acquired at once using the same settings. An additional problem, not exclusive of array spectrometers is the shape of the *slit function* which depends on the width and alignment of the instrument's entrance slit. The "annomalies" in slit function may decrease the effective wavelength resolution of spectrometers and introduce a small bias in cases of tail asymmetry.

There are several ways of getting around these problems, some more involved than others. We will give here a broad overview of the different problems and the corrections implemented in package 'ooacquire'.

4.7.1 Measurements—problems and solutions

Hot and dead pixels

Most array detectors will have a a few miss-functioning pixels. So called *hot pixels* return a relatively high reading in darkness or are exceptionally sensitive to light. So called *dead pixels* do not respond to radiation. Once identified, the raw counts data from these pixels can be replaced by the average of readings from the two neighbouring pixels. The errors introduced by this operation are usually minor except when measuring lasers or in the case of arrays with few pixels.

Non-linearity

The response of each detector pixel is not linear. When the wells are nearly full less charge is *trapped* in the well for each impinging photon. This needs to be corrected by applying a function that compensates for the curvature of the response curve. It is assumed that the same function is valid for all individual pixels in a given detector array. We use the polynomial supplied by Ocean Optics in their calibration certificates, and stored in the instruments' EEPROM. The linearization should be applied to raw counts before substraction of any dark readings, as the actual charge accumulated in the array wells determines the correction needed.

Dark electrical noise

To some extent array detectors are sensitive to stimuli other than the photons we are interested in measuring. One important factor is temperature, and the electrical signal returned by a detector kept in total darkness increases with increasing temperature. Any electronic circuit can be also disturbed by electrical fields and interference from the driving computer adding another, smaller source of random noise. This dark signal can be measured and subtracted from measurements. If the random dark noise rapidly varies in time we can reduce this variation by averaging several integrations, both for the dark measurement and the actual measurements. Noise can also be smoothed by averaging nearby pixels using a running mean or running median. Ocean Optics' literature calls this approach boxcar smoothing. Unless the spectrometer has considerably better wavelength resolution than needed for the application at hand, this type of smoothing should be avoided.

Dynamic range and resolution

The theoretical dynamic range in a single measurement is rarely achieved. Real values are smaller due to the above mentioned dark signal and its variation, and in some

cases stray light. Furthermore, the resolution with respect to number of photons per step in the count depends on the number of bits (binary digits) used to express the result of the count. For example an analogue to digital conversion (ADC) with 16 bits of resolution results in $2^16 = 65536$ discrete numbers or values. Not all spectrometers have electronics have ADCs with 16 bits of resolution. For example the miniature spectrometer model STS from Ocean Optics has an ADC with 14 bits of resolution ($2^14 = 16384$ possible discrete values).

The overall dynamic range of a spectrometer is much larger than for a single measurement, and is mainly given by the range of usable integration times and background optical and electrical noise. One way of reducing the dark noise is to cool either the detector or the whole spectrometer which allows the use of longer integration times and consequently the measurement of weaker light signals, such as fluorescence.

The dynamic range for a single measurement can be increased by *bracketing* the integration time used. In other words we measure under the same illumination conditions using the equivalent of what in photography would be called exposure values. The longer integration time will result in a spectrum with some regions clipped (over-exposed). By merging the spectra, using the long integration reading for the "darker" regions of the spectrum, and the short one for the brighter parts we may increase the effective dynamic range by an order of magnitude, or even more. However, one should be careful with this approach as the reading from pixels near those saturated (clipped) can be affected by charge leaking from them.

Stray light

Stray light is purely an optical phenomenon. Light scattering and reflections within the spectrometer result in radiation of the "wrong" wavelength hitting the pixels, resulting in non-zero or too high readings for some pixels. Being an error that is not present unless radiation is entering the spectrometer, it is not compensated by subtraction of a dark reading. If it cannot be controlled through optical means, it needs to be measured and subtracted separately from the dark signal. In the case of our Maya 2000 Pro spectrometers stray light in the UV region originates mainly from visible and specially infrared radiation. In such a case it can be measured by means of a UV-absorbing long pass filter. This measured stray light component can then used to correct the readings.

Slit function

Spectral resolution of an array spectrometer depends both on the slit and grating combination used and on how this resolved spectrum is projected onto different pixels. Say a spectrometer with a wide slit may have the wavelength resolution limited by

the slit function as even with a monochromatic light source like a laser several pixels can be illuminated. On the other hand an instrument with a narrow slit, and a limited number of relatively large pixels, will have its wavelength resolution limited by the pixels themselves, as light of different wavelengths will be received by different parts of the same individual pixel.

In the case than the pixel resolution is better or similar to the optical resolution, then if the shape of the slit function, usually with rather long tails affecting nearby pixels is characterized for different wavelengths, it can be corrected by deconvolution. This characterization needs to be done only once, unless the alignment of the optical components in changed.

Wavelength calibration

The correspondence between pixel positions and wavelengths is calibrated by comparison of certain known emission lines in lamps or sunlight. The wavelength will drift as a function of temperature, but the effect is relatively small unless the temperature change is large. In the case of array spectrometers, as they lack moving parts or a scanning mechanism, the wavelength calibration is quite stable in time as long as the temperature is the same. Ocean Optics provides a wavelength calibration but with the functions in package 'ooacquire' either a calibration stored in the instrument's EEPROM or one supplied by the user can be used.

Irradiance calibration

The sensitivity of each pixel to light from a suitable broad spectrum calibration lamp can be used to obtain a coefficient for each pixel to convert the output into irradiance or fluence values. These quantities are absolute quantities and consequently their measurement requires an absolute calibration against a light source and optical bench set up for which the irradiance or fluence are known with enough accuracy.

Transmittance, reflectance and absorptance

Transmittance, reflectance and absorptance are relative quantities we are normally measured against know clear or opaque, white, grey or black references. As long as we take into account the integration time, and the apply the linearization correction to the readings, no absolute calibration of the spectrometer is needed. It is important to realize that absorbance (A) and optical density (OD) are just different ways of expressing *internal* transmittance.

4.7.2 Data processing steps for irradiance

Part II

Tools used for calculations

5

Software

5.1 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 scripts. 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'. During the development of the packages, we used RStudio exclusively.

The typesetting is done with LaTeX and the source of this handbook was edited using both the shareware editor 'WinEdt' (which excels as a LaTeX editor) and 'RStudio' which is better suited to the debugging of the code examples. We also used LaTeX for our first handbook (Aphalo, Albert, Björn, Ylianttila, et al. 2012).

Combining R with Markdown (Rmarkdown: Rmd files) or LaTeX (Rnw files) to produce *literate* scripts is best for reproducible research and our suite of packages is well suited for this approach to data analysis. However, it is not required to go this far to be able to profit from R and our suite for simple analyses, but the set up we will describe here, is what we currently use, and 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 are many alternatives and the only piece that you cannot replace is R itself.

5.2 The different pieces

5.2.1 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 on a programming language, and being able to call and being called by programs and subroutine libraries written in several other programming languages, makes R easily extensible.

R has a well defined mechanism for "addons" 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.

The *engine* behind the production of the pages of this handbook is the R package 'knitr' which allows almost seamless integration of R code and text marked up using LaTeX. We have used in addition several other packages, both as building blocks in our packages, and for the production of the examples. The most notable ones are: 'tibble', 'dplyr', 'readr', 'lubridate', 'ggplot2', and 'ggtern'. Packages 'devtools' and 'testthat' significantly eased the task of package development and coding.

If you are not familiar with R, please, before continuing reading this handbook read a book on the R *language* itself—rather than a book on statistics with R. The book 'Learning R …as you learnt your mother tongue' (Aphalo 2016) takes the approach of learning the R language through exploration, which is the way many experienced R programmers unravel the intricacies of the language. In contrast the more concise book 'R Programming for Data Science' (Peng 2016) takes a more direct and possibly less challenging approach to teaching the R language. There is also a free introduction to R by Paradis (2005) available at https://cran. r-project.org/doc/contrib/Paradis-rdebuts_en.pdf and a 'student's Guide to R' by Horton (2015) at https://cran.r-project.org/doc/contrib/Horton+ Pruim+Kaplan_MOSAIC-StudentGuide.pdf.

5.2.2 RStudio

'RStudio' is an intergrated development environment (IDE). The difference between an IDE and an editor is that an IDE integrates additional tools that facilitate the interaction with R. RStudio highlights the R code according to the syntax, offers auto-

completion while typing, highlights invalid code on the fly. When a script is run, if an error is triggered, it automatically find the location of the error. RStudio also supports the concept of projects allowing saving of settings separately. It also integrates support for file version control (see next section).

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 run in the 'cloud', for example, as an AWS instance (Amazon Web Services) quite easily and cheaply, or on one's own server hardware. 'RStudio' is under active development, and constantly improved (visit `http://www.rstudio.org/` for an up-to-date description and download and installation instructions.

Two books (Hillebrand and Nierhoff 2015; Loo and Jonge 2012) describe and teach how to use RStudio without going in depth into data analysis or statistics, however, as 'RStudio' is under very active development new features are or will be missing from these books as time goes by. You will find tutorials and up-to-date cheat sheets at `http://www.rstudio.org/`.

5.2.3 Revision control: Git and Subversion

Revision control systems help by 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, was designed by Linus Tordvals of Linux fame, 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.

The books 'Git: Version Control for Everyone' (Somasundaram 2013) and 'Pragmatic Guide to Git' (Swicegood 2010) are good introductions to revision control with Git. Free introductory videos and *cheatsheets* are available at `https://git-scm.com/doc`.

5.2.4 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. With suitable build tools (e.g. 'RTools' for Windows) source packages can be installed and developed in any of the operating systems on which R runs.

5.2.5 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 additional font formats, and is the most popular TeX engine in China and other Asian countries. Both XₑLaTeX and LuaTeX are rapidly becoming popular also for typesetting texts in variants of Latin and Greek alphabets as these new TeX engines natively support large character sets and modern font formats such as TTF (True Type) and OTF (Open Type).

For the typesetting of this handbook we used several LaTeX packages, of which those that most affected appearance are 'KOMA-script', 'hyperref', 'booktabs', 'pgf/tikz' and 'biblatex'. The TeX used is MikTeX.

5.2.6 Markdown

Markdown is a simple markup language, which although offering somehow less flexibility that LaTeX is much easier to learn and which can be easily converted to various output formats such as HTMLand XTML in addition to PDF. RStudio supports editing markdown and the variants Rmarkdown and Bookdown. Documentation on Rmarkdown is available on-line at http://rmarkdown.rstudio.com/ and on Bookdown at https://bookdown.org/.

6

Photobiology R packages

6.1 Expected use and users

One aim of the suite is to provide a tools for teaching VIS and UV radiation physics and photobiology. Another aim is to make it easier for researchers in the field of photobiology to do calculations required for the description of irradiation conditions and for simulations useful for data validation and/or when designing experiments. The suite is a collection of classes, methods and functions, accompanied by data sets. In particular we hope the large set of example data will make it easy to carry out sanity checks of newly acquired and/or published data.

Given the expected audience of both students and biologists, rather than data analysts, or experienced programmers, we have aimed at designing a consistent and easy to understand paradigm for the analysis of spectral data. The design is based on our own user experience, and on feedback from our students and 'early adopters'.

6.2 The design of the framework

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* (Figure 6.1). All spectra have in common that observations are referenced to a wavelength value. However, there are different types spectral objects, e.g. for light sources and responses to light. 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.

A consistent naming scheme for methods as well as consistency in the order of arguments across the suite should reduce the number of *names* to remember. Data
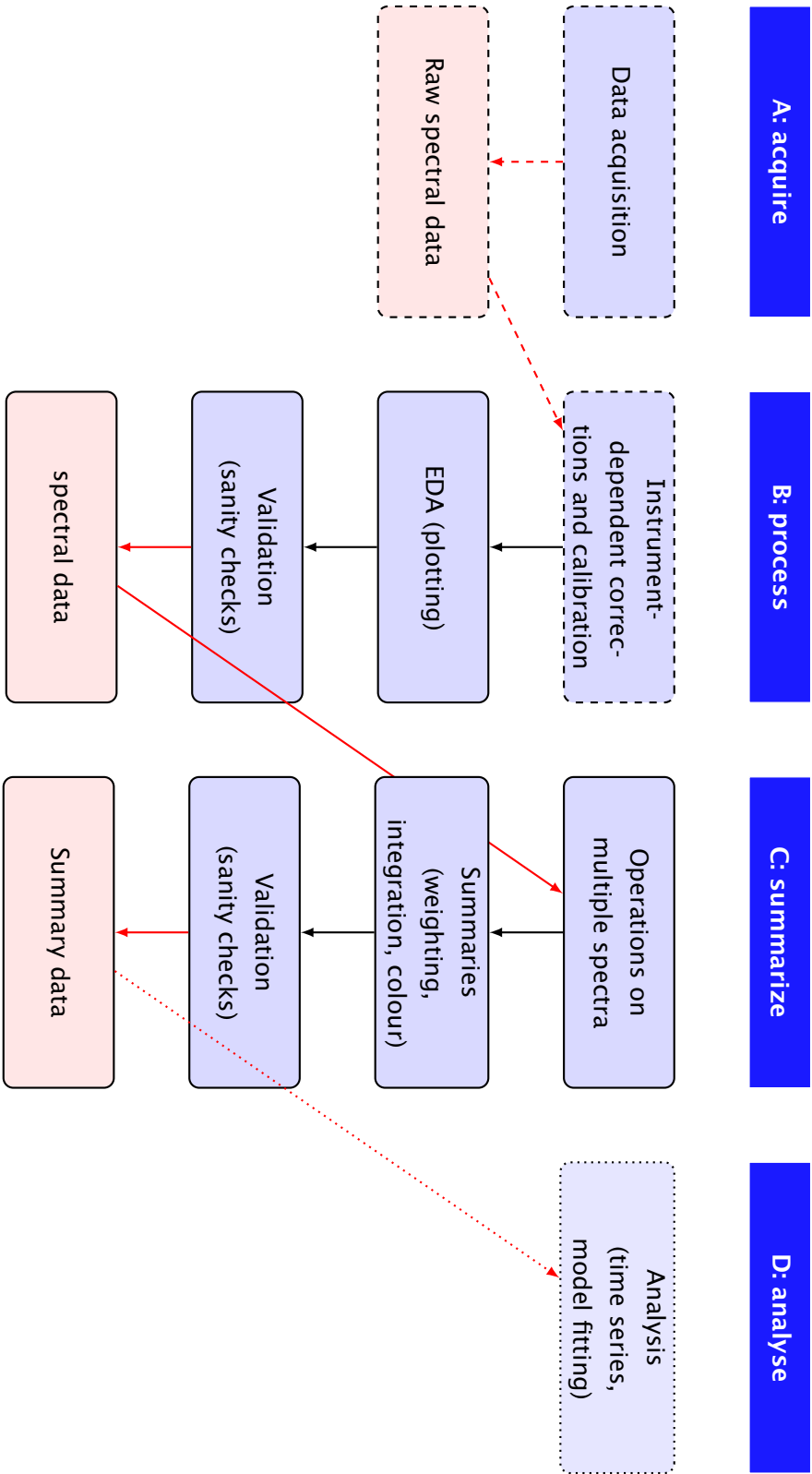
Figure 6.1: Data "pipeline" from acquisition to analysis of spectral data

Box 6.1: Elements of the framework used by all packages in the suite.

`_spct` Spectral objects are containers for different types of spectral data, data which is referenced to wavelength. These data normally originate in measurements or simulation with models.

`_mspct` Containers for spectral objects are used to store related spectral objects, such as time series of spectral objects or spectral images.

`wavebands` Waveband objects are containers of 'instructions' for the quantification of spectral data. In addition to the everyday definition as a range of wavelengths, we include the spectral weighting functions used in the calculation of what are frequently called weighted or effective exposures and doses.

maths operators and functions Used to combine and/or transform spectral data, and in some cases to apply weights defined by wavebands.

`apply` **methods** Used to apply functions individual spectra stored in collections of spectra.

summary methods and functions Different summary functions return different quantities through integration over wavelengths and take as arguments spectra and wavebands.

`plot` **methods** Simplify the construction of specialized plots of spectral data.

foreign data exchange functions For importing data output by measuring instruments and exchanging data with other R packages.

objects are *tidy* as defined by in (Wickham 2014), in other words data on a row always corresponds to a single observation event, although such an observation can consist in more than one measured or derived quantity. Data from different observations are stored in different objects, or if in the same object they are *keyed* using and index variable.

As an example, the same summary methods, are available for individual spectra (`_spct` objects) and and collections of spectra (`_mspct` objects), in the first case returning a numeric vector, and in the second case, a `data.frame` object.

Package '`photobiology`' can be thought as a framework defining a way of storing spectral data plus 'pieces' of code from which specific methods can be constructed, plus ready defined methods for frequently used operations. Extensibility and reuse is at the core of the design. This is achieved by using the weakest possible assumptions or expectations about data properties and avoiding as much as possible hard-coding of any constants or size limits. This, of course, has a cost in possibly slower execution speed. Within these constraints an effort has been made to remove performance bottlenecks by extensive testing and in isolated cases writing functions in C++.

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

Is all what is needed to obtain the CIE98-weighted energy irradiance simulating the effect of a polyester filter on the example solar spectrum, which of course, can be substituted by other spectral irradiance and filter 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 available at the same wavelengths values, and the BSWF is defined as a function. Interpolation of the spectral data and calculation of spectral 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 a 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 correc-
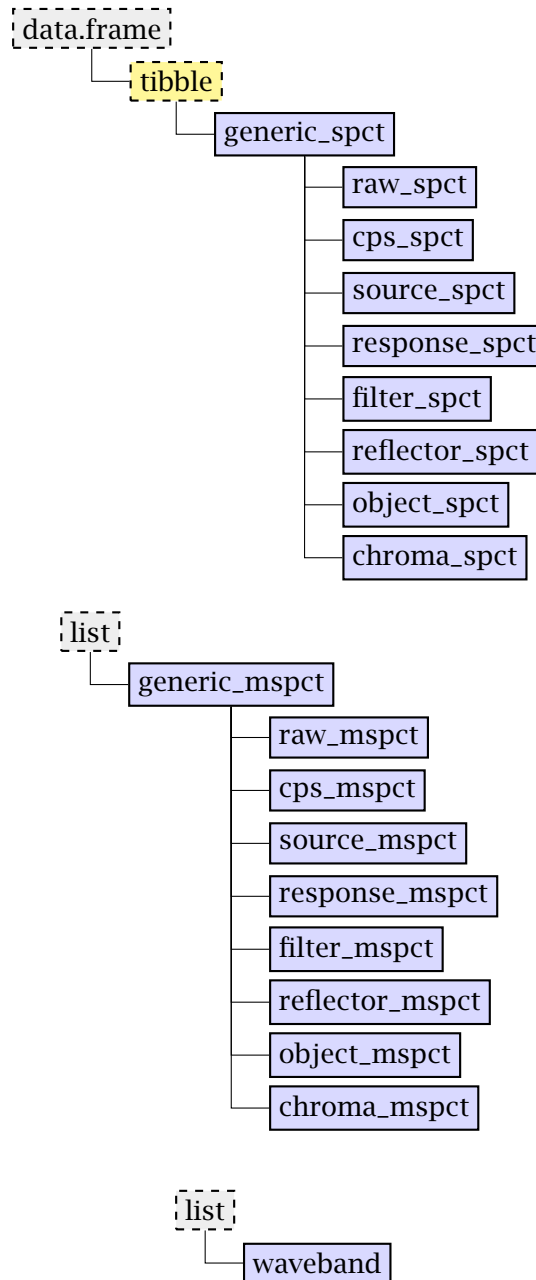
Figure 6.2: Object classes used in the packages. Objects of _spct classes are used to store spectra, in most cases a single spectrum. Objects of _mspct classes can be used to store *collections* of _spct objects, in most cases all belonging to the same class. Objects of class waveband contain information used for quantification: boundaries of a wavelength range and, optionally, spectral weighting functions. Gray-filled boxes represent classes defined in base R, yellow-filled boxes represent classes defined by contributed packages available through CRAN, Comprehensive R Archive Network, and blue-filled boxes represent classes defined in package 'photobiology'.

Table 6.1: Packages in the r4photobiology suite. Packages not yet released are marked with a red bullet •, and those at 'beta' stage with a yellow bullet •, those relatively stable with a green bullet •, and those in CRAN with the label CRAN in the first column of the table.

	Package	Type	Contents
•	photobiologyAll	dummy	loads other packages of the suite
CRAN	photobiology	funs + classes	basic functions, class definitions, class methods and example data
CRAN	photobiologyInOut	functions	data import/export functions
CRAN	photobiologyWavebands	definitions	quantification of radiation
CRAN	ggspectra	functions	extensions to package 'ggplot2'
•	photobiologySun	data	spectral data for solar radiation
CRAN	photobiologyLamps	data	spectral data for lamps
CRAN	photobiologyLEDs	data	spectral data for LEDs
•	photobiologyFilters	data	transmittance data for filters
•	photobiologySensors	data	response data for sensors
•	photobiologyReflectors	data	reflectance data for materials
CRAN	photobiologyPlants	funs + data	photobiology of plants
•	rOmniDriver	driver API	Ocean Optics spectrometers
•	ooacquire	data acquistion	Ocean Optics spectrometers

tion procedures to those data. A future package will provide an interface to the TUV model to allow easy simulation of the solar spectrum.

6.3 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 6.1).

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 nanometres (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 aim to be able to make most changes backwards compatible, in other words, we expect that they will not break existing user code.

6.4 The r4photobiology repository

Seven of the packages are already available through CRAN, the 'Comprehensive R Archive Network'. For distributing other packages in the suite, I have created a repository at http://r4photobiology.info/R. This repository follows the CRAN folder structure, so package installation can be done using normal R commands. Consequently dependencies are installed automatically and 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 r4photobiology 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 r4photobiology repository to the current session as follows, which will give you a menu of additional repositories to activate:

```
setRepositories(
  graphics = getOption("menu.graphics"),
  ind = NULL,
  addURLs = c(r4photobiology = "http://www.r4photobiology.info/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.

```
setRepositories(
  graphics = getOption("menu.graphics"),
  ind = c(1, 6),
  addURLs = c(r4photobiology = "http://www.r4photobiology.info/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` ', unless it already exists. 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. So previous example becomes:

```
utils::setRepositories(
  graphics = getOption("menu.graphics"),
  ind = c(1, 6),
  addURLs = c(r4photobiology = "http://www.r4photobiology.info/R"))
```

The `.Rprofile` file located in the current folder is *sourced* (i.e. executed) 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 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 r4photobiology repository as shown above, the photobiology packages can be installed and uninstalled through the normal RStudio menus and dialogues, and will be 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", "ggspectra"))
```

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 an user-account wide .Rprofile file, and a different one on those folders needing different settings. Many other R options can also be modified by means of commands in the .Rprofile file.

Part III

Cookbook of calculations

$<$

7

Math operators and functions

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.

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

---

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

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

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 16. 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.

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 7.1 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.spct

## Object: source_spct [533 x 2]
## Wavelength range 280 to 800 nm, step 0.07692308 to 1 nm
## Time unit 1s
##
## # A tibble: 533 × 2
##   w.length s.e.irrad
```

```
##       <dbl>      <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.0000         0
## 4 281.8462         0
## 5 282.0000         0
## # ... with 528 more rows
```

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 oneÂ´. 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

## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 517 more rows

2 * sun.spct

## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
```

```
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 517 more rows
```

```
sun.spct * 2
```

```
## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 517 more rows
```

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)
```

```
## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
```

```
##
## # A tibble: 522 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1 280.0000      -Inf
## 2 280.9231      -Inf
## 3 281.8462      -Inf
## 4 282.7692      -Inf
## 5 283.6923      -Inf
## # ... with 517 more rows
```

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.

```
sun.spct * filters.mspct$GG400

## Object: source_spct [533 x 2]
## Wavelength range 280 to 800 nm, step 0.07692308 to 1 nm
## Time unit 1s
##
## # A tibble: 533 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.0000         0
## 4 281.8462         0
## 5 282.0000         0
## # ... with 528 more rows
```

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

```
sun.spct * filters.mspct$GG400 * (100 - 9) / 100

## Object: source_spct [533 x 2]
## Wavelength range 280 to 800 nm, step 0.07692308 to 1 nm
## Time unit 1s
##
## # A tibble: 533 × 2
##    w.length s.e.irrad
```

```
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.0000         0
## 4 281.8462         0
## 5 282.0000         0
## # ... with 528 more rows
```

Calculations related to filters will be explained in detail in chapter 24. 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

## Object: source_spct [522 x 3]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010–06–22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 3
##   w.length s.e.irrad s.q.irrad
##      <dbl>     <dbl>     <dbl>
## 1 280.0000         0         0
## 2 280.9231         0         0
## 3 281.8462         0         0
## 4 282.7692         0         0
## 5 283.6923         0         0
## # ... with 517 more rows

sun.spct * 2

## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010–06–22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
```

```
## 1 280.0000        0
## 2 280.9231        0
## 3 281.8462        0
## 4 282.7692        0
## 5 283.6923        0
## # ... with 517 more rows
```

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)
```

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 vectorised, 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, we can achieve the same result, with simpler syntax, using spectral objects and the corresponding operators.

```
sun.spct + sun.spct

## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
```

74

```
## Time unit 1s
##
## # A tibble: 522 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 517 more rows
```

```
e2q(sun.spct + sun.spct)

## Object: source_spct [522 x 3]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Time unit 1s
##
## # A tibble: 522 × 3
##   w.length s.e.irrad s.q.irrad
##      <dbl>     <dbl>     <dbl>
## 1 280.0000         0         0
## 2 280.9231         0         0
## 3 281.8462         0         0
## 4 282.7692         0         0
## 5 283.6923         0         0
## # ... with 517 more rows
```

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. The class of the returned spectrum depends on the classes of the operands. In this case returned objects are `source_spct`.

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,

$$q = h \cdot \nu = h \cdot \frac{c}{\lambda} \qquad (7.1)$$

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

$$q' = h' \cdot \nu = h' \cdot \frac{c}{\lambda} \qquad (7.2)$$

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:

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

```
## [1] 919.5147
```

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

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

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

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

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

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

`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.

```
e2q(sun.spct, byref = FALSE)

## Object: source_spct [522 x 3]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 3
##   w.length s.e.irrad s.q.irrad
##      <dbl>     <dbl>     <dbl>
## 1 280.0000         0         0
## 2 280.9231         0         0
## 3 281.8462         0         0
## 4 282.7692         0         0
## 5 283.6923         0         0
## # ... with 517 more rows
```

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

```
sun.spct

## Object: source_spct [522 x 3]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 3
##   w.length s.e.irrad s.q.irrad
##      <dbl>     <dbl>     <dbl>
## 1 280.0000         0         0
## 2 280.9231         0         0
## 3 281.8462         0         0
## 4 282.7692         0         0
## 5 283.6923         0         0
## # ... with 517 more rows

e2q(sun.spct, "replace", byref = FALSE)

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

```
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 2
##   w.length s.q.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 517 more rows
```

`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 the handbook "Introduction" (Aphalo, Albert, Björn, Ylianttila, et al. 2012) it is written: "Example 1: red light at 600 nm has about 200 $kJ\,mol^{-1}$, therefore, 1 μmol photons has 0.2 J. Example 2: UV-B radiation at 300 nm has about 400 $kJ\,mol^{-1}$, 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 $J\,mol^{-1}$ we multiply the result by 10^{-3} to obtain $kJ\,mol^{-1}$:

```
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:

78

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

```
q2e(sun.spct, "replace", byref = FALSE)

## Object: source_spct [522 x 2]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1 280.0000          0
## 2 280.9231          0
## 3 281.8462          0
## 4 282.7692          0
## 5 283.6923          0
## # ... with 517 more rows
```

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.

```
my_sun.spct <- sun.spct
q2e(my_sun.spct, byref = TRUE)

## Object: source_spct [522 x 3]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 522 × 3
##    w.length s.e.irrad s.q.irrad
##       <dbl>     <dbl>     <dbl>
## 1 280.0000          0          0
```

```
## 2 280.9231          0          0
## 3 281.8462          0          0
## 4 282.7692          0          0
## 5 283.6923          0          0
## # ... with 517 more rows
```

`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.

Task: Using options to change default behaviour of maths operators and functions

Table 7.2 lists all the recognized options, and their default values. Within the suite all functions have a default value which is used when the options are undefined. Options are set using base R's function `options`, and queried with functions `options` and `getOption`. Using options can result in more compact and terse code, but the user should clearly document the use of non-default values for options to avoid surprising the reader of the code.

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

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

## Object: source_spct [37 x 2]
## Wavelength range 280 to 315 nm, step 0.9230769 to 1 nm
## Time unit 1s
##
## # A tibble: 37 × 2
##   w.length s.q.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 32 more rows
```

Table 7.2: Options affecting calculations by functions and operators in the 'photobiology' package and their possible values. Options controlling the printing of the returned values are also listed.

Option	default	function
Base R		
digits	7	$d - 3$ used by `summary`
Package 'tibble'		
tibble.print_max	$n_{\max} = 20$	nrow(*spct*) $> n_{\max}$ print $n_{\min}$ lines
tibble.print_min	$n_{\min} = 10$	nrow(*spct*) $> n_{\max}$ print $n_{\min}$ lines
R4photobioloy suite		
photobiology.radiation.unit	`"energy"`	output ($\mathrm{W\,m^{-2}\,nm^{-1}}$)
	`"photon"`	output ($\mathrm{mol\,m^{-2}\,s^{-1}\,nm^{-1}}$)
photobiology.filter.qty	`"transmittance"`	output (/1)
	`"absorptance"`	output (/1)
	`"absorbance"`	output (a.u. $\log_{10}$ base)
photobiology.use.hinges	`NULL`	guess automatically
	`TRUE`	do insert hinges
	`FALSE`	do not insert hinges

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

## Object: source_spct [37 x 2]
## Wavelength range 280 to 315 nm, step 0.9230769 to 1 nm
## Time unit 1s
##
## # A tibble: 37 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 32 more rows
```

⚠️ For filters, an option controls whether transmittance, the default, or absorbance is use in the operations, and the controls the returned quantity. It is important to remember that absorbance, A, is always expressed on a logarithmic

scale, while transmittance, T, is always expressed on a linear scale. So to simulate the effect of stacking two layers of polyester film, we need to sum, or in this case as there are two layers of the same material, multiply by two the spectral absorbance values, while we need to multiply the spectral transmittances of stacked filters, or use a power when the layers are identical.

```
options(photobiology.filter.qty = "absorbance")
polyester.spct * 2

## Object: filter_spct [611 x 2]
## Wavelength range 190 to 800 nm, step 1 nm
##
## # A tibble: 611 × 2
##    w.length        A
##       <int>     <dbl>
## 1       190 3.917215
## 2       191 4.000000
## 3       192 3.917215
## 4       193 3.647817
## 5       194 3.591760
## # ... with 606 more rows

options(photobiology.filter.qty = "transmittance")
polyester.spct ^ 2

## Object: filter_spct [611 x 2]
## Wavelength range 190 to 800 nm, step 1 nm
##
## # A tibble: 611 × 2
##    w.length      Tfr
##       <int>     <dbl>
## 1       190 0.000121
## 2       191 0.000100
## 3       192 0.000121
## 4       193 0.000225
## 5       194 0.000256
## # ... with 606 more rows
```

Either option can be unset, by means of the `NULL` value[3].

```
options(photobiology.radiation.unit = NULL)
options(photobiology.filter.qty = NULL)
```

---

[3]If you are planning to continue working through the examples in later sections, do reset the options as shown in this chunk, as otherwise, the results of later calculations will differ from those shown.

The proper use of trimming of wavebands is important, and option `photobiology.waveband.trim` makes changing the behaviour of the `trim_spct` function and other functions accepting wavebands easier. The need to carefully assess the validity of trimming and how it can affect the interpretation of results is further discussed in Chapter 10 and Chapter 11.

Other options affect the optimization of performance vs. precision of calculations and can be useful especially when processing huge numbers of spectra. Some options defined in base R and package 'dplyr' affect printing of output (Table 7.2).

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 **??**.

```
is_effective(UVA())

## [1] FALSE

sun.spct * UVA()

## Object: source_spct [86 x 2]
## Wavelength range 315 to 400 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 86 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      315 0.1127901
## 2      316 0.1020587
## 3      317 0.1487690
## 4      318 0.1413919
## 5      319 0.1569692
## # ... with 81 more rows
```

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()

## Object: source_spct [122 x 2]
## Wavelength range 280 to 400 nm, step 0.9230769 to 1 nm
## Time unit 1s
## Data weighted using 'CIE98.298' BSWF
##
## # A tibble: 122 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## 5 283.6923         0
## # ... with 117 more rows
```

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

8

Spectra: simple summaries and features

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(photobiologyWavebands)
library(ggplot2)
library(ggspectra)
library(photobiologyLamps)
library(photobiologyFilters)
library(photobiologyReflectors)
library(polynom)
```

8.2 Task: Printing spectra

Spectral objects are printed with a special `print` method that is an extension to the print method for `tibble` objects, consequently, it is possible to use options from package 'tibble' to control printing. The first option set below, `tibble.print_max`, sets the number of rows above which only 'head' rows are printed and `tibble.print_min` sets the number of rows printed when the head is printed. Another option, `tibble.width` sets width of the output printed for `tibble` objects with many columns.

```
options(tibble.print_max = 4, tibble.print_min = 4)
```

The number of rows printed can be also controlled through an explicit argument to the second parameter of `print`, `head`, and `tail`. Setting an option by means of `options` changes the default behaviour of `print`, but explicit arguments can still be used for changing this behaviour in an individual statement.

8.3 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)

## Summary of object: source_spct [522 x 3]
## Wavelength range 280 to 800 nm, step 0.9230769 to 1 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit: 1s
##
##     w.length        s.e.irrad
## Min.   :280.0   Min.   :0.0000
## 1st Qu.:409.2   1st Qu.:0.4115
## Median :539.5   Median :0.5799
## Mean   :539.5   Mean   :0.5160
## 3rd Qu.:669.8   3rd Qu.:0.6664
## Max.   :800.0   Max.   :0.8205
##    s.q.irrad
## Min.   :0.000e+00
## 1st Qu.:1.980e-06
## Median :2.929e-06
## Mean   :2.407e-06
## 3rd Qu.:3.154e-06
## Max.   :3.375e-06
```

8.4 Task: Integrating spectral data

Package 'photobiology' provides specific functions for frequently used quantities, but in addition 'general purpose' function is available to add flexibility for special cases. Function `integrate_spct` takes into account each individual wavelength step, so it returns valid results even for spectra measured at arbitrary and varying wavelength steps. This function operates on all `numeric` variables contained in a spectral object except for `w.length`. The returned value is expressed as a total per spectrum. See section 4.1 for details on the integration algorithm used and an explanation of why in most cases adding up the spectral values leads to wrong estimates.

```
integrate_spct(sun.spct)

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

8.5 Task: Averaging spectral data

Package 'photobiology' provides specific functions for frequently used quantities, but in addition 'general purpose' function is available to add flexibility for special cases. Function `average_spct` takes into account each individual wavelength step, so it returns valid results even for spectra measured at arbitrary and varying wavelength steps. This function operate on all `numeric` variables contained in a spectral object except for `w.length`. The returned value is expressed per nanometre.

```
average_spct(sun.spct)

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

8.6 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] 280 800

midpoint(sun.spct)

## [1] 540

max(sun.spct)

## [1] 800

min(sun.spct)

## [1] 280
```

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] 520
```

87

```
stepsize(sun.spct)
```

```
## [1] 0.9230769 1.0000000
```

8.7 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" "tbl_df"
## [4] "tbl"          "data.frame"
```

```
class(polyester.spct)
```

```
## [1] "filter_spct"  "generic_spct" "tbl_df"
## [4] "tbl"          "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.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.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.spct)
```

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

```
is.filter_spct(sun.spct)
```

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

```
is.filter_spct(polyester.spct)
```

```
## [1] TRUE
```

8.8 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. See sections 4.4 on page 39 and 4.3 on page 38 for an explanation of these operations.

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

## [1] "total"
```

and `reflector_spct` objects.

```
getRfrType(gold.spct)

## [1] "total"
```

8.9 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)

## Object: source_spct [6 x 3]
## Wavelength range 280 to 284.61538 nm, step 0.9230769 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 6 × 3
##   w.length s.e.irrad s.q.irrad
## *    <dbl>     <dbl>     <dbl>
## 1 280.0000         0         0
## 2 280.9231         0         0
## 3 281.8462         0         0
## 4 282.7692         0         0
## # ... with 2 more rows

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

## Object: source_spct [6 x 2]
## Wavelength range 280 to 284.61538 nm, step 0.9230769 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 6 × 2
##   w.length s.q.irrad
## *    <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.8462         0
## 4 282.7692         0
## # ... with 2 more rows

is_energy_based(my.spct)

## [1] FALSE

is_photon_based(my.spct)

## [1] TRUE
```

We now consider `filter_spct` objects.

```
is_transmittance_based(polyester.spct)

## [1] TRUE

is_absorbance_based(polyester.spct)

## [1] FALSE
```

8.10 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.
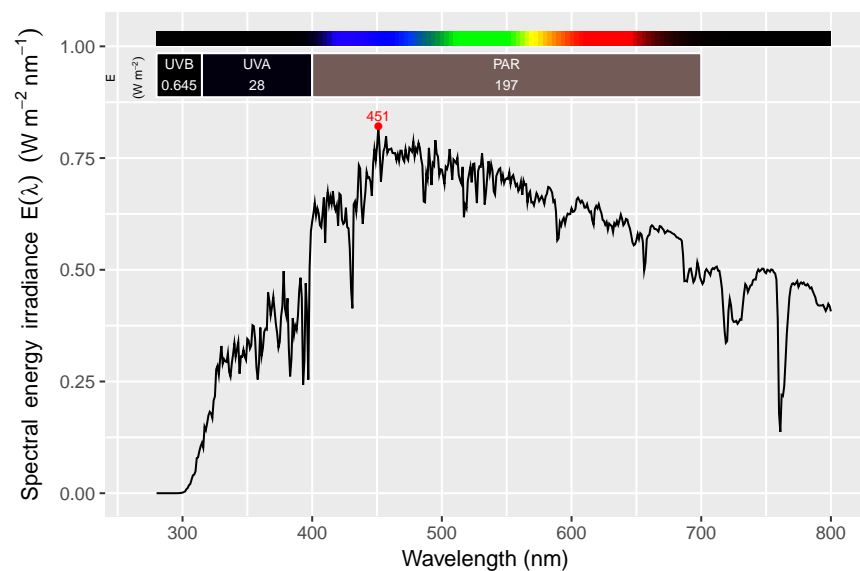
```
is_effective(sun.spct)
```

```
## [1] FALSE
```
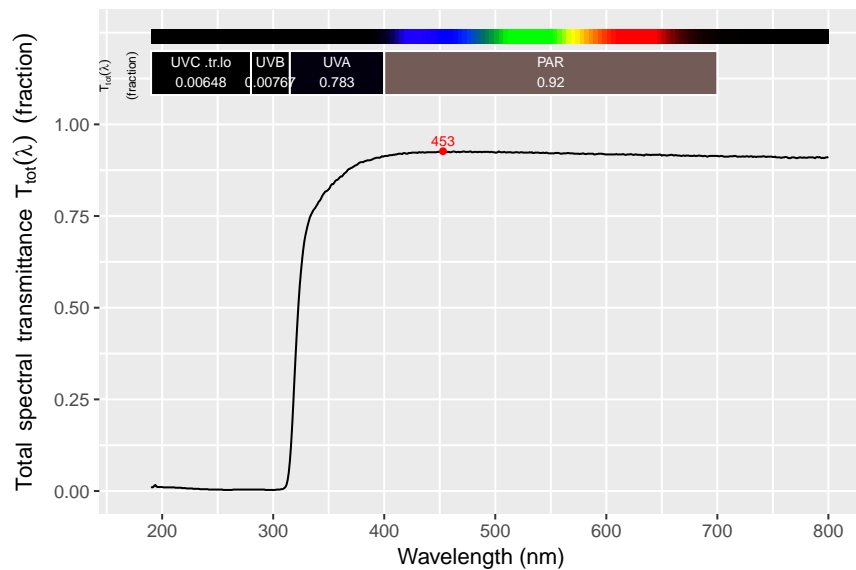
8.11 Task: Plotting a spectrum

Method plot is defined for waveband objects, and can be used to visually check their properties. Plotting is discussed in detail in chapter 16.

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



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

8.12 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`. As some comments contain new line characters, to get them printed nicely we need to use `cat`.

```r
names(sun.spct)

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

cat(comment(sun.spct))

## Simulated solar spectrum based on real weather conditions.
## Data author: Dr. Anders Lindfors
## Finnish Meteorological Institute.
## See help file for references.
```

8.13 Task: Extract 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.

We first show the use of function `peaks` that returns a subset of the spectral containing peaks (local maxima). The parameter `span` determines the number of discrete `w.length` values used to find a local maximum (the higher the value used, the fewer maxima are detected), with a `NULL` span returning the overall maximum. The `span` 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. The parameter `ignore_threshold` indicates the fraction of the total span along the spectral variable, such as spectral irradiance for `source_spct` objects, 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)[1].

Using default `span` and then a relatively large value of 51.

```
peaks(lamps.mspct$philips.tl12)

## Object: source_spct [10 x 2]
## Wavelength range 253.5 to 390.5 nm, step 2 to 52.5 nm
## Time unit 1s
## Spectral data normalized to 1 at 313 nm
##
## # A tibble: 10 × 2
##    w.length    s.e.irrad
## *     <dbl>        <dbl>
## 1     253.5 1.530899e-04
## 2     258.0 8.651685e-07
## 3     260.5 1.609551e-06
## 4     262.5 7.471910e-07
## # ... with 6 more rows

peaks(lamps.mspct$philips.tl12, span = 51)

## Object: source_spct [2 x 2]
## Wavelength range 313 to 365.5 nm, step 52.5 nm
## Time unit 1s
## Spectral data normalized to 1 at 313 nm
##
## # A tibble: 2 × 2
##    w.length s.e.irrad
## *     <dbl>     <dbl>
## 1     313.0 1.0000000
## 2     365.5 0.3005618
```

Using `NULL` as argument for span is interpreted as meaning the whole wavelength range of the spectral data, and always returns only the highest values in the spectrum.

---

[1]In the current example setting `ignore_threshold` equal to 0.75 given that the range of the spectral irradiance data goes from 0.00 μmol m^{-2} s^{-1} nm^{-1} to 0.82 μmol m^{-2} s^{-1} nm^{-1}, causes any peaks having a spectral irradiance of less than 0.62 μmol m^{-2} s^{-1} nm^{-1} to be ignored.

```
peaks(lamps.mspct$philips.tl12, span = NULL)

## Object: source_spct [1 x 2]
## Wavelength range 313 to 313 nm, step NA nm
## Time unit 1s
## Spectral data normalized to 1 at 313 nm
##
## # A tibble: 1 × 2
##   w.length s.e.irrad
## *    <dbl>     <dbl>
## 1      313         1
```

Using threshold we can limit the search for peaks to only part of the y-range of the data. This is useful when peaks in "low ares" of the curve are of no interest.

```
peaks(sun.spct,
      span=31,
      ignore_threshold=0.2)

## Object: source_spct [11 x 2]
## Wavelength range 378 to 774 nm, step 17 to 85 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 11 × 2
##   w.length s.e.irrad
## *    <dbl>     <dbl>
## 1      378 0.4969714
## 2      416 0.6761818
## 3      451 0.8204633
## 4      478 0.7869773
## # ... with 7 more rows

peaks(sun.spct,
      span=31,
      ignore_threshold=0.8)

## Object: source_spct [7 x 2]
## Wavelength range 416 to 605 nm, step 17 to 51 nm
## Measured on 2010-06-22 09:51:00 UTC
## Measured at 60.20942 N, 24.96424 E
## Time unit 1s
##
## # A tibble: 7 × 2
##   w.length s.e.irrad
## *    <dbl>     <dbl>
## 1      416 0.6761818
## 2      451 0.8204633
```

```
## 3       478 0.7869773
## 4       495 0.7899872
## # ... with 3 more rows
```

It is also possible to convert how the returned spectral irradiance is expressed.

```
peaks(lamps.mspct$philips.tl12, unit.out = "photon")

## Object: source_spct [13 x 2]
## Wavelength range 253.5 to 390.5 nm, step 2 to 32 nm
## Time unit 1s
## Spectral data normalized to 1 at 313 nm
##
## # A tibble: 13 × 2
##   w.length     s.q.irrad
## *    <dbl>         <dbl>
## 1    253.5 3.244072e-10
## 2    258.0 1.865892e-12
## 3    260.5 3.504922e-12
## 4    262.5 1.639559e-12
## # ... with 9 more rows
```

All the examples earlier in this section also apply to other types of spectra, and to `valleys` in addition to `peaks`.

```
valleys(filters.mspct$KG5)

## Object: filter_spct [6 x 2]
## Wavelength range 424 to 3000 nm, step 34 to 900 nm
##
## # A tibble: 6 × 2
##   w.length          Tfr
## *    <dbl>        <dbl>
## 1      424 8.478600e-01
## 2      458 8.615100e-01
## 3      529 8.742400e-01
## 4     1250 6.823387e-06
## # ... with 2 more rows

peaks(filters.mspct$KG5, filter.qty = "absorbance")

## Object: filter_spct [6 x 2]
## Wavelength range 424 to 3000 nm, step 34 to 900 nm
##
## # A tibble: 6 × 2
##   w.length          A
## *    <dbl>      <dbl>
## 1      424 0.07167585
## 2      458 0.06473968
```

```
## 3       529 0.05836933
## 4      1250 5.16600000
## # ... with 2 more rows
```

A `peaks` method is also defined in this package for numeric vectors, returning the values at the peaks.

```
peaks(sun.spct$s.e.irrad)

##  [1] 0.1822031 0.3295190 0.3129253 0.3352353
##  [5] 0.3380052 0.3207918 0.3453572 0.3758625
##  [9] 0.3707068 0.4491898 0.4393233 0.4969714
## [13] 0.4362110 0.3915446 0.4822105 0.4699886
## [17] 0.6497388 0.6615421 0.6742498 0.6761818
## [21] 0.6701269 0.6388873 0.7336607 0.7188581
## [25] 0.8204633 0.7984935 0.7711277 0.7665312
## [29] 0.7693357 0.7724634 0.7869773 0.7832759
## [33] 0.7728111 0.7899872 0.7701737 0.7466557
## [37] 0.7510876 0.7302733 0.7376088 0.7603297
## [41] 0.7429248 0.7272464 0.7178863 0.7117599
## [45] 0.6991590 0.6750915 0.6713390 0.6644287
## [49] 0.6853736 0.6256272 0.6372767 0.6614323
## [53] 0.6468436 0.6464099 0.6233510 0.6138918
## [57] 0.5819163 0.5995383 0.5977089 0.5879885
## [61] 0.5029201 0.5164799 0.5013394 0.5070675
## [65] 0.4345295 0.3855997 0.4676589 0.5006212
## [69] 0.5025733 0.5000141 0.5007593 0.4746771
## [73] 0.4716414 0.4680026 0.4213304 0.4236281
```

The parameters of the numeric specializations behave in the same way as in the case of spectra. As an example we use `span`.

```
peaks(sun.spct$s.e.irrad, span = NULL)

## [1] 0.8204633
```

8.13.1 Task: finding the location of peaks as an index into vectors with spectral data

Low level functions for finding peaks and valleys in numeric vectors are also defined in 'photobiology'. We recommend using the functions in the previous section whenever possible, as the use of the functions `find_peaks` and `get_peaks`, requires the user to validate the 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] 37 39 43 49 52 54
```

8.13.2 Task: Extracting peaks and valleys using vectors

Extracting peaks and valleys, is more conveniently done with functions `peaks` and `valleys`. Function `get_peaks` described in this section, is used internally in package 'photobiology'. It returns a `data.frame` object with columns `x`, `y` and `labels`, with `labels`. The other arguments are as described above.

```
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
```

With `span = NULL` only the tallest peak is returned.

```
head(with(sun.spct,
          get_peaks(w.length, s.e.irrad, span = NULL)))
```

```
##     x         y label
## 1 451 0.8204633   451
```

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
```

8.14 Task: Refining the location of peaks and valleys

The functions described in the previous sections locate the observation with the locally highest, or locally smallest y-value. This is in most cases *not* 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 observed 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 possible methods of peak-position refinement.

Some of the functions that could be fitted are suitable for both symmetrically shaped and asymmetrically shaped peaks, while others are suitable only for cases when one peak shoulder is the mirror image of the other. Frequently it is not necessary, or even advantageous, to fit the spectral data all the way down to the baseline, when the aim is to obtain the location of a peak rather than its area.

8.14.1 Bell-shaped function

In the example below, we will fit a non-linear function to the peak. In the first statement we locate the tallest peak as described above. In the second statement we refine its location. For this approach to work, the peaks should be well above the baseline, relatively narrow, and not very close to each other. We use the spectral irradiance emitted by an UV-C (*germicidal*) lamp as an example. We fit a function of the form:

$$I = d + a \cdot \mathrm{e}^{-0.5 \cdot ((\lambda - c)/b)^2} \tag{8.1}$$

```
peak <- peaks(lamps.mspct$germicidal, span = NULL)[1, "w.length"]
wl.tmp <- seq(round(peak - 5), round(peak + 5), length.out = 101)
neighbourhood.spct <- clip_wl(lamps.mspct$germicidal, range = wl.tmp)
fit <- nls(s.e.irrad ~ d + a1*exp(-0.5*((w.length-c1)/b1)^2),
           start=list(a1=3.1, b1=1, c1=peak, d=0),
           data=neighbourhood.spct)
fit

## Nonlinear regression model
##   model: s.e.irrad ~ d + a1 * exp(-0.5 * ((w.length - c1)/b1)^2)
##    data: neighbourhood.spct
##        a1         b1        c1          d
##   0.98742    0.49677 253.87095    0.01249
##  residual sum-of-squares: 0.005453
##
## Number of iterations to convergence: 8
## Achieved convergence tolerance: 1.143e-06

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

```
## [1] 253.8709
```

```
peak
```

```
## [1] 253.95
```

In this case the fitted position of the peak was close to the pixel with maximum reading as the wavelength resolution of the instrument is good. The result suggests a small wavelength calibration error for the spectrometer that can be calculated as:

```
signif(fit$m$getPars()[["c1"]], 7) - 253.6517 # nm
```

```
## [1] 0.2192
```

```
predicted <- predict(fit, data.frame(w.length = wl.tmp))
fitted_peak.spct <- source_spct(w.length  = wl.tmp,
                                s.e.irrad = predicted)
```

```
ggplot(data = fitted_peak.spct) +
  geom_line(data = fitted_peak.spct, colour = "blue") +
  geom_point(data = neighbourhood.spct, colour = "red", size = 3) +
  xlim(range(wl.tmp))
```
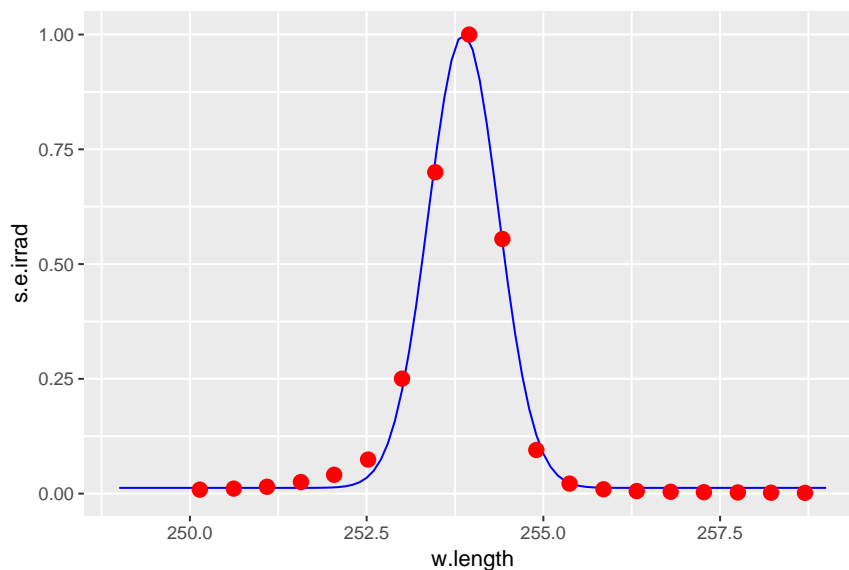

The fit above is not as good as one would like, both at the peak and at the short-wavelength shoulder. Of course other functions can be used, and additional examples of functions and peak shapes will be added here in later drafts.

8.14.2 Spline with a single node

I will try here the use of Prunty's equations.

8.14.3 Spline with three nodes

```
try(detach(package:photobiologyReflectors))
try(detach(package:photobiologyFilters))
try(detach(package:photobiologyLamps))
try(detach(package:ggspectra))
try(detach(package:ggplot2))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
```

9

Wavebands: simple summaries and features

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(ggspectra)
```

9.2 Task: Printing wavebands

A `print` method for `waveband` objects is defined in package 'photobiology', which in the example below is called implicitly.

```
VIS()

## VIS.ISO
## low (nm) 380
## high (nm) 760
## weighted none

CIE()

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

To print the internals (the underlying components) of the object, one can use method `unclass`.

```
unclass(VIS())

## $low
## [1] 380
```

```
##
## $high
## [1] 760
##
## $weight
## [1] "none"
##
## $SWF.e.fun
## NULL
##
## $SWF.q.fun
## NULL
##
## $SWF.norm
## NULL
##
## $norm
## NULL
##
## $hinges
## [1] 380 380 760 760
##
## $name
## [1] "VIS.ISO"
##
## $label
## [1] "VIS"
```

```r
unclass(CIE())
```

```
## $low
## [1] 250
##
## $high
## [1] 400
##
## $weight
## [1] "SWF"
##
## $SWF.e.fun
## function (w.length)
## {
##     CIE.energy <- numeric(length(w.length))
##     CIE.energy[w.length <= 298] <- 1
##     CIE.energy[(w.length > 298) & (w.length <= 328)] <- 10^(0.094 *
##         (298 - w.length[(w.length > 298) & (w.length <= 328)]))
##     CIE.energy[(w.length > 328) & (w.length <= 400)] <- 10^(0.015 *
##         (139 - w.length[(w.length > 328) & (w.length <= 400)]))
##     CIE.energy[w.length > 400] <- 0
```

```
##     return(CIE.energy)
## }
## <bytecode: 0x000000001a022b38>
## <environment: namespace:photobiologyWavebands>
##
## $SWF.q.fun
## function (w.length)
## {
##     SWF.e.fun(w.length) * SWF.norm/w.length
## }
## <bytecode: 0x00000000166f0648>
## <environment: 0x000000001b6436d8>
##
## $SWF.norm
## [1] 298
##
## $norm
## [1] 298
##
## $hinges
## [1] 250 250 298 328 400 400
##
## $name
## [1] "CIE98.298"
##
## $label
## [1] "CIE98"
```

9.3 Task: Summaries related to object properties

In the case of the `summary` method and their corresponding print method, specializations for waveband objects are provided.

```
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
```

9.4 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
```

9.5 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 *effective exposures*.

```
is_effective(vis.wb)
```

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

```
is_effective(cie.wb)
```

```
## [1] TRUE
```

9.6 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"
```
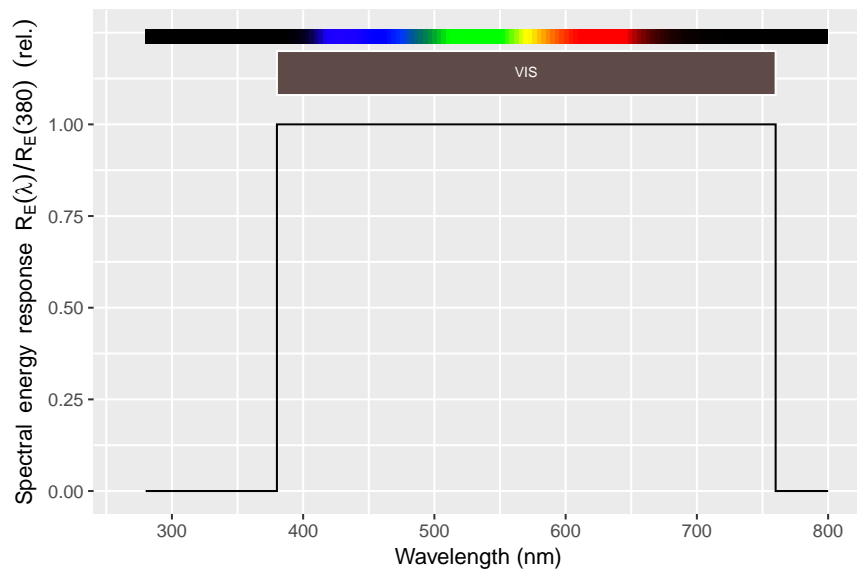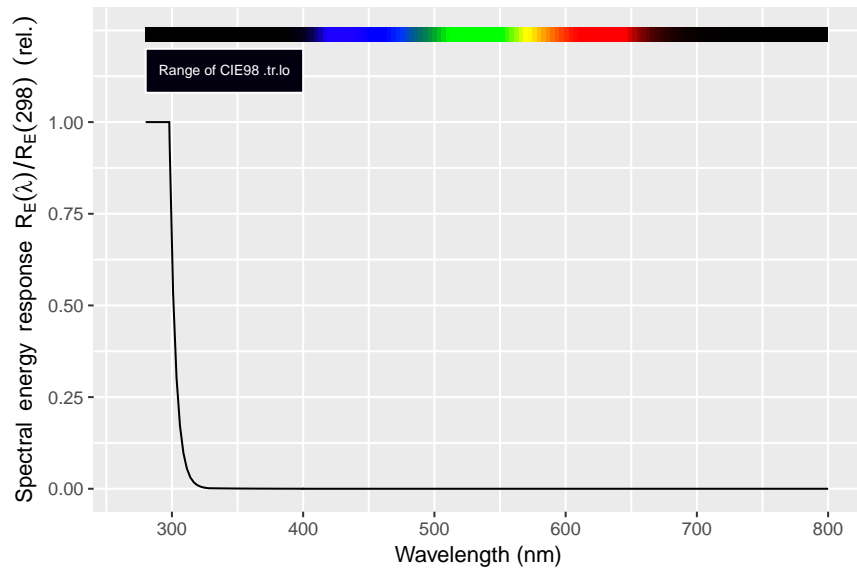
9.7 Task: Plotting a waveband

Method `plot` is defined for `waveband` objects, and can be used to visually check their properties. Plotting is discussed in detail in chapter 16.

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

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



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

10

Irradiance (not weighted)

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)
library(photobiologyLamps)
library(photobiologyLEDs)
library(lubridate)
```

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 119 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 113

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)

##   Red.CMF
## "#900000"
```

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
```

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
```

---

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

```
## 0.6445105
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"

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

##   UVB.none
##   1.337179
## 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
## 0.4819927
## 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 11 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) 280
## 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
##         68.4895
## 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
##     68.4895
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "energy irradiance total"
```

115

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.280.453.3
## low (nm) 280
## high (nm) 453
## weighted none
##
## $wb2
## range.453.3.626.7
## low (nm) 453
## high (nm) 627
## weighted none
##
## $wb3
## range.626.7.800
## low (nm) 627
## 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
```

---

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

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

```
## 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
##
## [[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

119

this example we will use example data for sunlight to calculate $E_{400\,\text{nm}<\square<700\,\text{nm}}$. 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
##             28.62872
## attr(,"time.unit")
## [1] "second"
```

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

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

## Warning in is.na(x): is.na() applied to non-(list or vector) of type 'NULL'

## 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    UVA.ISO
##  0.6445105 27.9842061
## 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
## 196.6343

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

##      PAR
## 196.6343
```

121

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
## 196.6343
```

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 $E_{400\,\mathrm{nm} < \square < 700\,\mathrm{nm}}$. 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 $\mathrm{\mu mol\,m^{-2}\,s^{-1}}$ 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
## 894.1352

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

##       PAR
## 894.1352
```

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

10.8 Task: irradiance for more than one waveband

As discussed above, it is possible to calculate simultaneously the irradiance 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
##   451.1083   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
## 451.1083

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
##   451.1083   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
## 1.675362
## 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
##   79.38159   49.26860   37.55207
## 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
## 79.38159 49.26860 37.55207
```

10.9 Task: calculate fluence for an irradiation event

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

$$F_{\lambda_1 < \lambda < \lambda_2} = \int_{\lambda_1}^{\lambda_2} E(\lambda) \times t \, \mathrm{d}\lambda \tag{10.3}$$

Let's assume that we want to calculate photosynthetically active radiation (PAR) energy fluence, 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 $F_{400\,\mathrm{nm}<\square<700\,\mathrm{nm}}$. The function used for this task when working with spectral objects is `e_fluence` returning energy irradiance. The "names" of the returned valued is set according to

the waveband used, and `sun.spct` is a `source_spct` object. The use of function `fluence` facilitates the calculation as it accepts the length of time of the exposure as a `lubridate::duration`, making it easy to enter the duration using different units, or even calculate the duration as the difference between two times. Of course, the spectral irradiance should be measured at the position where the material being exposed was located during irradiation. The following example is for a red fluorescent tube as sometimes used in seed germination experiments to study phytochrome-mediated responses.

Here we calculate the red light fluence corresponding to 5 min exposure to sunlight at the time the example spectral data was measured.

```
fluence(sun.spct,
        exposure.time = duration(5, "minutes"))

##      Total
## 80737.47
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "300s (~5 minutes)"
```

While the example spectral data for sunlight are expressed in absolute units, example spectral data for lamps in packages 'photobiologyLamps' and 'photobiologyLEDs' are normalized. To be able to calculate irradiance or fluence from these spectra they need first to the scaled. Scaling can be based on a different waveband. An example follows.

We assume that PAR photon irradiance is known for the experimental conditions, and is equal to 20 $\mu mol\,m^{-2}\,s^{-1}$. From this value and the normalized spectrum we can reconstruct actual spectral irradiance for the experiment.

```
red_lamp.spct <- fscale(lamps.mspct$philips.tld36w.15,
                        f = q_irrad,
                        w.band = PAR(),
                        target = 20e-6)
setScaled(red_lamp.spct, FALSE) # to avoid warnings
```

In the example above we used a waveband object defining a range of wavelengths, however, a waveband object describing a BSWF could also have been used in its place.

```
fluence(red_lamp.spct, w.band = list(Red("Smith10"), Red("Smith20"), Red("ISO")),
        exposure.time = duration(5, "minutes"))

##  Red.Smith10  Red.Smith20      Red.ISO
##     372.3421     608.6529    1039.0630
## attr(,"radiation.unit")
```

```
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "300s (~5 minutes)"
```

Above we calculate fluences for three different definitions of "red light" for the same experimental condition. Clearly, they are different.

Please see the sections 11.8 for additional details.

10.10 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 Morgan and Smith (1981), using photon irradiance[4]:

$$\zeta = \frac{Q_{655nm<\lambda<665nm}}{Q_{725nm<\lambda<735nm}} \tag{10.4}$$

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)
##                       1.266704
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

In the case of ratios whether spectral irradiance has been normalized or linearly scaled, or not, does not affect the result.

```
q_ratio(lamps.mspct$philips.tld36w.15, Red("Smith10"), Far_red("Smith10"))

##  Red.Smith10: FarRed.Smith10(q:q)
##                       100.6783
## attr(,"radiation.unit")
## [1] "q:q ratio"

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

---

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

```
##  Red.Smith10: FarRed.Smith10(q:q)
##                          100.6783
## 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()))
```

```
##  UVB.ISO: UV.ISO.tr.lo(q:q)
##                0.019369458
##      UVB.ISO: VIS.ISO(q:q)
##                0.001541437
## 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)
##              0.4150475                  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)
##                0.15087813
##    Blue.ISO: VIS.ISO(q:q)
##                0.13711571
##   Green.ISO: VIS.ISO(q:q)
##                0.20259364
##  Yellow.ISO: VIS.ISO(q:q)
##                0.06106049
##  Orange.ISO: VIS.ISO(q:q)
##                0.05545498
##     Red.ISO: VIS.ISO(q:q)
##                0.41504754
## 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] 1.266704
```

10.11 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)
##                       1.401142
## attr(,"radiation.unit")
## [1] "e:e ratio"
```

other examples in section 10.10 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] 1.401142
```

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

10.12 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 $\mathrm{mol\,J}^{-1}$. In this case
only one waveband definition is used to calculate the quotient:

$$\bar{q'} = \frac{Q_{\lambda_1 < \lambda < \lambda_2}}{E_{\lambda_1 < \lambda < \lambda_2}} \tag{10.5}$$

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 $\mu mol\,J^{-1}$ 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)
##     3.433902e-06     3.968591e-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] 4.547199e-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] 4.547199
```

10.13 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
##      69.69043       68.48950       58.45434
## 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
##      69.69043       68.48950       58.45434
## 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
##      69.69043       68.48950       58.45434
## 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
##      0.3544165      0.3483091      0.2972744
## 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
##      0.3544165      0.3483091      0.2972744
## 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
##      69.69043       68.48950       58.45434
## 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
##      69.69043       68.48950       58.45434
## 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
##      69.69043      68.48950      58.45434
```

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
##      0.3544165      0.3483091      0.2972744
```

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
##      35.44165      34.83091      29.72744
```

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
##      35.44165      34.83091      29.72744
```

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

```
## range.400.500 range.500.600
##      50.43455      49.56545
```

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`:

133

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

## range.400.700
##      196.6343
```

10.14 Task: calculate the spectral overlap between two light sources

The first case assumes that we use the same photon irradiance for both sources, in this case two different types of LEDs.

The first step is to scale both spectra so that the photon irradiance is the same, in this case equal to one.

```
green.spct <- fscale(leds.mspct$NHXRGB090_G,
                     range = c(400, 700),
                     f = "total",
                     unit.out = "photon")
blue.spct <- fscale(leds.mspct$NHXRGB090_B,
                     range = c(400, 700),
                     f = "total",
                     unit.out = "photon")

overlapBG.s.q.irrad <- ifelse(green.spct$s.q.irrad < blue.spct$s.q.irrad,
                              green.spct$s.q.irrad,
                              blue.spct$s.q.irrad)

overlapBG.spct <- source_spct(w.length = blue.spct$w.length,
                              s.q.irrad = overlapBG.s.q.irrad)

integrate_spct(overlapBG.spct) /
    (integrate_spct(green.spct) + integrate_spct(blue.spct)) * 1e2

## q.irrad
## 3.93787
```

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

11

Irradiance (weighted or effective)

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:

```
library(photobiology)
library(photobiologyWavebands)
library(photobiologyLamps)
library(lubridate)
```

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 10.6, 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. Package 'photobiology' and the suite make 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) (Green et al. 1974):

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

##  GEN.G.300.tr.lo
##       0.1028401
## 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 (Caldwell 1971). 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
##      0.02397171
## 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.

137

11.5 Task: define wavebands that use weighting functions

In sections 10.4 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
##      0.08181583
## 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
##      0.1028401        0.1473621
## 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
##      0.02397171       0.10284005
## 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
##    2.578989e-07
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

11.8 Task: calculate daily effective energy exposure

11.8.1 From spectral daily exposure

To calculate daily exposure values, if we have available spectral daily exposure (time-integrated spectral irradiance for a whole day) we need to apply the same code as used above, but using the 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 units 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"`).

11.8.2 From spectral irradiance

To calculate daily exposure values, from a known constant irradiance, we need to take into account the total length of exposure per day. This is equivalent to calculating fluence.

We start by scaling the normalized lamp spectrum to a known irradiance, in this case we assume, total UV irradiance from the unfiltered lamp is $1 \, \text{W m}^{-2}$.

```
qpanel_lamp.spct <- fscale(lamps.mspct$qpanel.uvb313,
                            f = e_irrad,
                            w.band = UV(),
                            target = 1)
setScaled(qpanel_lamp.spct, FALSE) # to avoid warnings
```

```
fluence(qpanel_lamp.spct, GEN.G(),
        exposure.time = duration(6, "hours"))

##  GEN.G.300
##    4872.507
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "21600s (~6 hours)"
```

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

```
fluence(qpanel_lamp.spct, GEN.G(),
        exposure.time = duration(6, "hours")) * 1e-3

##  GEN.G.300
##    4.872507
## attr(,"radiation.unit")
## [1] "energy fluence (J m-2)"
## attr(,"exposure.duration")
## [1] "21600s (~6 hours)"
```

by comparing these result to those for effective irradiances above, it can be seen that the `exposure.duration` supplied is copied to the result, allowing us to know the length of exposure used for the calculation.

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

12

Transmission and reflection

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:

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

12.2 Introduction

In this chapter we explain how to do calculations related to the description of absorption and reflection of UV and VIS radiation. Is important to be aware that in the same way as solar radiation has direct and diffuse components because of interactions in the atmosphere, the same phenomena affect light traversing a glass or plastic filter, or the tissues of an organism.

Depending on how transmittance is measured, it includes or not the effect of the reflections at the filter surfaces (called total and internal transmittance, respectively). If transmittance is measured with an integrating sphere, all scattered transmitted radiation is measured. However, if no integrating sphere is used, depending on the distance to the detector and how much scattering is present, the transmittance may be drastically underestimated.

An equivalent problem due to scattering occurs when measuring reflectance. Depending on the type of surface and material reflectance and total transmitatnce can strongly depend on the angle of incidence of the light beam. A special case are *interference* filters, as not only the transmittance itself but also the wavelengths transmitted depend on the angle of incidence of radiation.

If you are not familiar with the phenomena described above, and the different ways in which optical properties of filters and reflectors can be characterized, please, read the introductory chapter in Part A, before attempting the calculation described below.

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".

```
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. In the 'r4photobiology suite' absorbance is always calculate according to equation 12.2 using $\log_{10}$. User input during construction of `filter_spct` objects from spectral absorbance data is also assumed to be expressed in this same way.

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 'photobiology' defines generic functions and specialized functions, that can be used on numeric vectors and on `filter_spct` objects. The 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 is shown in the next section.

12.4 Task: spectral absorbance from spectral transmittance

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

```
my_gg400.spct <- filters.mspct$GG400
T2A(my_gg400.spct)

## Object: filter_spct [1,001 x 3]
## Wavelength range 200 to 5200 nm, step 1 to 50 nm
##
## # A tibble: 1,001 × 3
##   w.length   Tfr     A
##      <dbl> <dbl> <dbl>
## 1     200 1e-05     5
## 2     201 1e-05     5
## 3     202 1e-05     5
## 4     203 1e-05     5
## # ... with 997 more rows

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

As in addition to the `T2A` method for `filter_spct` there is a `T2A` method available for numeric vectors.

```
my_gg400.spct <- filters.mspct$GG400
my_gg400.spct$A <- T2A(my_gg400.spct$Tfr)
my_gg400.spct

## Object: filter_spct [1,001 x 3]
## Wavelength range 200 to 5200 nm, step 1 to 50 nm
##
## # A tibble: 1,001 × 3
##   w.length   Tfr     A
##      <dbl> <dbl> <dbl>
## 1     200 1e-05     5
## 2     201 1e-05     5
## 3     202 1e-05     5
## 4     203 1e-05     5
## # ... with 997 more rows
```

or even on single numeric values:

```
T2A(0.001)

## [1] 3
```

12.5 Task: spectral transmittance from spectral absorbance

Please, see section 12.4 for more details in the description of the method T2A which does the opposite conversion than the method A2T needed for this task, but which works similarly.

```
A2T(a.gg400.spct)

## Object: filter_spct [1,001 x 3]
## Wavelength range 200 to 5200 nm, step 1 to 50 nm
##
## # A tibble: 1,001 × 3
##   w.length     A   Tfr
##      <dbl> <dbl> <dbl>
## 1     200     5 1e-05
## 2     201     5 1e-05
## 3     202     5 1e-05
## 4     203     5 1e-05
## # ... with 997 more rows

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

## Object: filter_spct [1,001 x 2]
## Wavelength range 200 to 5200 nm, step 1 to 50 nm
##
## # A tibble: 1,001 × 2
##   w.length   Tfr
##      <dbl> <dbl>
## 1     200 1e-05
## 2     201 1e-05
## 3     202 1e-05
## 4     203 1e-05
## # ... with 997 more rows
```

12.6 Task: transmitted spectrum from spectral transmittance and spectral irradiance

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

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

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

```r
class(filters.mspct$GG400)
```

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

```r
filtered_sun.spct <- sun.spct * filters.mspct$GG400
class(filtered_sun.spct)
```

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

```r
head(filtered_sun.spct)
```

```
## Object: source_spct [6 x 2]
## Wavelength range 280 to 282.76923 nm, step 0.07692308 to 0.9230769 nm
## Time unit 1s
##
## # A tibble: 6 × 2
##   w.length s.e.irrad
## *    <dbl>     <dbl>
## 1 280.0000         0
## 2 280.9231         0
## 3 281.0000         0
## 4 281.8462         0
## # ... with 2 more rows
```

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

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

```
##  UV.ISO.tr.lo
##      86.49506
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

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

```
##  UV.ISO.tr.lo
##      2.623568
## attr(,"time.unit")
## [1] "second"
```

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

q_irrad(sun.spct * filters.mspct$GG400, UV()) * 1e6

##  UV.ISO.tr.lo
##      2.623568
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

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

##    Total
## 1135.486
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

12.7 Task: reflected spectrum from spectral reflectance and spectral irradiance

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

If in the examples from the previous section one replaces the transmittance spectrum by a reflection spectrum, one obtains the spectrum of the reflected radiation given a certain spectrum of incident light.

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

12.9 Task: combined spectral transmittance of two or more filters

12.9.1 Ignoring reflectance

12.9.2 Considering reflectance

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

12.11 Task: simulating the spectral irradiance under a LED luminaire

To conclude the chapter, we show how combining different operations described in this and earlier chapters one can do interesting simulations of estimations.

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 individual light source. If we want then to add the effect of a filter we must multiply by the filter transmittance. Bellow we simulate a (bad?) light source for plant cultivation.

We first would need to scale the normalized spectral irradiance for each LED based on their relative relative absolute output. For simplicity, we assume they will powered so that they will output equal amounts of energy as visible radiation. Even under this assumption, as the width of emission peaks depend on the type of LED, we need to scale these data. We will scale them using defaults.

```
red_led.spct <- fscale(leds.mspct$NHXRGB090_R)
blue_led.spct <- fscale(leds.mspct$NHXRGB090_B)
```

We next assume that we use five red LEDs for each blue LED, and protect them with a sheet of clear Plexiglas.

```
my_luminaire.spct <-
  (red_led.spct * 5 + blue_led.spct) *
      filters.mspct$Clear_0A000_XT
class(my_luminaire.spct)

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

We now assume that the PAR photon irradiance on a horizontal plane below the luminaire is 250 $\mu mol\, m^{-2}\, s^{-1}$.

```
my_luminaire.spct <- fscale(my_luminaire.spct,
                            f = q_irrad,
                            w.band = PAR(),
                            target = 250e-6)
setScaled(red_lamp.spct, FALSE) # to avoid warnings
class(my_luminaire.spct)

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

Finally we calculate photon irradiances and photon ratios for some bands of interest for plant responses.

```
q_ratio(my_luminaire.spct,
          list(Red(), Blue(), Green()), PAR())

##    Red.ISO: PAR(q:q)    Blue.ISO: PAR(q:q)
##        0.852914493            0.112481864
##  Green.ISO: PAR(q:q)
##        0.002995064
## attr(,"radiation.unit")
## [1] "q:q ratio"

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

## Warning in irrad_spct(spct, w.band = w.band, unit.out = "photon", quantity = quantity,
: Summarized spectral data have been rescaled

##          PAR      Red.ISO      Blue.ISO    Green.ISO
## 249.8794445 213.1257997   28.1069057    0.7484049
## attr(,"time.unit")
## [1] "second"
## attr(,"radiation.unit")
## [1] "photon irradiance total"
```

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

13

Astronomy

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)
library(ggplot2)
library(ggmap)

cache <- FALSE
```

13.2 Introduction

13.2.1 Time coordinates

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"

ymd("2014-03-20")

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

ymd("14-03-20")

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

```
ymd("2014-3-20")

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

ymd("2014/3/20")

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

dmy("20.03.2014")

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

dmy("20032014")

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

mdy("03202014")

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

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.

⚠ Times and dates are stored in R objects in Universal Time Coordinates (UTC), but the time zone (`tz`) used for input and output can vary widely. The UTC 'time zone' corresponds to the Greenwich meridian without any shifts due to daylight-saving time throughout the year. The time offset between local time and UTC can vary throughout the year due to day-light saving times—winter vs. summer time—, or accross years through changes in legislation—e.g. date when when day-light saving time starts and ends, or changes in the geographical borders between time zones, or a country adopting a different time zone. Different names and abbreviations are used, for the same time zones. Examples of time zones are EET or Eastern European Time, CET or Central European Time. Functions in package 'lubridate' are of help for getting around these conversions but one should be always very careful as many functions use as default the 'locale' and 'TZ' settings queried from the operating system as defaults. This means that data saved at a different geographic location or using a computer with wrong settings, or saving data using a remote (e.g. cloud) server located in a different time zone, can all result in misinterpretation of time data. Furthermore, the same R script may yield different output when run in a different time zone, unless time zone is explicitly

passed as an argument. This problem does not only affect times, but also dates.

```r
ymd("20140320") # no time zone set, Date object

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

ymd("20140320", tz = "Europe/Helsinki") # Eastern Europe

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

ymd("20140320", tz = "America/Buenos_Aires") # Argentina

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

ymd("20140320", tz = "Asia/Tokyo") # Japan

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

# time set according to time zone
ymd_hm("20140320 00:00") # assumes local tz, POSIXct object

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

ymd_hm("20140320 00:00", tz = "Europe/Helsinki") # central Europe

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

ymd_hm("20140320 00:00", tz = "America/Buenos_Aires") # Argentina

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

ymd_hm("20140320 00:00", tz = "Asia/Tokyo") # Central Australia

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

In the example above, the data are read using the supplied time zone, and printed back in a the same time zone. An example follows whose result depends on the local time zone where this file is run.

```r
# this is just a date object, which ignores the time zone
ymd("20140320") # assumes local tz, POSIXct object

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

with_tz(ymd("20140320"), tzone = "America/Buenos_Aires")

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

# this is a POSIXct date + time object
ymd_hm("20140320 00:00") # assumes local tz, POSIXct object
```

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

with_tz(ymd_hm("20140320 00:00"), tzone = "America/Buenos_Aires")

## [1] "2014-03-19 21:00:00 ART"

# current system time zone
Sys.timezone()

## [1] "UTC"
```

> ⚠️ We should make explicit the time zones used for both input and output to avoid any ambiguities.

```
with_tz(ymd("20140320", tz = "America/Buenos_Aires"), tzone = "Europe/London")

## [1] "2014-03-20 03:00:00 GMT"

with_tz(ymd("20140320", tz = "Asia/Tokyo"), tzone = "America/Buenos_Aires")

## [1] "2014-03-19 12:00:00 ART"

with_tz(ymd_hm("20140320 00:00", tz = "America/Buenos_Aires"), tzone = "Asia/Tokyo")

## [1] "2014-03-20 12:00:00 JST"

with_tz(ymd_hm("20140320 00:00", tz = "Asia/Tokyo"), tzone = "America/Buenos_Aires")

## [1] "2014-03-19 12:00:00 ART"
```

13.2.2 Geographic coordinates

For astronomical calculations we also need as argument geographical coordinates. One can supply latitude and longitude values, acquired with a GPS instrument or read from a map. However, when the location is searchable through Google Maps, it is also possible to obtain the coordinates on-the-fly. A query can be run from within R using packages 'RgoogleMaps', or 'ggmap', as done here—of course this requires internet access. When inputting 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). Latitudes N are given by positive numbers and and latitudes S by negative numbers. Longitudes W of Greenwich are given as positive numbers and longitudes

E of Greenwich as negative numbers. The latitude and longitude must be in variables named `lat` and `lon` in a data frame. A third variable, `address`, is recognized and copied unchanged to the output. The address is optionally included in the search result from Google maps. The address returned is not necessarily the searched term, and is useful when using partial addresses as search terms, as the hit, may not be the intended location.

```
geocode("Helsinki")

##       lon      lat
## 1 24.93838 60.16986

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

##       lon     lat
## 1 25.01673 60.2253

geocode("Viikinkaari 1, 00790 Helsinki, Finland", output = "latlona")

##       lon     lat
## 1 25.01673 60.2253
##                                address
## 1 viikinkaari 1, 00790 helsinki, finland
```

13.2.3 Algorithms and peculiarities

The code uses the well known algorithms by Meeus (Meeus 1998) implemented in the R language as a translation of code used in NOAA's calculation worksheets. We have compared the results from our functions against the NOAA on-line calculator. Julian day calculations do not use the same algorithms as NOAA, but instead, use the `julian` function from base R and time zone and daylight saving times based on package 'lubridate'. Handling of time zones and daylight saving times is very tricky for past dates, as country boundaries have changed as well as legislation over the years and centuries. The further back in history one goes, the less reliable calculations expressed in local time become. Of course UTC times, and local solar time values can be trusted for any epoch for which the algorithm is valid. The algorithm is precise enough to ensure valid results into the far past, but the calculation of julian dates may be less or more precise in R than other implementations. Agreement with the NOAA calculator is good as far back as we have tested, dates about 1000 years before now.

13.3 Task: calculating the length of the photoperiod

Functions `day_length` and `night_length` have the same parameter signature. They are vectorized for the `date` parameter and for `geocode` with multiple rows, and in the this last case also `tz` is vectorized.

Calculating the length of the current day is easy. We use the Greenwich meridian, and 60 degrees in the Northern and Southern hemispheres. We use function `today` from package 'lubridate'.

```r
day_length(today(), geocode = data.frame(lat = 60, lon = 0))
```

```
## [1] 5.93108
```

```r
day_length(today(), geocode = data.frame(lat = -60, lon = 0))
```

```
## [1] 18.79534
```

In the case of daylength calculations the longitude is almost irrelevant, and defaults to zero degrees. As the date defaults to 'today' the code above can be simplified.

```r
day_length(geocode = data.frame(lat = -60, lon = 0))
```

```
## [1] 18.79534
```

Function `geocode` from package 'ggmap' returns suitable values in a `data.frame` based on search term(s). The returned value is a data frame, which can be passed as argument for parameter `geocode`. The use of Google search has some restrictions that you will need to check before any intense or commercial use of their service.

```r
my.city <- geocode('Helsinki', output = "latlona")
my.city
```

```
##        lon      lat          address
## 1 24.93838 60.16986 helsinki, finland
```

We can calculate the photoperiod for the current day as

```r
day_length(geocode = my.city)
```

```
## [1] 5.876761
```

Or if we give a date explicitly using functions from package 'lubridate'.

```r
day_length(ymd("2015-06-09"),
           geocode = my.city)
```

```
## [1] 18.71578
```

```
day_length(dmy("9.6.2015"),
           geocode = my.city)
```

```
## [1] 18.71578
```

Or for several consecutive days by supplying a vector of dates as argument.

```
my.dates <- seq(ymd("2015-01-01"), ymd("2015-12-31"), by = "month")
day_length(my.dates, geocode = my.city)
```

```
##  [1]  5.993966  7.969853 10.463575 13.311352
##  [5] 16.022538 18.354633 18.775172 16.873553
##  [9] 14.170592 11.458596  8.680355  6.419090
```

Or to get the results as a data frame.

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

```
photoperiods.df <-
  data.frame(date = my.dates,
             photoperiod = day_length(my.dates, geocode = my.city))
```

The six lines at the top of the output are

```
head(photoperiods.df, 6)
```

```
##         date photoperiod
## 1 2015-01-01    5.993966
## 2 2015-02-01    7.969853
## 3 2015-03-01   10.463575
## 4 2015-04-01   13.311352
## 5 2015-05-01   16.022538
## 6 2015-06-01   18.354633
```

The complementary function `night_length` gives

```
night_length(ymd("2015-06-09"),
             geocode = my.city)
```

```
## [1] 5.284221
```

Using the functions as described above, 'measures' the photoperiod according to a boundary between day and night at a solar elevation angle equal to zero. An additional parameter of the functions, described in section 13.4, allows setting this twilight angle in degrees or by name according to different twilight angle definitions.

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

Functions `sunrise_time`, `sunset_time`, and `noon_time` have the same parameter signature—take the same arguments—but the values they return differ.

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.

⚠️ 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 that in use at the location and date of interest. Longitude is not enough, as time zones depend on country boundaries, and in some cases administrative or other boundaries within countries. The period of the year when daylight saving time has been in effect depends on legislation in effect at the date of interest, as well as on the administrative boundaries at the time. It is best to use UTC whenever considering times in the past or future.

```r
sunrise_time(today(tzone = "UTC"),
             geocode = data.frame(lat = 60, lon = 0),
             tz = "UTC")

## [1] "2016-12-27 09:03:29 UTC"

sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki")

## [1] "2016-12-27 09:23:29 EET"

sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "UTC")

## [1] "2016-12-27 07:23:29 UTC"
```

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

```r
sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
```

```
              tz = "Europe/Helsinki",
              twilight = "civil")
```

```
## [1] "2016-12-27 08:26:05 EET"
```

```
sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki",
             twilight = -10)
```

```
## [1] "2016-12-27 07:47:32 EET"
```

```
sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki",
             twilight = +12)
```

```
## [1] NA
```

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"`. The default for `date` is the current day in time zone UTC. Using defaults the code is simpler, but this is not a good approach for scripts.

```
sunrise_time(geocode = data.frame(lat = 60, lon = 0))
```

```
## [1] "2016-12-27 09:03:29 UTC"
```

By default a POSIXct object is returned. This object described a date and time in UTC. It is portable and safe. However, if one needs the time of the day as numeric value, one can supply additional arguments.

```
sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki",
             unit.out = "days")
```

```
## [1] 0.3913087
```

```
sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki",
             unit.out = "hours")
```

```
## [1] 9.391408
```

```
sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki",
             unit.out =  "minutes")
```

```
## [1] 563.4845

sunrise_time(today(tzone = "Europe/Helsinki"),
             geocode = data.frame(lat = 60, lon = 25),
             tz = "Europe/Helsinki",
             unit.out = "seconds")

## [1] 33809.07
```

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 <- sunrise_time(my.dates,
                                   geocode = my.city,
                                   tz = "Europe/Helsinki")
```

The six lines at the top of the output are

```
head(time_at_sunrise.df, 6)

## [1] "2015-01-01 09:24:00 EET"
## [2] "2015-02-01 08:34:44 EET"
## [3] "2015-03-01 07:18:40 EET"
## [4] "2015-04-01 06:44:45 EEST"
## [5] "2015-05-01 05:16:40 EEST"
## [6] "2015-06-01 04:07:26 EEST"
```

The complementary functions `sunset_time` and `noon_time` take exactly the same arguments as `sunrise_time`, but `noon_time` ignores any argument supplied for `twilight`.

```
sunrise_time(today(tzone = "UTC"),
             geocode = data.frame(lat = 60, lon = 0),
             tz = "UTC")

## [1] "2016-12-27 09:03:29 UTC"

sunset_time(today(tzone = "UTC"),
            geocode = data.frame(lat = 60, lon = 0),
            tz = "UTC")

## [1] "2016-12-27 14:59:20 UTC"

noon_time(today(tzone = "UTC"),
          geocode = data.frame(lat = 60, lon = 0),
          tz = "UTC")

## [1] "2016-12-27 12:01:25 UTC"
```

Box 13.1: Astronomical calculations of the sun position

Function `sun_angles`, which is a modified version of function `sunAngle` from package 'ode' is used to calculate the elevation of the sun at given time and geographical coordinates. To calculate the times corresponding to a given solar elevation, the algorithm first searches for the local solar noon by finding the maximal solar elevation, and then searches for sunrise in the first half of the day and for sunset in the second half local solar day. Sunset and sunrise are by default based on a solar elevation angle equal to zero. However, an argument can be passed to parameter `twilight` to set the angle according to different conventions—civil, nautical, astronomical—, by giving one or two values for the angles. Day and night durations are calculated as time durations between sunrise and sunset (the default) or dusk and dawn or some other angle. The algorithm currently used is valid for years 1950–2050.

Function `day_night` returns a list with the different times with a single call. As other functions described in this chapter, `day_night` is vectorised for the `date` parameter.

```
day_night(today(tzone = "UTC"),
          geocode = data.frame(lat = 60, lon = 0),
          tz = "UTC")

##         day  tz twilight.rise twilight.set
## 1 2016-12-27 UTC          none         none
##   longitude latitude address   sunrise      noon
## 1         0       60    <NA> 9.234849 12.02361
##     sunset daylength nightlength
## 1 14.81238  5.577531    18.42247
```

As in earlier examples, we use `geocode` to obtain the latitude and longitude of cities, although for precise calculations for large cities using a street address or at least a postal code in addition to the name of the city is preferable. The calculations below are 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.

```
geocode_BA <- geocode("Buenos Aires")
day_night(ymd("2013-12-21"),
          geocode = geocode_BA,
          tz = "America/Buenos_Aires")

##         day                  tz twilight.rise
## 1 2013-12-21 America/Buenos_Aires          none
##   twilight.set longitude  latitude address
## 1         none -58.38156 -34.60368    <NA>
##    sunrise     noon   sunset daylength
```

161

```
## 1 5.704087 12.86422 20.02434  14.32026
##   nightlength
## 1    9.679743
```

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

```
day_night(ymd("2013-12-21"),
          geocode = geocode_BA,
          tz = "America/Argentina/Buenos_Aires",
          unit.out = "hour")

##        day                          tz
## 1 2013-12-21 America/Argentina/Buenos_Aires
##   twilight.rise twilight.set longitude  latitude
## 1         none         none -58.38156 -34.60368
##   address sunrise     noon   sunset daylength
## 1    <NA> 5.704087 12.86422 20.02434  14.32026
##   nightlength
## 1    9.679743
```

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

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

##       lon      lat
## 1 24.93838 60.16986
```

```
day_length(ymd("2013-09-21"),
           geocode = geocode_He)

## [1] 12.32003
```

```
day_length(ymd("2013-09-21"),
           geocode = geocode_He,
           twilight = "civil")

## [1] 13.71643
```

```
day_length(ymd("2013-09-21"),
           geocode = geocode_He,
           twilight = "nautical")

## [1] 15.40104
```

```
day_length(ymd("2013-09-21"),
           geocode = geocode_He,
           twilight = "astronomical")

## [1] 17.24508
```

162

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"),
          geocode = geocode_He,
          twilight = c(20, 0))

## [1] 9.218363
```

13.5 Task: calculating the position of the sun

Function `sun_angles` not only returns solar elevation, it returns 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 by default for the output can vary as described in page 152, so it is more reliable specify the time coordinates used for the output with parameter `tz`, using arguments valid for pacakge `lubridate` — which is used internally by package 'photobiology'.

```
geocode_Jo <- geocode("80100 Joensuu, Finland")
geocode_Jo

##        lon      lat
## 1 29.77829 62.62812

my_time <- ymd_hms("2014-05-29 18:00:00", tz="EET")
sun_angles(my_time,
          geocode = geocode_Jo)

##                  time  tz solartime longitude
## 1 2014-05-29 18:00:00 EET  17:01:39  29.77829
##   latitude address  azimuth elevation
## 1 62.62812    <NA> 267.5987  25.85347
```

If we do not supply a time as argument, `sun_angles` calculates the current position of the sun—or at the time indicated by the computer. In this case giving the position of the sun in the sky of Joensuu at the time this .PDF file was generated.

```
sun_angles(geocode = geocode_Jo)

##                  time  tz solartime longitude
## 1 2016-12-27 21:29:26 UTC  23:26:56  29.77829
##   latitude address  azimuth elevation
## 1 62.62812    <NA> 348.0848 -50.23391
```

163

13.6 Task: plotting sun elevation through a day

Function `sun_angles` described above is vectorised, 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 only solar elevation, plotted for Helsinki through the course of 23 June 2014. We first create a vector of times, using `seq` which can be used with dates in addition to numbers. In the case of dates the argument passed to parameter `by` is specified as a string.

```
opts_chunk$set(opts_fig_wide_full)
```

```
hours <- seq(from = ymd("2014-06-23", tz = "EET"),
             by = "3 min",
             length.out = 24 * 20)
angles_He <- sun_angles(hours,
                        geocode = geocode_He)
head(angles_He)

##                    time  tz solartime longitude
## 1 2014-06-23 00:00:00 EET   22.62694  24.93838
## 2 2014-06-23 00:03:00 EET   22.67694  24.93838
## 3 2014-06-23 00:06:00 EET   22.72693  24.93838
## 4 2014-06-23 00:09:00 EET   22.77692  24.93838
## 5 2014-06-23 00:12:00 EET   22.82691  24.93838
## 6 2014-06-23 00:15:00 EET   22.87691  24.93838
##   latitude address   azimuth elevation
## 1 60.16986    <NA> 341.1021 -4.725760
## 2 60.16986    <NA> 341.7823 -4.844501
## 3 60.16986    <NA> 342.4632 -4.959033
## 4 60.16986    <NA> 343.1450 -5.069336
## 5 60.16986    <NA> 343.8276 -5.175388
## 6 60.16986    <NA> 344.5110 -5.277170
```

We also create an auxiliary data frame with data for plotting and labeling the different conventional definitions of *twilight*.

```
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 of solar elevations through a day, using the data frames created above.

```
ggplot(angles_He,
       aes(x = time, 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")
```
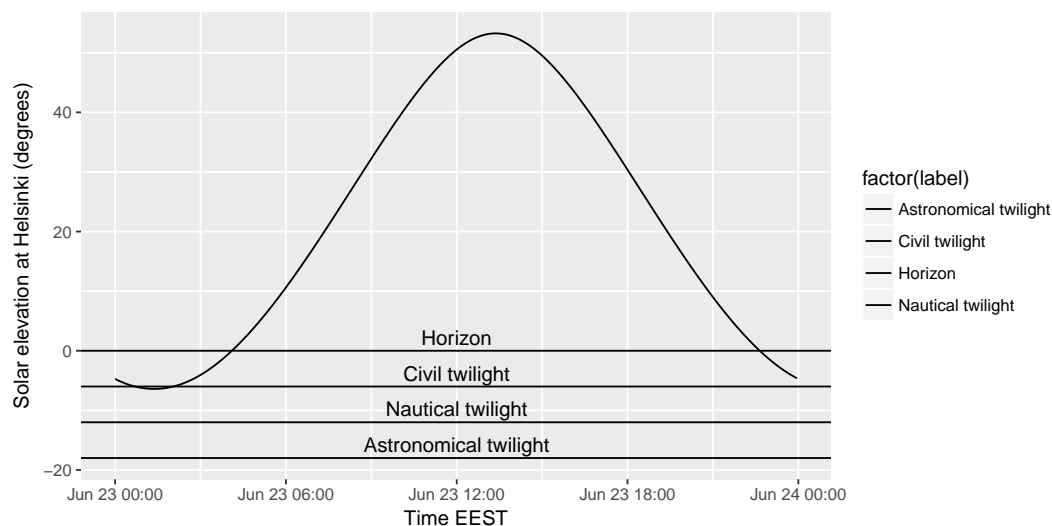
We draw a plot of solar azimuths through a day, using the data frames created above.

```
ggplot(angles_He,
       aes(x = time, y = azimuth)) +
  geom_line() +
  scale_y_continuous(breaks = c(0,90,180,270,360),
                     limits = c(-20,380)) +
  labs(y = "Solar azimuth at Helsinki (degrees)",
       x = "Time EEST")
```
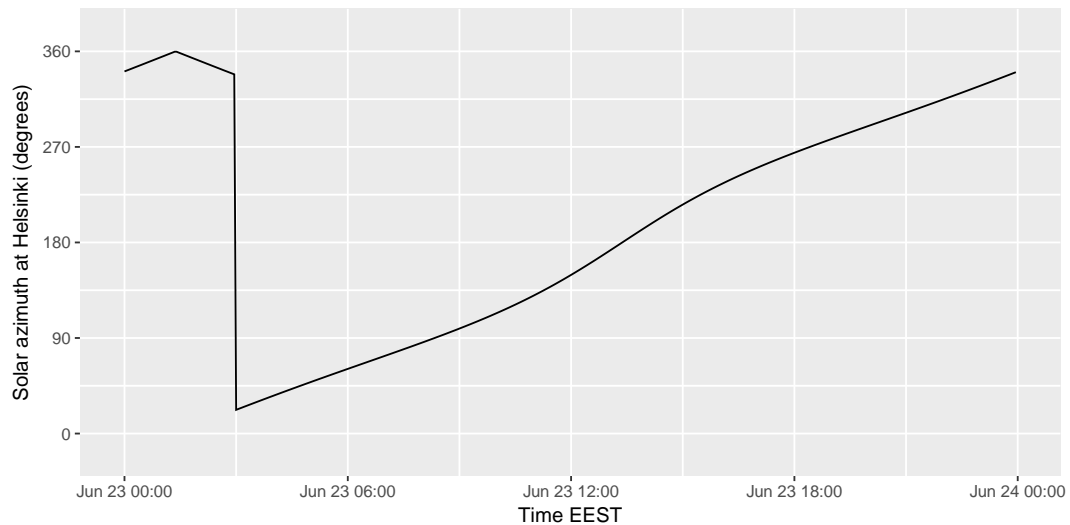
13.7 Task: plotting day or night length through the year

We use function `day_night()` to calculate simulataneously several values, although this function is slower it returns a data frame that makes coding easier. We create a `days` vector, and create a vector of geocodes by searching city names. We take advantage of vectorization to obtain a dataframe. We add a factor with the names of the locations.

```
days <- seq(from = ymd("2016-01-01"), to = ymd("2016-12-31"), by = 7)
cities <- c("Ivalo, Finland", "Helsinki, Finland", "Athens, Greece")
geocodes <- geocode(cities, output = "latlona")

## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Ivalo,%20Finland&sens
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Helsinki,%20Finland&s
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Athens,%20Greece&sens

times <-
   day_night(date = days,
             geocode = geocodes,
             tz = "EET",
             unit.out = "hour")
```

We can list the variables in the data frame.

```
names(times)

##  [1] "day"           "tz"
##  [3] "twilight.rise" "twilight.set"
##  [5] "longitude"     "latitude"
```

```
## [7] "address"       "sunrise"
## [9] "noon"          "sunset"
## [11] "daylength"    "nightlength"
```

We plot day length using 'ggplot2'. The last two lines of code are optional. We use `geom_point` to highlight that the calcualtions have not been done for each day. Ivalo is above the northern polar circle, so in winter nights last for 24 h and in summer days last for 24 h.

```
ggplot(times,
       aes(x = day, y = daylength, colour=address)) +
  geom_line() +
  scale_y_continuous(breaks=c(0,12,24), limits=c(0,24)) +
  labs(x = "Date", y = "Day length (h)", colour="Location")
```
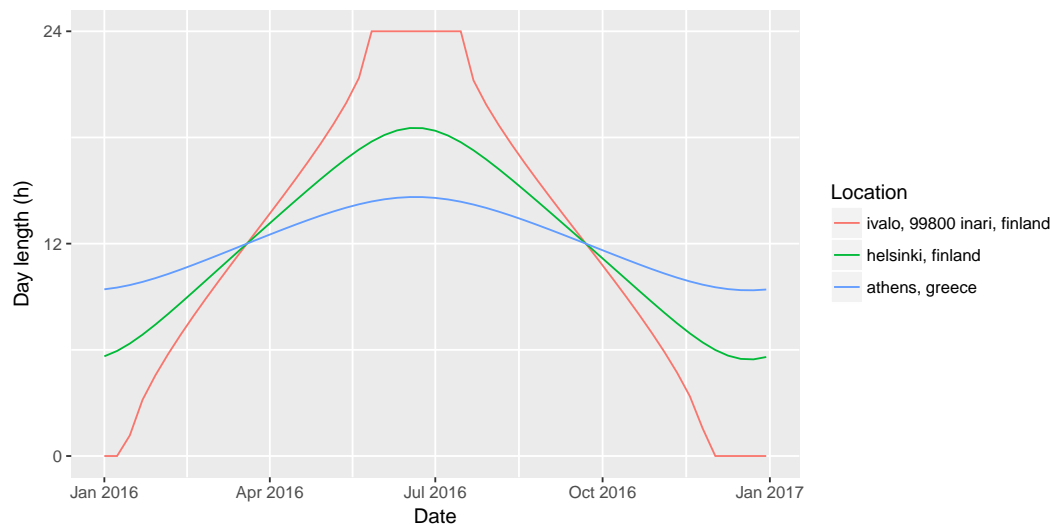

For plotting `nightlength` we just need to map it to the y aesthetic instead pf `daylength` and its corresponding axis label.

13.8 Task: plotting local time at sunrise

We reuse the data frame from the previous section. The breaks in the lines are the result of the changes between winter and summer time coordinates in the EET zone. The points indicate the calculated values, once per week.

```
ggplot(times,
       aes(x = day, y = sunrise, colour=address)) +
  geom_line() + geom_point() +
  scale_y_continuous(breaks=c(0,6,12), limits=c(0,14)) +
  labs(x = "Date", y = "Local time at sunrise (h)", colour="Location")
```

```
## Warning: Removed 7 rows containing missing values (geom_path).
## Warning: Removed 15 rows containing missing values
## (geom_point).
```
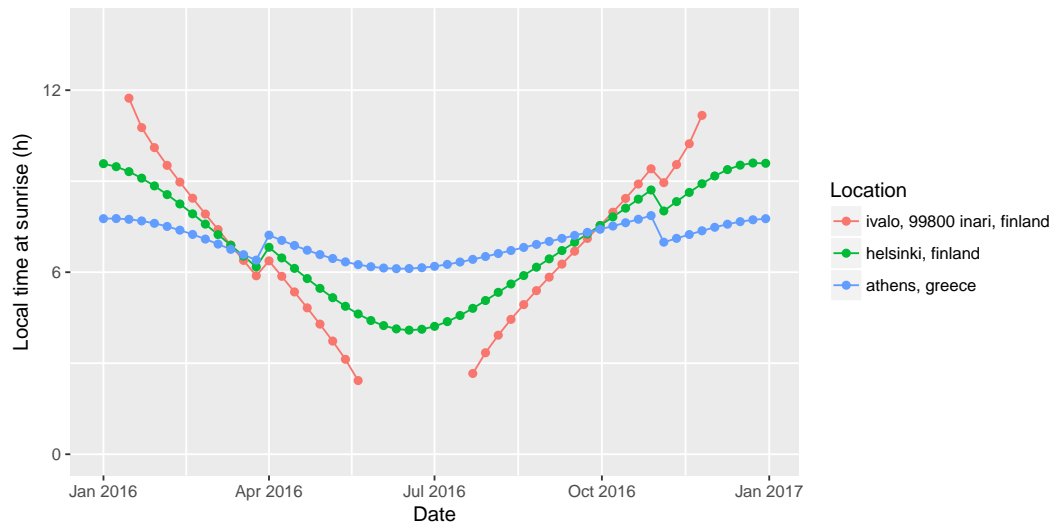

By replacing the mapping of `sunrise_time` by `sunset_time` to y in the code above and editing the axis label of the plot one can produce a similar plot of sunset times.

In general scale limts must be selected with care when dealing with civil time co-ordinates as it can be shifted by a few hours from solar time at some geographic locations.

13.9 Task: plotting solar time at sunrise

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

14

Colour

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:

```
library(colorSpec)
library(photobiology)
library(photobiologyWavebands)
library(photobiologyInOut)
library(ggspectra)
```

14.2 Introduction

Although package 'photobiology' defines some functions for color related calculations, special R packages exist for color calculations. Package 'colorSpec' is very comprehensive for human vision, illumination, and also covers cameras and display screens. We will use this package in some of the examples in this chapter.

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 trichromic 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.

14.3 Task: calculating an RGB colour from a single wavelength

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

```
color(550) # green

## [1] "#00FF00"

color(630) # red

## [1] "#FF0000"

color(380) # UVA

## [1] "#000000"

color(750) # far red

## [1] "#000000"

color(c(550, 630, 380, 750)) # vectorized
## [1] "#00FF00" "#FF0000" "#000000" "#000000"
```

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

Method `color` can be used also in this case, but we need to supply the range of wavelengths as a `waveband` object. The details of how to construct a `waveband` object are described in This function assumes a flat energy spectral irradiance curve within the range. Some examples: Examples for wavelength ranges:

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

## range.400.700.CMF
##         "#735B57"

color(PAR())

##   PAR.CMF
## "#735B57"

color(Yellow())

## Yellow.CMF
##  "#FFBB00"
```

14.5 Task: calculating an RGB colour for spectrum

Method `color` is also defined for spectral objects.

Examples for spectra, in this case the solar spectrum:

```
color(sun.spct)

## source CMF
##  "#544F4B"

color(sun.spct, type = "CMF") # colour matching function

## source CMF
##  "#544F4B"

color(sun.spct, type = "CC")  # colour coordinates

## source CC
## "#544F4B"

color(sun.spct * yellow_gel.spct)

## source CMF
##  "#946000"
```

14.6 Standard CIE illuminants

Package 'colorSpec' defines all the standard CIE illuminants, so if needed for calculations with methods and operators from package 'photobiology' they need to be converted. The conversion can easily be done with functions `colorSpec2spct()` or `colorSpec2mspct()`, the first returning a single spectral object, possibly containing

171

several spectra in long form, and the second returns a collection of spectra containing one or more member spectra. Some examples of single spectra.

```
A.1nm.spct <- colorSpec2spct(A.1nm)
A.1nm.spct

## Object: source_spct [421 x 3]
## Wavelength range 360 to 780 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 421 × 3
##   w.length  s.e.irrad spct.idx
##      <dbl>      <dbl>   <fctr>
## 1     360 0.06144620        A
## 2     361 0.06299550        A
## 3     362 0.06457250        A
## 4     363 0.06617743        A
## # ... with 417 more rows
```

```
plot(A.1nm.spct)
```
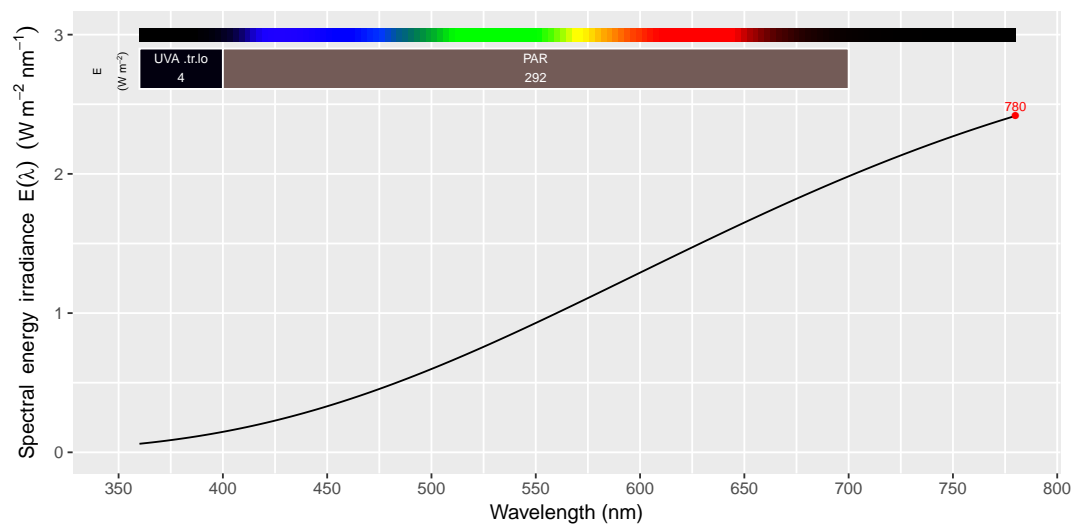


```
B.5nm.spct <- colorSpec2spct(B.5nm)
```

```
D50.5nm.spct <- colorSpec2spct(D50.5nm)
```

Two examples of multiple spectra. Note bellow the use of double square brackets, to obtain a single spectrum—using single square brackets would instead subset the collection of spectra, possibly returning a collection with a single member spectrum.

```
daylight1964.mspct <- colorSpec2mspct(daylight1964)

## Warning in range_check(x, strict.range = strict.range): Negative spectral energy
irradiance values; minimun s.e.irrad = -14
## Warning in range_check(x, strict.range = strict.range): Negative spectral energy
irradiance values; minimun s.e.irrad = -2.9

names(daylight1964.mspct)

## [1] "S0" "S1" "S2"

plot(daylight1964.mspct[[1]])
```
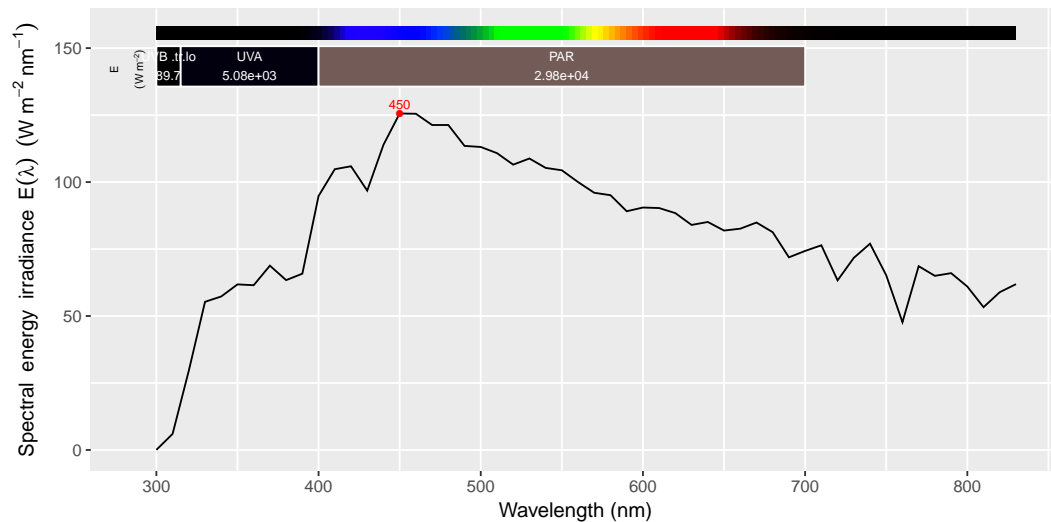


```
daylight2013.mspct <- colorSpec2mspct(daylight2013)

## Warning in range_check(x, strict.range = strict.range): Negative spectral energy
irradiance values; minimun s.e.irrad = -13
## Warning in range_check(x, strict.range = strict.range): Negative spectral energy
irradiance values; minimun s.e.irrad = -2.7

names(daylight2013.mspct)

## [1] "S0" "S1" "S2"
```

```
Fs.5nm.mspct <- colorSpec2mspct(Fs.5nm)
names(Fs.5nm.mspct)

##  [1] "F1"  "F2"  "F3"  "F4"  "F5"  "F6"  "F7"
##  [8] "F8"  "F9"  "F10" "F11" "F12"

plot(Fs.5nm.mspct[["F1"]])
plot(Fs.5nm.mspct[["F11"]])
```
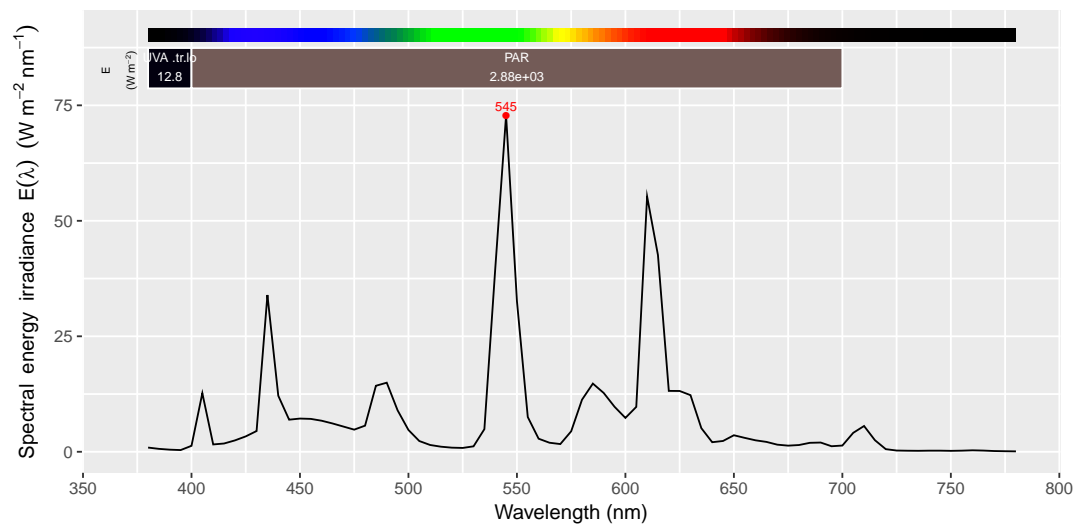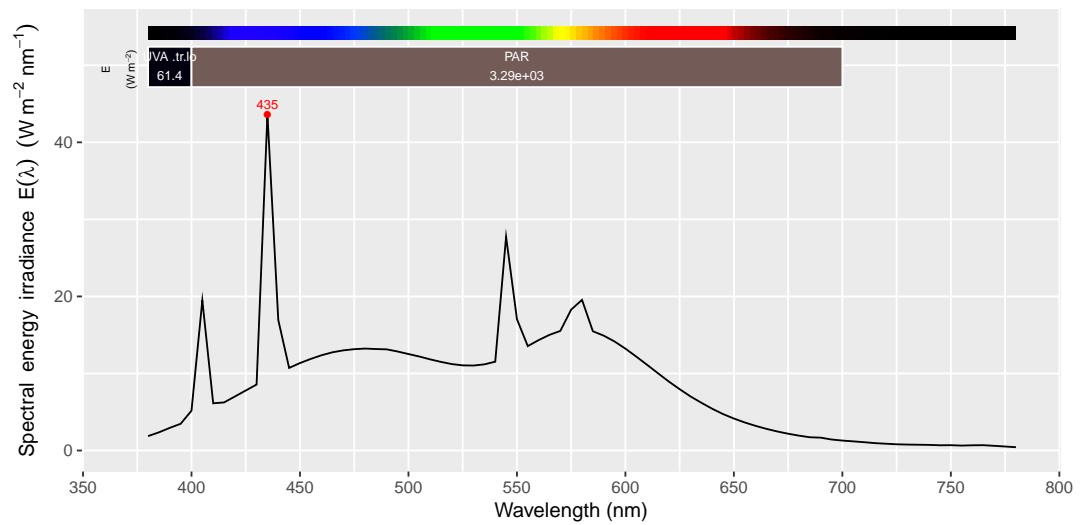
14.7 A sample of colours

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

```
color_chart(color(390:829), ncol = 20, text.color = NA)
```
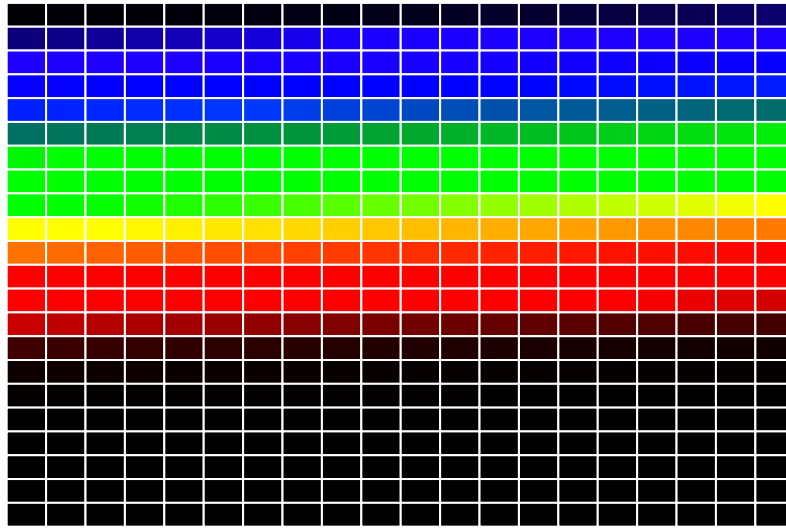
```
wls <- 400:699
color_chart(w_length2rgb(wls, color.name = as.character(wls)),
            ncol = 10, use.names = TRUE, text.size = 3)
```

```
try(detach(package:ggspectra))
try(detach(package:photobiologyInOut))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
try(detach(package:colorSpec))
```

15

Colour based indexes

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(photobiology)
library(ggspectra)
library(photobiologyPlants)
library(hsdar)
```

15.2 What are colour-based indexes?

15.3 Task: Calculation of the value of a known index from spectral data

We will start with a very well-known index used in remote sensing, Normalized Difference Vegetation Index (NDVI). We must be aware that an NDVI value calculated from spectral data on "first principles" may deviate from that obtained by means of non-spectral wide- or narrow-band sensors as used in used in satellites. Package 'hsdar' supplies spectral responses for satellites. So for remote sensing applications the use of this package is recommended.

We here demonstrate how to transfer a spectrum to 'hsdar', and one example of index calculation with this package. The 'hsdar' package can not only be used for individual spectra but also for hyperspectral images. Be aware, that this package seems to aim at data of rather low spectral resolution.

```
Solidago_hs.spct <- with(Solidago_altissima.mspct$upper_adax, speclib(Rfr, w.length))
plot(Solidago_hs.spct)
ndvi <- vegindex(Solidago_hs.spct, "NDVI")
ndvi

## [1] 0.7594845
```
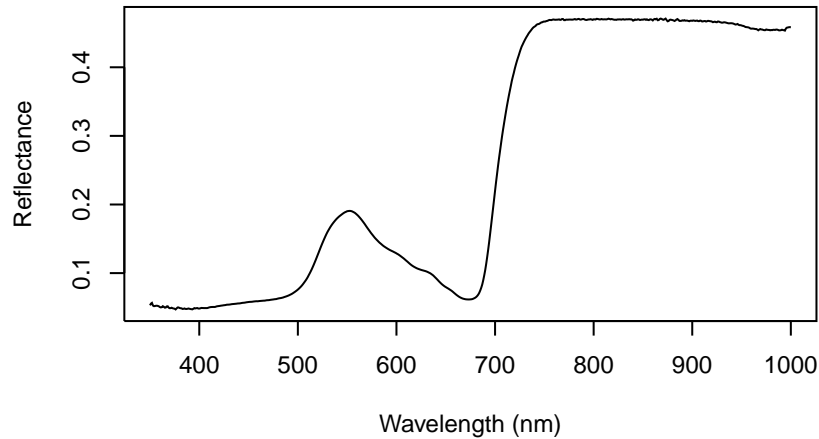
We now calculate a similar index by integrating reflectance for two wavebands,

```
normalized_diff_ind(Solidago_altissima.mspct$upper_adax,
                    waveband(c(700, 1100)),
                    waveband(c(400, 700)),
                    reflectance)

## NDI  reflectance  [ 700.1100 ] – [ 400.700 ]
##                                 0.6355334
```

This returns a different value because, the wavebands are un-weighted, while weighting functions would be needed to reproduce NDVI.

15.4 Task: Estimation of an optimal index for discrimination

15.5 Task: Fitting a simple optimal index for prediction of a continuous variable

15.6 Task: PCA or PCoA applied to spectral data

15.7 Task: Working with spectral images

```
try(detach(package:hsdar))
try(detach(package:photobiologyPlants))
try(detach(package:ggspectra))
try(detach(package:photobiology))
```

16

Plotting spectra and colours

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(scales)
library(ggrepel)
library(gridExtra)
library(dplyr)
library(photobiology)
library(photobiologyFilters)
library(photobiologyWavebands)
library(photobiologyPlants)
library(photobiologyReflectors)
library(ggspectra)
```

16.2 Set up

We create an object with two spectra, to be used to show grouping and faceting work.

```
two_suns.spct <- rbindspct(list(sun1 = sun.spct, sun2 = sun.spct * 2))
```

16.3 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 package 'ggplot2' and plotting using the *grammar of graphics*, please read the chapter on plotting in the book *Learning R …as your learnt your mother tongue* (Aphalo 2016).

16.4 Using `plot()` methods with spectra

Package 'ggspectra' defines specializations of the generic `plot` function of R. These methods are available for all the different spectral object classes defined in package 'photobiology' except for `generic_spct`. They return a `ggplot` object, to which additional layers can be added if desired. An example of it simplest use follows in the next task. As all the spectral objects have spectral quantities expressed in known units, and an attribute indicating the time unit, the axis labels are produced automatically. If summaries are added, the units in their labels are also produced automatically. The plots produced can be to some extent tailored through options or by adding or replacing `ggplot` layers, the intention is for these methods to be a total that is simple, but that caters for the majority of the most frequent tasks.

---

[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.
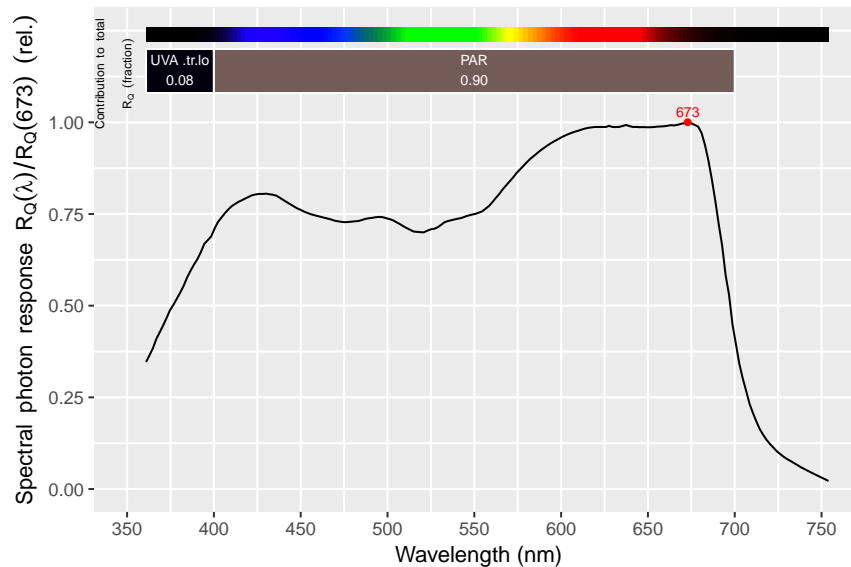
16.4.1 Task: plotting of `source_spct` objects

The defaults arguments cater the exploratory plotting of spectra, returning a plot that is heavily annotated, and using the most common units of expression. The two plots that follow show spectral irradiance, and spectral daily exposure, respectively. The objects plotted contain meta data that informs the time unit, so applying the same `plot` method, yields two plots with different units in the y-axis label.

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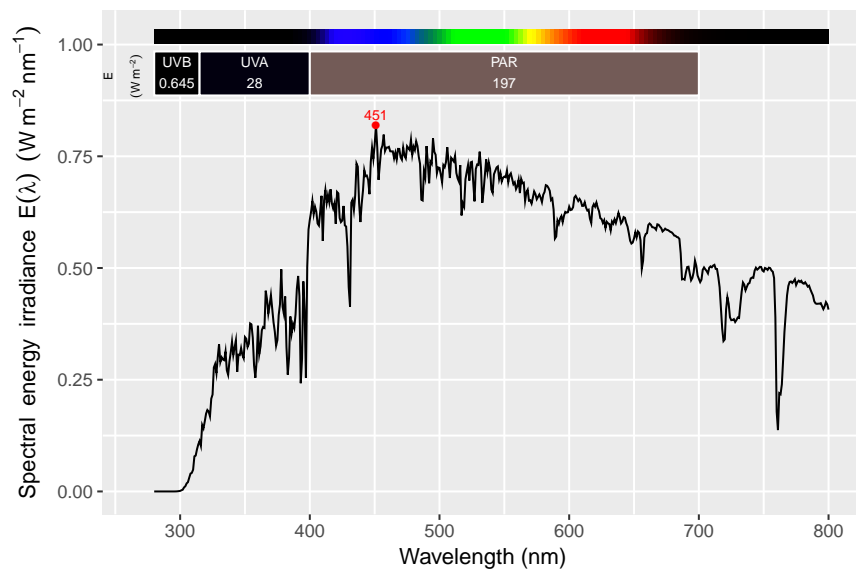


```
plot(sun.daily.spct)
```
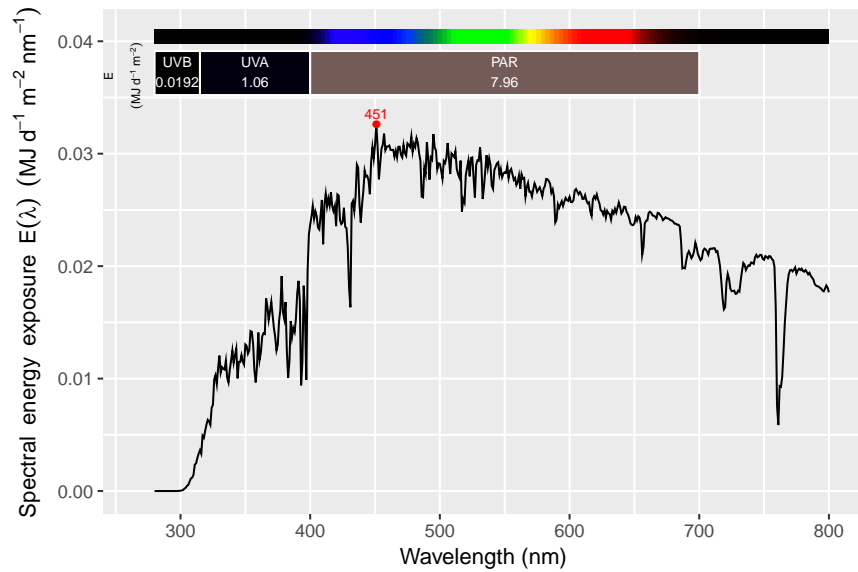
The parameter `unit` can be set to `"photon"` to obtain a plot depicting spectral photon irradiance. This works irrespective of whether the `source_spct` object contains spectral data expressed in photon or energy units—data are converted as needed.

```
plot(sun.spct, unit.out = "photon")
```
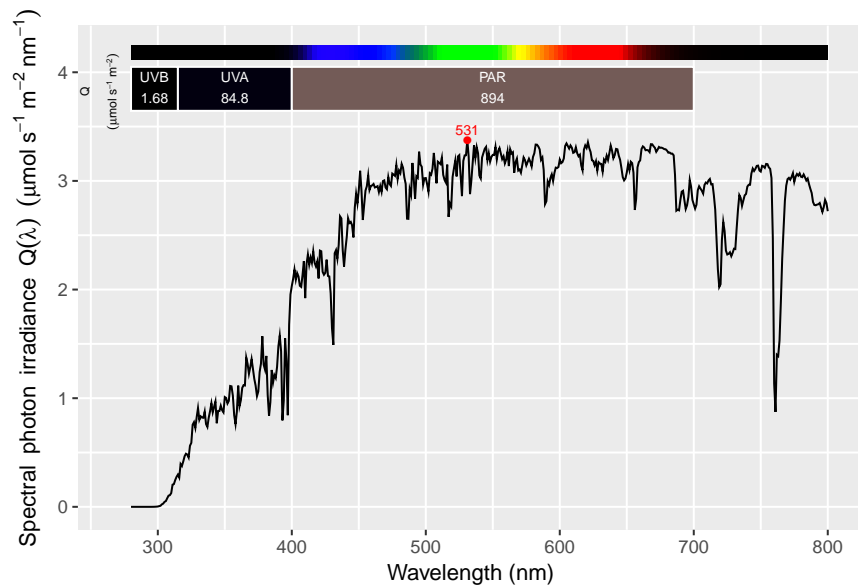

A list of wave bands, or a single wave band, to be used for annotation can be supplied through the `bands` parameter. A `NULL` waveband results in no waveband labels, while the next example shows how to obtain the total irradiance.

```
plot(sun.spct, w.band = PAR(), unit.out = "photon")
```
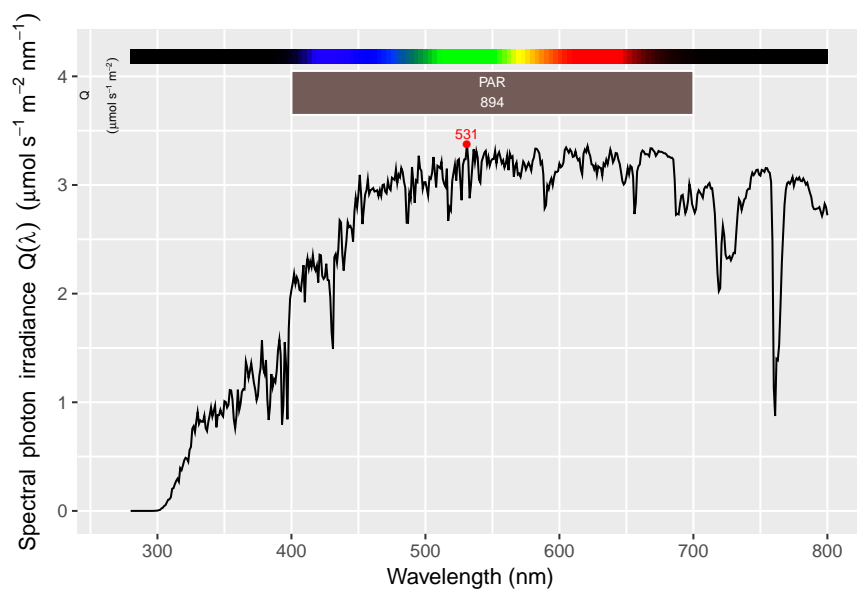


```
plot(sun.spct, w.band = NULL)
```
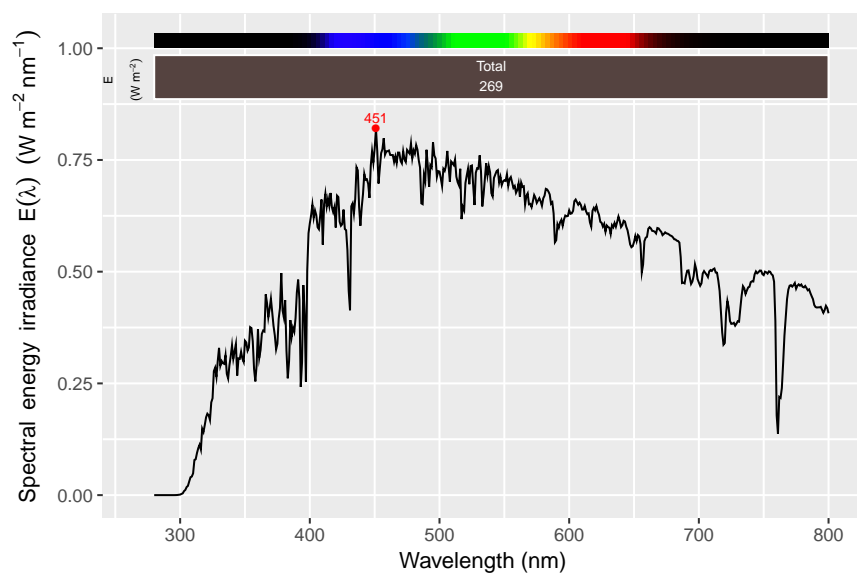


```
plot(sun.spct, w.band = waveband(sun.spct))
```
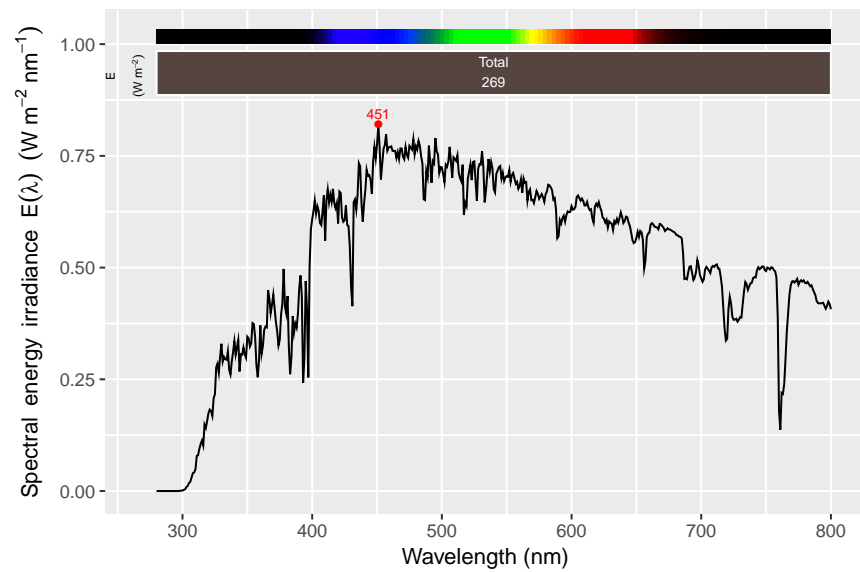
Of course the arguments to these parameters can be supplied in different combinations, and combined with other functions as needed. This last example shows how to plot using photon-based units, selecting only a specific region of the spectrum, annotated with the red and far-red photon irradiances, using Prof. Harry Smith's definitions for these two wavebands using 20-nm wide bands.

```
plot(trim_wl(sun.spct, waveband(c(600,800))),
     w.band = list(Red("Smith20"), Far_red("Smith20")), unit.out = "photon")
```
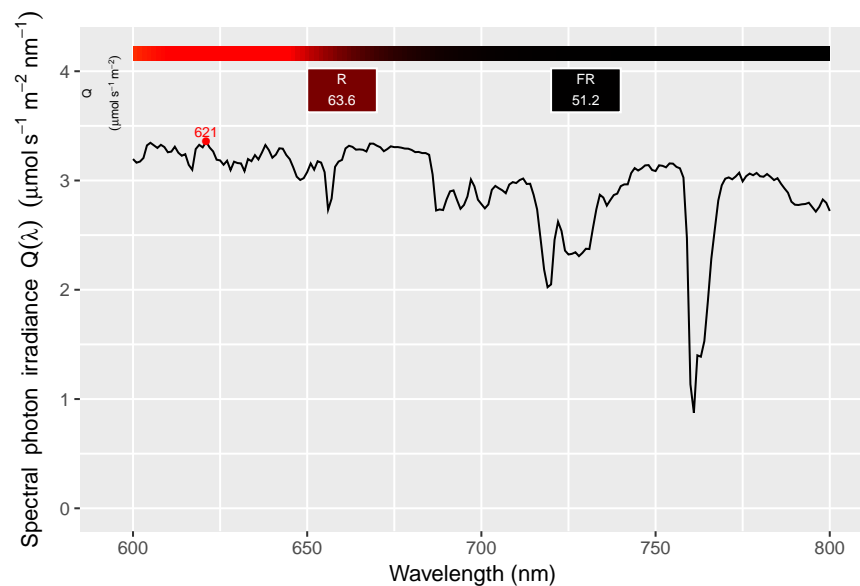

The next two examples show how to annotate an irradiance spectrum plot by equal sized wavebands.

```
plot(sun.spct,
     w.band = split_bands(c(300,800), length.out = 5), unit.out = "photon")
```
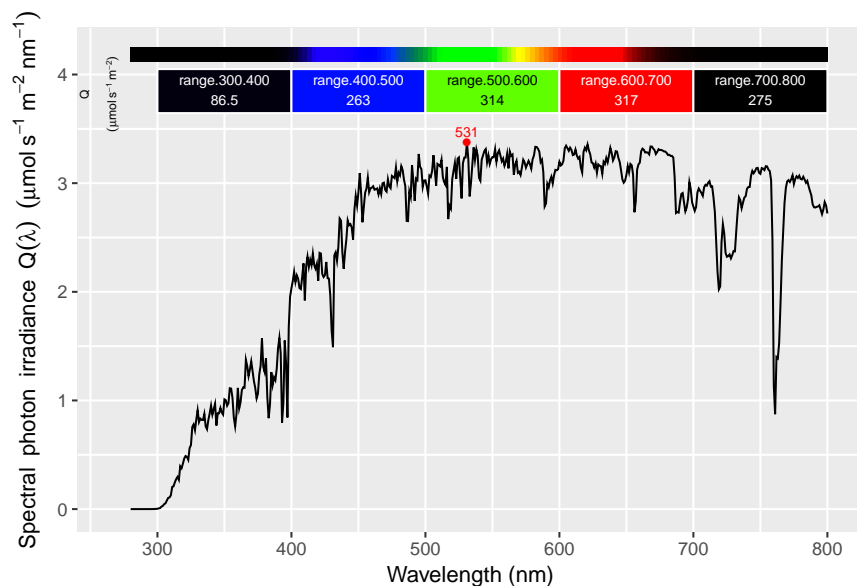


```
plot(trim_wl(sun.spct, PAR()),
     w.band=split_bands(PAR(), length.out = 6), unit.out = "photon")
```



```
plot(sun.spct,
     w.band = split_bands(c(300,800), length.out = 5), unit.out = "photon")
```

16

185
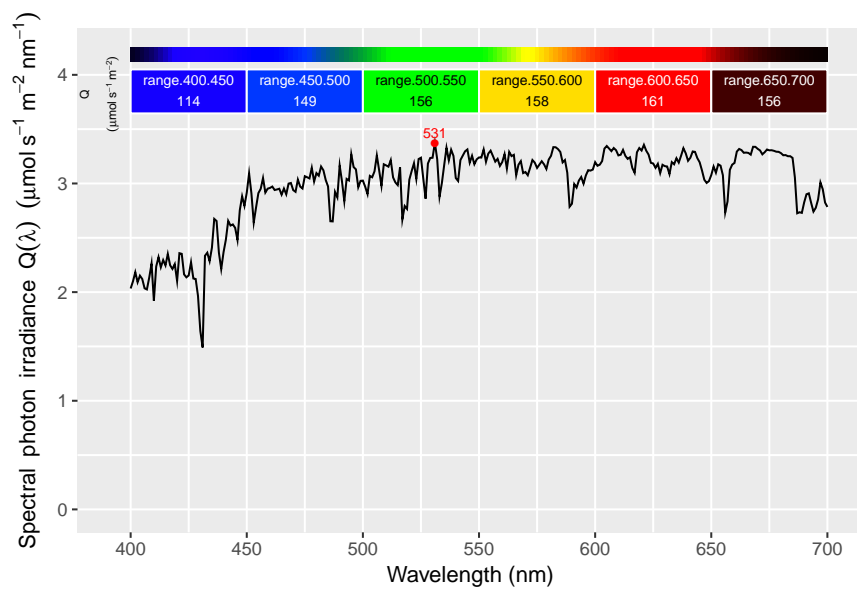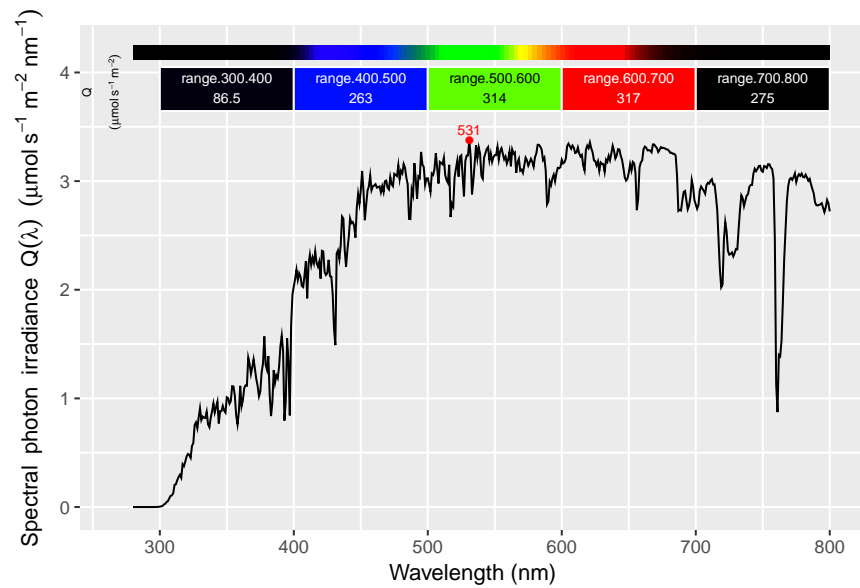
Which annotations are included, depends on the argument supplied, as exemplified below.

```
plot(sun.spct,
     annotations = c("=", "color.guide", "segments", "labels"),
     w.band = VIS_bands("ISO"))
```
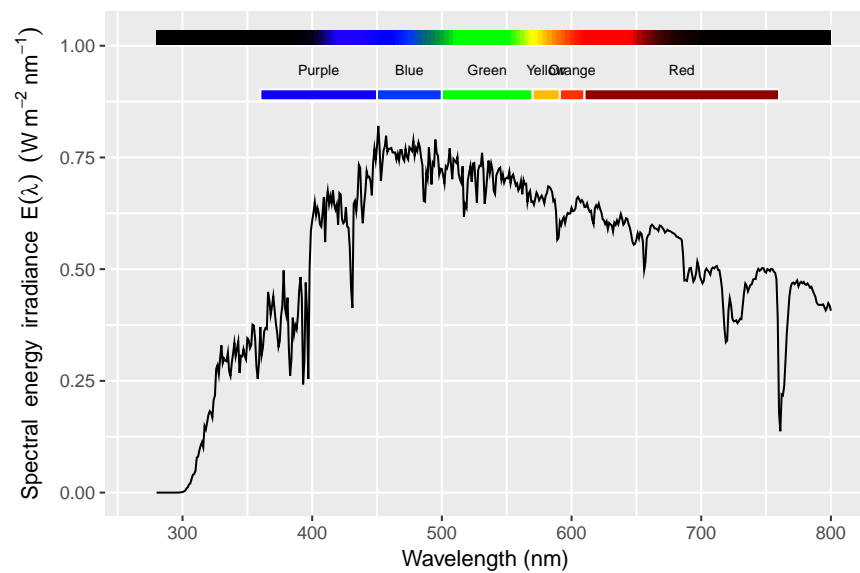


```
plot(sun.spct,
     annotations = c("-", "summaries"),
     w.band = VIS_bands("ISO"))
```
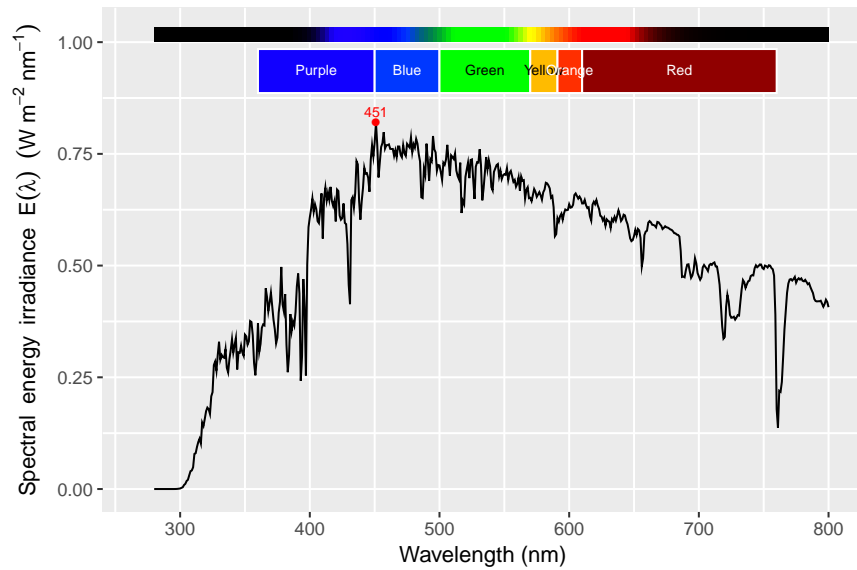
```
plot(sun.spct,
     annotations = c("+", "peak.labels", "segments"),
     w.band = VIS_bands("ISO"))
```
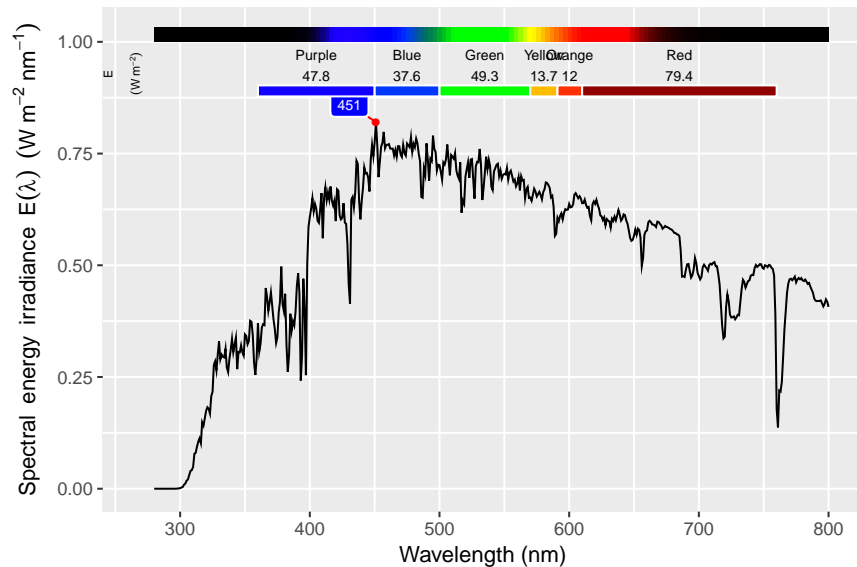

There are additional meta-data values stored in `source_spct` objects related to rescaling, normalization and weighting of the spectral data. These all affect axis labels, and in the case of weighting, the name of the spectral weighting function is added to the plot as an annotation. The following examples exemplify some of these, plus the use of `range` to limit the range of wavelengths included in the plot.

```
plot(sun.spct * CIE(), range = c(290,400))
```
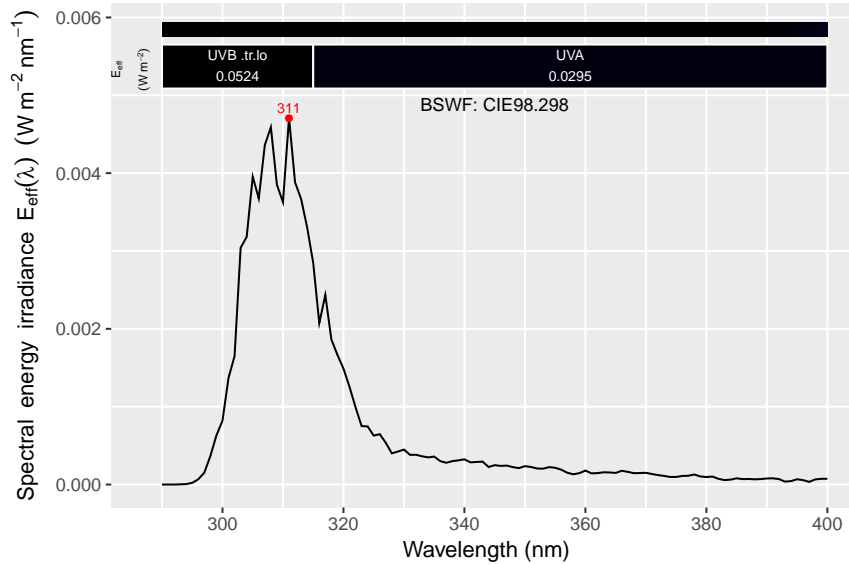

> As the current implementation uses `ggplot` 'statistic's, waveband irradiance annotations respect global aesthetics and facets. If used for simultaneous plotting in the same panel of several spectra (stored *longitudinally* in a single R object), then the *summaries* annotation should not be used as the values would be overplotted and ambiguous.

16.4.2 Task: plotting of normalized `source_spct` objects

Normalization consists in scaling a spectrum so that the spectral quantity takes a given value, usually one, at a certain wavelength. See section 4.4 on page 39 for a more detailed explanation.

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

```r
plot(normalize(sun.spct, norm = "max"))
```
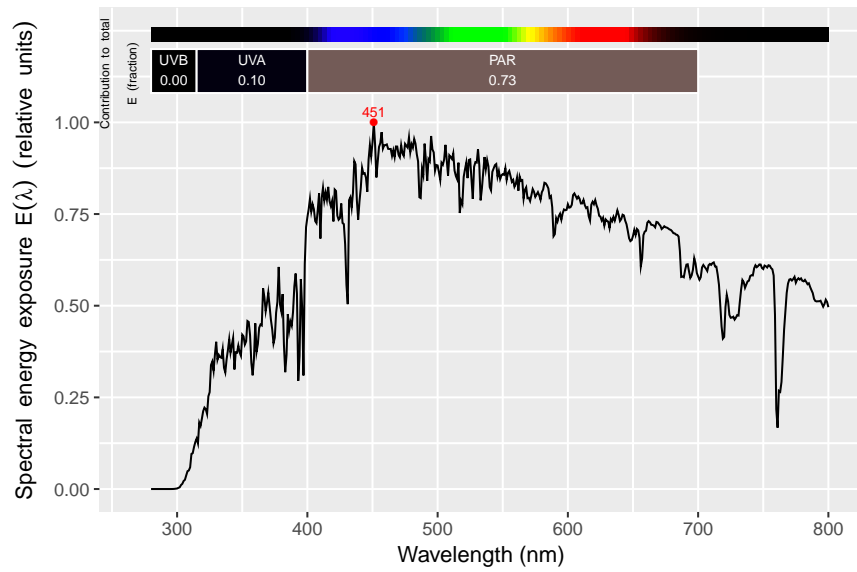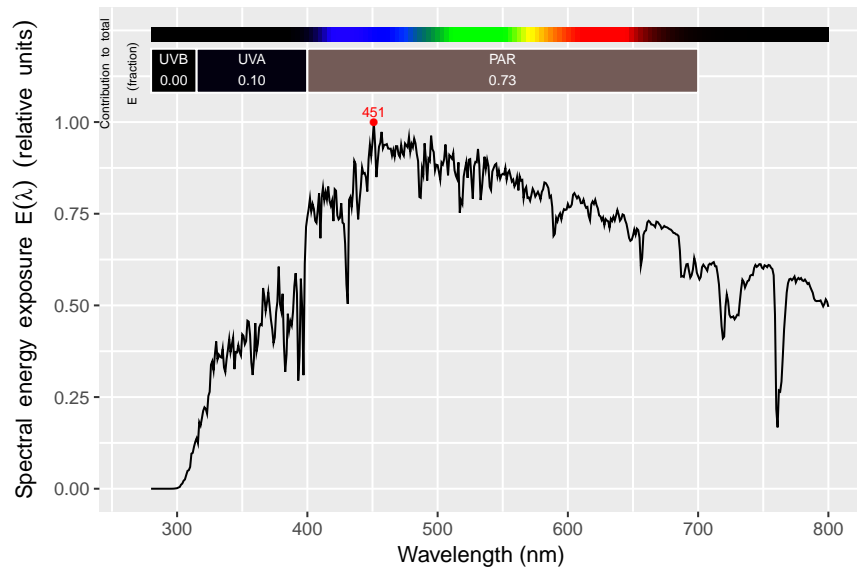
```r
plot(normalize(sun.spct, norm = 700))
```
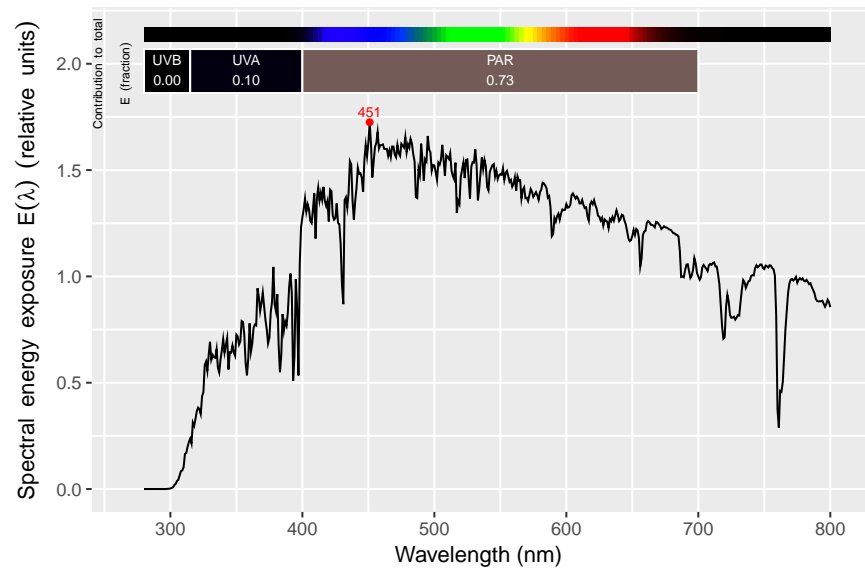
```r
plot(normalize(sun.spct, norm = "max"), range = VIS())
```
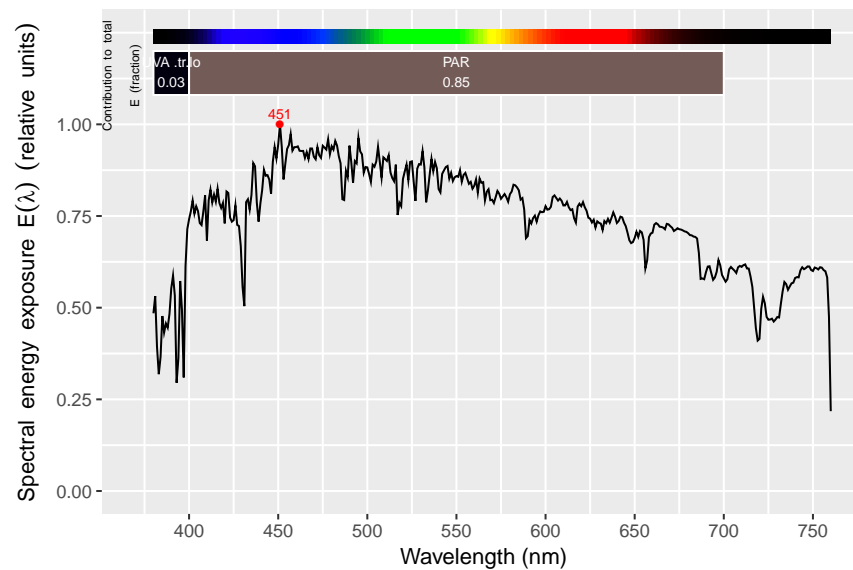


```r
plot(normalize(sun.spct, norm = "max", range = Red()),
     range = VIS(), w.band = VIS_bands())

## Warning in trim_tails(data$x, data$y, use.hinges = TRUE, low.limit = range[1], :
## Ignoring high.limit as it is too high.
```
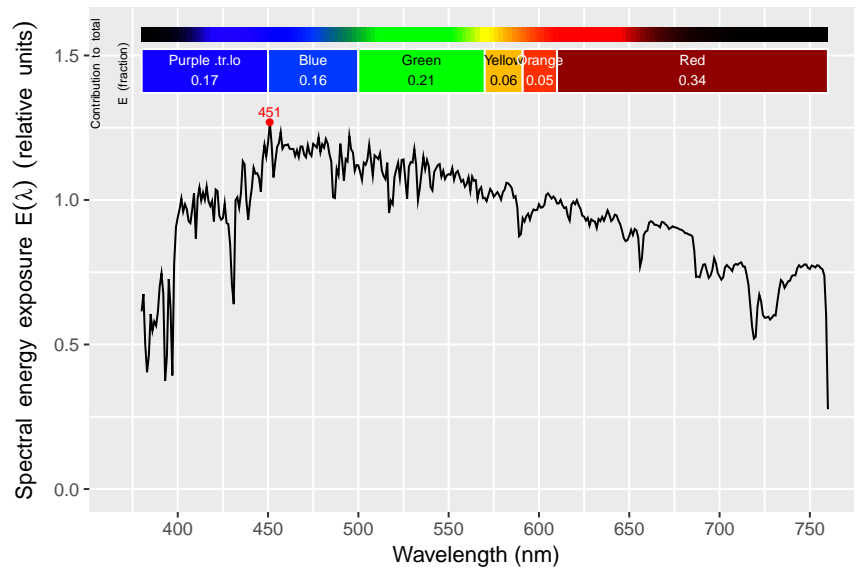
16.4.3 Task: plotting of `response_spct` objects

The defaults arguments cater the exploratory plotting of spectra, returning a plot that is heavily annotated. The two plots that follow show spectral response, expressed on an energy or photon basis. We use as example the action spectrum of photosynthesis in leaves.

```
plot(McCree_photosynthesis.mspct$oats)
```
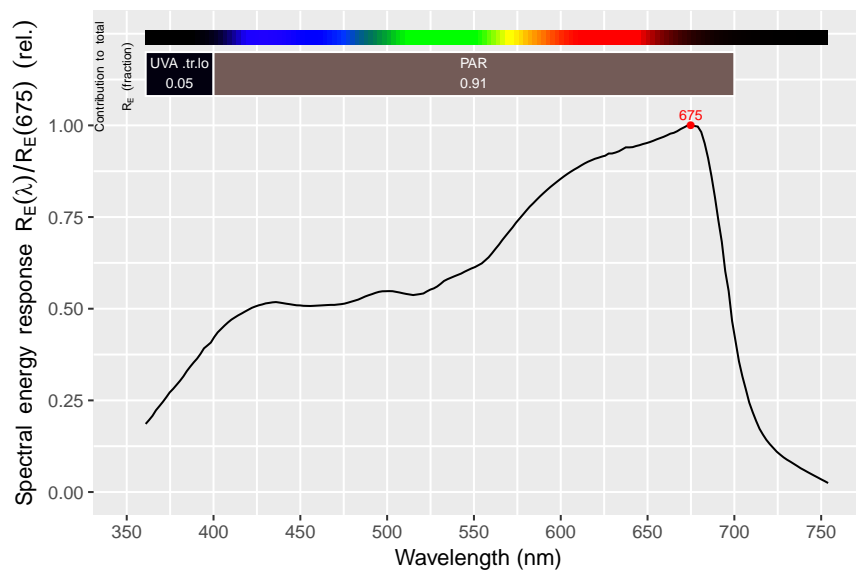
```
plot(McCree_photosynthesis.mspct$oats, unit.out = "photon")
```

For details on how to modify the wavebands used for labels and summaries, and customization of annotations, please see the previous section 16.4.2 on page 188. As for other `plot` methods described in this chapter normalization and scaling metadata are reflected in the plot axis-labels.

16.4.4 Task: plotting of `filter_spct` objects

The defaults arguments cater the exploratory plotting of spectra, returning a plot that is heavily annotated, and using the most common units of expression. The three plots that follow show spectral transmittance, spectral absorptance and spectral absorbance, respectively. The objects plotted contain meta data that informs wether transmittance is expressed as *internal* or *total*, so applying the same `plot` method, will produce plots with different units in the y-axis label in these two cases. The difference between internal and total transmittance is explained in section 12.3.

```
plot(yellow_gel.spct)
```
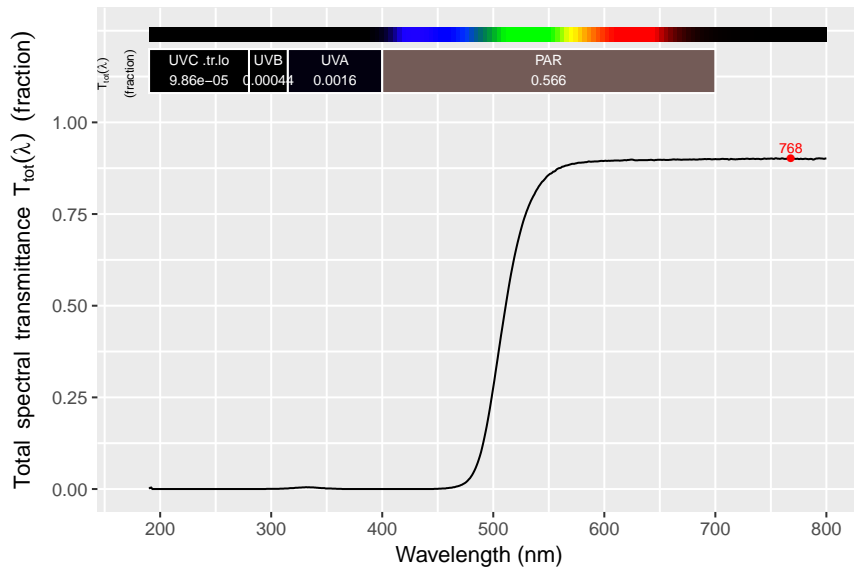
```r
plot(yellow_gel.spct, plot.qty = "absorbance")
```
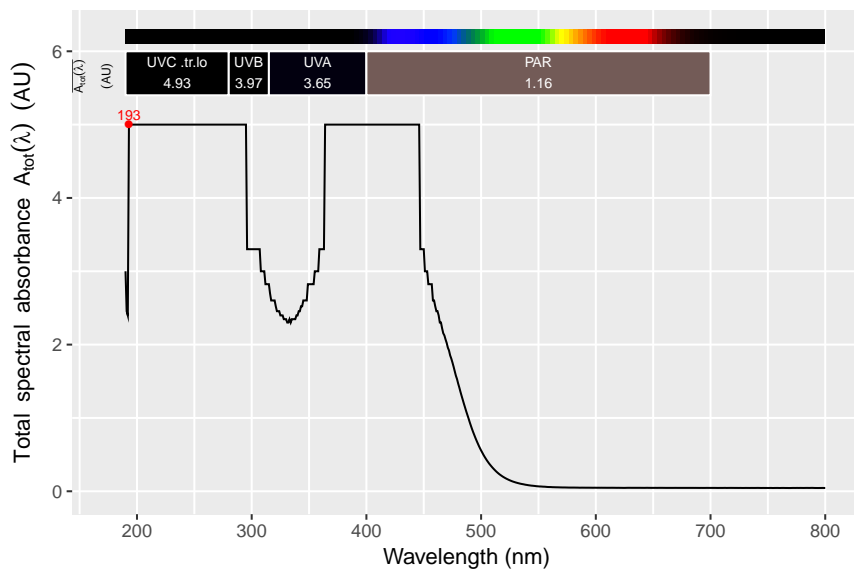

Beware that *absorptance* being the complement to *internal transmittance* can be calculated only from transmittance alone if it is expressed as internal. If total transmittance is available, then reflectance must be also known before absorptance can be calcaluated. For this reason we use different spectral data for this example. We also illustrate how we can force the range of wavelengths included in the plot to match the range of another spectral object.

```
plot(filters.mspct$GG495, range = yellow_gel.spct)
```
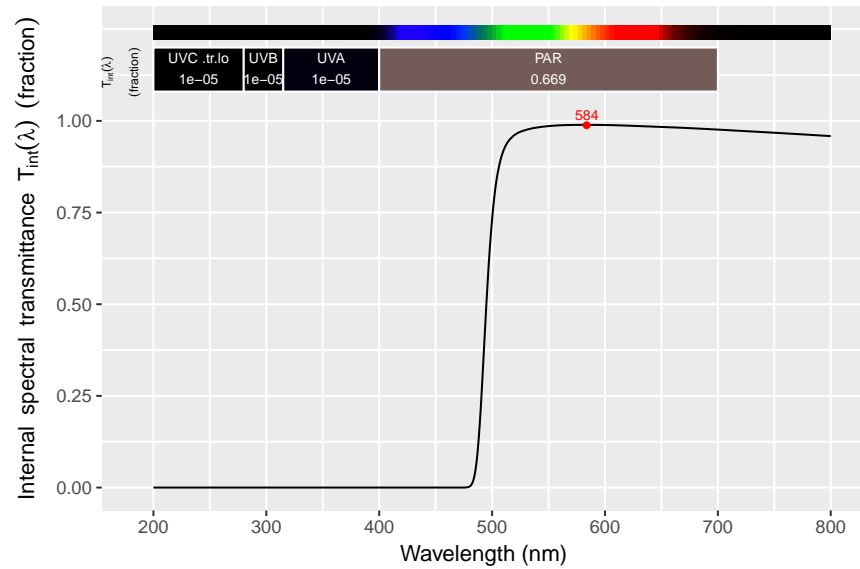


```
plot(filters.mspct$GG495, plot.qty = "absorptance",
     range = yellow_gel.spct)
```
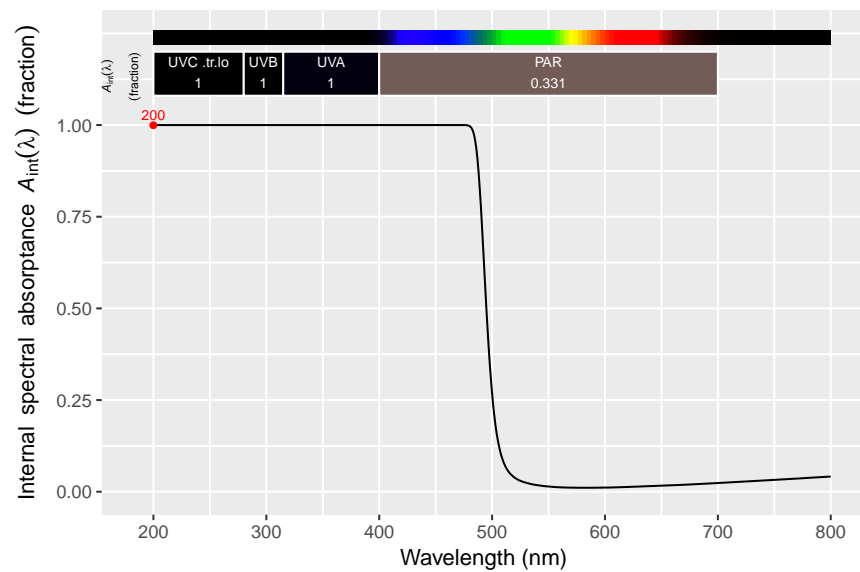

By default transmittance and absorptance are expressed as fractions of one. However, plotting of values expressed as percentages is also possible.

```
plot(yellow_gel.spct, pc.out = TRUE)
```
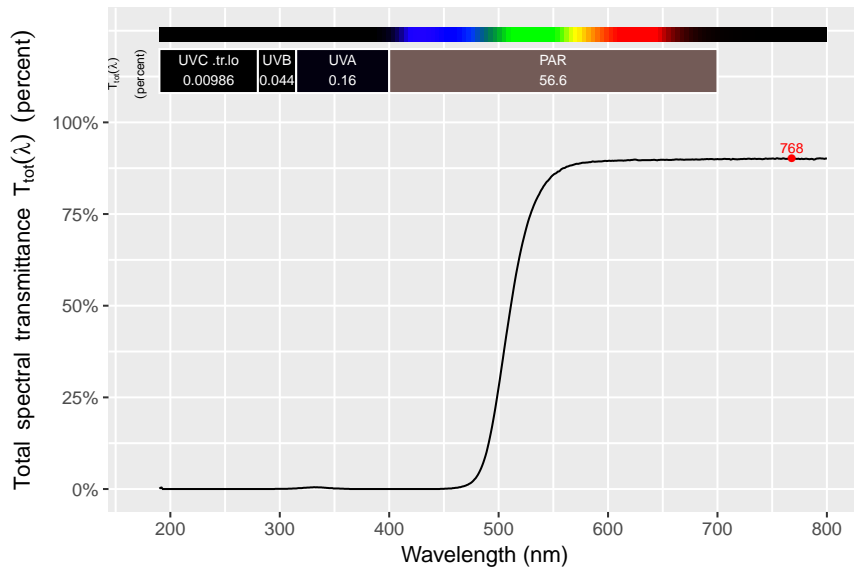
```
plot(filters.mspct$GG495, plot.qty = "absorptance",
     pc.out = TRUE, range = yellow_gel.spct)
```
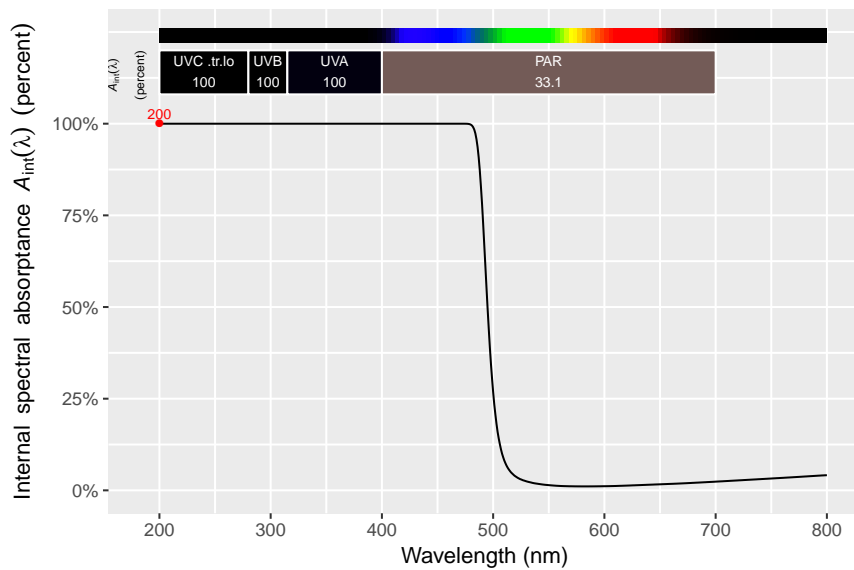

For details on how to modify the wavebands used for labels and summaries, and customization of annotations, please see the previous section .

16.4.5 Task: plotting of `reflector_spct` objects

At the moment, the only `plot.qty` supported is *reflectance*.

```
plot(gold.spct)
```
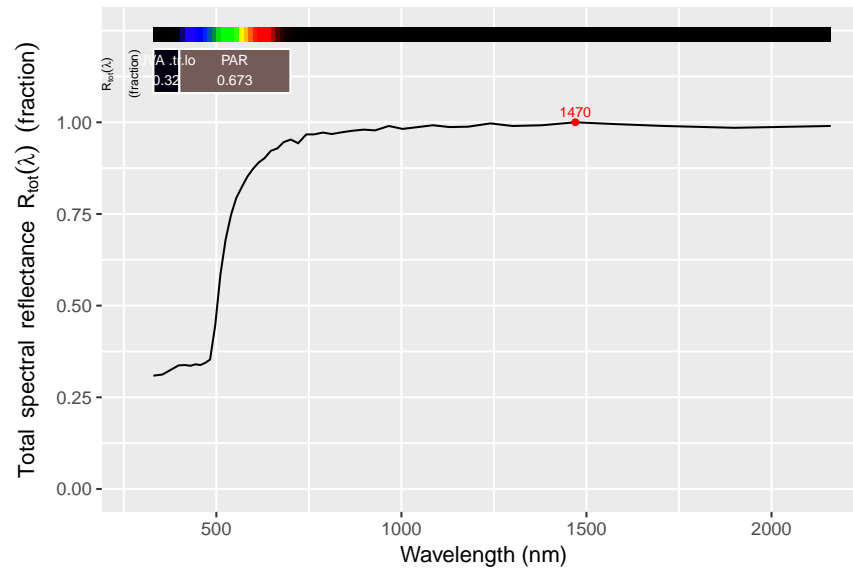


```
plot(gold.spct, pc.out = TRUE)
```
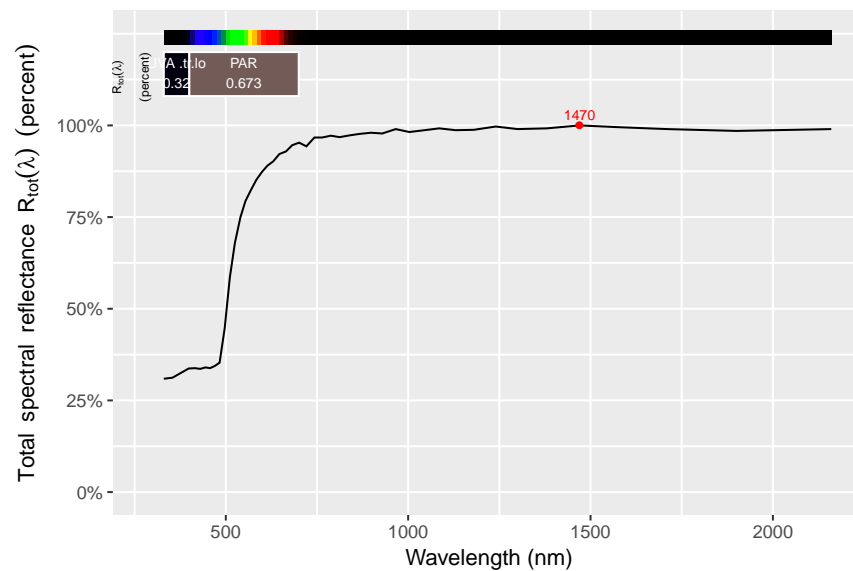

For details on how to modify the wavebands used for labels and summaries, and customization of annotations, please see the previous section 16.4.2 on page 188.

16.4.6 Task: plotting of `object_spct` objects

We will use as example the spectral properties of a plant leaf. This method is the most flexible with respect to the quantity plotted, as `object_spct` objects contain more than one spectral quantity, and as $R + A + T = 1$, the three quantities are always available for plotting. By default, all three are plotted.

```
plot(Solidago_altissima.mspct$upper_adax)
```
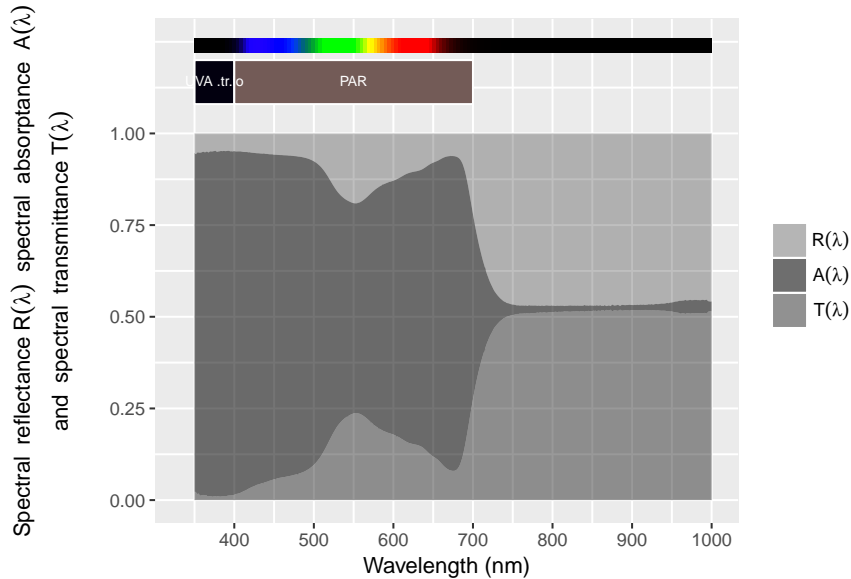

The default for `plot.qty` is `"all"` (meaning Rrf, Tfr, and Afr).

```
plot(Solidago_altissima.mspct$upper_adax, plot.qty = "all")
```


197
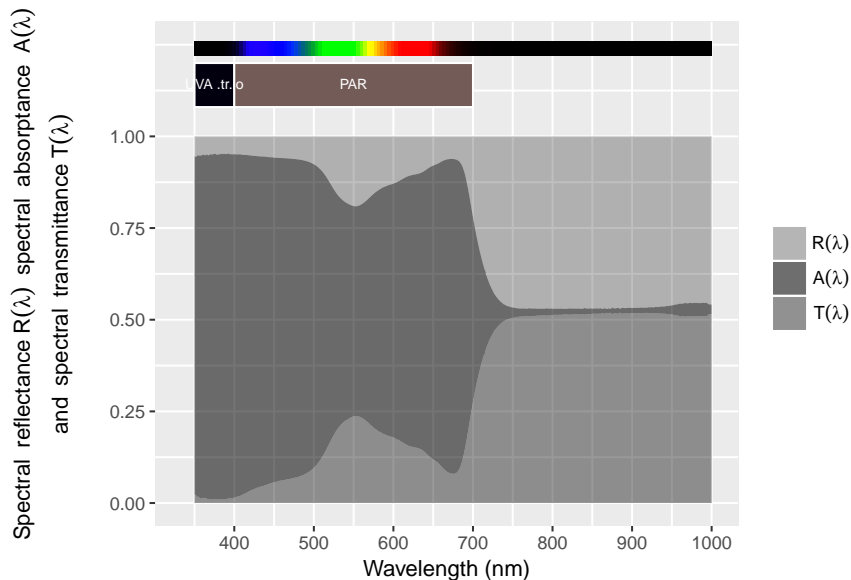
When the three quantities are plotted, the default is to stack them, but this can be changed.

```
plot(Solidago_altissima.mspct$upper_adax, plot.qty = "all", stacked = FALSE)
```
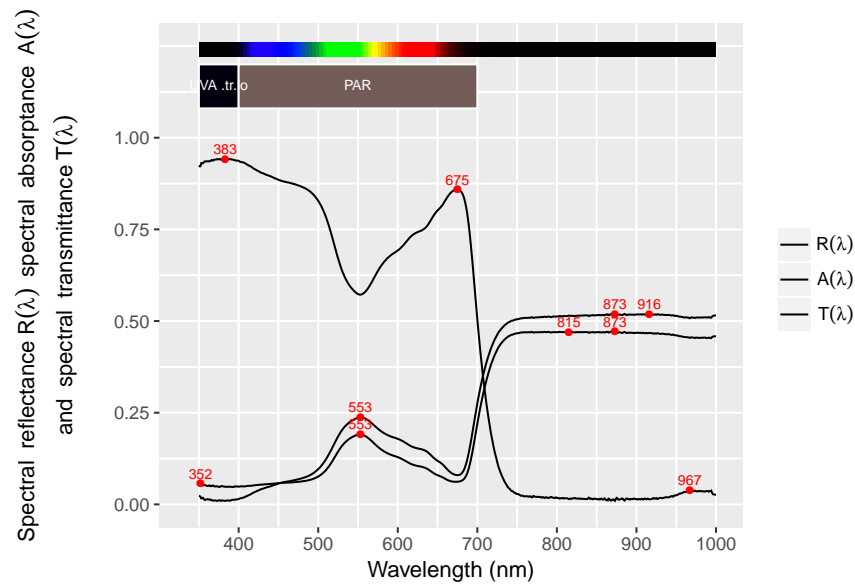

The three quantities can also be plotted individually, as well as absorbance.

```
plot(Solidago_altissima.mspct$upper_adax, plot.qty = "reflectance")
```



```
plot(Solidago_altissima.mspct$upper_adax, plot.qty = "absorptance")
```
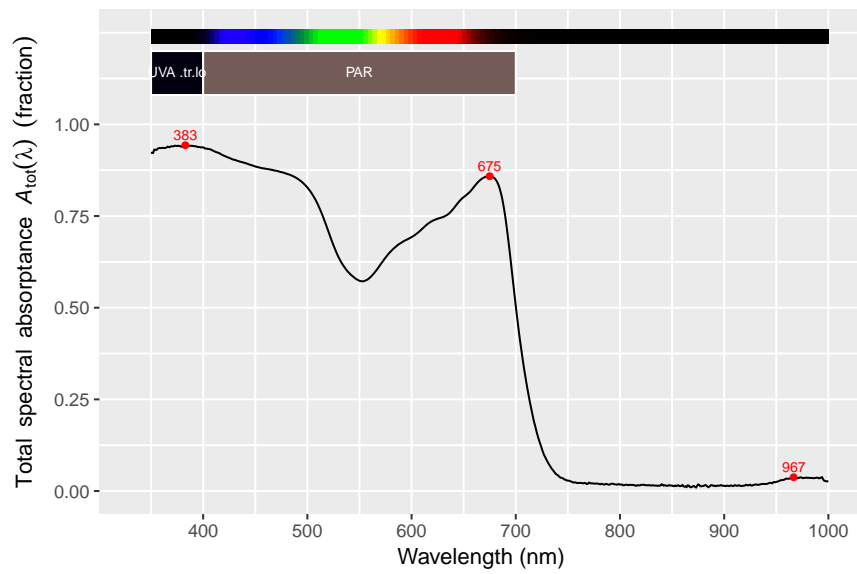
```
plot(Solidago_altissima.mspct$upper_adax, plot.qty = "transmittance")
```
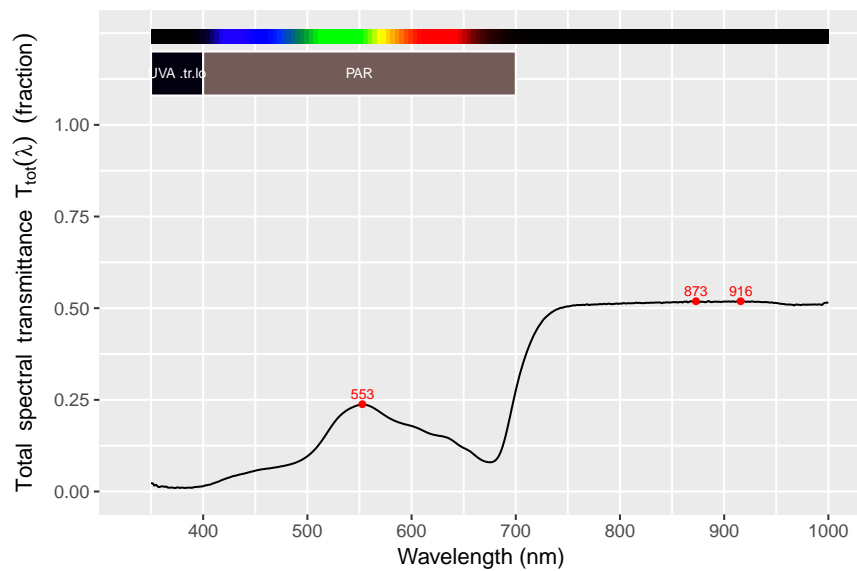


```
plot(Solidago_altissima.mspct$upper_adax, plot.qty = "absorbance")
```
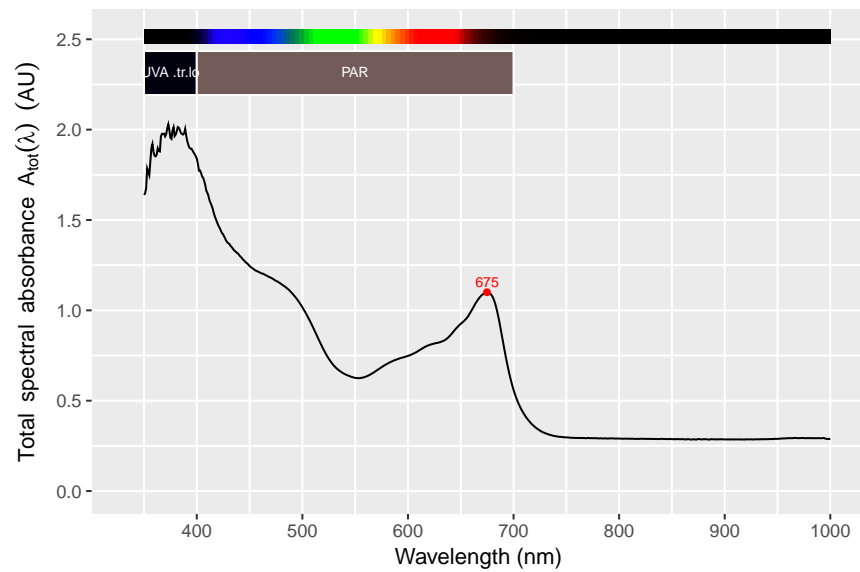
As for `filter_spct` and `reflector_spct` quantities can be plotted as percentages. We give only one example here.

```
plot(Solidago_altissima.mspct$upper_adax, pc.out = TRUE)
```
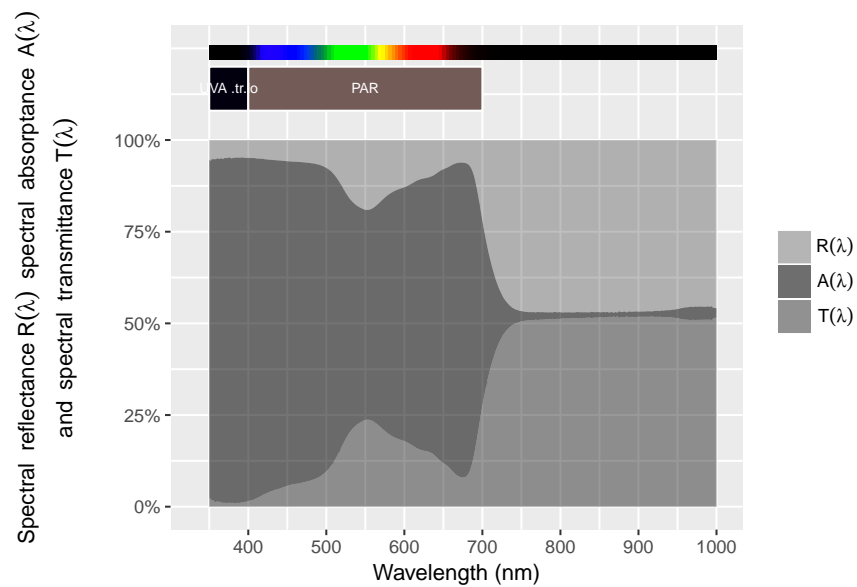

For details on how to modify the wavebands used for labels and summaries, and customization of annotations, please see the previous section 16.4.2 on page 188.

16.5 Plotting spectra with `ggplot`

Package 'ggspectra' not only provides simple to use `plot` methods, but also 'building block' that simplify the construction of custom plots of spectral data with functions from package 'ggplot2'. In this section we give examples of plotting tasks carried out using this more flexible approach. In this chapter's examples we will exploit the power of the grammar of graphics to build figures piece by piece, reusing some of the 'pieces' or groups of 'pieces' several times to highlight the flexibility of this approach. We also exemplify the 'philosophy' of defining everything that needs to be consistent across figures only once. In this section we use a `source_spct` object, `sun.spct` in most examples, but the same approaches work with objects of any of the spectral classes from package 'photobiology'. When plotting spectral objects the use of `aes` is optional, as 'ggspectra' defines specializations of method `ggplot` with suitable defaults aesthetics.

16.5.1 Task: plotting `source_spct` objects

We start with a very simple example, and later add layers to the plot little by little. We create a 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.

```
ggplot(data = sun.spct) +
    geom_line()
```

---

[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.
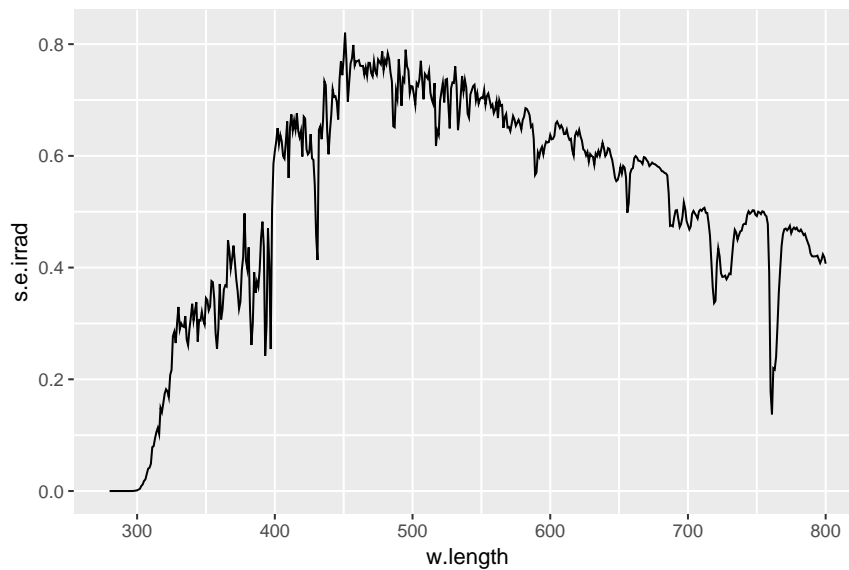
⚠ If package 'ggtern' is loaded simultaneously with package 'ggmisc' the examples in this chapter will give an error. The conflict originates in that 'ggtern' redirects calls to method `ggplot` to itself, and does not dispatch the specialized methods defined in 'ggpmisc'. The solution is to give the name of the namespace explicitly as shown below.

```
ggplot2::ggplot(data = sun.spct) +
                geom_line()
```

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 the documentation with `help(plotmath)` or read a book or watch tutorial for more details.

```
ggplot(data = sun.spct) +
    geom_line() +
  labs(
    y = expression(Spectral~~energy~~irradiance~~(W~m^{-2}~nm^{-1})),
    x = "Wavelength (nm)")
```
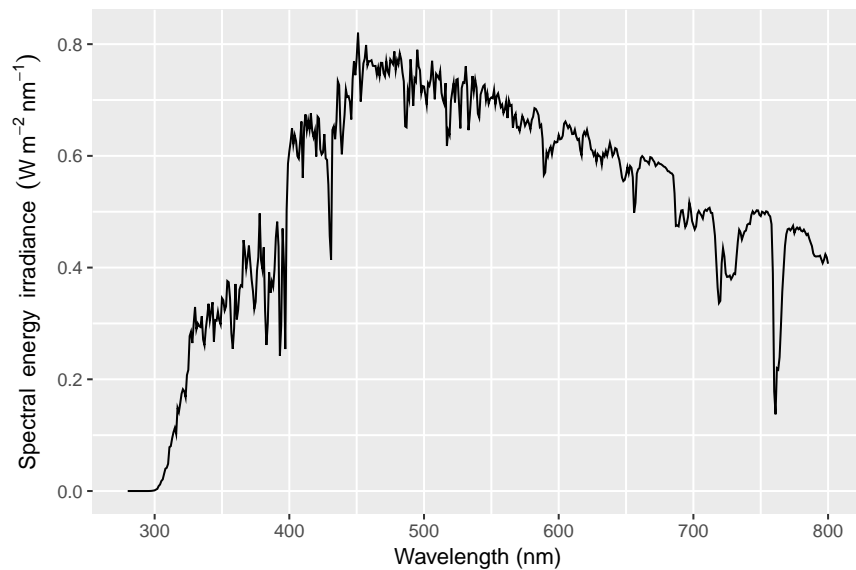
16.5.2 Task: Saving axis-label definitions for re-use

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 obtain 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)"
```

```
ggplot(data = sun.spct) +
    geom_line() +
    labs(x = xlab_nm, y = ylab_watt)
```
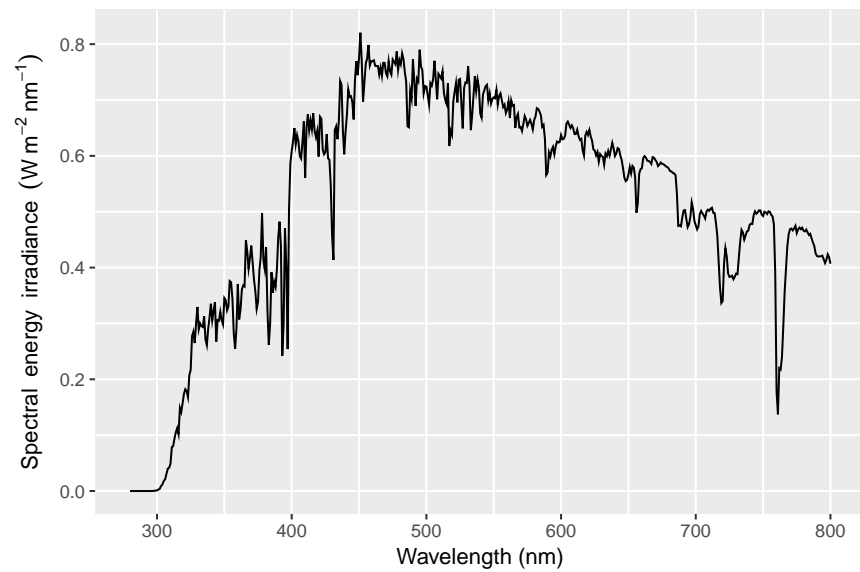
```
ggplot(data = sun.spct, unit.out = "photon") +
    geom_line() +
    labs(x = xlab_nm, y = ylab_umol)
```
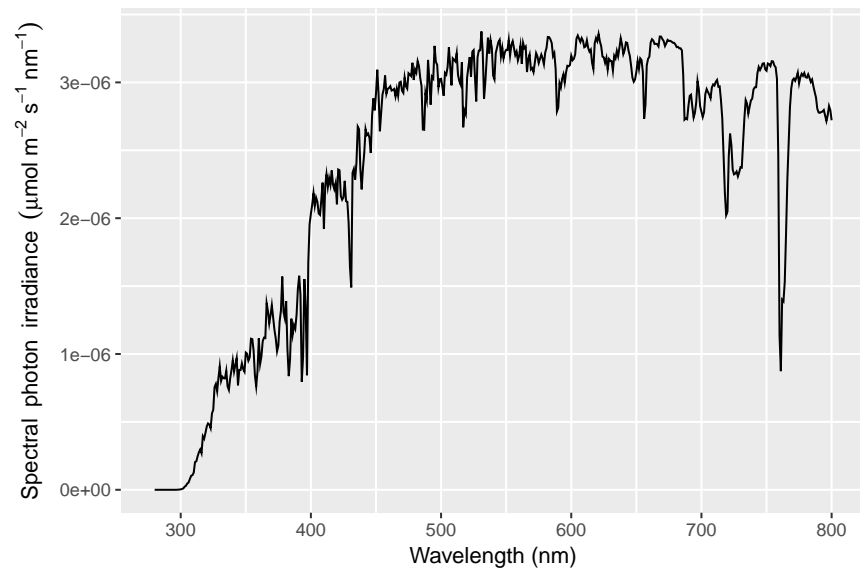

16.5.3 Task: plotting a spectrum as discrete columns

It is also possible to split a range of wavelengths into a list of wavebands and use these in a column plot. In this example we split the solar spectrum into a series of wavebands each 25 nm-wide.

```
wl.range <- c(250, 800)
num.bands <- spread(wl.range) / 25
many.bands <- split_bands(wl.range, length.out = num.bands)
```

```
ggplot(sun.spct) +
  stat_wb_column(color = "white", w.band = many.bands) +
  labs(
    y = expression(Spectral~~energy~~irradiance~~(W~m^{-2}~nm^{-1})),
    x = "Wavelength (nm)") +
  scale_fill_identity() +
  scale_color_identity()
```
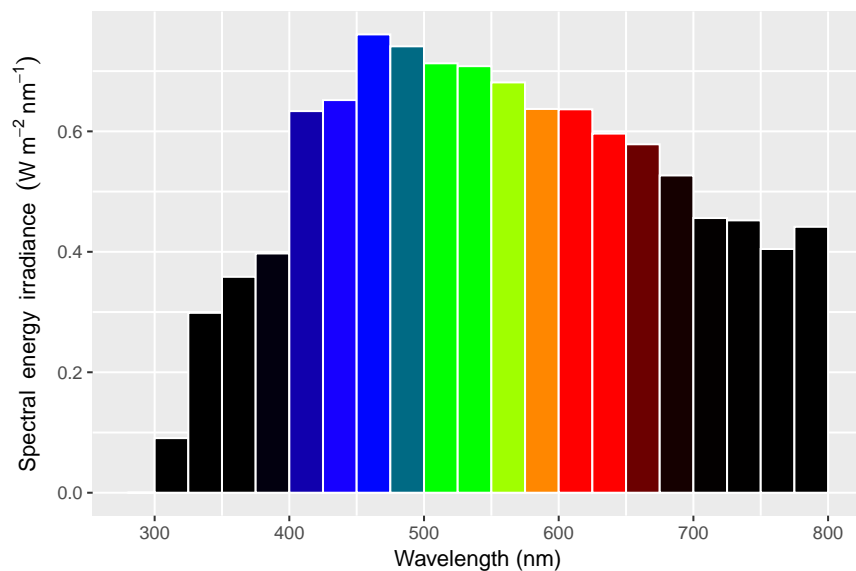

Or for PAR one could use a plot like the following with 10 nm-wide columns.

```
wl.range <- range(PAR())
num.bands <- spread(PAR()) / 25
many.bands <- split_bands(wl.range, length.out = num.bands)
```

```
ggplot(trim_wl(sun.spct, range = PAR())) +
  stat_wb_column(color = "white", w.band = many.bands) +
  labs(
    y = expression(Spectral~~energy~~irradiance~~(W~m^{-2}~nm^{-1})),
    x = "Wavelength (nm)") +
  scale_fill_identity()

## Warning in trim_tails(data$x, data$y, use.hinges = TRUE, low.limit = range[1], :
## Ignoring high.limit as it is too high.
```
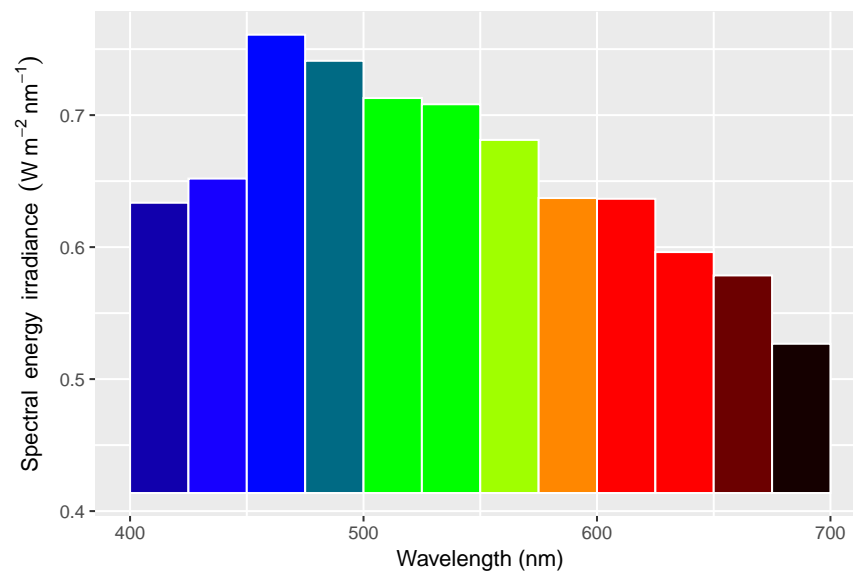
16.5.4 Task: using a log scale

Here without need to recreate the figure, we add a logarithmic scale for the y-axis and two of the just saved axis-labels, printing the plot on the fly. 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.

```
ggplot(data = sun.spct) +
      geom_line() +
      scale_y_log10(limits=c(1e-3, 1e0))

## Warning: Transformation introduced infinite values in continuous y-axis
```
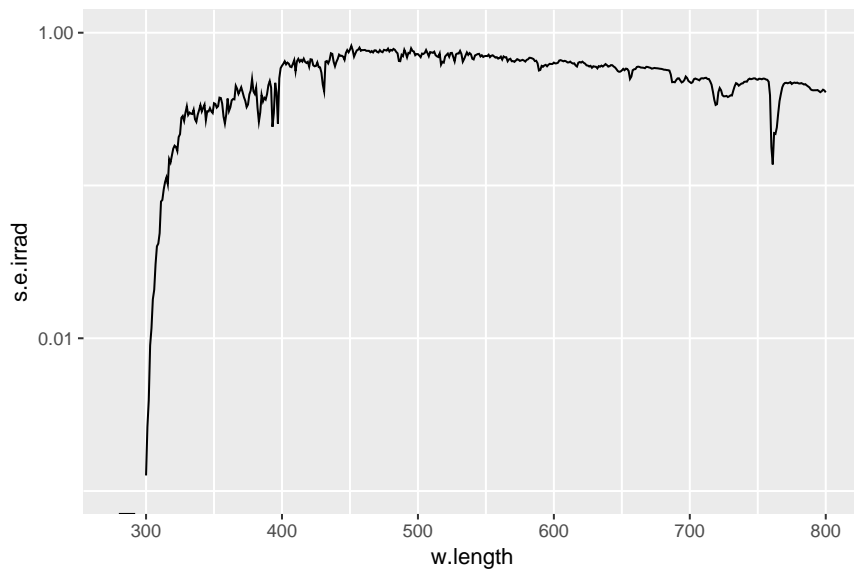
The code above generates some harmless warnings, which are due some y values not being valid input for `log10`, the function used for the re-scaling, or because they fall outside the scale limits.

16.5.5 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.

As we are re-scaling the data on the fly, we need to override the default `aes` as a whole, we have to specify both the x and y aesthetics. If we want to have a particular formatting for axis labels, we can write our own function and pass it as argument. We save to individual figures, one with spectral irradiance expressed on a photon or quantum basis, and one an energy basis.

```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 <-
```

```r
ggplot(data=sun.spct, aes(x=w.length, y=s.q.irrad)) +
geom_line() +
scale_y_continuous(labels = num_one_dec) +
labs(x = xlab_nm)
```

We use function `multiplot` to make a single plot from two separate ggplot objects, 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)
```
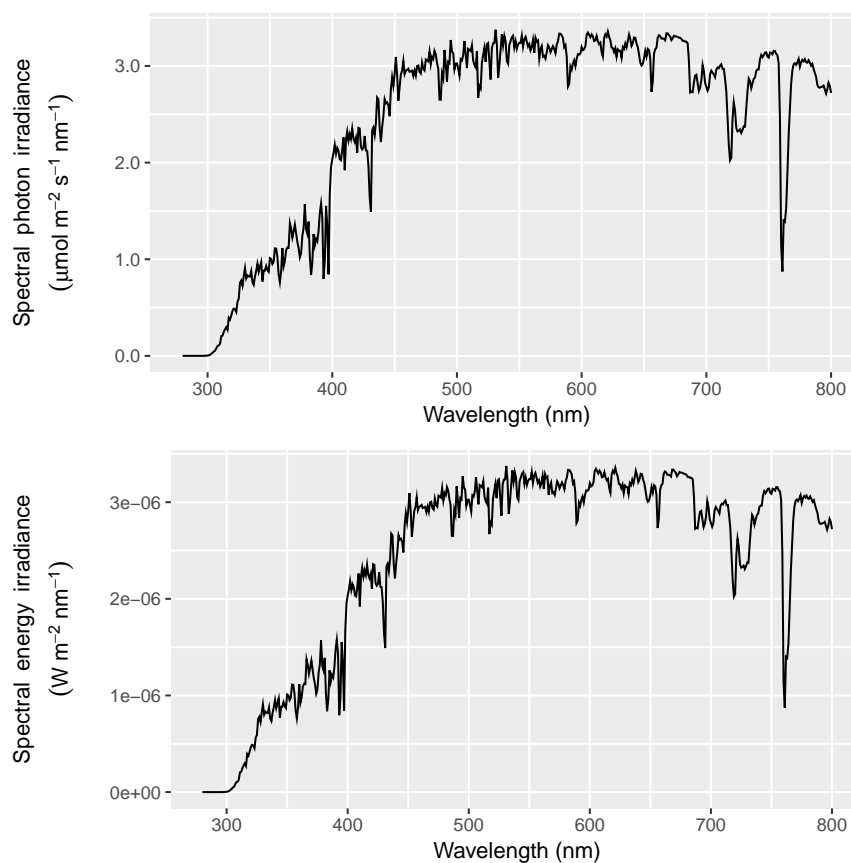


```r
multiplot(fig_sun.q  + labs(y = ylab_umol),
          fig_sun.e1 + labs(y = ylab_watt),
          cols = 2)
```

Table 16.1: Spectral features-extraction `ggplot` statistics defined in package 'ggspectra'

stat	description
stat_peaks	Subset and annotate local maxima
stat_valleys	Subset and annotate local minima
stat_label_peaks	Tag and annotate local maxima
stat_label_valleys	Tag and annotate local minima
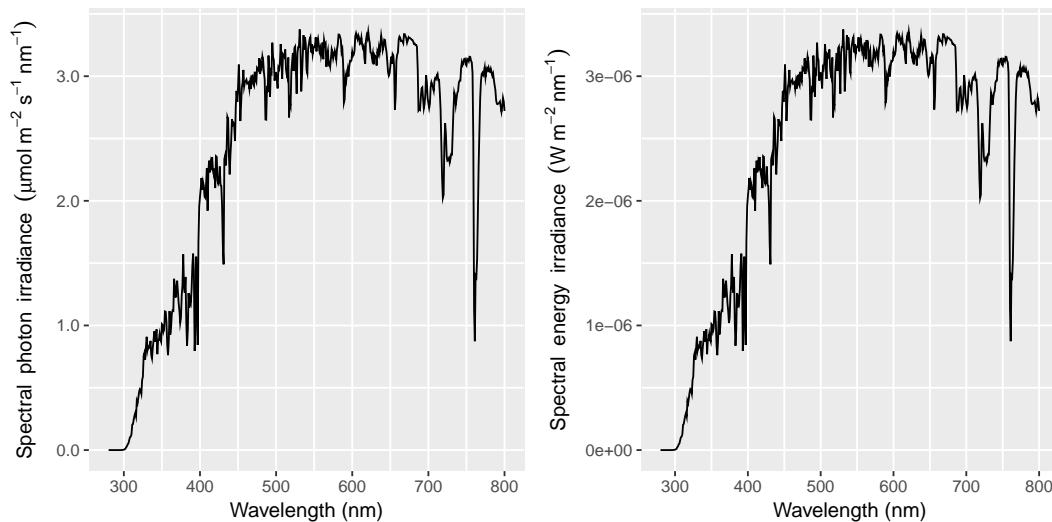
16.5.6 Task: annotating peaks and valleys in spectra

Here we show an example of the use of `stat_peaks` and `stat_valleys` from package 'ggspectra'. It uses the same parameter names and take the same arguments as methods `peaks` and `valleys` described in section 8.13 on page 93—please, consult this section for details on how to adjust how many and which local maxima and local minima are highlighted.

We reuse once more `fig_sun.e` saved in section 16.5 and we start with the simple example of highlighting the overall maximum in a spectrum. By default `geom_point` is used.

```
head(with(sun.spct,
          get_peaks(w.length, s.e.irrad, span = NULL)))

##     x         y label
## 1 451 0.8204633   451
```

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_peaks(span=NULL)
```
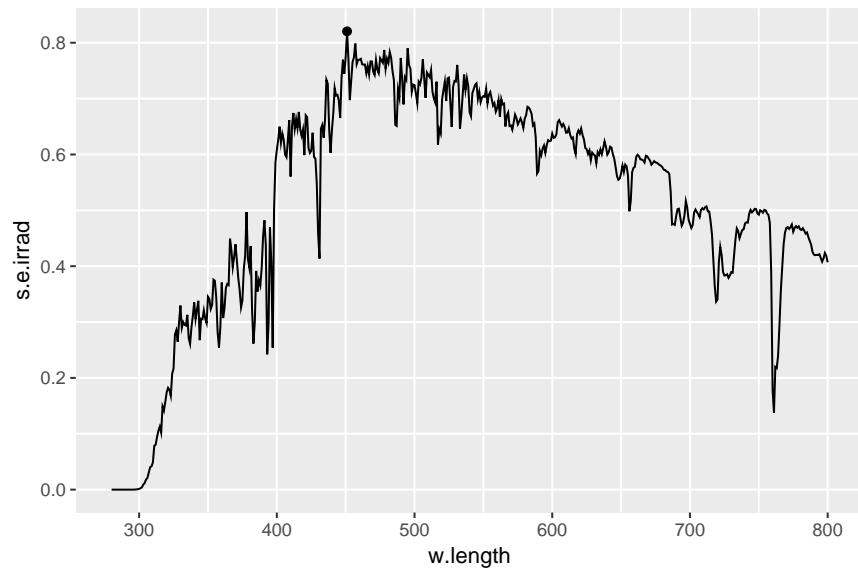

We can change aesthetics, for example the colour.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_peaks(span=NULL, color = "red")
```
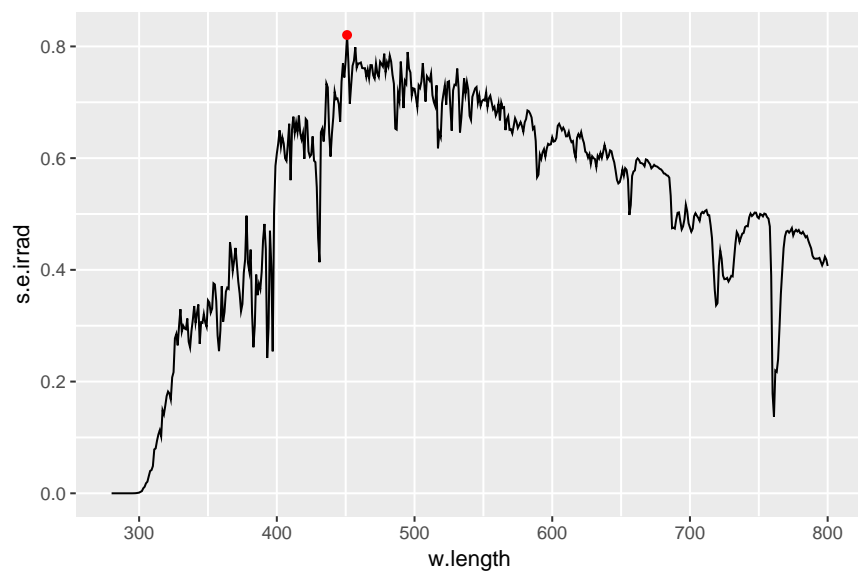

The span, is the number of consecutive wavelength values in the data used to locate each individual maximum. The larger the value used, the 'coarser' the features detec-

ted. It is frequently needed to override the default so as to have only the "meaningful"
features highlighted.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_peaks(span=31, color = "red")
```
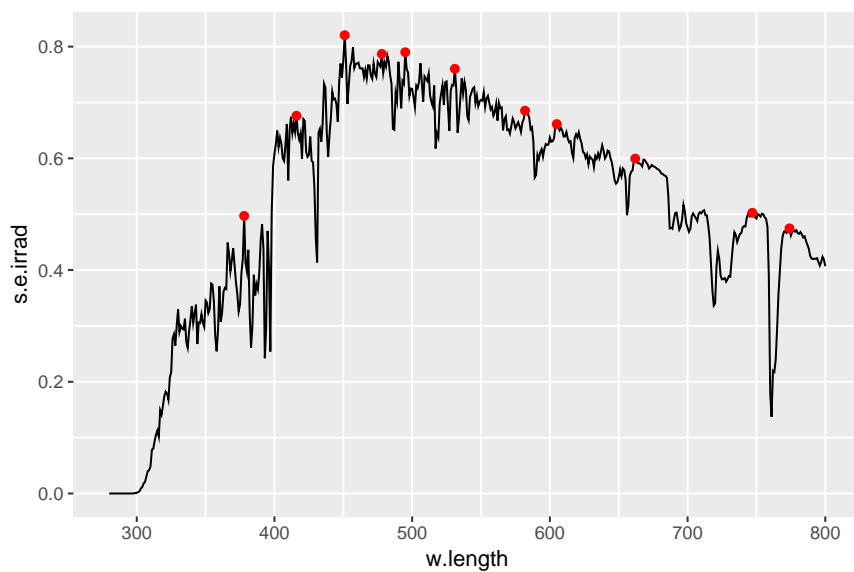

Below we continue 'playing' with package 'ggplot2' to show different ways
of plotting the peaks and valleys. Being 'ggplot2'-compatible `stat_peaks` and
`stat_valleys` accept a `geom` argument and all the aesthetics valid for the chosen
`geom`. By overriding the default geom argument `"point"` with `"text"` we obtain
labels for peaks and valleys. Be aware that consistently with package 'ggplot2', the
`geom` parameter takes as argument a character string giving the name of the `geom`,
rather than the name of the function—in this case `geom_text`.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_peaks(geom = "text", vjust = 0, colour="red", span=31) +
    stat_valleys(geom = "text", vjust = 1, colour="blue", span=51)
```

211
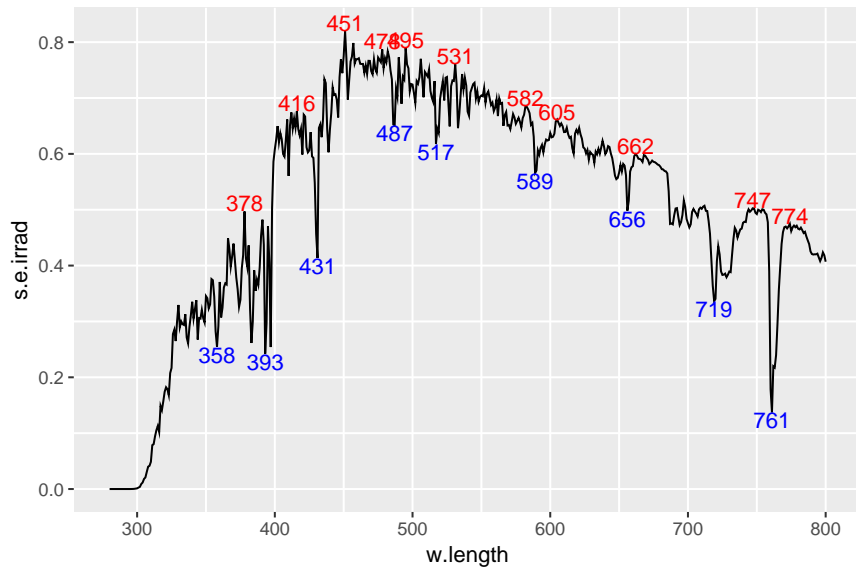
We can use the default `geom`, `geom_point`, but change a few additional aesthetics: we set `shape` to a character, and set its size to 6.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_peaks(colour = "red", shape = "|", size = 6, span = 31)
```
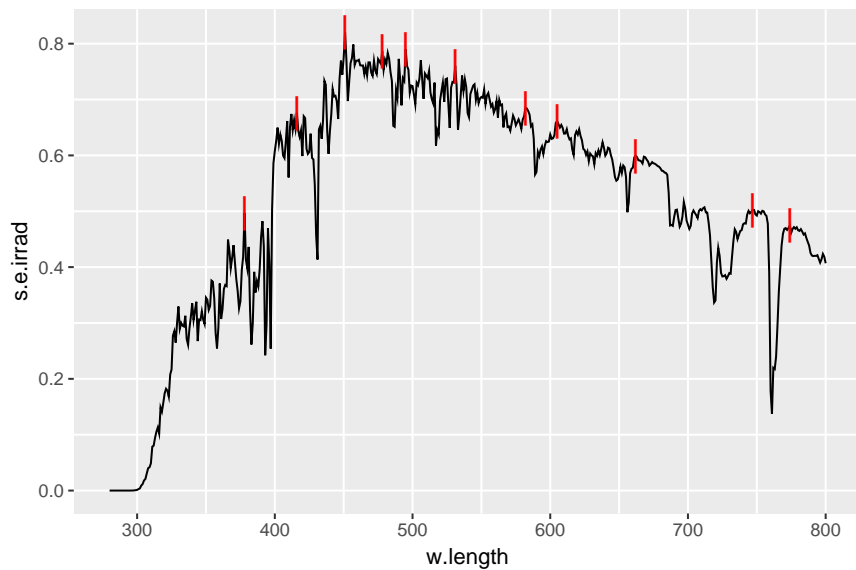

We can add the same `stat` two or more times to a `ggplot` object, in this example, each time with a different `geom`. First we add points to mark the peaks, and afterwards add labels showing the wavelengths at which they are located using geom `"text"`. For the `shape`, or type of symbol, we use one 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[3] In addition we expand the y-axis scale so that all labels fall within the plotting area.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_peaks(colour = "red", shape = 23,
               fill = "white", size = 1.5, span = 31) +
    stat_peaks(colour = "red", geom = "text", vjust = -1, span = 31) +
    expand_limits(y = 0.9)
```
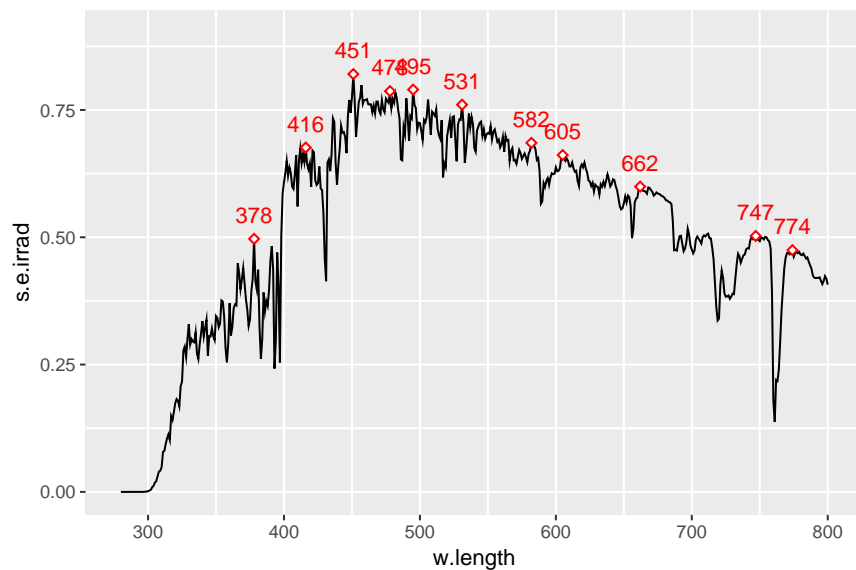

We continue with an example with rotated labels, using different colours for peaks and valleys. As we put peak labels above the spectrum and valleys bellow it, we need to use `hjust` values of opposite sign, but the exact values used were simply adjusted by trial and error until the figure looked as desired.

```
ggplot(data = sun.spct) +
    geom_line() +
```

---

[3]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.

```
    stat_peaks(geom = "text", angle = 90, hjust = -0.1, colour = "red", span = 31) +
    stat_valleys(geom = "text", angle = 90, hjust = 1.1, color = "blue", span = 51) +
    expand_limits(y = 1.0)
```
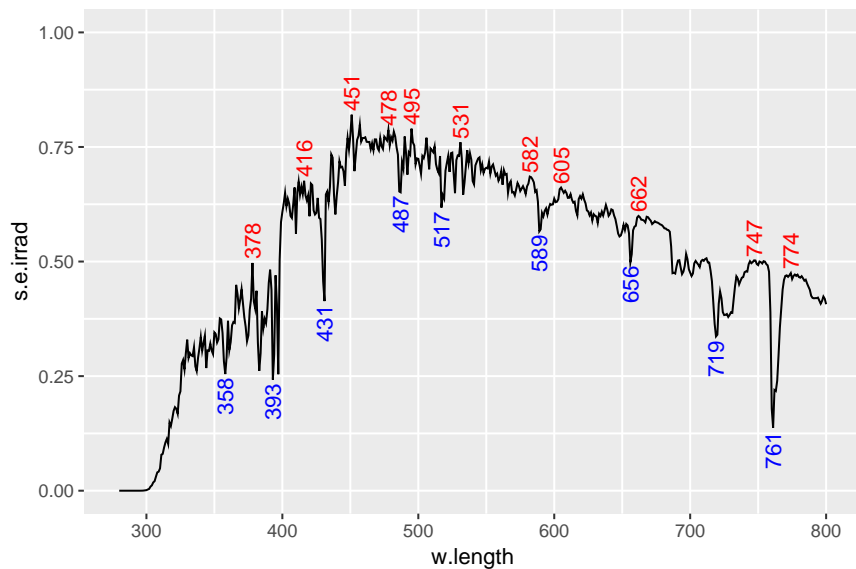

⚠ Be aware that the 'justification' direction, as discussed in the footnote, is referenced to the position of the text, and for this reason to move the rotated labels upwards we need to use `hjust` as the desired displacement is horizontal with respect to the orientation of the text of the label.

The labels mapped to aesthetic can be *computed* on the fly.

```
ggplot(sun.spct) +
  geom_line() +
  stat_peaks(span = NULL, color = "red") +
  stat_peaks(span = NULL, geom = "text", vjust = -0.5, color = "red",
             aes(label = paste(..y.label.., "at", ..x.label.. , "nm"))) +
  expand_limits(y = c(NA, 0.9))
```
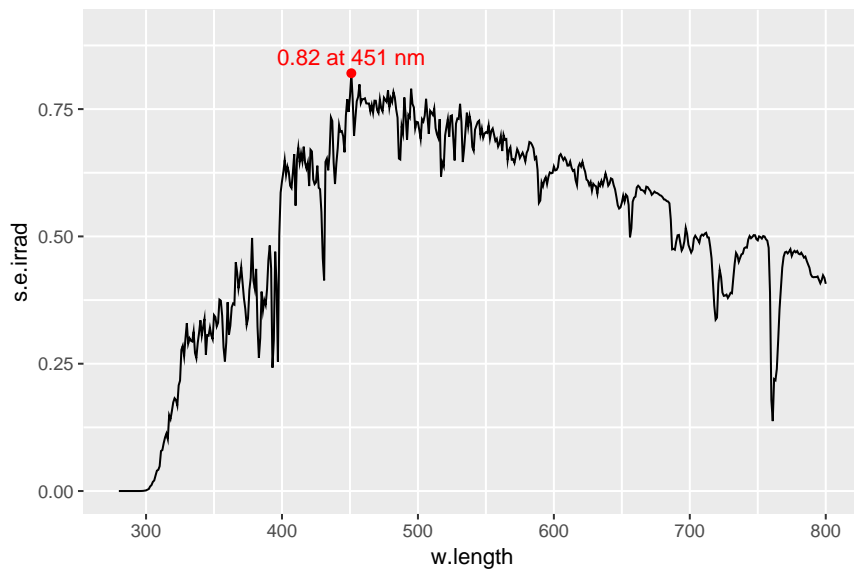
We conclude with a more elaborate plot, with the peaks labelled with wavelengths and color using a repulsive *geom* from package 'ggrepel'.

```
ggplot(sun.spct) +
  geom_line() +
  stat_peaks(shape = 21, span = 25, size = 2) +
  stat_label_peaks(geom = "label_repel", span = 25,
                   size = 3.5, nudge_y = 0.075, segment.colour = "grey20") +
  stat_valleys(shape = 21, span = 25, size = 2) +
  stat_label_valleys(geom = "label_repel", span = 25,
                     size = 3.5, nudge_y = -0.075, segment.colour = "grey20") +
  scale_fill_identity() + scale_color_identity() +
  expand_limits(y = c(-0.08, 0.9))
```
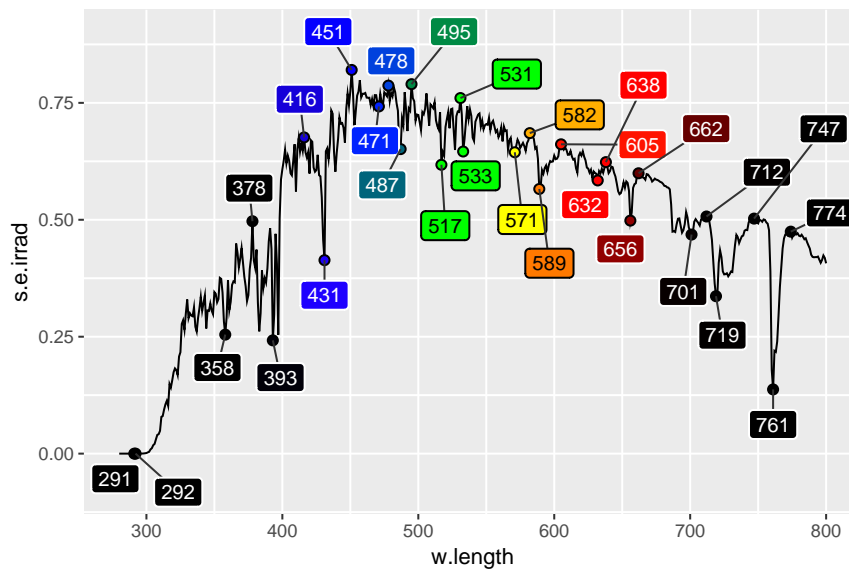
See section 16.8.1 on page sec:plot:effective for an example these stats together with facets.

16.6 Annotating wavebands and wavelengths

Several ggplot-compatible *statistics* are defined in package 'ggspectra' (Table 16.2). An additional high level function `decoration` is also provided to facilitate combining the different stats.

16.6.1 Task: annotate a plot with waveband names as labels

Stat `stat_wb_label` can be used to highlight a waveband in a plot of spectral data. The argument for `w.band` should be a `waveband` object, or a list of such objects. The argument for parameter `geom` should be supplied as a character string. The default `aes` provides defaults for different *geoms*, and the aesthetics can be set also to some of the other computed values. The positions on the x-axis are calculated automatically assuming that it describes wavelengths in nanometres. The vertical position has defaults that are rarely the best, but that allow output that helps with manual positioning. 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`.

Here is an example for PAR using defaults except for `w.band` and `color`.

Table 16.2: Summary `ggplot` statistics defined in package 'ggspectra'

statistic	*description*
Graphic elements	
stat_wb_box	Draw a box showing wavelength range and color
stat_wb_column	Draw a column with area equal to integral and color
stat_wb_hbar	Draw a bar showing mean spectral irradiance, wavelength range and color
stat_wl_strip	Draw bar with colours based wavelength (nm) or wavebands
Summary labels	
stat_wb_contribution	Contribution of the integral of each waveband to the integral under the whole *plotted* curve
stat_wb_relative	Integral for each waveband expressed relative to the sum of all wavebands
stat_wb_label	Label with name of wavelength ranges or wavebands
stat_wb_e_irrad	Energy irradiance for each waveband ($\mathrm{W\,m^{-2}}$)
stat_wb_e_sirrad	Spectral energy irradiance for each waveband ($\mathrm{W\,m^{-2}\,nm^{-1}}$)
stat_wb_q_irrad	Photon irradiance for each waveband ($\mathrm{\mu mol\,m^{-2}\,s^{-1}}$)
stat_wb_q_sirrad	Spectral photon irradiance for each waveband ($\mathrm{\mu mol\,m^{-2}\,s^{-1}\,nm^{-1}}$)
stat_wb_irrad	Irradiance for each wavebands (varies)
stat_wb_sirrad	Spectral irradiance for each wavebands (varies)
stat_wb_total	Integral of the spectral quantity over wavelength for each waveband
stat_wb_mean	Mean of the spectral quantity for each waveband
stat_wl_summary	Average area under curve for regions

16

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_wb_label(w.band = PAR(), color = "black")
```
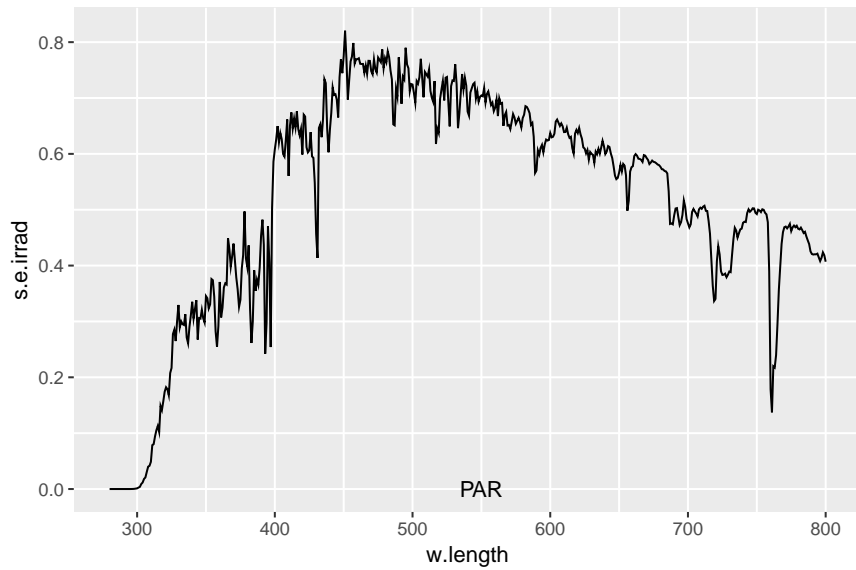
Using the color corresponding to the waveband.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_wb_label(w.band = PAR(), aes(color = ..wb.color..)) +
    scale_color_identity()
```
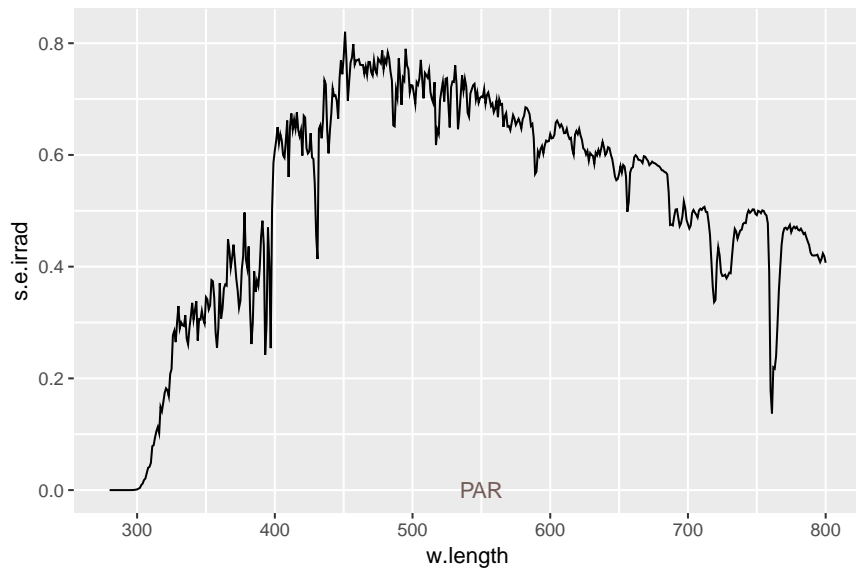

The next example does the annotation using two different *geoms*, `"rect"` for marking the region, and `"text"` for the labels. In this example, the order in which the graphical elements are added is important, as it determines the drawing order. In this case, it is what determines that the text is written on top of the rectangle instead

of the rectangle covering the text.

```
ggplot(data = sun.spct) +
  geom_line() +
  stat_wl_strip(w.band = PAR(), ymax = 0.95, ymin = 0.85) +
  stat_wb_label(w.band = PAR(), ypos.fixed = 0.9, color = "white") +
  ylim(NA, 0.95) +
  scale_fill_identity()
```
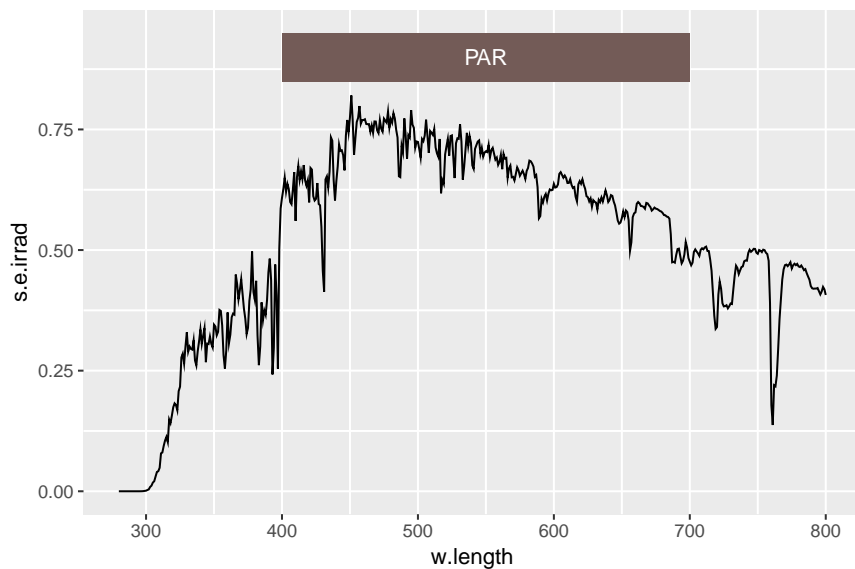

Or with automatic color settings, which in this cases results in the same plot.

```
ggplot(data = sun.spct) +
  geom_line() +
  stat_wl_strip(w.band = PAR(), ymax = 0.95, ymin = 0.85) +
  stat_wb_label(w.band = PAR(), ypos.fixed = 0.9) +
  ylim(NA, 0.95) +
  scale_fill_identity() + scale_color_identity()
```
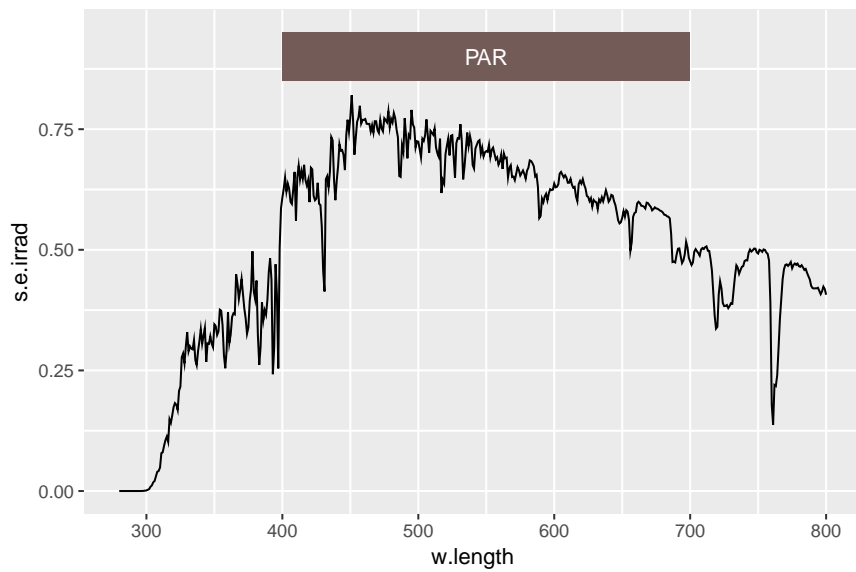
Now an example that is more complex, and demonstrates the flexibility of plots produced with 'ggplot2'. We add annotations for all the colors defined in the ISO standard. For each one we use, as above, two *geoms*. We can also see in this example that the annotations look nicer on a white background, which can be obtained with `theme_bw`. In this case, the effect of automatic selection of black or white text is important.

```
ggplot(data = sun.spct) +
    geom_line() +
    stat_wl_strip(w.band = VIS_bands(), ymax = 1.05, ymin = 0.85) +
    stat_wb_label(w.band = VIS_bands(), ypos.fixed = 0.95, angle = 90) +
    scale_fill_identity() + scale_color_identity() +
    ylim(NA, 1.05)
```
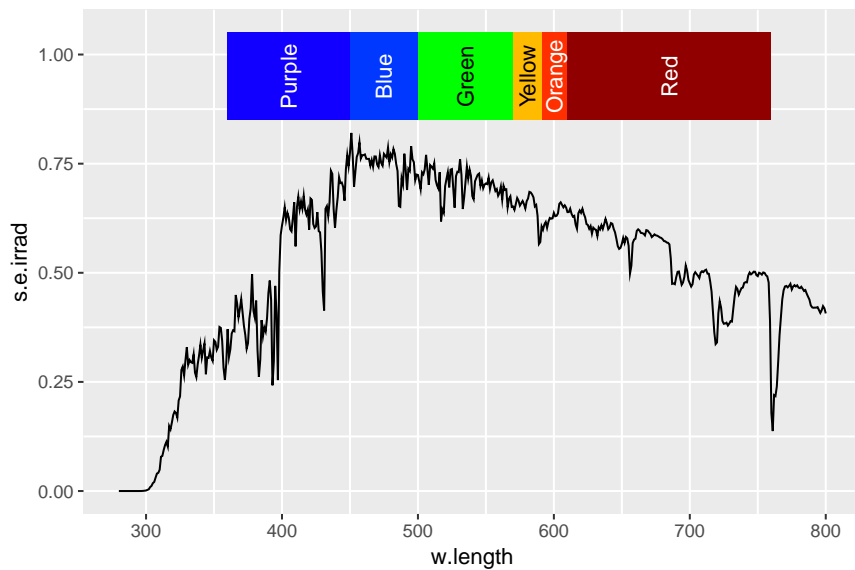
geom_vline without need of any ggplot *statistic* allows to delimit a waveband. We use method range for waveband objects and geom_vline to plot two vertical lines highlighting the wavelength boundaries of the waveband.

```
ggplot(data=sun.spct, unit.out = "photon") +
    geom_line() +
    geom_vline(xintercept=range(PAR()))
```
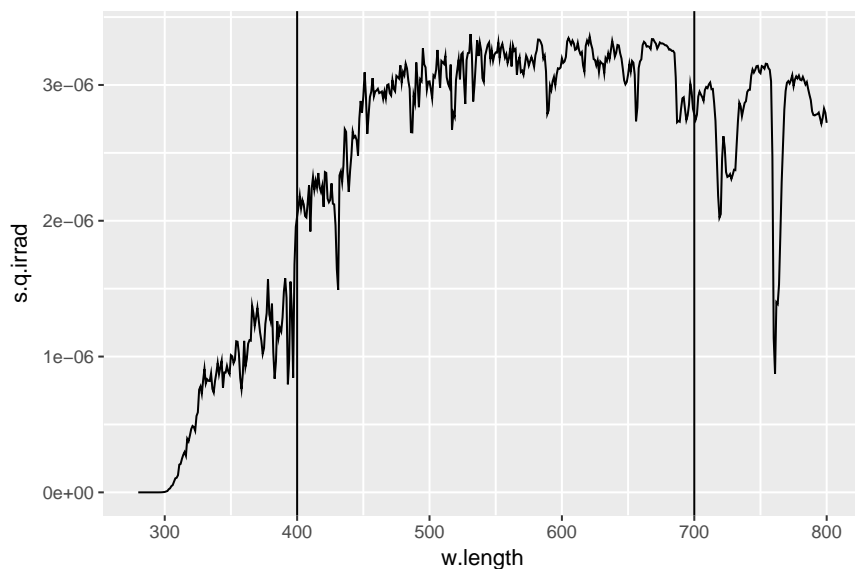

We change aesthetics to obtain dashed lines, and add a label using stat_wb_label as described above.

```
ggplot(data=sun.spct, unit.out = "photon") +
    geom_line() +
    geom_vline(xintercept=range(PAR()), linetype="dashed") +
    stat_wb_label(w.band = PAR(), color = "black")
```
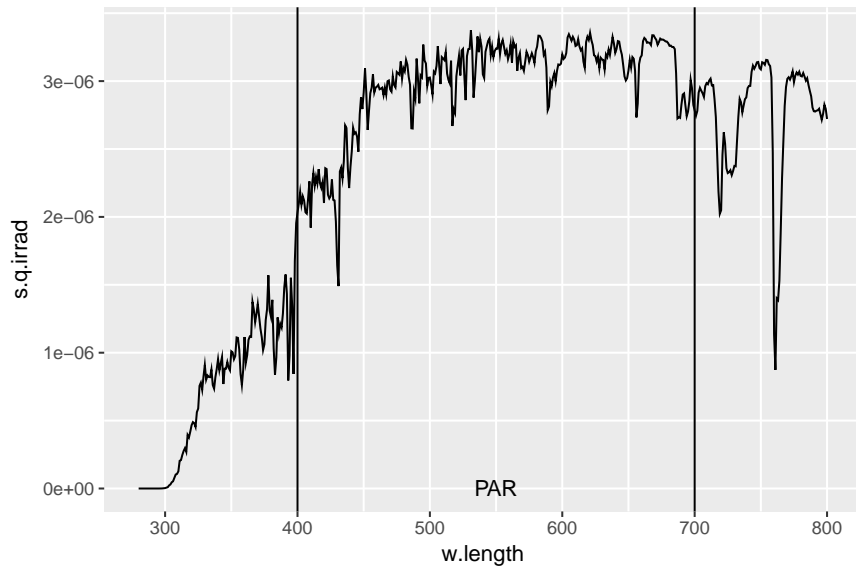

16.6.2 Task: annotate a plot with waveband summary values as labels

Next we use calculated values in the label, first for a simple example with a single waveband. Because we use expressions to obtain superscripts we need to add `parse=TRUE` to the call. We use photon based units for expressing both the spectral irradiance and the integral.

```
ggplot(sun.spct, unit.out = "photon") +
    geom_line() +
    stat_wl_summary(geom = "hline", color = "red") +
    stat_wl_summary(label.fmt = "Mean = %.3g", color = "red", vjust = -0.3)
```
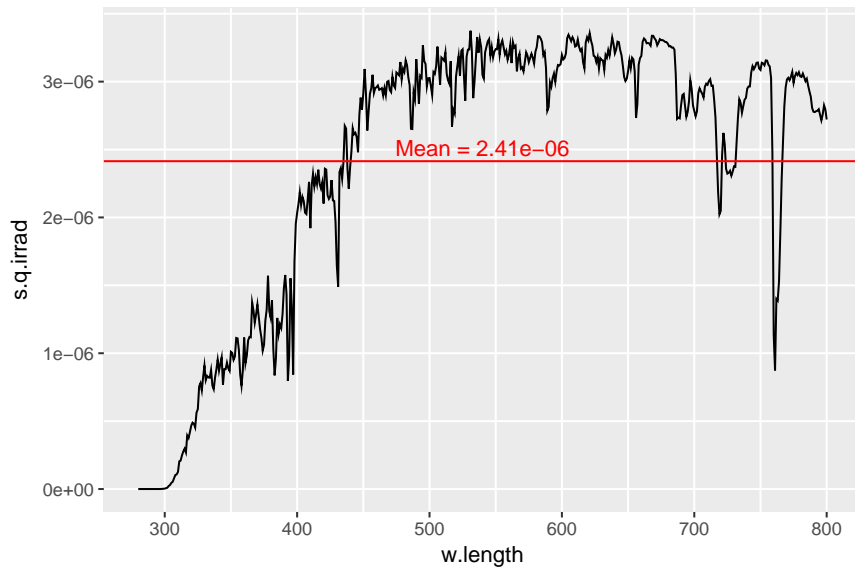
With the two-spectra data object, we use *facets* to plot each spectrum on a separate panel. By default, the y-scale is common to all panels.

```
ggplot(two_suns.spct) + aes(color = spct.idx) +
    geom_line() +
    stat_wl_summary(geom = "hline") +
    stat_wl_summary(label.fmt = "Mean = %.3g", vjust = 1.2, show.legend = FALSE) +
    facet_grid(spct.idx~.)
```
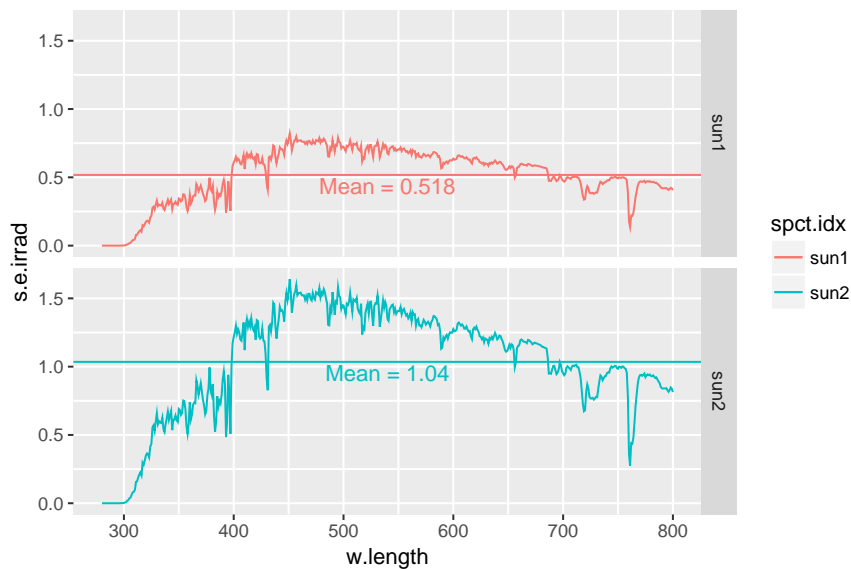

Here we set free scales, so that they can differ between panels.

```r
ggplot(two_suns.spct) + aes(color = spct.idx) +
    geom_line() +
    stat_wl_summary(geom = "hline") +
    stat_wl_summary(label.fmt = "Mean = %.3g", vjust = 1.2, show.legend = FALSE) +
    facet_grid(spct.idx~., scales = "free_y")
```
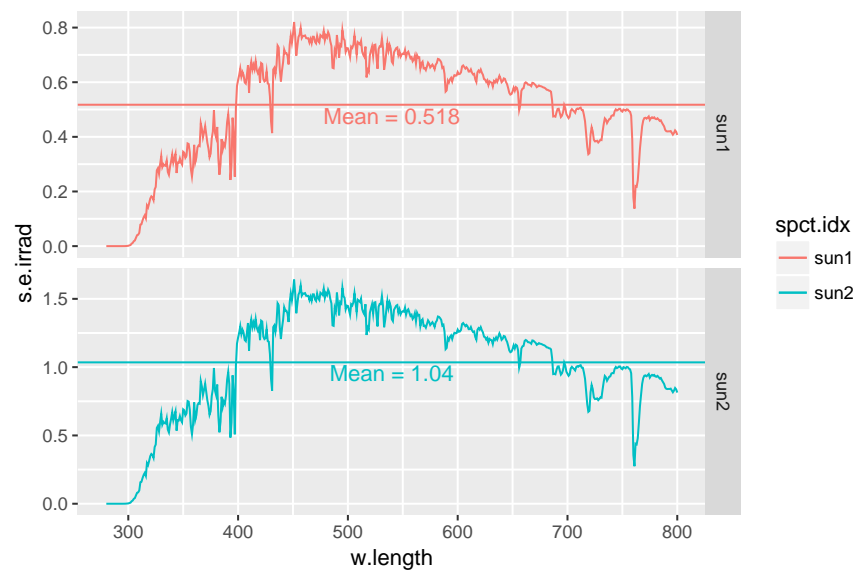


```r
ggplot(yellow_gel.spct) +
    geom_line() +
    stat_wb_hbar(w.band = Plant_bands()) +
    stat_wb_mean(w.band = Plant_bands(), angle = 90, hjust = 0, ypos.fixed = 0.1,
                 aes(label = paste(..wb.name.., " = ", ..y.label.., sep = ""),
                     color = "black")) +
scale_color_identity()
```
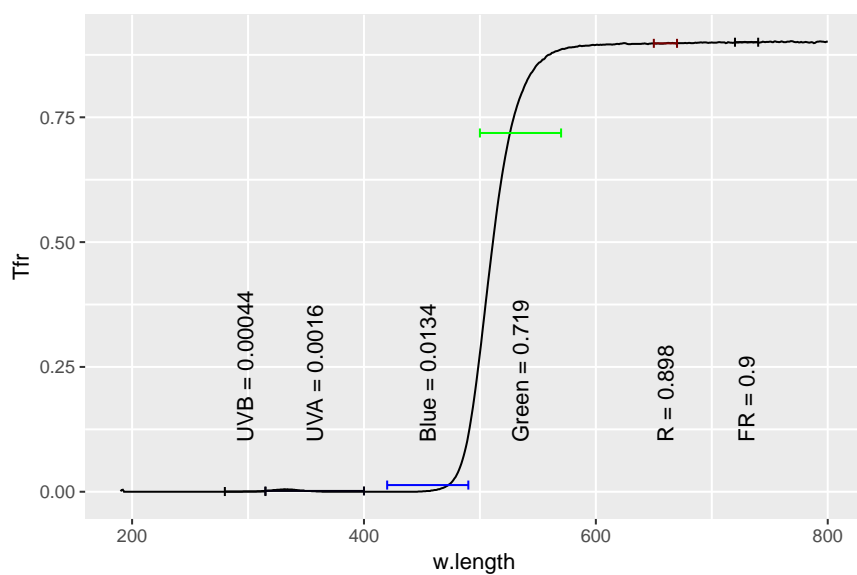
```
ggplot(sun.spct, unit.out = "photon") +
    geom_line() +
    stat_wb_box(w.band = PAR()) +
    stat_wb_irrad(w.band = PAR(),
                  unit.in = "photon", time.unit = "second",
                  aes(label = sprintf("%s = %.3g", ..wb.name.., ..yint.. * 1e6))) +
    scale_color_identity() +
    scale_fill_identity()
```
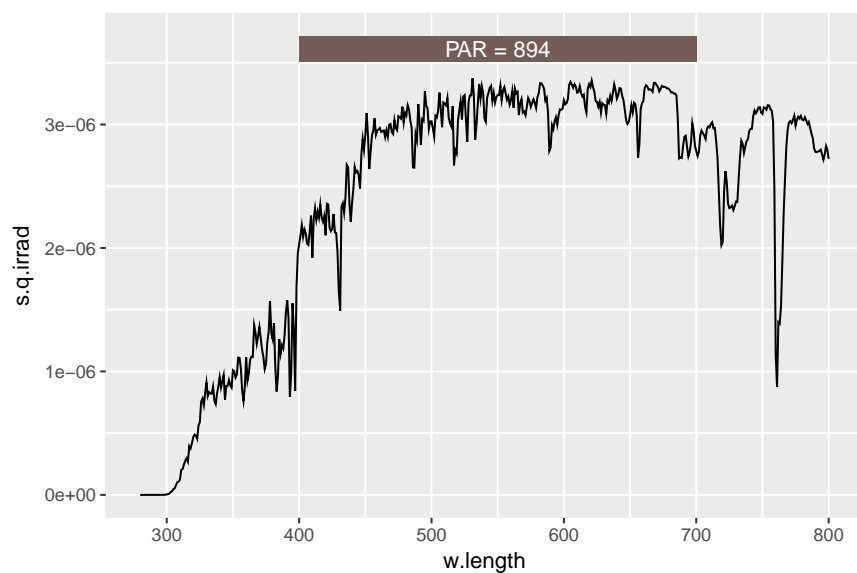
```
ggplot(sun.spct, unit.out = "photon") +
stat_wb_column(w.band = PAR(), alpha = 0.5) +
    stat_wb_irrad(w.band = PAR(),
                  unit.in = "photon", time.unit = "second",
                  aes(label = sprintf("%s = %.3g", ..wb.name.., ..yint.. * 1e6),
                      color = ..wb.color..)) +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```
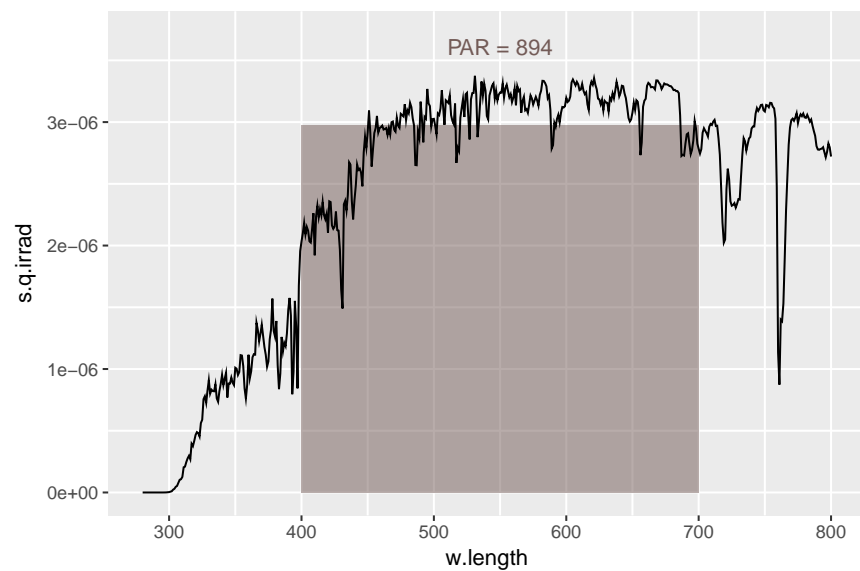


```
ggplot(sun.spct, unit.out = "photon") +
    stat_wb_box(w.band = VIS_bands(),
                   color = "white") +
    stat_wb_q_irrad(w.band = VIS_bands(),
               label.mult = 1e6, size = 2.5) +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```
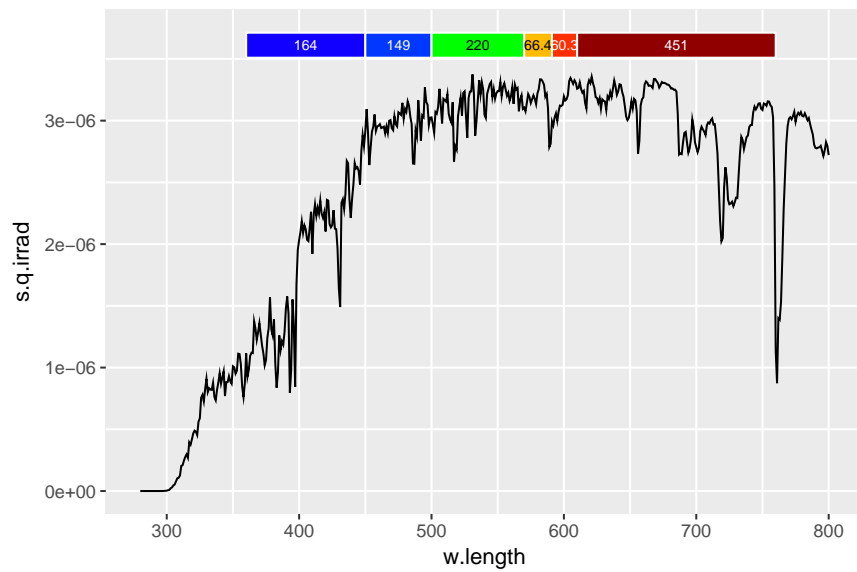
```r
ggplot(sun.spct, unit.out = "photon") +
    stat_wb_column(w.band = PAR(), alpha = 0.5) +
    stat_wb_q_irrad(w.band = PAR(),
                aes(label = sprintf("Total %s = %.3g", ..wb.name.., ..yint.. * 1e6)),
                ypos.mult = 0.55) +
    stat_wb_q_sirrad(w.band = PAR(),
            mapping = aes(label = sprintf("Mean %s = %.3g", ..wb.name.., ..ymean.. * 1e6)),
                ypos.mult = 0.45) +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```


227
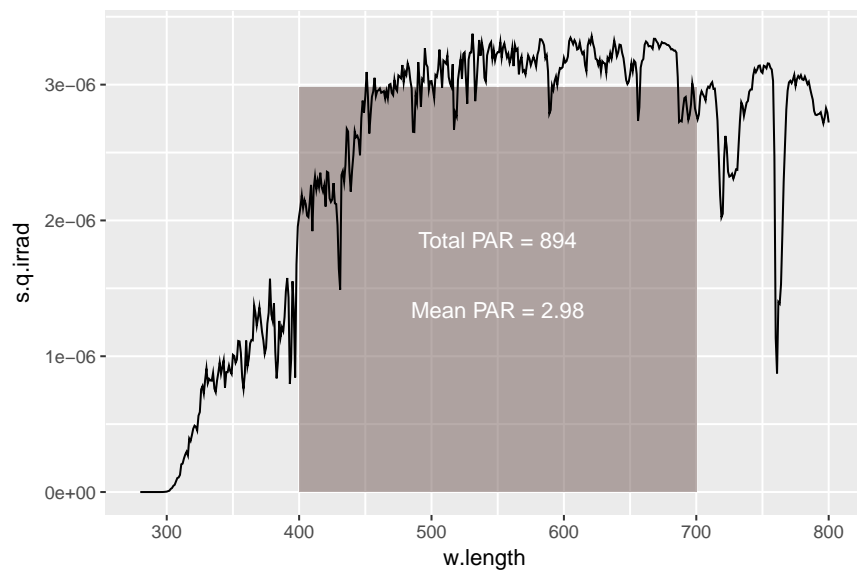
```
ggplot(sun.spct, unit.out = "photon") +
    stat_wb_column(w.band = VIS_bands(), alpha = 0.45) +
    stat_wb_q_irrad(w.band = VIS_bands(), angle = 90,
                    label.mult = 1e6, ypos.fixed = 1e-7, hjust = 0,
                    aes(label = paste(..wb.name.., ..y.label.., sep = " = "))) +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```
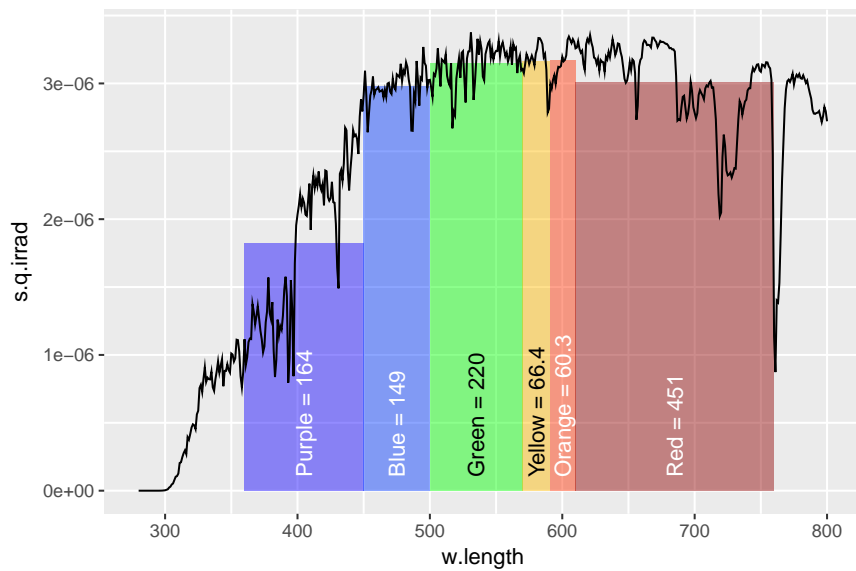


```
my.bands <- split_bands(c(300,800), length.out = 10)
ggplot(sun.spct) +
    stat_wb_column(w.band = my.bands, color = NA, alpha = 0.75) +
    stat_wb_e_irrad(w.band = my.bands, angle = 90,
                    ypos.fixed = 0.05, hjust = 0) +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```
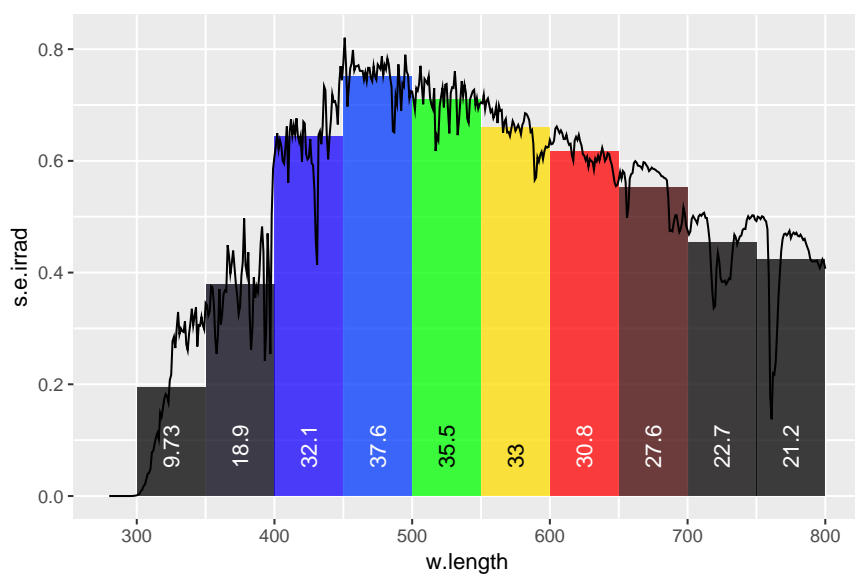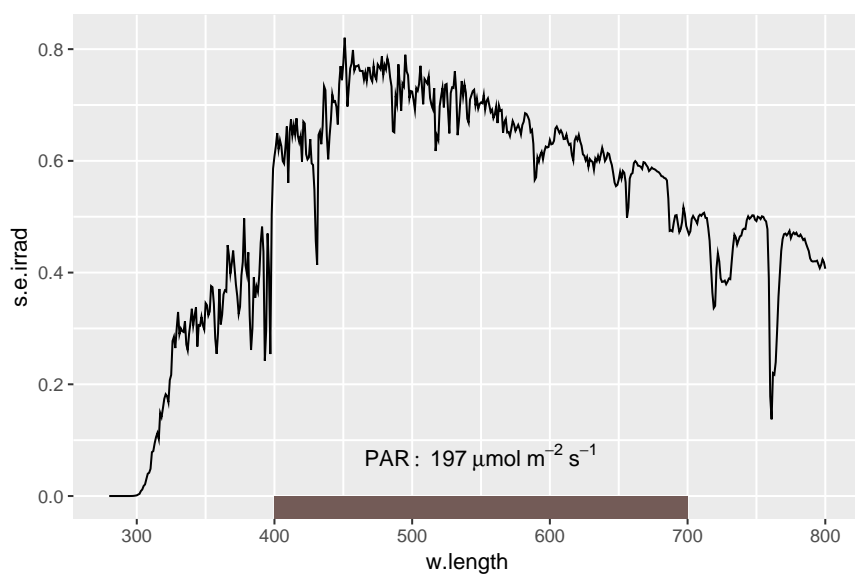
```
ggplot(sun.spct) +
    stat_wb_column(w.band = PAR(), alpha = 0.2) +
    stat_wb_irrad(w.band = PAR(),
                  aes(label=paste("PAR:~", ..y.label..,
                                  "*~mu*mol~m^{-2}~s^{-1}", sep="")),
                  ypos.mult = 0.5,
                  parse=TRUE,
                  time.unit = "second",
                  unit.in = "energy") +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```
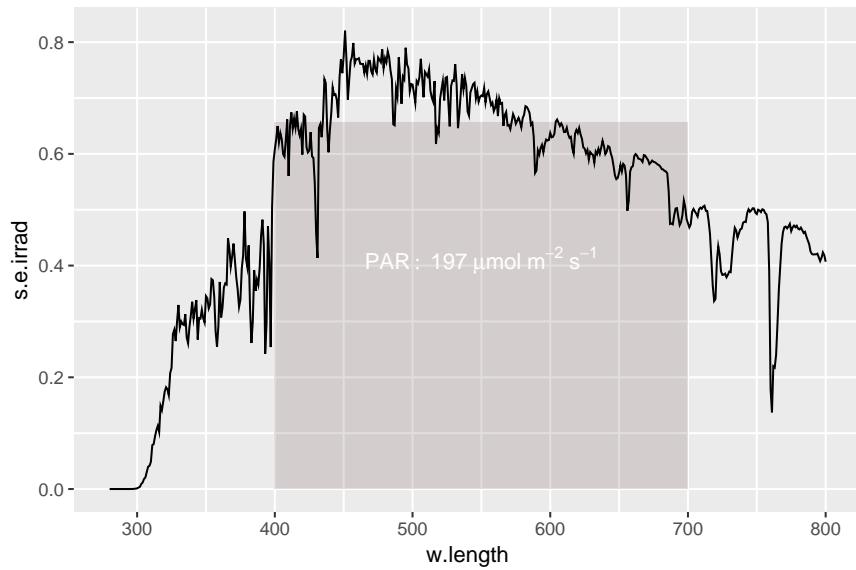
Using smaller rectangles for annotation, yields plots where the spectrum itself is easier to see than when the rectangle overlaps the spectrum. We achieve this by supplying as argument both `ymax` and `ymin`, and slightly reducing the size of the text with `size = 4`.

```r
ggplot(sun.spct) +
    stat_wl_strip(w.band = PAR(),
                  ymin = -0.05, ymax = 0) +
    stat_wb_irrad(w.band = PAR(), size=4,
                  aes(label=paste("PAR:~", ..y.label..,
                                  "*~mu*mol~m^{-2}~s^{-1}", sep="")),
                  ypos.fixed = 0.07,
                  colour="black",
                  parse=TRUE,
                  time.unit = "second",
                  unit.in = "energy") +
    geom_line() +
    scale_fill_identity()
```

A bigger rectangle better shows the range of wavelengths.

```
ggplot(sun.spct) +
    stat_wl_strip(w.band = PAR(),alpha = 0.2,
                  ymin = -Inf, ymax = +Inf) +
    stat_wb_irrad(w.band = PAR(), size=4,
                  aes(label=paste("PAR:~", ..y.label..,
                                  "*~mu*mol~m^{-2}~s^{-1}", sep="")),
                  colour="black",
                  parse=TRUE,
                  time.unit = "second",
                  unit.in = "energy") +
    geom_line() +
    scale_fill_identity()
```
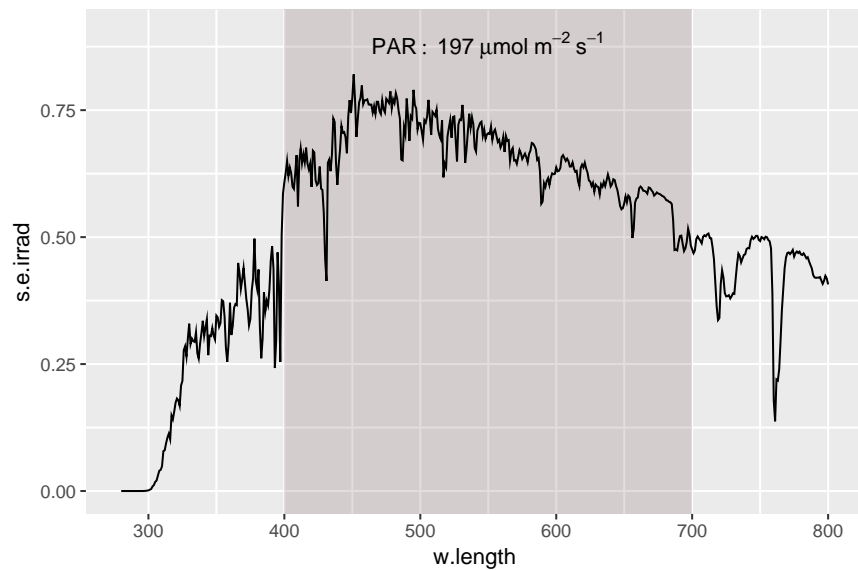
This type of annotations can be also easily done for effective exposures or doses, but in this example as we position the annotations manually, we can use ggplot2's 'normal' `annotate` function. We use `xlim` to restrict the plotted region of the spectrum to the range of wavelengths of interest.

```
ggplot(sun.spct, unit.out = "photon") +
    geom_vline(xintercept = range(CIE()), linetype = "dashed") +
    stat_wb_e_irrad(w.band = CIE(), color = "black",
                unit.in = "photon", time.unit = "second",
                aes(label = sprintf("%s\n%.3g", ..wb.name.., ..yeff.. * 1e6))) +
    geom_line() +
    scale_color_identity() +
    scale_fill_identity()
```
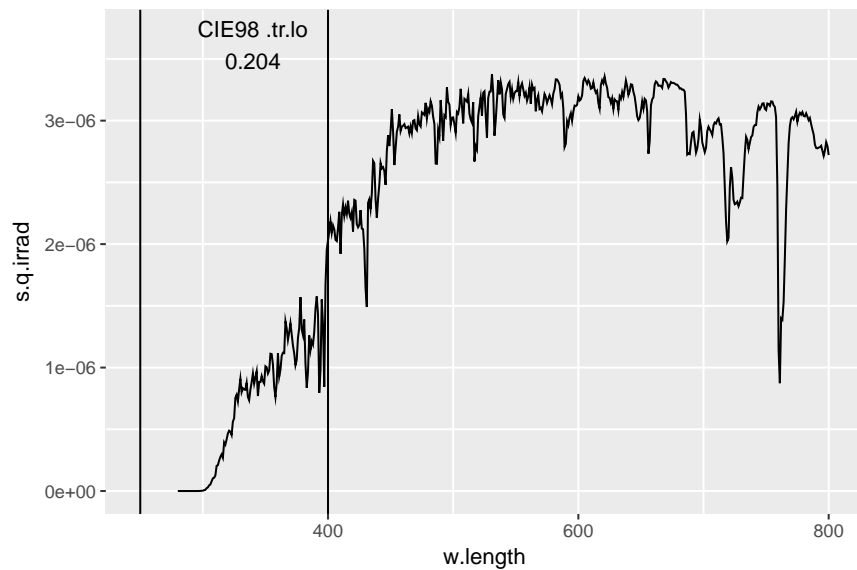
And finally some figures with multiple spectra, and effective doses.

```
filtered.lst <- list(none = sun.daily.spct,
                     pet = sun.daily.spct * polyester.spct,
                     yellow = sun.daily.spct * yellow_gel.spct)
filtered.spct <- rbindspct(filtered.lst, idfactor = "filter")
```

```
ggplot(filtered.spct, unit.out = "energy") +
  stat_wb_hbar(w.band = CIE(), ypos.fixed = 28000) +
  stat_wb_e_irrad(w.band = CIE(), color = "black",
                  unit.in = "energy", time.unit = "day",
                  ypos.fixed = 31000) +
  geom_line() +
  facet_wrap(~filter, ncol = 1) +
  scale_color_identity()
```

233
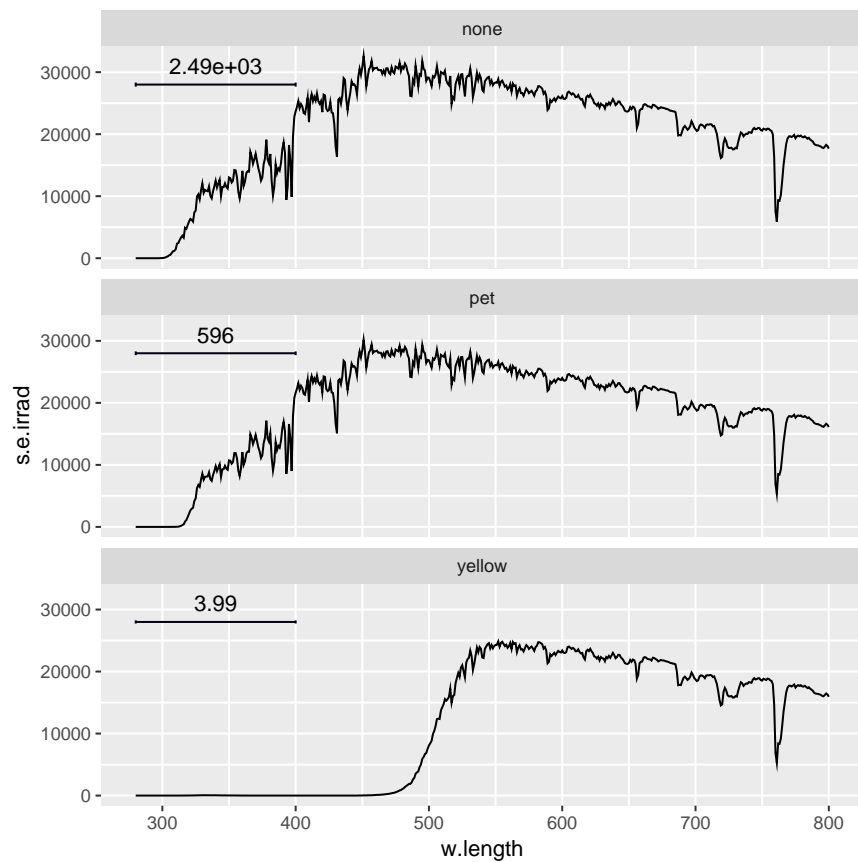
```
ggplot(filtered.spct, unit.out = "energy") +
  geom_line() +
  stat_wb_hbar(w.band = CIE(), ypos.fixed = 0, color = "red") +
  stat_wb_e_irrad(w.band = CIE(), color = "red", size = 2,
                  unit.in = "energy", time.unit = "day", label.mult = 1e-3,
                  label.fmt = "CIE %3.2f kJ/day",
                  ypos.fixed = -3000) +
  facet_wrap(~filter, ncol = 1) +
  scale_color_identity()
```
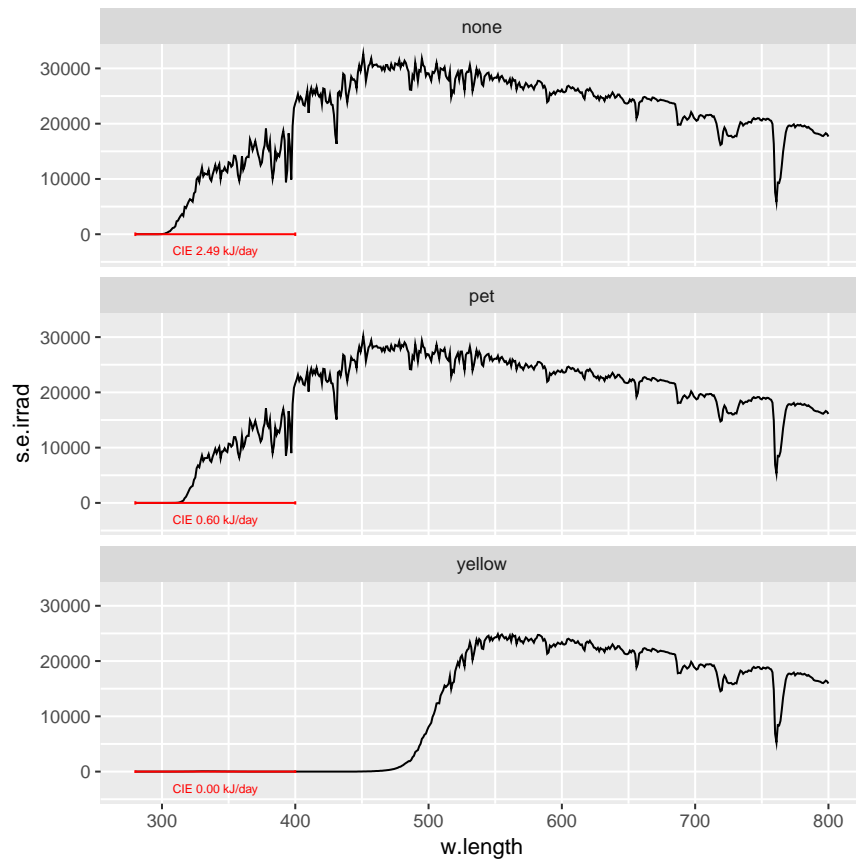
16.7 Using colour as data in plots

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 16.7.1 we show code based on tagging spectral data, and in section 16.7.2 the case of using different data for plotting the guide or key is described.

### 16.7.1 Task: Plots using colour for the spectral data

We will first show how to generate the colour information from wavelengths by means of *statistics* defined in 'ggspectra' and later give some examples of using colors by combining functions from packages 'photobiology' and 'ggplot2'. `stat_wl_summary()` and `stat_wb_mean()` return a *reduced* data set with fewer rows than the original data. `stat_wl_strip()` depending on the input, can also return a data set with more rows than the input data. This is the default behaviour, with `w.band` with argument `NULL`. This *stat* operates only on the variable mapped to the x *aesthetic*, a y-mapping is not required as input. The *stats* with `wb` in their name have a parameter `w.band` to which `waveband` objects or lists of `waveband` objects can be passed as arguments, while *stats* with `wl` in their name have also a parameter `range` to which a `numeric` vector of length two can be passed, or any R object on which function `range()` returns such a vector, interpretable as a range of wavelengths expressed in nanometres.

```
my.data <- data.frame(x = 300:800)
ggplot(my.data, aes(x)) +
    stat_wl_strip(ymin = -1, ymax = 1) +
    scale_fill_identity() + scale_color_identity() +
    scale_y_continuous(breaks = NULL)
```
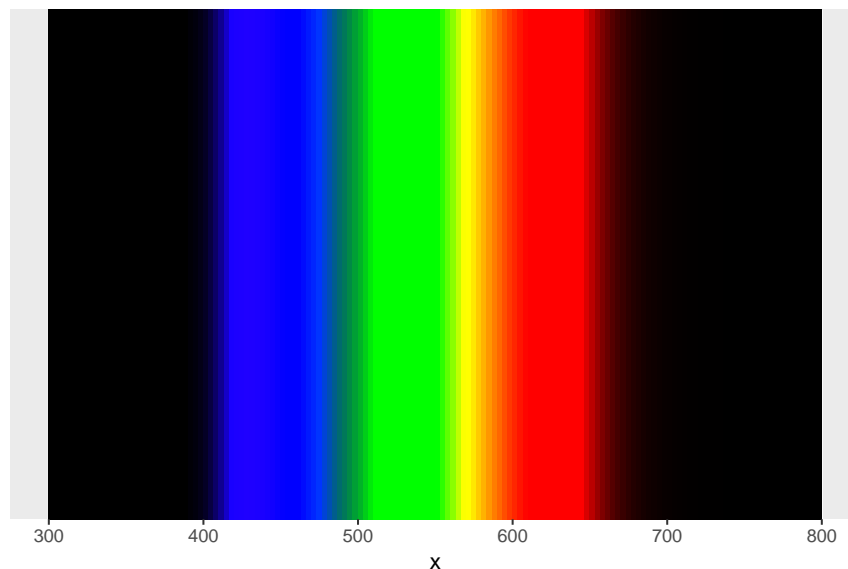

The default *geom* is `"rect"` which with suitable mappings of `ymin` and `ymax` *aesthetics* can be used to add a colour guide to the x-axis (if its scale maps wavelengths in nanometres). When `w.band` is `NULL` a long series of contiguous wavebands is generated to create the illusion of a continuous colour gradient.

```
ggplot(sun.spct) +
    stat_wl_strip(ymin = -Inf, ymax = -0.025) +
    geom_line() +
    scale_fill_identity()
```
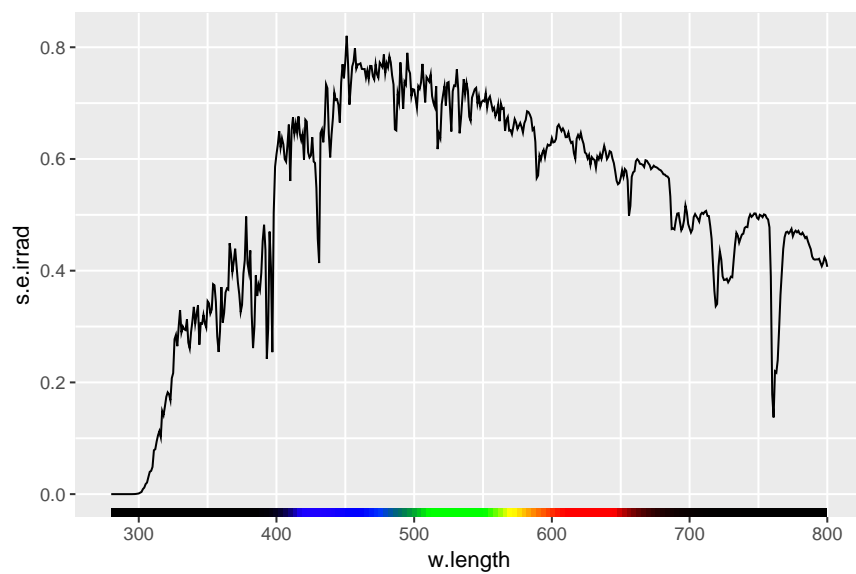

When a list of wavebands is supplied, it is used for the calculation of colours. These calculations do not use the spectral irradiance, they simply assume a flat spectrum, so the represent the colours of the wavelengths, rather than the colour of the light described by the spectral irradiance.

```
ggplot(sun.spct) +
    stat_wl_strip(w.band = VIS_bands(), ymin = -Inf, ymax = -0.025) +
    geom_line() +
    scale_fill_identity()
```

237
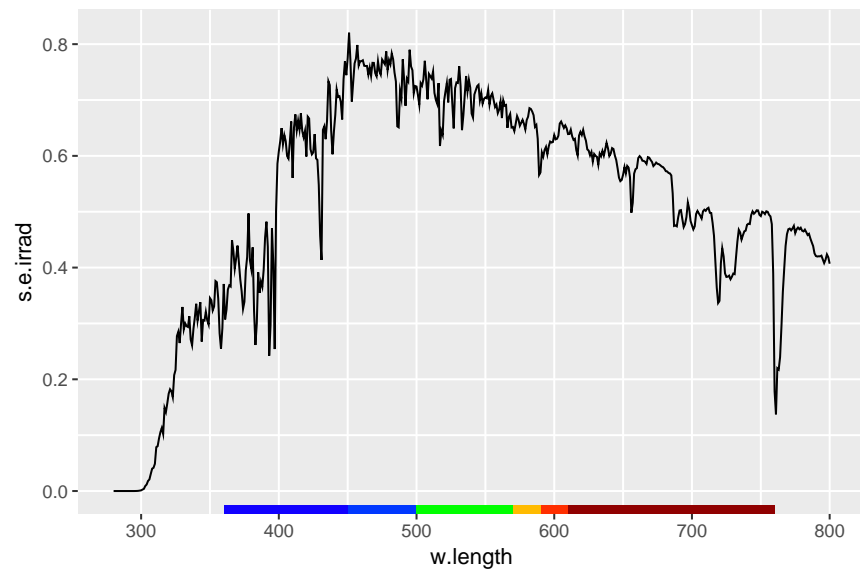
stat_wl_strip() can also used to produce a background layer with colours corresponding to *wavebands*. Here we use `alpha = 0.5` to add transparency.

```
ggplot(sun.spct) +
    stat_wl_strip(w.band = VIS_bands(), ymin = -Inf, ymax = Inf, alpha = 0.4) +
    geom_line() +
    scale_fill_identity()
```
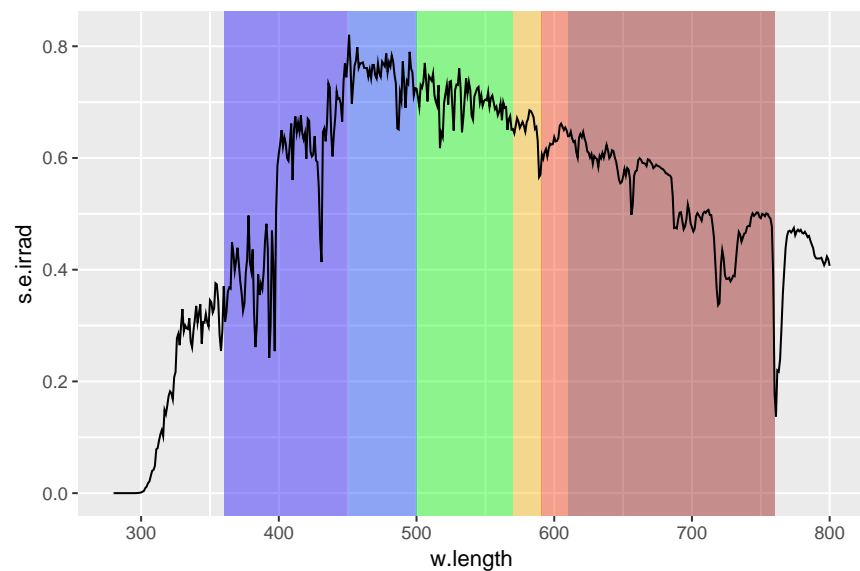

By default an almost continuous colour gradient can be generated for the background.

```
ggplot(sun.spct) +
stat_wl_strip(alpha = 0.4, ymin = -Inf, ymax = Inf) +
    geom_line() +
    scale_fill_identity()
```
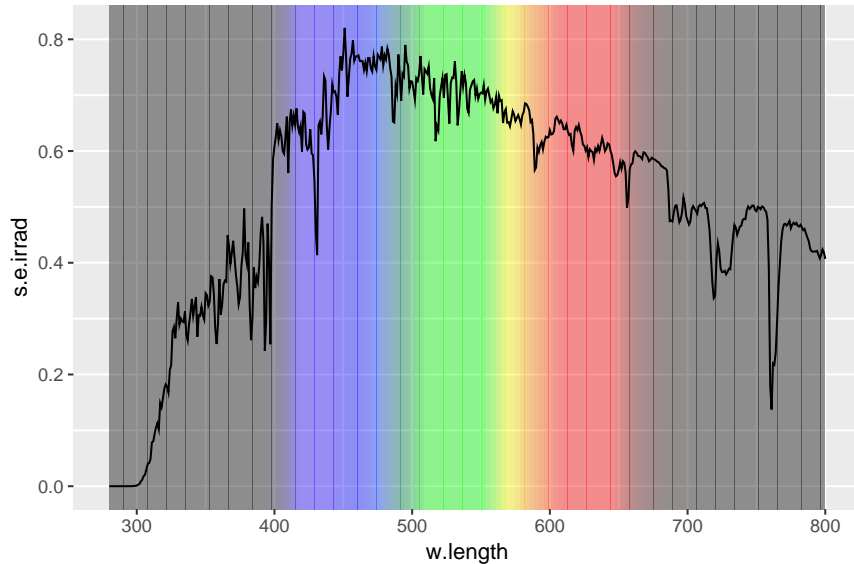

Next, we show the most flexible, but also most verbose way of plotting using colors derived from the wavelength values in the data. Instead of using a *statistic* in the plot, we can *tag* the spectra before hand with color information. This can be useful if we wish to apply some additional processing to the generated color data or if the same spectrum will be plotted repeatedly as tagging the data before plotting avoid recomputing the color for each geom or plot that uses them.

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 <- tag(sun.spct)
cp.sun.spct

## Object: source_spct [523 x 6]
## Wavelength range 280 to 800 nm, step 1.023182e-12 to 1 nm
## Time unit 1s
##
## # A tibble: 523 × 6
##   w.length s.e.irrad s.q.irrad wl.color wb.color
##      <dbl>     <dbl>     <dbl>    <chr>    <chr>
## 1 280.0000         0         0  #000000  #554340
## 2 280.9231         0         0  #000000  #554340
## 3 281.8462         0         0  #000000  #554340
## 4 282.7692         0         0  #000000  #554340
## # ... with 519 more rows, and 1 more variables:
```

```
## #   wb.f <fctr>
```

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 <- tag(sun.spct, UVB())
levels(uvb.sun.spct[["wb.f"]])

## [1] "UVB"
```

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.

```
uvb.sun.spct <- 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

levels(uvb.sun.spct[["wb.f"]])

## [1] "UVB"
```

We provide no example for this, but tagging an already tagged spectrum replaces the old tagging data with the new one, with a warning.

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 <- tag(sun.spct, Plant_bands())
levels(plant.sun.spct[["wb.f"]])

## [1] "UVB"    "UVA"    "Blue"   "Green" "R"
## [6] "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.tags")

## $time.unit
## [1] "second"
##
## $wb.key.name
## [1] "Bands"
```

```
##
## $wl.color
## [1] TRUE
##
## $wb.color
## [1] TRUE
##
## $wb.num
## [1] 1
##
## $wb.colors
## [1] "#554340"
##
## $wb.names
## [1] "Total"
##
## $wb.list
## $wb.list[[1]]
## Total
## low (nm) 280
## high (nm) 800
## weighted none
```

```r
attr(uvb.sun.spct, "spct.tags")
```

```
## $time.unit
## [1] "second"
##
## $wb.key.name
## [1] "Bands"
##
## $wl.color
## [1] TRUE
##
## $wb.color
## [1] TRUE
##
## $wb.num
## [1] 1
##
## $wb.colors
## [1] "black"
##
## $wb.names
## [1] "UVB"
##
## $wb.list
## $wb.list[[1]]
## UVB.ISO
```

```
## low (nm) 280
## high (nm) 315
## weighted none
```

We now tag a spectrum for use in our first plot example.

```
par.sun.spct <- tag(sun.spct, PAR())
```

We can also use other `geom` s like `geom_area` in the next chunk, together with, as an example, a grey fill scale from 'ggplot2'. We need to add `scale_fill_identity` so that the values of the `fill` aesthetic are used as colour definitions rather that factor levels to be mapped a different colour scale.

```
ggplot(par.sun.spct) +
 geom_area(color = NA, aes(fill = wb.color)) +
 geom_line() +
 scale_fill_identity(na.value = NA)
```
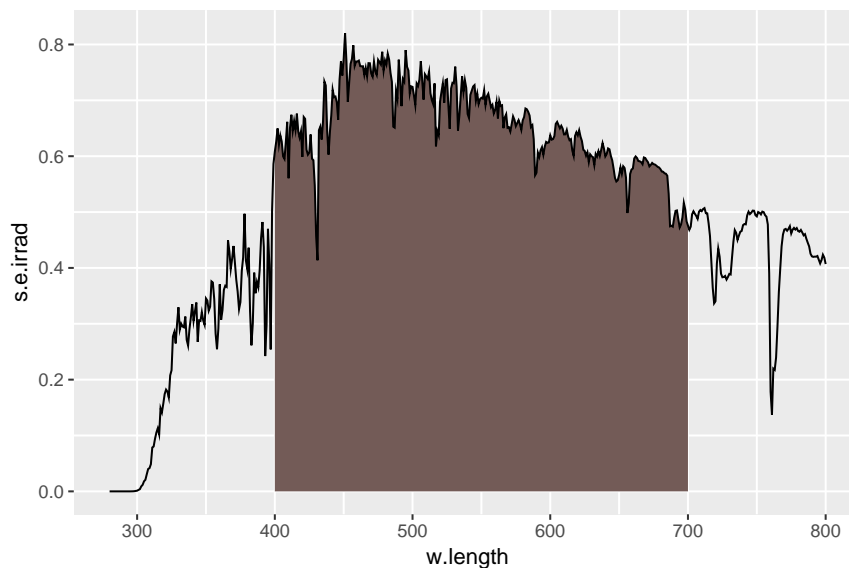

We tag each observation in the solar spectrum with human vision colours as defined by ISO.

```
tg.sun.spct <- tag(sun.spct, VIS_bands())
```

The colours when displayed on screens (additive devices, as they emit light) are most frequently defined by three *channels*: red, green and blue (RGB). These three numbers can be represented in different ways. In R it is common to use character strings of hexadecimal digits. The choice of these three colours is not arbitrary, they

are those perceived by the three photoreceptors in our eyes[4].

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` s used by 'ggplot2' do not accept a `fill` aesthetic, while shape `21` gives circles that can be *filled*.

```
ggplot(tg.sun.spct) +
  geom_line() +
  geom_point(aes(fill = wb.color), shape = 21) +
  scale_fill_identity()
```
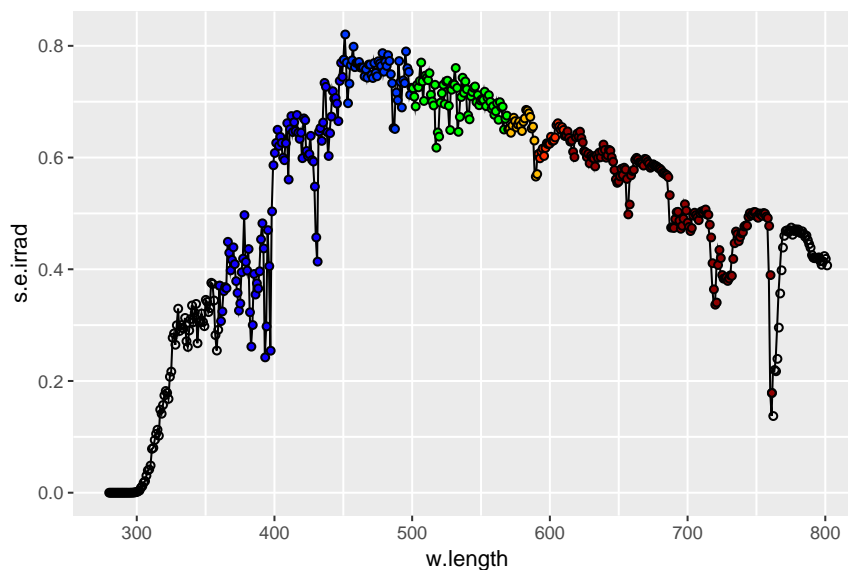

Using `geom_area` we can fill the area under the curve according to the colour of different wavebands, we set the fill only for this geom, so that the `NA` s do not affect other plotting. To get a single black curve for the spectrum we use `geom_line` . This approach works as long as wavebands do not share the same value for the color, which means that it is not suitable either when any band is outside the visible range, or when using many narrow wavebands.

```
ggplot(tg.sun.spct) +
  scale_fill_identity() + scale_color_identity() +
  geom_line() +
  geom_spct(aes(fill = wb.color), alpha = 0.75)
```

---

[4]In other words almost all screen devices and printed material based on dithering of three colours just trick our eyes into seeing certain colours, even if they do not actually exist on the screen or paper. Consequently, we cannot expect other animals, like our pets to make any sense out of TV screens or colour photographs!
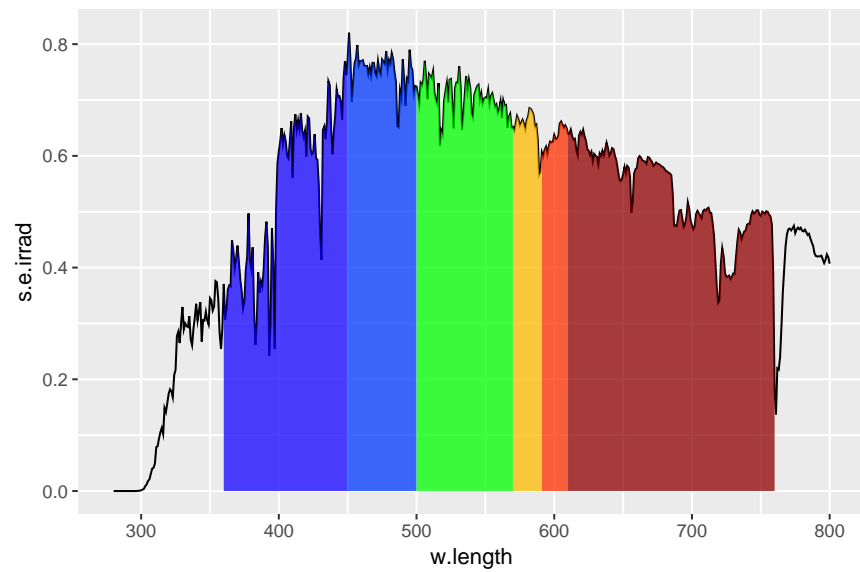
In the next example we tag the solar spectrum with colours using the definitions of plant sensory 'colours'.

```r
pl.sun.spct <- tag(sun.spct, Plant_bands())
```

Here we plot the wavebands corresponding to plant sensory 'colours', using the spectrum we tagged in the previous code chunk.

```r
ggplot(pl.sun.spct) +
geom_spct(aes(fill = wb.color), alpha = 0.75) +
  geom_line() +
  scale_fill_identity() + scale_color_identity()
```
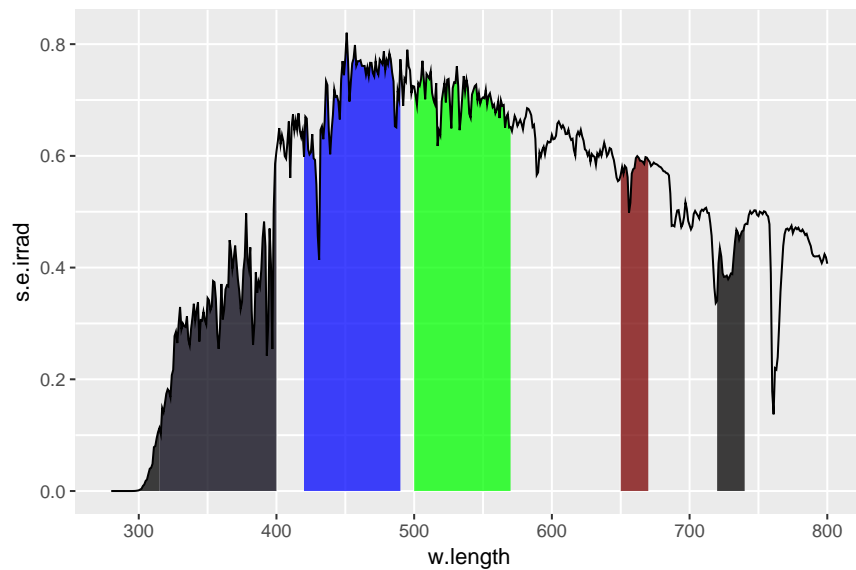
We can also use the factor `wb.f` which has value `NA` outside the wavebands, changing the colour used for `NA` to `NA` which renders it invisible. We can change the labels used for the wavebands in two different way, when plotting by supplying a labels argument to the scale used, or when tagging the 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.

```
ggplot(pl.sun.spct) +
  geom_spct(aes(fill = wb.f)) +
  geom_line() +
  scale_fill_grey(na.value = NA, name = "",
                  labels=c("UVB", "UVA", "Blue", "Green", "Red", "Far red"))
```
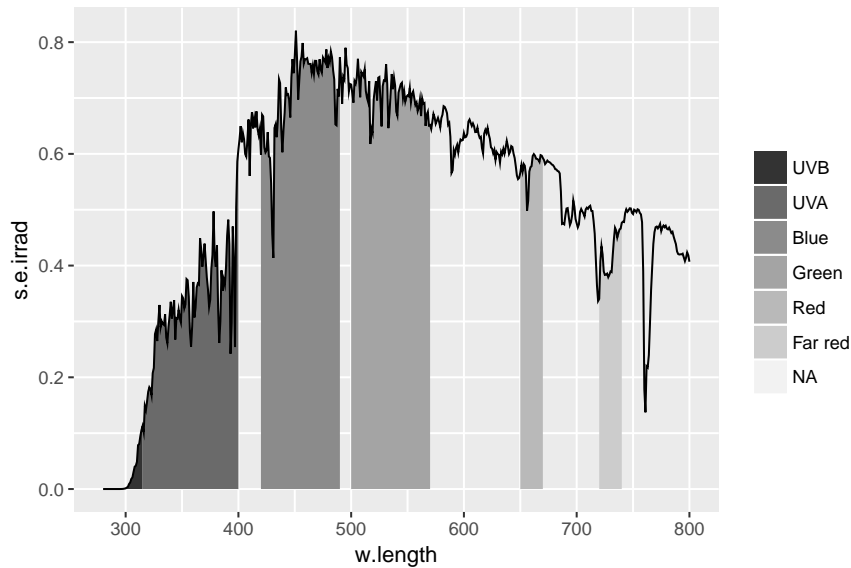
When using a factor we can play with the scale definitions and represent the wavebands in any way we may want. For example we can use `split_bands` to split a waveband or spectrum into many adjacent narrow bands and get an almost continuous gradient, but we need to get around the problem of repeated 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.

```
dense.sun.spct <- 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. However, the longer spectrum should not be used for statistical calculations, not even plotting using `geom_smooth`.

```
splt.sun.spct <- tag(sun.spct, split_bands(VIS(), length.out = 150))
```

In the code above, we made a copy of `sun.spct` using a different name so as not hide the object in package 'photobiology'.

```
ggplot(splt.sun.spct) +
    geom_spct(aes(fill = wb.color), alpha = 0.75) +
    geom_line() +
    scale_fill_identity() + scale_color_identity()
```
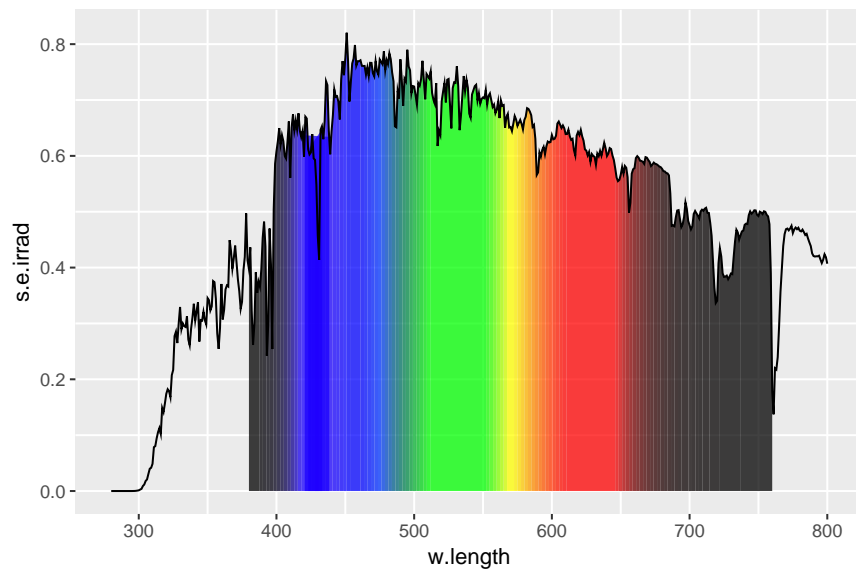
In this other example we tag the whole spectrum, dividing it into 200 wavebands.

```
splt1.sun.spct <- tag(sun.spct, split_bands(sun.spct, length.out = 200))
```

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.

```
ggplot(splt1.sun.spct) +
    geom_spct(aes(fill = wb.color), alpha = 0.75) +
    geom_line() +
    scale_fill_identity() + scale_color_identity()
```
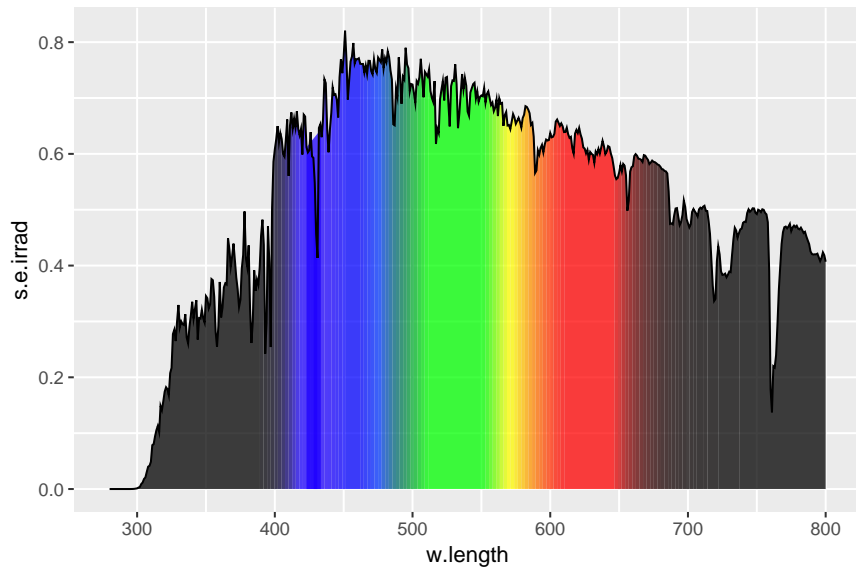
The next example uses `geom_point` and `color` to color the data points according to the waveband they are included in.

```
ggplot(tg.sun.spct) +
    geom_line() +
    geom_point(aes(color=wb.color), na.rm = TRUE) +
    scale_colour_identity()
```
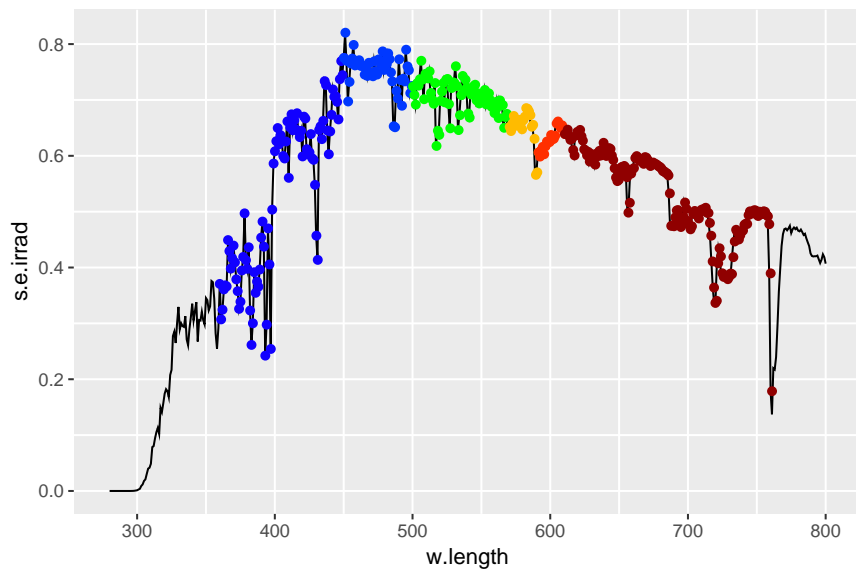

When plotting individual observations as points, instead of using colours from wavebands, we will want to plot the colour calculated for each individual wavelength value, which `tag` adds to the spectrum, whether a waveband definition is supplied

or not. In this case we need to use `scale_color_identity` .

```
ggplot(data=tg.sun.spct) +
    scale_color_identity() +
    geom_point(aes(color = wl.color))  +
    geom_line()
```
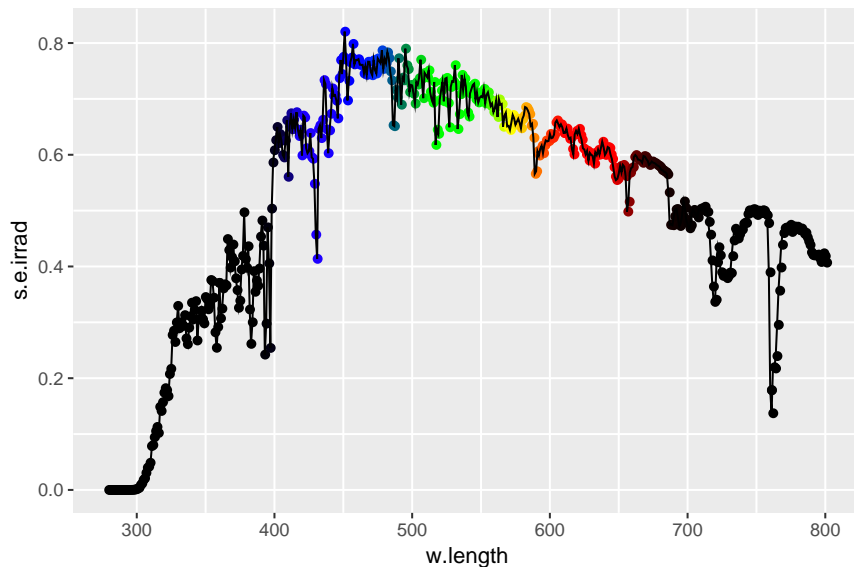

Other possibilities are for example, using one of the symbols that can be filled, and then for example for symbols with a black border and a colour matching its wavelength as a fill aesthetic. It is also possible to use `alpha` with points.

16.7.2 Task: 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 `geom` s. 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())
```

```
## Object: generic_spct [4 x 8]
```

249

```
## Wavelength range 400 to 700 nm, step 1.023182e-12 to 300 nm
##
## # A tibble: 4 × 8
##   w.length counts   cps s.e.irrad s.q.irrad   Tfr
##      <dbl> <dbl> <dbl>     <dbl>     <dbl> <dbl>
## 1      400     0     0         0         0     0
## 2      400     0     0         0         0     0
## 3      700     0     0         0         0     0
## 4      700     0     0         0         0     0
## # ... with 2 more variables: Rfl <dbl>,
## #   s.e.response <dbl>

wb2spct(Plant_bands())

## Object: generic_spct [22 x 8]
## Wavelength range 280 to 740 nm, step 1.023182e-12 to 85 nm
##
## # A tibble: 22 × 8
##   w.length counts   cps s.e.irrad s.q.irrad   Tfr
##      <dbl> <dbl> <dbl>     <dbl>     <dbl> <dbl>
## 1      280     0     0         0         0     0
## 2      280     0     0         0         0     0
## 3      315     0     0         0         0     0
## 4      315     0     0         0         0     0
## # ... with 18 more rows, and 2 more variables:
## #   Rfl <dbl>, s.e.response <dbl>
```

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())

## Object: generic_spct [4 x 12]
## Wavelength range 400 to 700 nm, step 1.023182e-12 to 300 nm
##
## # A tibble: 4 × 12
##   w.length counts   cps s.e.irrad s.q.irrad   Tfr
##      <dbl> <dbl> <dbl>     <dbl>     <dbl> <dbl>
## 1      400     0     0         0         0     0
## 2      400     0     0         0         0     0
## 3      700     0     0         0         0     0
## 4      700     0     0         0         0     0
## # ... with 6 more variables: Rfl <dbl>,
## #   s.e.response <dbl>, wl.color <chr>,
## #   wb.color <chr>, wb.f <fctr>, y <dbl>

wb2tagged_spct(Plant_bands())
```

```
## Object: generic_spct [22 x 12]
## Wavelength range 280 to 740 nm, step 1.023182e-12 to 85 nm
##
## # A tibble: 22 × 12
##   w.length counts   cps s.e.irrad s.q.irrad   Tfr
##      <dbl>  <dbl> <dbl>     <dbl>     <dbl> <dbl>
## 1      280      0     0         0         0     0
## 2      280      0     0         0         0     0
## 3      315      0     0         0         0     0
## 4      315      0     0         0         0     0
## # ... with 18 more rows, and 6 more variables:
## #   Rfl <dbl>, s.e.response <dbl>,
## #   wl.color <chr>, wb.color <chr>, wb.f <fctr>,
## #   y <dbl>
```

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())
```

```
## Object: generic_spct [1 x 15]
## Wavelength range 550 to 550 nm, step NA nm
##
## # A tibble: 1 × 15
##   w.length counts   cps s.e.irrad s.q.irrad   Tfr
##      <dbl>  <dbl> <dbl>     <dbl>     <dbl> <dbl>
## 1      550      0     0         0         0     0
## # ... with 9 more variables: Rfl <dbl>,
## #   s.e.response <dbl>, wl.color <chr>,
## #   wb.color <chr>, wb.name <chr>, wb.f <fctr>,
## #   wl.high <dbl>, wl.low <dbl>, y <dbl>
```

```
wb2rect_spct(Plant_bands())
```

```
## Object: generic_spct [6 x 15]
## Wavelength range 297.5 to 730 nm, step 60 to 125 nm
##
## # A tibble: 6 × 15
##   w.length counts   cps s.e.irrad s.q.irrad   Tfr
##      <dbl>  <dbl> <dbl>     <dbl>     <dbl> <dbl>
## 1    297.5      0     0         0         0     0
## 2    357.5      0     0         0         0     0
## 3    455.0      0     0         0         0     0
## 4    535.0      0     0         0         0     0
## # ... with 2 more rows, and 9 more variables:
## #   Rfl <dbl>, s.e.response <dbl>,
## #   wl.color <chr>, wb.color <chr>,
## #   wb.name <chr>, wb.f <fctr>, wl.high <dbl>,
## #   wl.low <dbl>, y <dbl>
```

16

251

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
## [1] "#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 `geom`s and also examples combining the use of tagged spectra as described in the previous section with the 'annotations' described here.

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 wave-bands. 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))

ggplot(data=sun.spct) +
  geom_line() +
  geom_rect(data  =wl.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                fill = wb.color),
            color = NA) +
  scale_fill_identity()
```
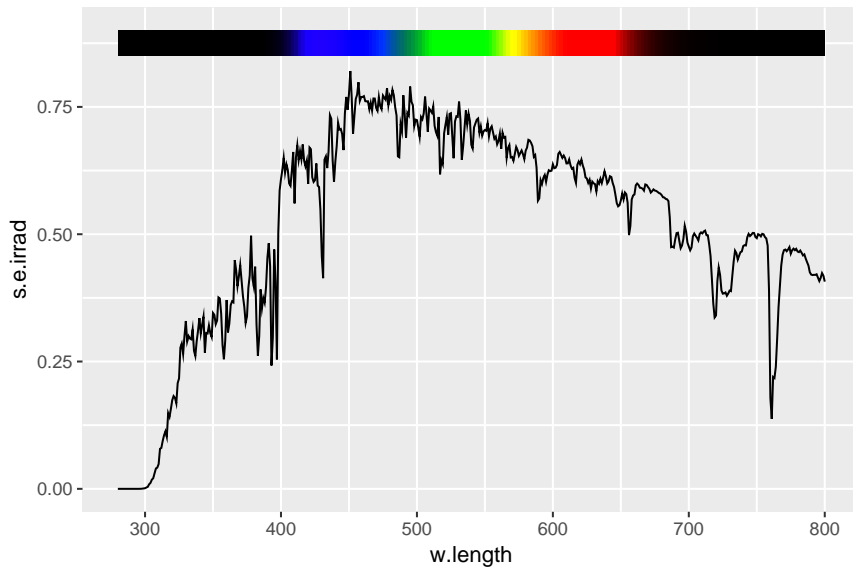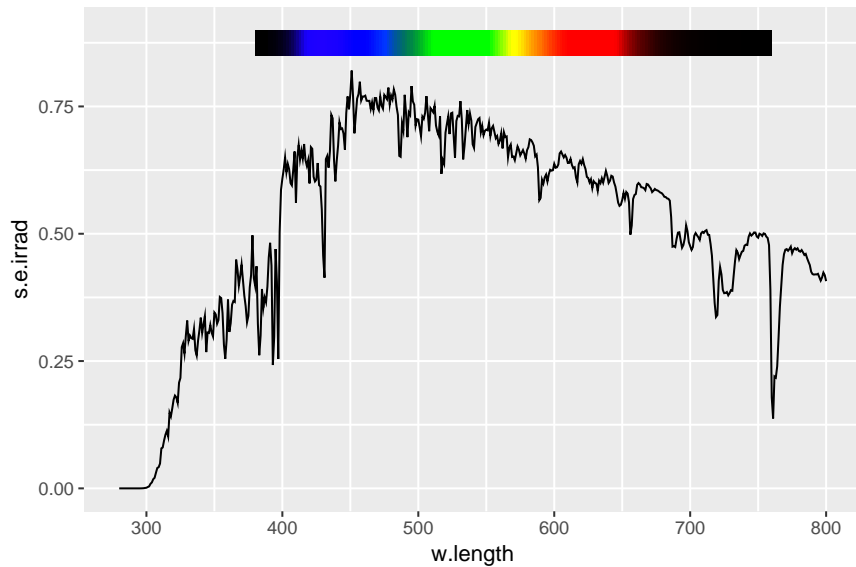

This second example differs very little from the previous one, but by using a wave-band definition instead of a spectrum as argument to `split_bands`, we restrict the region covered by the colour fill to that of the waveband. In fax a vector of length two, or any object for which a `range` method is available can be used as input to this function.

```
wl.guide.spct <- wb2rect_spct(split_bands(VIS(), length.out = 200))
```

253

```
ggplot(data=sun.spct) +
    geom_line() +
    geom_rect(data=wl.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                fill=wb.color),
            color = NA) +
    scale_fill_identity()
```


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` .
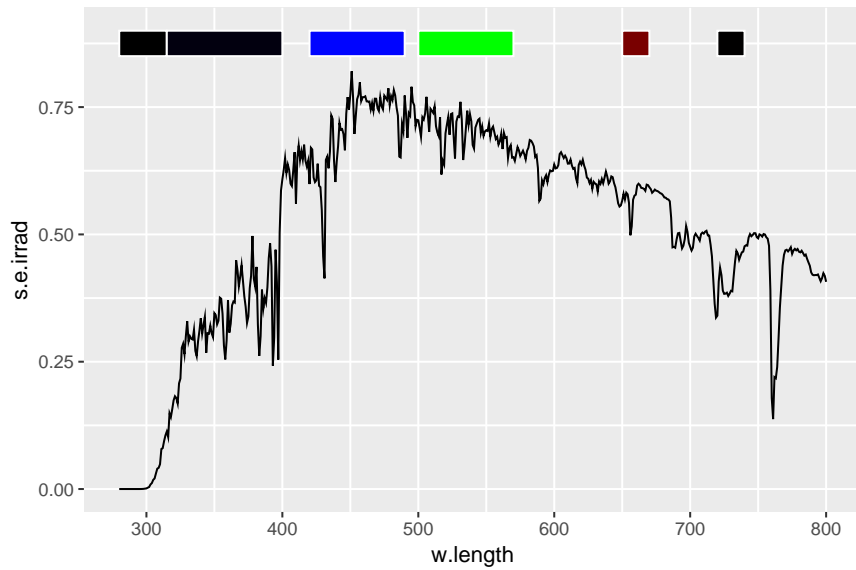
```
iso.guide.spct <- wb2rect_spct(VIS_bands())

ggplot(data=sun.spct) +
    geom_line() +
    geom_rect(data=iso.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                fill = wb.color),
            color = NA) +
    scale_fill_identity()
```
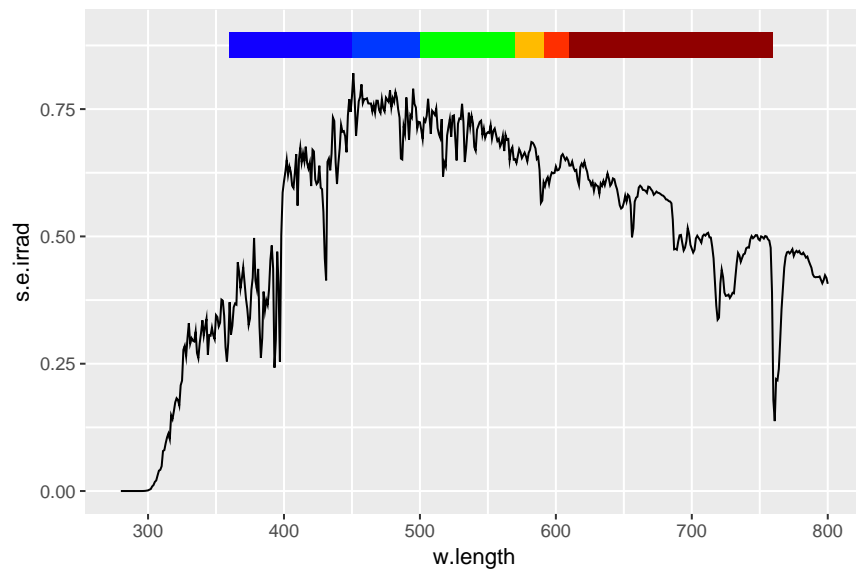
We use as an example plant's sensory colours, to show the case when the wavebands in the list are not contiguous.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

ggplot(data = sun.spct) +
    geom_line() +
    geom_rect(data = plant.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                fill = wb.color),
            color = "white") +
    scale_fill_identity()
```

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())

ggplot(data=sun.spct) +
  geom_line() +
  geom_rect(data=plant.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9,
                fill = wb.color),
            color = "white") +
  geom_text(data=plant.guide.spct,
            aes(y = y + 0.875, label = as.character(wb.f),
            color = black_or_white(wb.color)), size = 3) +
  scale_fill_identity() + scale_color_identity()
```
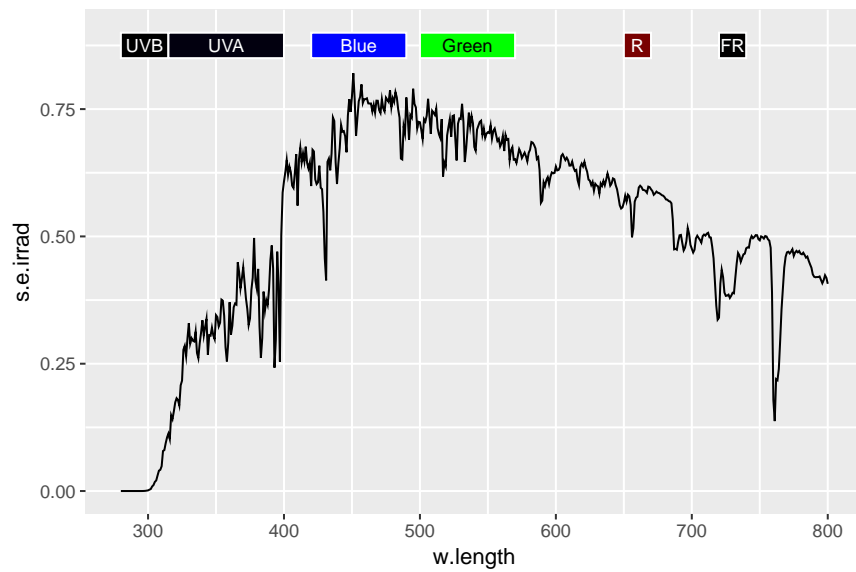
Here we add `alpha` or transparency to make the colours paler, and use black text
and lines.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

ggplot(data=sun.spct) +
  geom_line() +
  geom_rect(data = plant.guide.spct,
       aes(xmin = wl.low, xmax = wl.high,
           ymin = y + 0.85, ymax = y + 0.9,
           fill = wb.color),
       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_identity()
```
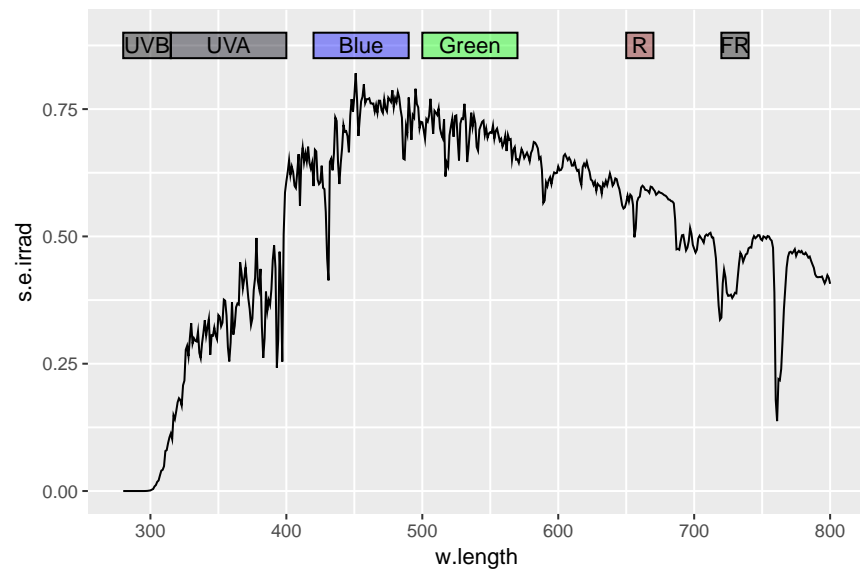
We change the guide so that all rectangles are filled with the same shade of grey by moving `fill` out of `aes` and setting it to a constant.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

ggplot(data = sun.spct) +
  geom_line() +
  geom_rect(data = plant.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y + 0.85, ymax = y + 0.9),
            color = "black", fill = "grey90") +
  geom_text(data = plant.guide.spct,
            aes(y = y + 0.875, label = as.character(wb.f)),
            color = "black", size=4)
```
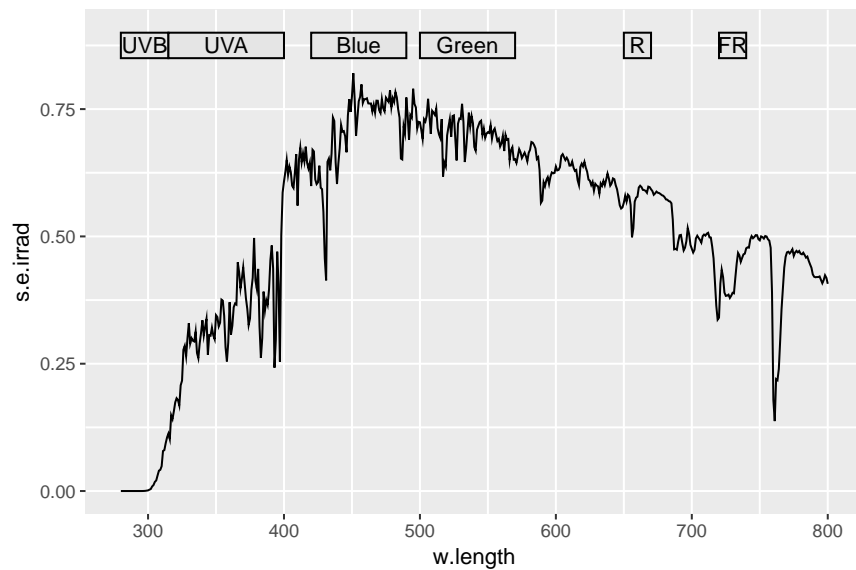
We can obtain annotations similar to those in 16.6 in page 216 created with `annotate_waveband` using geoms.

By changing the order in which the geoms are added compared to previous examples, we make sure that the line is in the top layer of the plot.

```
plant.guide.spct <- wb2rect_spct(Plant_bands())

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,
           fill=wb.color),
       color = "white", alpha=0.5) +
  scale_fill_identity() +
  geom_text(data = plant.guide.spct,
       aes(y = y + 0.88, label = as.character(wb.f)),
       color = "black") +
  geom_line()
```
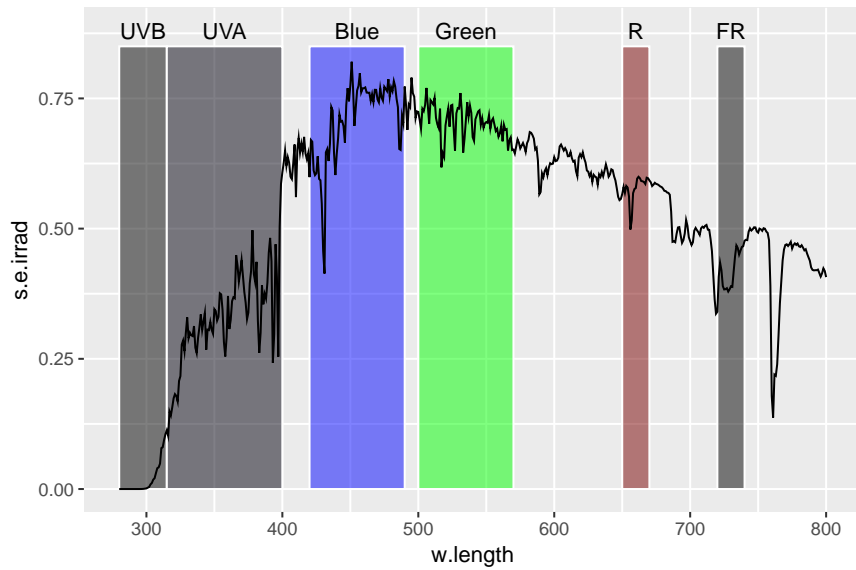
In the examples above we used predefined lists of wavebands, but one can, of course, use any list of waveband definitions, for example explicitly created with `list` and `new_waveband`, or `list` and any combination of user-defined and predefined wavebands. Even single waveband definitions are allowed.

```
par.guide.spct <- wb2rect_spct(PAR())

ggplot(data=sun.spct) +
  geom_line() +
  geom_rect(data=par.guide.spct,
            aes(xmin = wl.low, xmax = wl.high,
                ymin = y - 0.1, ymax = y),
            color = "black", fill = "grey90") +
  geom_text(data = par.guide.spct,
            aes(y = y - 0.05, label = as.character(wb.f)),
            color = "black")
```
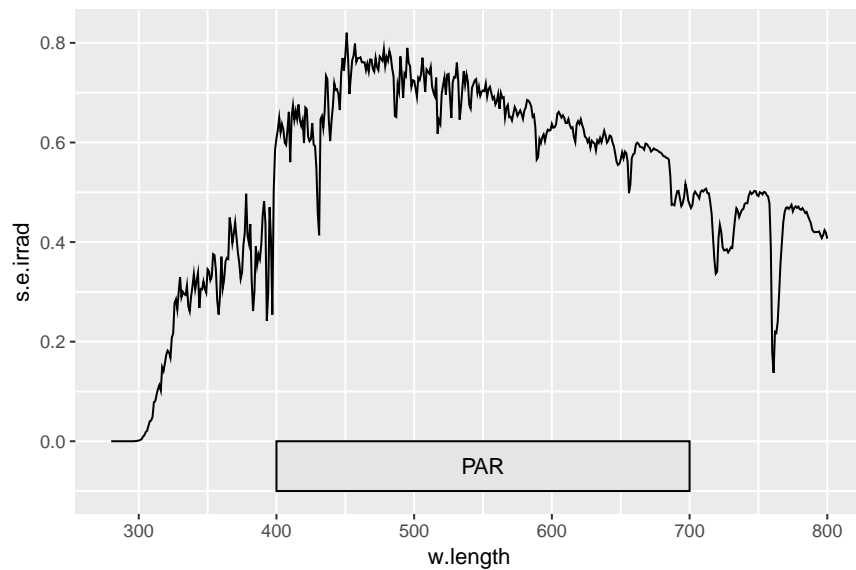
We can also use `geom_segment` to draw lines, including arrows. In this example we also set a different font `family` and label text. We can replace the label text which is by default obtained from the waveband definition by assigning a name to the waveband as member of the list. We use single quotes so that the long name containing space characters is accepted by `list`.

```
par.guide1.spct <-
  wb2rect_spct(list('Photosynthetically active radiation' = PAR()))

ggplot(data=sun.spct) +
  geom_line() +
  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")
```
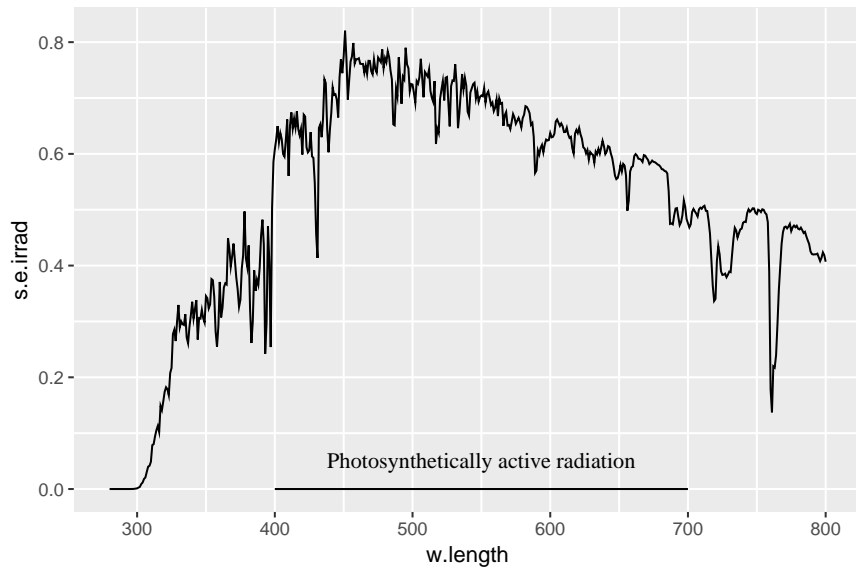
In this section we have used until now function `wb2rect_spct` to create 'spectral' annotation data from waveband definitions. Two other functions are available, that are needed or easier to use in some cases. One such case is when we have a list of wavebands and we would like to mark their boundaries 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())

ggplot(data=sun.spct) +
  geom_line() +
  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")
```
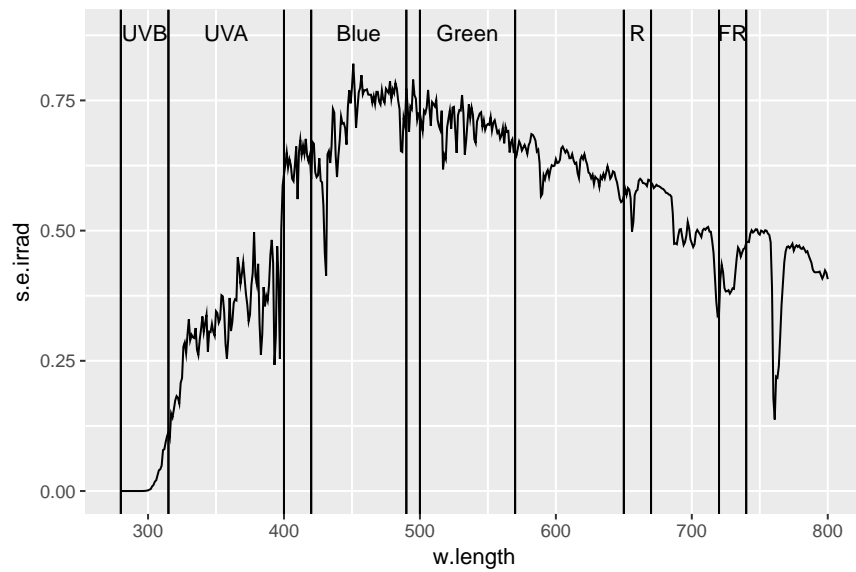
Function `wb2tagged_spct` returns the same data as `wb2spct` but 'tagged'. As shown in the next code chunk, tagging allows us to use waveband-dependent colours to the vertical lines.

```
plant.boundaries.spct <- wb2tagged_spct(Plant_bands())
plant.guide.spct <- wb2rect_spct(Plant_bands())

ggplot(data=sun.spct) +
  geom_line() +
  geom_vline(data = plant.boundaries.spct,
             aes(xintercept = w.length, color = wb.color),
             size = 1, linetype = "dashed") +
  geom_label(data = plant.guide.spct,
        aes(y = y + 0.88, label = as.character(wb.f), color = wb.color),
        size = 4) +
  scale_color_identity()
```

263
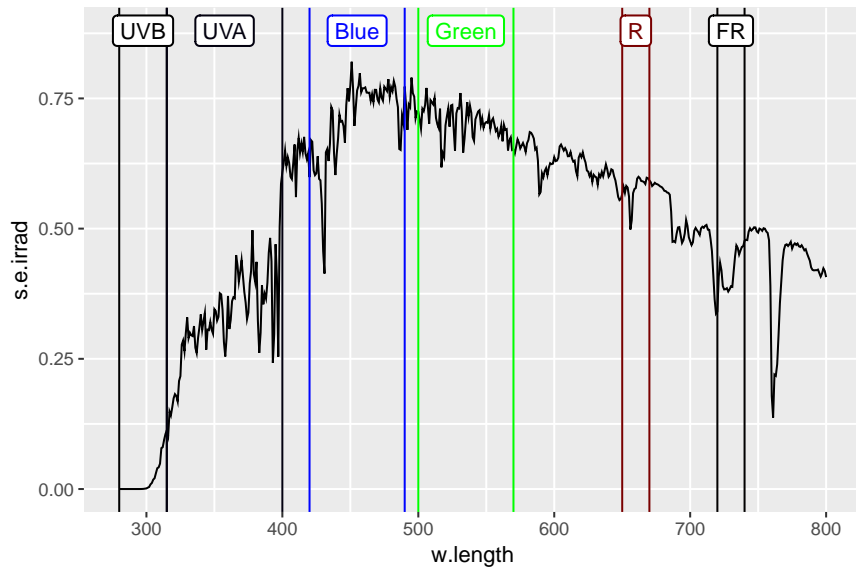
Of course it is possible to combine tagged data spectra and tagged spectra created from wavebands. The tagging is consistent, so, as demonstrated in the next figure, the same aesthetic 'link' works for both spectra. In this case the fill scale and the setting of fill to `wb.f` work across different 'data' 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 earlier examples where we have added layers to a previously saved plot, here we show the whole code needed to build the figure.

```r
my.sun.spct <- tag(sun.daily.spct, list(UVB(), UVA()))
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_spct() +
  geom_line() +
  labs(x = "Wavelength (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
```

Unweighted solar radiation (daily accumulated spectrum)
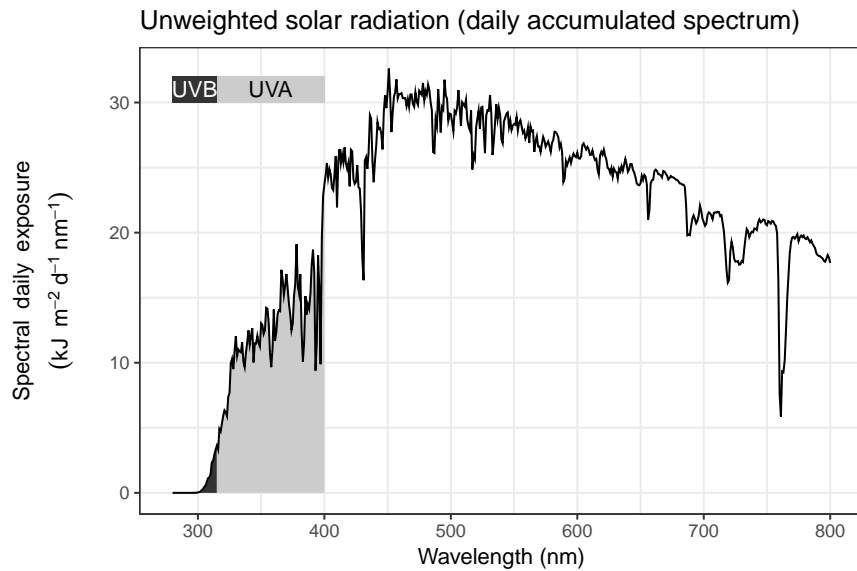
Possible variations are almost endless, so we invite the reader to continue exploring how the functions from package 'photobiology' can be used together with those from package 'ggplot2', to obtain beautiful plots of spectra. As an example here we show new versions of two plots from the previous section, 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.
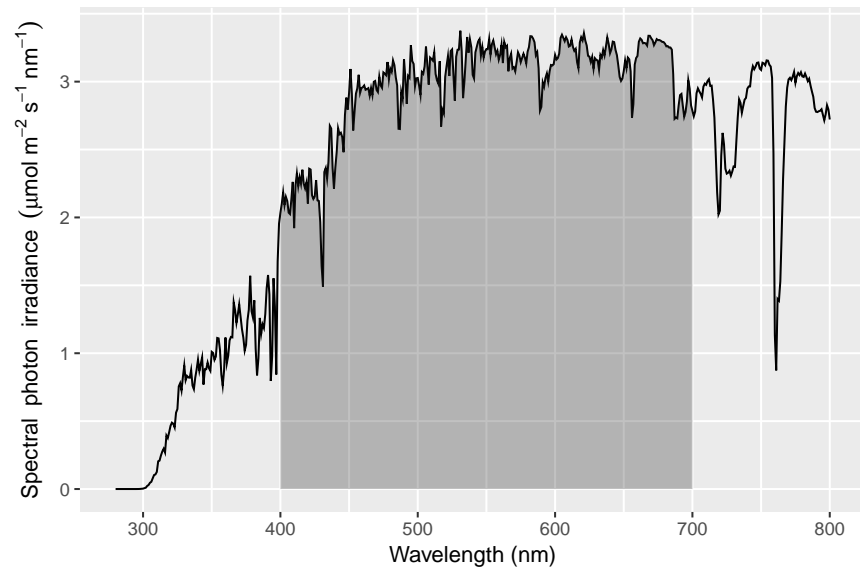
```
par <- q_irrad(sun.spct, PAR()) * 1e6

fig_sun.tgrect1 <-
  ggplot(data = par.sun.spct,
         aes(x = w.length, y = s.q.irrad * 1e6)) +
  geom_line() +
  geom_spct(color = NA, alpha = 0.3, aes(fill = wb.f))  +
  scale_fill_grey(na.value = NA, guide = "none") +
  labs(
    y = ylab_umol,
    x = "Wavelength (nm)")

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()
```
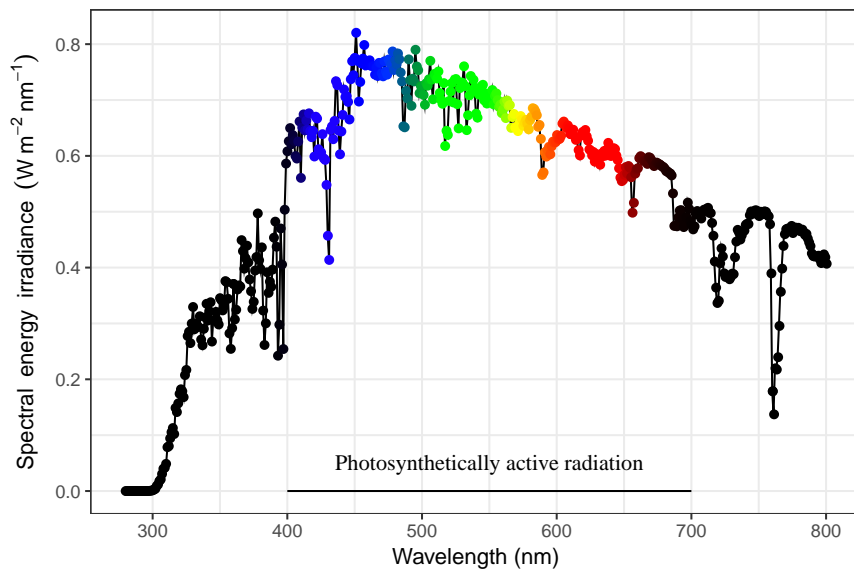
16.8 Plotting the result of operations on spectral data

### 16.8.1 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 <-
    tag(sun.spct * CIE(), UV_bands())
sun.eff.cie.pe.spct <-
    tag(sun.spct * polyester.spct * CIE(), UV_bands())
sun.eff.cie.spct <-
    rbindspct(list('no filter' = sun.eff.cie.nf.spct,
                   'polyester' = sun.eff.cie.pe.spct),
              idfactor = "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_spct() +
  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
```

CIE 1998 erythemal BSWF
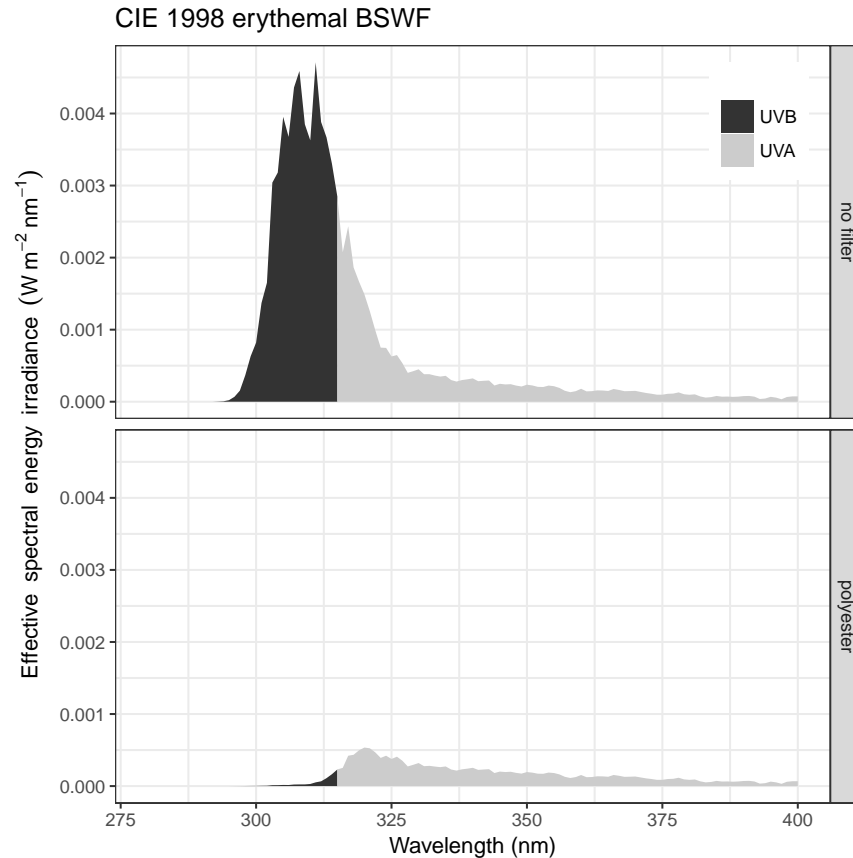
One should be aware that these are estimated values and in practice stray light reduces the efficiency of the filters for blocking radiation, and the amount of stray light depends on many factors including the relative positions of plants, filter and sun.

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.

16.8.2 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.

```r
cie.nf.irrad <- e_irrad(sun.spct * CIE(),
                        list(UVB(), UVA()))
cie.pe.irrad <- e_irrad(sun.spct * polyester.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.

```r
cie.dt <- data_frame(
  cie.irrad = c(cie.nf.irrad, cie.pe.irrad),
  filter = factor(rep(c('none', 'polyester'), c(2,2)),
                  levels=c('none', 'polyester')),
  w.band = factor(rep(c('UVB', 'UVA'), 2),
                  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.

```r
cie.dt

## # A tibble: 4 × 3
##      cie.irrad    filter w.band
##          <dbl>    <fctr> <fctr>
## 1 0.0523585311      none    UVB
## 2 0.0294572964      none    UVA
## 3 0.0006758325 polyester    UVB
## 4 0.0157202386 polyester    UVA


fig_cie_bars0 <- ggplot(data = cie.dt,
                        aes(y = cie.irrad * 1e3,
                            x = filter,
                            fill = w.band)) +
  scale_fill_brewer(palette = "PRGn") +
```
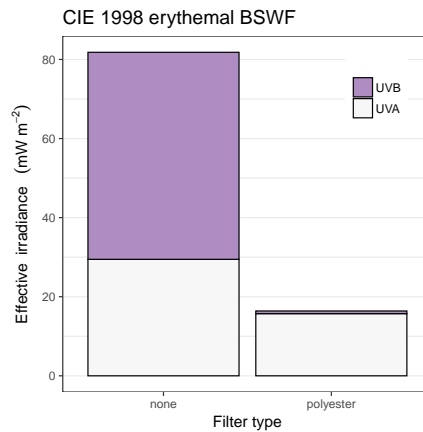
```
  geom_bar(stat = "identity", colour = "black") +
  labs(x = "Filter type",
       y = expression(Effective~~irradiance~~~(mW~m^{-2})),
       title = "CIE 1998 erythemal BSWF",
       fill = "") +
  theme_bw(13) +
  theme(legend.position = c(0.85, 0.85)) +
  theme(panel.grid.minor.x = element_blank(),
        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_col(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
```
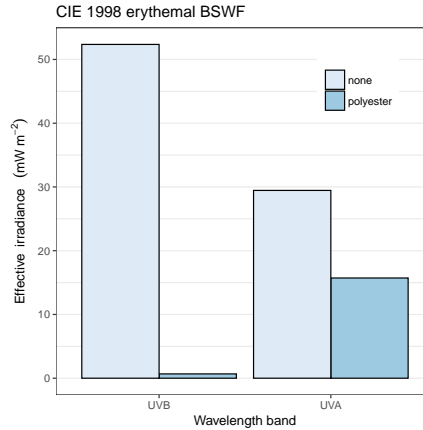

16.8.3 Task: plotting a spectrum using colour bars

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_frame(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
```
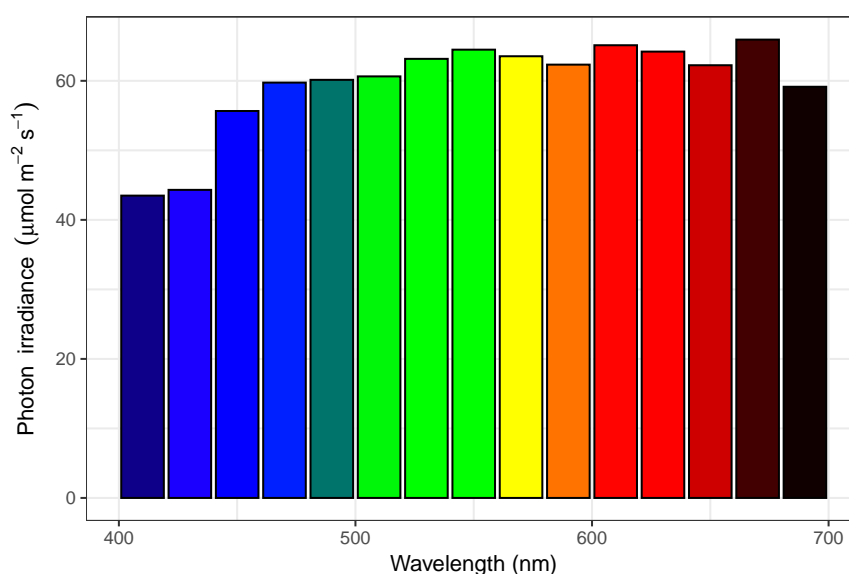

In the case of the example spectrum with equal wavelength steps, one could have directly summed the values, however, the approach shown here is valid for any type of spacing of the values along the wavelength axis, including variable one, like is the case for array spectrometers.

16.9 Task: plotting colours in Maxwell's triangle

16.9.1 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 method `color` (see section 14.3), from a waveband (see section 14.4), for a wavelength range by first constructing a `wavband` object (see section 14.4), and

from a spectrum (see section 14.5). 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`. Geometries, 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.

We need to load 'ggtern' here because at the moment it conflicts with 'ggspectra' as it redefines `ggplot` in a way that does not match its current definition in 'ggplot2' as an S3 method.

```
library(ggtern)

## --
## Consider donating at: http://ggtern.com
## Even small amounts (say $10-50) are very much appreciated!
## Remember to cite, run citation(package = 'ggtern') for further info.
## --
##
## Attaching package: 'ggtern'
## The following objects are masked from 'package:gridExtra':
##
##     arrangeGrob, grid.arrange
## The following objects are masked from 'package:ggplot2':
##
##     %+%, aes, annotate, calc_element,
##     ggplot_build, ggplot_gtable, ggplotGrob,
##     ggsave, layer_data, theme, theme_bw,
##     theme_classic, theme_dark, theme_gray,
##     theme_light, theme_linedraw,
##     theme_minimal, theme_void
```

In the first example we use colors pre-defined in R.

```
colours <- c("red", "green", "yellow", "white",
             "orange", "purple", "seagreen", "pink")
rgb.values <- col2rgb(colours)
color.data <- data.frame(colour = colours,
                         R = rgb.values[1, ],
                         G = rgb.values[2, ],
                         B = rgb.values[3, ])
ggtern(data = color.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() +
```

```
theme_nomask()
```
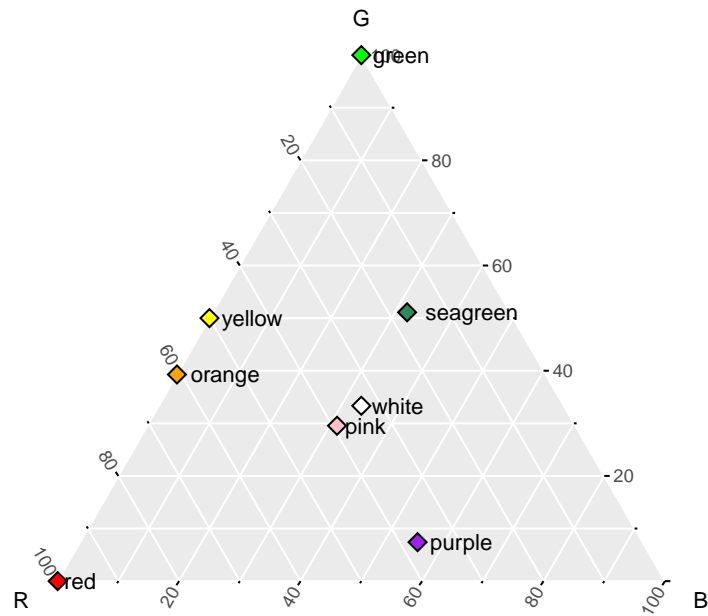

In the second example, we calculate the colours for leaves as seen in sunlight.

```
betula_reflected.mspct <-
  convolve_each(as.reflector_mspct(Betula_ermanii.mspct),
                sun.spct)
color.values <- color(betula_reflected.mspct, type = "CC")
rgb.values <- col2rgb(color.values[[2]])
leaf.data <- data.frame(leaf = color.values[[1]],
                        color = color.values[[2]],
                        R = rgb.values[1, ],
                        G = rgb.values[2, ],
                        B = rgb.values[3, ])
ggtern(data = leaf.data,
       aes(x = R, y = G, z = B, fill = color)) +
       geom_point(shape = 21, size = 3) +
  labs(x = "R", y = "G", z = "B") + scale_fill_identity()
```
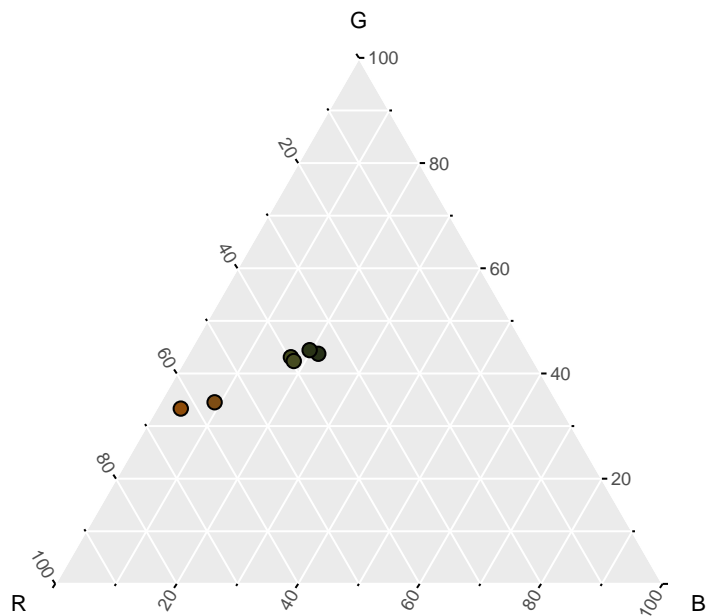
```
try(detach(package:ggtern))
try(detach(package:ggspectra))
try(detach(package:photobiologyReflectors))
try(detach(package:photobiologyPlants))
try(detach(package:photobiologyFilters))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiology))
try(detach(package:dplyr))
try(detach(package:gridExtra))
try(detach(package:scales))
try(detach(package:ggplot2))
```

17

Radiation physics

17.1 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(photobiology)
library(ggspectra)
library(photobiologyFilters)
```

17.2 Introduction

17.3 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):

$$L_s(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{e^{(hc/k_B T\lambda)} - 1} \tag{17.1}$$

with Boltzmann's constant $k_B = 1.381 \times 10^{-23}$ JK^{-1}, Planck's constant $h = 6.626 \times 10^{-34}$ Js 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}:

```
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:

```
black_body_spectrum(500, 5000)

## [1] 1.212443e+13
```

The function is vectorised:

```
black_body_spectrum(c(300,400,500), 5000)

## [1] 3.354907e+12 8.759028e+12 1.212443e+13
```

```
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]:

```
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 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)")
```

---

[1]Exercise: calculate each of the four values individually to work out how the two vectors are being used.
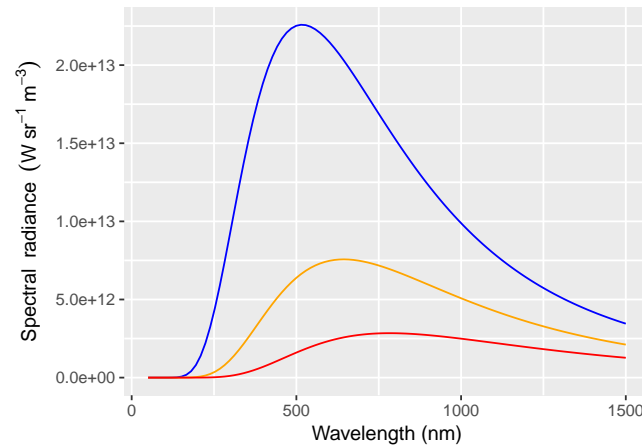
Wien's displacement law, gives the peak wavelength of the radiation emitted by a black body as a function of its absolute temperature.

$$\lambda_{max} \cdot T = 2.898 \times 10^6 \, \text{nm K} \tag{17.2}$$

A function implementing this equation takes just a few lines of code:

```r
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:

```r
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)")
```
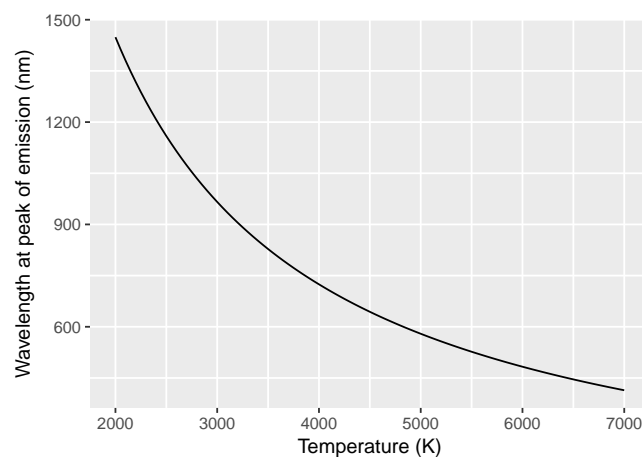

279

```r
try(detach(package:photobiologyFilters))
try(detach(package:ggspectra))
try(detach(package:photobiology))
try(detach(package:ggplot2))
```

Part IV

Data acquisition and exchange

18

Importing and exporting 'R' data

18.1 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(tibble)
library(photobiology)
library(photobiologySun)
library(photobiologyWavebands)
library(photobiologyInOut)
library(lubridate)
library(ggplot2)
library(ggspectra)
library(hyperSpec)
library(colorSpec)
library(pavo)
library(fda)
library(fda.usc)
```

18.2 Base R

18.2.1 Task: Import one spectrum from a `data.frame`

If the variables in the `data.frame` or `tibble` object are already named and expressed in the units expected by the 'r4photobiology suite' (see Table **??**) then it is possible to convert the object into a spectral object directly as shown in task **??** on page **??** or to create a spectral object without modifying the original data frame object as shown in **??** on page **??**.

If the names of the variables or units of expression are not those expected, the data frame, or a copy can be suitably converted as needed with R commands.

```
my.df <- data.frame(a = 201:250, b = c("A", "B"), c = 120)
names(my.df) <- c("w.length", "TREA", "s.e.irrad")
my.spct <- as.source_spct(my.df)
my.spct
```

```
## Object: source_spct [50 x 3]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 3
##   w.length   TREA s.e.irrad
##      <int> <fctr>     <dbl>
## 1      201      A       120
## 2      202      B       120
## 3      203      A       120
## 4      204      B       120
## # ... with 46 more rows
```

An alternative, more convenient when scaling is needed is to *extract* the vectors and use them in the *constructor*.

```
my.df <- data.frame(a = 201:250, b = c("A", "B"), c = 12000)
source_spct(w.length = my.df$a, s.e.irrad = my.df$c / 100)

## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <int>     <dbl>
## 1      201       120
## 2      202       120
## 3      203       120
## 4      204       120
## # ... with 46 more rows
```

18.2.2 Task: Export one spectrum to a `data.frame`

The spectral classes are `data.frame` objects, so in many cases they will behave as such and can be passed as argument to functions expecting a data frame as input. In cases when one does not want any of the special methods to be called, it is possible to strip the additional class attributes.

```
my.spct <- sun.spct
rmDerivedSpct(my.spct)
class(my.spct)

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

class(sun.spct)

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

18.2.3 Task: Import one spectrum from a `matrix`

To convert a matrix containing data for a single spectrum, with wavelengths and for example spectral irradiance in two columns, we can convert the matrix to a data frame, and then use the approach already described for converting data frames.

```r
my.mat <- matrix(c(201:250, rep(120, 50)), ncol = 2)
dim(my.mat)

## [1] 50  2

colnames(my.mat) <- c("w.length", "s.e.irrad")
my.spct <- as.source_spct(as_tibble(my.mat))
my.spct

## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      201       120
## 2      202       120
## 3      203       120
## 4      204       120
## # ... with 46 more rows
```

If the matrix contains one row for each variable, we can transpose it with function `t`.

```r
my.mat <- matrix(c(201:250, rep(120, 50)), nrow = 2, byrow = TRUE)
dim(my.mat)

## [1]  2 50

rownames(my.mat) <- c("w.length", "s.e.irrad")
my.spct <- as.source_spct(as_tibble(t(my.mat)))
my.spct

## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      201       120
## 2      202       120
```

18

285

```
## 3       203       120
## 4       204       120
## # ... with 46 more rows
```

⚠️ Above in `as.source_spct(as_tibble(t(my.mat)))` we have three nested function calls. `t` takes as argument `my.mat`, then the result of this computation becomes the input argument to `as_tibble` and then the result returned by it, becomes the input argument to `as.source_spct` and the value returned is then 'printed' or displayed.

Of course the approach using a *constructor* can be also used for matrices.

```
my.mat <- matrix(c(201:250, rep(120, 50)), nrow = 2, byrow = TRUE)
dim(my.mat)

## [1]  2 50

source_spct(w.length = my.mat[1, ], s.e.irrad = my.mat[2, ])

## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1       201       120
## 2       202       120
## 3       203       120
## 4       204       120
## # ... with 46 more rows
```

⚠️ In the code statement `my.mat[1, ]` and `my.mat[2, ]` we use indexes to extract the first and second rows from the matrix, because the matrix we created in the first line of the code chunk with argument `byrow = TRUE` contains the wavelengths in the first or top row, and the spectral irradiance data in the second row.

18.2.4 Task: Export one spectrum to `matrix`

For an individual spectrum, this trivial, as one can use base R's `as.matrix`

```
my.mat <- as.matrix(sun.spct)
dim(my.mat)

## [1] 522    3

nrow(my.mat)

## [1] 522
```

And if the intention is to have the variables as rows in the matrix, we use `t` to transpose it.

```
my.mat <- t(as.matrix(sun.spct))
dim(my.mat)

## [1]    3 522

nrow(my.mat)

## [1] 3
```

18.2.5 Task: Import a collection of spectra from a `matrix`

In many cases, when matrices are used, the wavelength values are stored in a separate vector as we assume for this example.

```
my.mat <- matrix(c(rep(120, 50), rep(240, 50)), ncol = 2)
wl <- 201:250
dim(my.mat)

## [1] 50   2

mat2mspct(my.mat, wl, "source_spct", "s.e.irrad")

## Object: source_mspct [1 x 2]
## --- Member: spct_1 ---
## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      201       120
## 2      202       120
## 3      203       120
## 4      204       120
## # ... with 46 more rows
```

18

```
## --- Member: spct_2 ---
## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      201       240
## 2      202       240
## 3      203       240
## 4      204       240
## # ... with 46 more rows
##
## --- END ---
```

Next we assume that the first column of the matrix contains the wavelength values and another two columns spectral irradiance data.

```
my.mat <- matrix(c(201:250, rep(120, 50), rep(240, 50)), ncol = 3)
dim(my.mat)

## [1] 50  3

mat2mspct(my.mat[ , 2:3], my.mat[ , 1], "source_spct", "s.e.irrad")

## Object: source_mspct [1 x 2]
## --- Member: spct_1 ---
## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      201       120
## 2      202       120
## 3      203       120
## 4      204       120
## # ... with 46 more rows
## --- Member: spct_2 ---
## Object: source_spct [50 x 2]
## Wavelength range 201 to 250 nm, step 1 nm
## Time unit 1s
##
## # A tibble: 50 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      201       240
```

```
## 2       202         240
## 3       203         240
## 4       204         240
## # ... with 46 more rows
##
## --- END ---
```

If the wavelengths are stored in the last, or some other column in the matrix, we just need to need to modify the indexes used above to extract the different columns from the matrix.

18.2.6 Task: Export a collection of spectra to `matrix`

For this operation one can directly use function `mspct2mat`, as shown in the first code chunk of section 18.6 on page 302.

18.3 Package 'hyperSpec'

18.3.1 To 'hyperSpec'

Can export to `hyperSpec` objects only collections of spectra where all members have identical `w.length` vectors, as objects of class `hypeSpec` store a single vector of wavelengths for the whole collection of spectra. We use as example data `gap.mspct` from package 'photobiologySun'.

```
gap.hspct <- mspct2hyperSpec(gap.mspct, "s.e.irrad")

## Warning in .local(.Object, ...): Spectra in data are overwritten by argument spc.

class(gap.hspct)

## [1] "hyperSpec"
## attr(,"package")
## [1] "hyperSpec"

plot(gap.hspct)

## Warning in plotspc(x, ...): Number of spectra exceeds spc.nmax. Only the first 50
## are plotted.
```

289
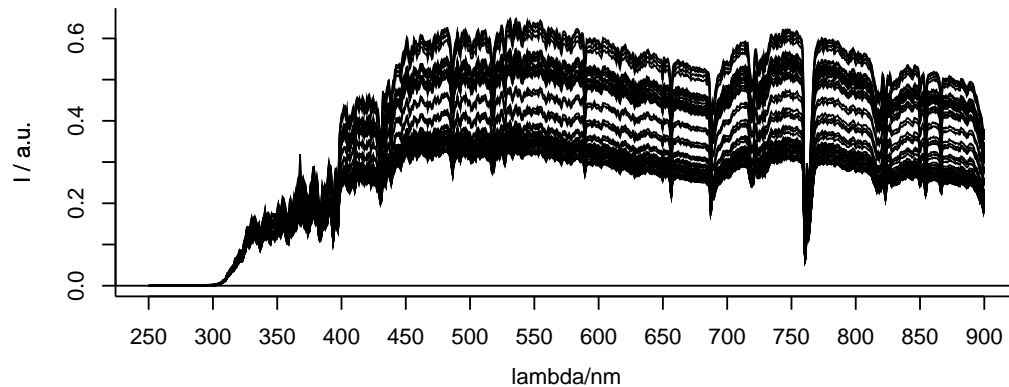
18.3.2 From 'hyperSpec'

Can import only data with wavelength in nanometres. Other quantities and units are not supported by the 'photobiology' classes for spectral data. See package 'hyperSpec' vignette "laser" for details on the data and the conversion of the original wavelength units into nanometres.
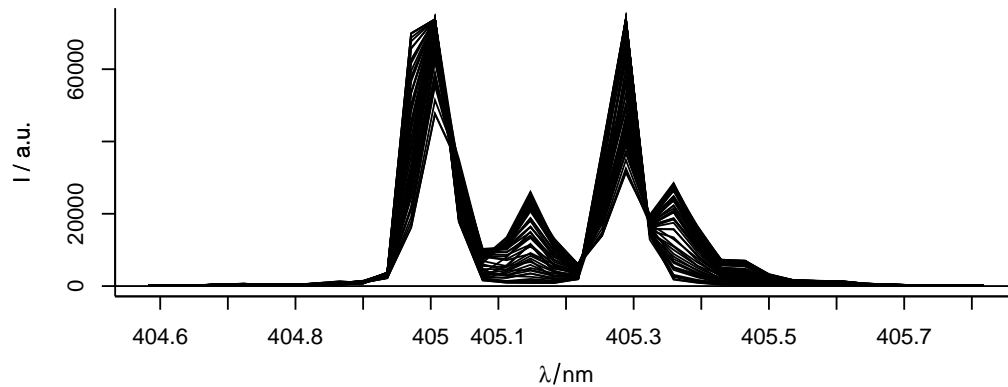
```
class(laser)

## [1] "hyperSpec"
## attr(,"package")
## [1] "hyperSpec"

laser

## hyperSpec object
##     84 spectra
##     2 data columns
##     36 data points / spectrum
## wavelength: lambda/nm [numeric] 404.5828 404.6181 ... 405.8176
## data:   (84 rows x 2 columns)
##     1. t: t / s [numeric] 0 2 ... 5722
##     2. spc: I / a.u. [matrix36] 164.650 179.724 ... 112.086

plot(laser)

## Warning in plotspc(x, ...): Number of spectra exceeds spc.nmax. Only the first 50
## are plotted.
```

We assume here, that the quantity for the spectral emission of the laser is spectral *energy* irradiance, expressed in $mW\,cm^{-2}\,nm^{-1}$. This is likely to be wrong but for the sake of showing how the conversion takes place is irrelevant. The parameter `multiplier` can be passed a numeric argument to rescale the original data. The default multiplier is 1.

```
wl(laser) <- list (
  wl = 1e7 / (1/405e-7 - wl (laser)),
  label = expression (lambda / nm)
)
laser

## hyperSpec object
##     84 spectra
##     2 data columns
##     36 data points / spectrum
## wavelength: lambda/nm [numeric] 411.7467 411.7473 ... 411.7677
## data:   (84 rows x 2 columns)
##     1. t: t / s [numeric] 0 2 ... 5722
##     2. spc: I / a.u. [matrix36] 164.650 179.724 ... 112.086

plot(laser)

## Warning in plotspc(x, ...): Number of spectra exceeds spc.nmax. Only the first 50
are plotted.

laser.mspct <-
  hyperSpec2mspct(laser, "source_spct", "s.e.irrad", multiplier = 1e-3)
ggplot(laser.mspct[[1]]) +
  geom_line() +
  stat_peaks(geom = "text", vjust = -1, label.fmt = "%.6g nm", color = "red")
```
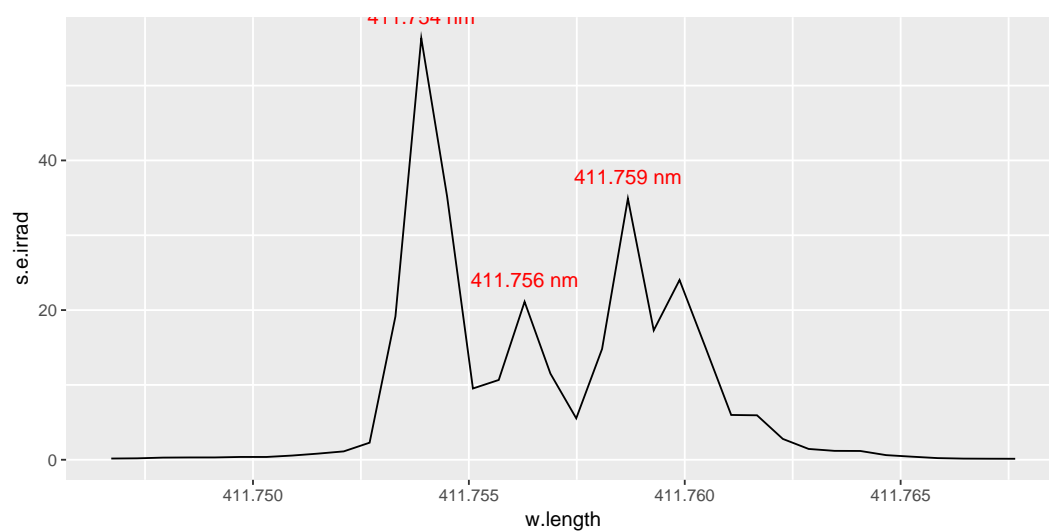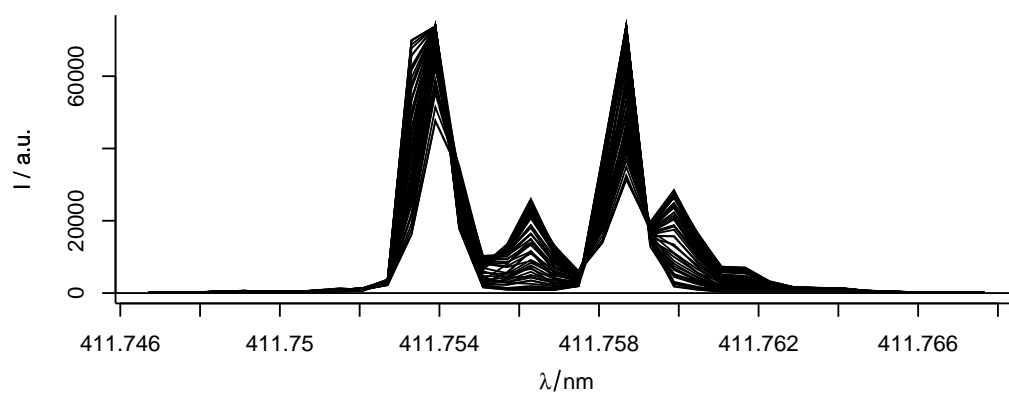
18.4 Package 'colorSpec'

18.4.1 From 'colorSpec'

```
fluorescent.mspct <- colorSpec2mspct(Fs.5nm)
print(fluorescent.mspct, n = 3, n.members = 3)

## Object: source_mspct [12 x 1]
## --- Member: F1 ---
## Object: source_spct [81 x 2]
## Wavelength range 380 to 780 nm, step 5 nm
## Time unit 1s
```

```
##
## # A tibble: 81 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1      380       1.87
## 2      385       2.36
## 3      390       2.94
## # ... with 78 more rows
## --- Member: F2 ---
## Object: source_spct [81 x 2]
## Wavelength range 380 to 780 nm, step 5 nm
## Time unit 1s
##
## # A tibble: 81 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1      380       1.18
## 2      385       1.48
## 3      390       1.84
## # ... with 78 more rows
## --- Member: F3 ---
## Object: source_spct [81 x 2]
## Wavelength range 380 to 780 nm, step 5 nm
## Time unit 1s
##
## # A tibble: 81 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1      380       0.82
## 2      385       1.02
## 3      390       1.26
## # ... with 78 more rows
## ........................
## 9 other member spectra not shown
##
## --- END ---
```

```
colorSpec2mspct(Hoya)
```

```
## Object: filter_mspct [4 x 1]
## --- Member: R-60 ---
## Object: filter_spct [46 x 2]
## Wavelength range 300 to 750 nm, step 10 nm
##
## # A tibble: 46 × 2
##    w.length   Tfr
##       <dbl> <dbl>
## 1      300     0
```

293

```
## 2       310     0
## 3       320     0
## 4       330     0
## # ... with 42 more rows
## --- Member: G-533 ---
## Object: filter_spct [46 x 2]
## Wavelength range 300 to 750 nm, step 10 nm
##
## # A tibble: 46 × 2
##   w.length    Tfr
##       <dbl> <dbl>
## 1       300     0
## 2       310     0
## 3       320     0
## 4       330     0
## # ... with 42 more rows
## --- Member: B-440 ---
## Object: filter_spct [46 x 2]
## Wavelength range 300 to 750 nm, step 10 nm
##
## # A tibble: 46 × 2
##   w.length    Tfr
##       <dbl> <dbl>
## 1       300     0
## 2       310     0
## 3       320     0
## 4       330     0
## # ... with 42 more rows
## --- Member: LB-120 ---
## Object: filter_spct [46 x 2]
## Wavelength range 300 to 750 nm, step 10 nm
##
## # A tibble: 46 × 2
##   w.length      Tfr
##       <dbl>    <dbl>
## 1       300 0.00003
## 2       310 0.00580
## 3       320 0.08100
## 4       330 0.30400
## # ... with 42 more rows
##
## --- END ---
```

```r
fluorescent.spct <- colorSpec2spct(Fs.5nm)
plot(fluorescent.spct, annotations = c("peaks", "color.guide")) + aes(linetype = spct.idx)
```
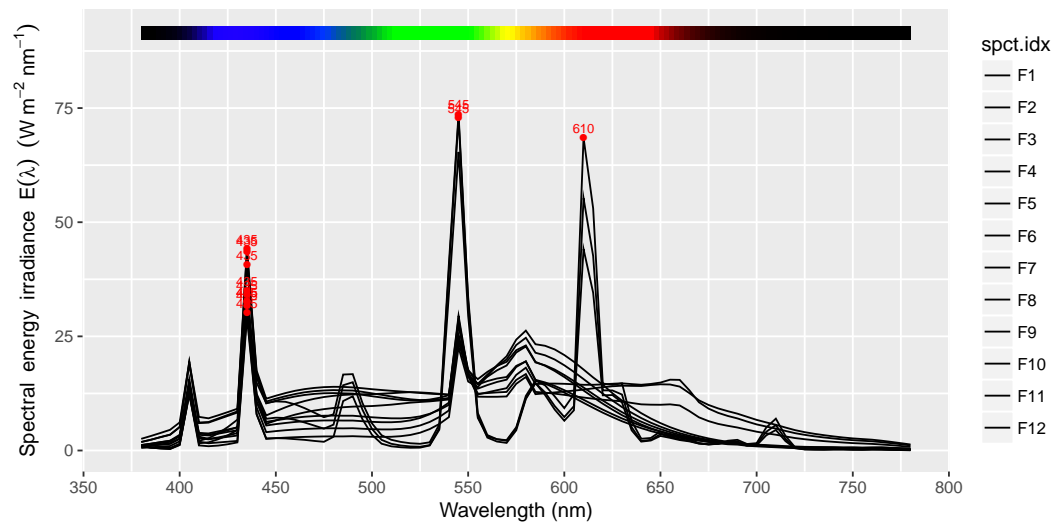
```
colorSpec2chroma_spct(xyz1931.5nm)

## Object: chroma_spct [81 x 4]
## Wavelength range 380 to 780 nm, step 5 nm
##
## # A tibble: 81 × 4
##         x       y       z w.length
## *   <dbl>   <dbl>   <dbl>    <dbl>
## 1 0.0014 0e+00 0.0065      380
## 2 0.0022 1e-04 0.0105      385
## 3 0.0042 1e-04 0.0201      390
## 4 0.0076 2e-04 0.0362      395
## # ... with 77 more rows
```

18.4.2 To 'colorSpec'

```
sun.cspec <- spct2colorSpec(sun.spct)
plot(sun.cspec, color = "blue")
```

295
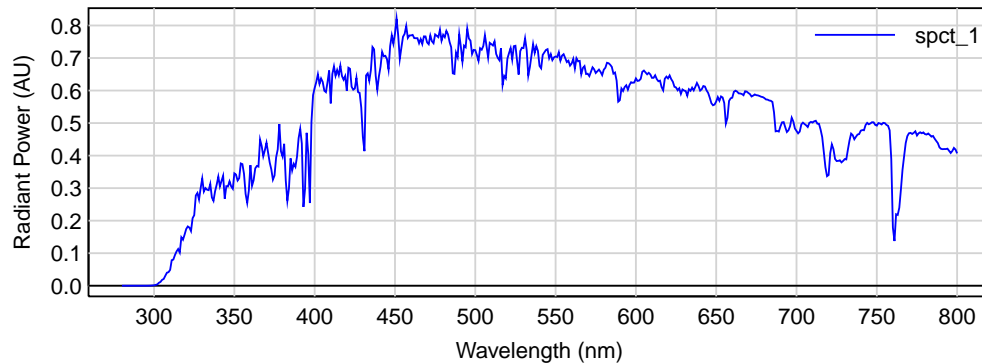
```
spct2colorSpec(yellow_gel.spct)

##
## colorSpec object.    The organization is 'vector'.  Object size is 11488 bytes.
## the object describes 1 transparent materials, and the quantity is 'transmittance'.
## Wavelength range: 190 to 800 nm.  Step size is 1 nm.
##
## 1 spectra
## 611 data points / spectrum
##
##    Material   Min    Max LambdaMax Integral
## 1   spct_1 1e-05 0.9025    768.5 260.3194
```

```
chroma_spct2colorSpec(beesxyzCMF.spct)

##
## colorSpec object.    The organization is 'matrix'.  Object size is 4128 bytes.
## the object describes a responder to light with 3 output channels, and the quantity is 'power->neural'.
## Wavelength range: 300 to 700 nm.  Step size is 5 nm.
##
## 3 spectra
## 81 data points / spectrum
##
##    Channel   Min Max LambdaMax E.response
## 1       x 0.000   1       340     68.965
## 2       y 0.000   1       435    103.850
## 3       z 0.006   1       560    135.620
```

18.5 Package 'pavo'

18.5.1 From 'pavo'

In this example we convert an `rspec` object from package 'pavo' into a collection of spectra and then we plot it with `ggplot` methods from package 'ggspectra' (an extension to 'ggplot2'). The data are the spectral reflectance of the plumage from seven different individual birds of the same species, measured in three different body parts.

```r
data(sicalis)
class(sicalis)

## [1] "rspec"      "data.frame"

names(sicalis)

## [1] "wl"     "ind1.C" "ind1.T" "ind1.B" "ind2.C"
## [6] "ind2.T" "ind2.B" "ind3.C" "ind3.T" "ind3.B"
## [11] "ind4.C" "ind4.T" "ind4.B" "ind5.C" "ind5.T"
## [16] "ind5.B" "ind6.C" "ind6.T" "ind6.B" "ind7.C"
## [21] "ind7.T" "ind7.B"
```

We convert the data into a collection of spectra, and calculate summaries for three spectra.

```r
sicalis.mspct <- rspec2mspct(sicalis, "reflector_spct", "Rpc")
summary(sicalis.mspct[[1]])

## Summary of object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
##     w.length         Rfr
##  Min.   :300   Min.   :0.001798
##  1st Qu.:400   1st Qu.:0.008288
##  Median :500   Median :0.031709
##  Mean   :500   Mean   :0.052848
##  3rd Qu.:600   3rd Qu.:0.098775
##  Max.   :700   Max.   :0.114807

summary(sicalis.mspct[[2]])

## Summary of object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
##     w.length         Rfr
##  Min.   :300   Min.   :0.006783
##  1st Qu.:400   1st Qu.:0.030112
##  Median :500   Median :0.096994
```

18

297

```
##  Mean   :500   Mean    :0.105449
##  3rd Qu.:600   3rd Qu.:0.179691
##  Max.   :700   Max.    :0.183823
```

```
summary(sicalis.mspct[[3]])
```

```
## Summary of object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
##      w.length        Rfr
##  Min.   :300   Min.    :0.001191
##  1st Qu.:400   1st Qu.:0.022293
##  Median :500   Median :0.085235
##  Mean   :500   Mean    :0.116253
##  3rd Qu.:600   3rd Qu.:0.212554
##  Max.   :700   Max.    :0.224162
```

We convert the subset of the collection corresponding to the first individual into a single spectra object for plotting with `ggplot` .

```
ggplot(rbindspct(sicalis.mspct[1:3])) +
  aes(linetype = spct.idx) +
  ylim(0,0.3) +
  geom_line()
```
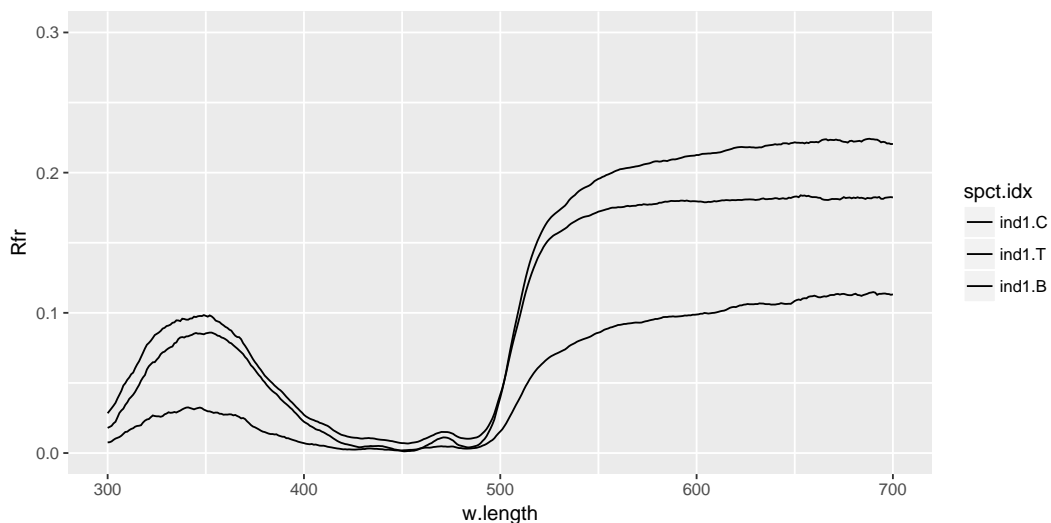

Here we extract the "crown" data from all individuals and plot these spectra in a single plot.

```
print(sicalis.mspct[c(TRUE, FALSE, FALSE)])
```

```
## Object: reflector_mspct [7 x 1]
## --- Member: ind1.C ---
```

```
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length         Rfr
##      <dbl>       <dbl>
## 1      300 0.007594984
## 2      301 0.007727841
## 3      302 0.008288000
## 4      303 0.009414667
## # ... with 397 more rows
## --- Member: ind2.C ---
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length         Rfr
##      <dbl>       <dbl>
## 1      300 0.002971167
## 2      301 0.002326722
## 3      302 0.003227833
## 4      303 0.003535167
## # ... with 397 more rows
## --- Member: ind3.C ---
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length          Rfr
##      <dbl>        <dbl>
## 1      300 0.0005947619
## 2      301 0.0000000000
## 3      302 0.0011853968
## 4      303 0.0009427302
## # ... with 397 more rows
## --- Member: ind4.C ---
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length         Rfr
##      <dbl>       <dbl>
## 1      300 0.003750159
## 2      301 0.003465714
## 3      302 0.004133333
## 4      303 0.004339333
## # ... with 397 more rows
## --- Member: ind5.C ---
```

18

```
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length        Rfr
##      <dbl>        <dbl>
## 1      300 0.004225349
## 2      301 0.005361222
## 3      302 0.006554556
## 4      303 0.006816556
## # ... with 397 more rows
## --- Member: ind6.C ---
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length        Rfr
##      <dbl>        <dbl>
## 1      300 0.0006326984
## 2      301 0.0006136508
## 3      302 0.0001934921
## 4      303 0.0008594921
## # ... with 397 more rows
## --- Member: ind7.C ---
## Object: reflector_spct [401 x 2]
## Wavelength range 300 to 700 nm, step 1 nm
##
## # A tibble: 401 × 2
##   w.length        Rfr
##      <dbl>        <dbl>
## 1      300 0.001680730
## 2      301 0.001043270
## 3      302 0.001702476
## 4      303 0.001939810
## # ... with 397 more rows
##
## --- END ---

ggplot(rbindspct(sicalis.mspct[c(TRUE, FALSE, FALSE)])) +
  aes(linetype = spct.idx) +
  ylim(0,0.15) +
  geom_line() +
  ggtitle("'crown' reflectance spectra")
```

'crown' reflectance spectra
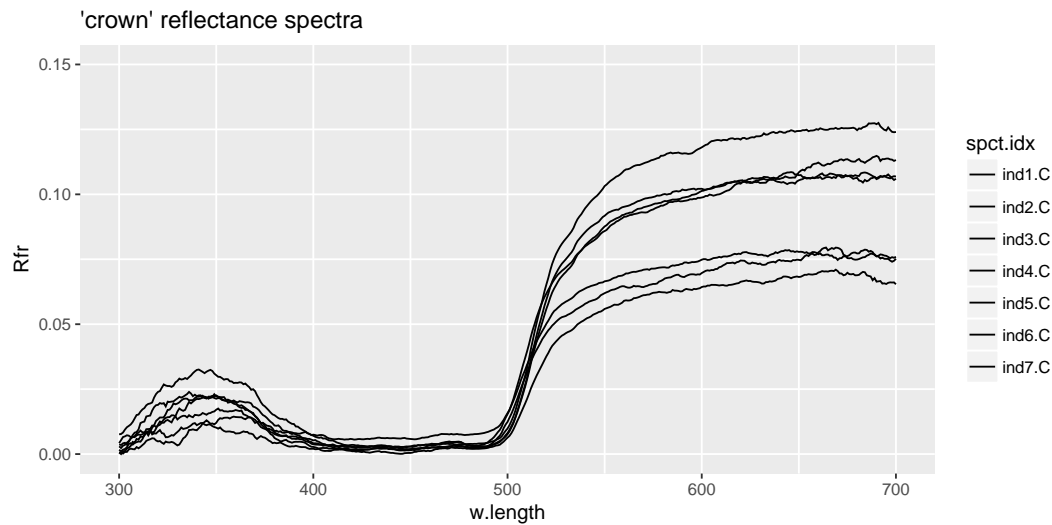
We calculate the mean reflectance in wavebands corresponding to ISO colors obtaining a data frame. We then add to this returned data frame a factor indicating the body parts.

```
refl.by.band <- reflectance(sicalis.mspct, w.band = list(Red(), Green(), Blue(), UVA()))
refl.by.band$body.part <- c("crown", "throat", "breast")
```

```
refl.red <- reflectance(sicalis.mspct, w.band = Red())
names(refl.red)[2] <- "red.reflectance"
refl.red$body.part <- c("crown", "throat", "breast")
ggplot(refl.red, aes(x = body.part, y = red.reflectance)) +
  stat_summary(fun.data = "mean_se", color = "red") +
  geom_point(alpha = 0.5)
```


18
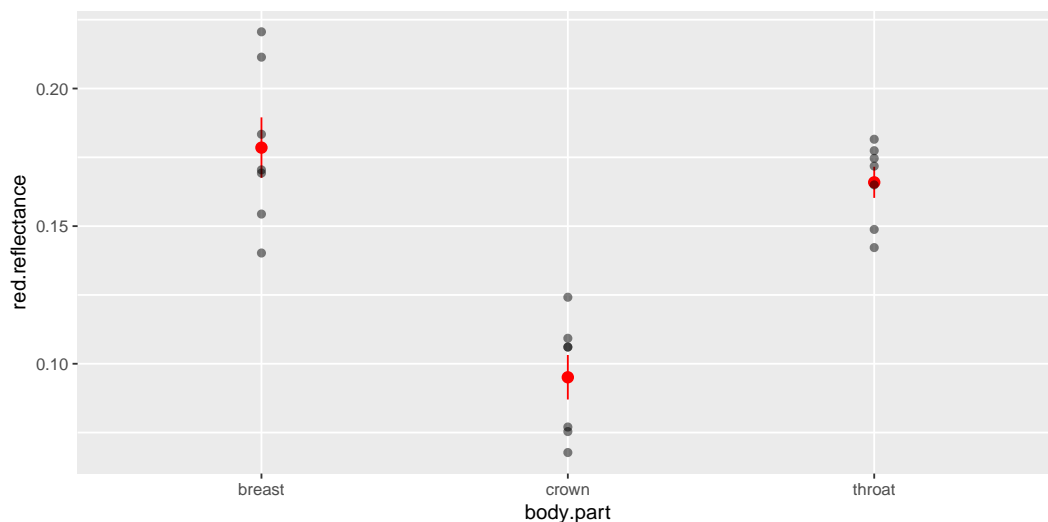
18.6 Packages 'fda' and 'fda.usc'

Functional data analysis is a specialized method that can be used to compare and classify spectra. We here exemplify the selection of the 'deepest spectrum' from a collection of spectra. The data interconversion can be done with a simple function. Package 'fda' expects the spectra in a single matrix object, with each spectrum as a row. We will use once again `gap.mspct` for this example.

```
gap.mat <- mspct2mat(gap.mspct, "s.e.irrad", byrow = TRUE)
dim(gap.mat)

## [1]   72 1425

names(dimnames(gap.mat))

## [1] "spct"     "w.length"

# convert the matrix to fdata
gap.fdata <- fdata(gap.mat)
```

We search for "deepest curve" using different methods.

```
# Returns the deepest curve following FM criteria
func_med_FM <- func.med.FM(gap.fdata)
# Returns the deepest curve following mode criteria
func_med_mode <- func.med.mode(gap.fdata)
# Returns the deepest curve following RP criteria
func_med_RP <- func.med.RP(gap.fdata)
```

We plot using plot method from package 'fda'.

```
plot(func_med_FM)
```


depth.FM median
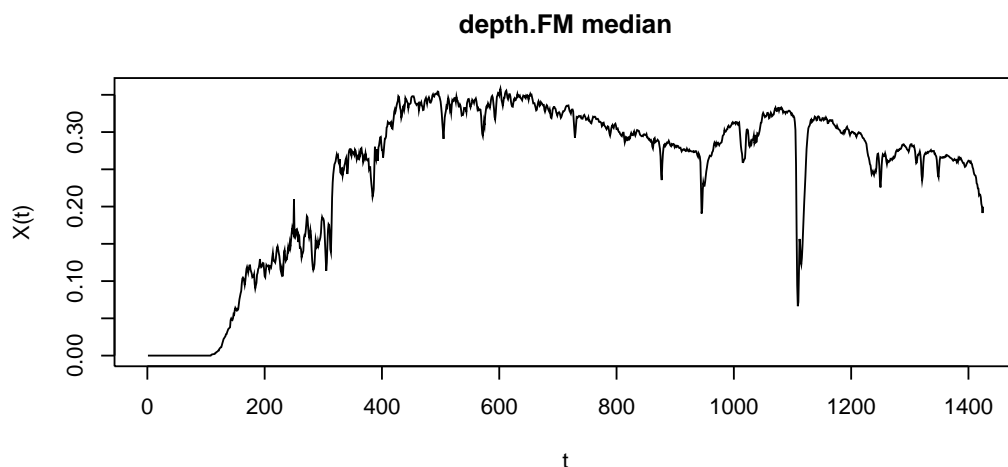
We convert the returned curves into a `source_spct` objects.

```
func_med_FM.spct <-
  source_spct(w.length = attr(gap.mat, "w.length"),
              s.e.irrad = func_med_FM$data[1, ])
func_med_mode.spct <-
  source_spct(w.length = attr(gap.mat, "w.length"),
              s.e.irrad = func_med_mode$data[1, ])
func_med_RP.spct <-
  source_spct(w.length = attr(gap.mat, "w.length"),
              s.e.irrad = func_med_RP$data[1, ])
```

We plot one spectrum using plot method from package 'ggspectra'.

```
plot(func_med_mode.spct)
```
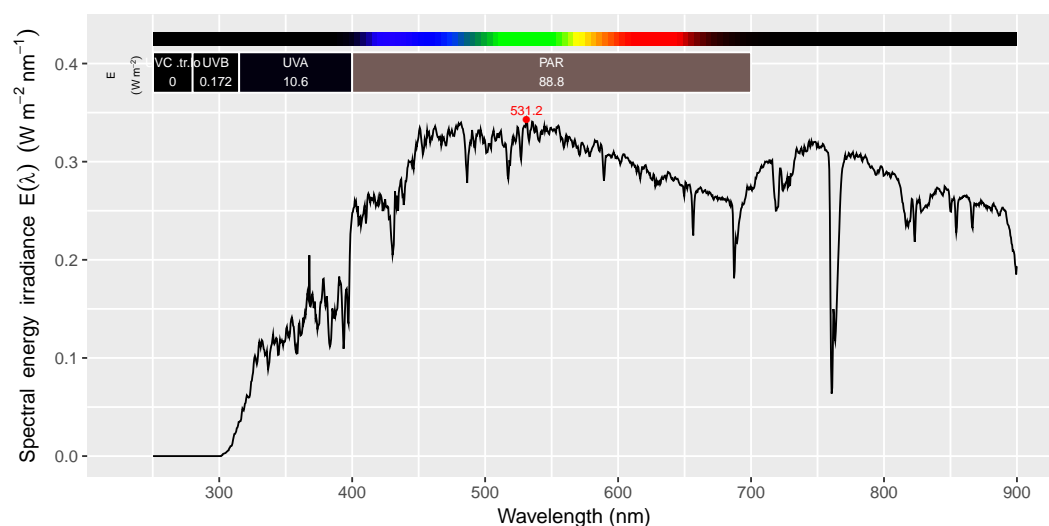

We calculate one summary.

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

##  Red.Smith10: FarRed.Smith10(q:q)
##                       0.8268562
## attr(,"radiation.unit")
## [1] "q:q ratio"
```

We create a collection of spectra.

```
gap_fda.mspct <- source_mspct(list(med_FM = func_med_FM.spct,
                 med_mode = func_med_mode.spct,
                 med_RP = func_med_RP.spct))
```

Calculate spectral summaries.

```
ratios <- q_ratio(gap_fda.mspct, Red("Smith10"), Far_red("Smith10"))
names(ratios) <- c("method", "R:FR")
ratios

## # A tibble: 3 × 2
##     method    `R:FR`
##     <fctr>     <dbl>
## 1   med_FM 0.8334827
## 2 med_mode 0.8268562
## 3   med_RP 0.8334827
```

We bind the three spectra to be able to plot them together.

```
gap_fda.spct <-
  rbindspct(gap_fda.mspct,
            idfactor = "method")
```

We plot three spectra using plot method from package 'ggspectra'.

```
plot(gap_fda.spct,
     annotations = c("color.guide", "boxes", "labels")) +
  aes(linetype = method)
```
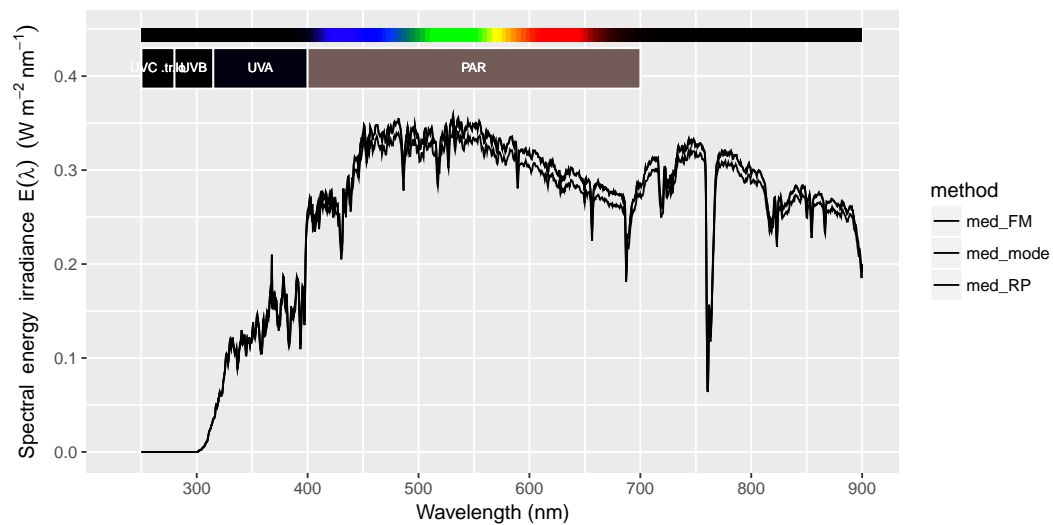


```
try(detach(package:fda.usc))
try(detach(package:fda))
try(detach(package:pavo))
try(detach(package:colorSpec))
try(detach(package:hyperSpec))
try(detach(package:ggspectra))
try(detach(package:ggplot2))
try(detach(package:lubridate))
```

```r
try(detach(package:photobiologyInOut))
try(detach(package:photobiologyWavebands))
try(detach(package:photobiologySun))
try(detach(package:photobiology))
try(detach(package:tibble))
```

19

Importing and exporting 'foreign' data

19.1 Introduction

The tasks described in this chapter concern the exchange of spectral information with other pieces of software. Most of the reading and writing operations described here are lossy or may require some help from the user to maintain data validity. For saving and restoring R objects of any class one should use R's built-in functions which ensure that the objects read-in (or restored) will be fully equivalent to the saved ones.

19.2 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(readr)
library(readxl)
library(photobiology)
library(photobiologyWavebands)
library(photobiologyInOut)
library(ggspectra)
#library(ooacquire}
#library(Yoctopuce}
library(lubridate)
```

19.3 Reading and writing common file formats

19.3.1 Task: Read and write spectra from text files

To read text files containing spectral data we use either base R's functions (`read.table`, `read.csv`, `read.csv2`) or preferably the corresponding functions from 'readr' (`read_table`, `read_csv`, `read_csv2`, `read_tsv`, `read_delim`, `read_fwf`). These functions return data frame and tibble objects, respectively, that can be converted into spectral objects as described in chapter 18.

To write a spectrum to a text file we can use base R's functions or from package 'readr' for *writing* data frames to text files. Once more, the functions in package 'readr' (`write_csv` , `write_excel_csv` , `write_tsv` , `write_delim`) are to be preferred.

⚠️ Writing spectral objects to text files does not preserve the metadata stored in attributes. It is lossy operation! To save to disk R objects in a way that they can be restored unchanged into the same or another *workspace* use R's function `save` . Such objects can be read without loss of information across different versions of R, and across different operating systems.

19.3.2 Task: Read a spectrum from an Excel workbook

Over the years different packages have become available for importing data from worksheets and workbooks. It has also been popular to export the worksheets as CSV (comma separated values) text files and then read these text files into R. The problem with the use of .CSV files is that the column separators and decimal markers depend on the locale in use when saved. Currently the best package for importing data from Excel workbooks (saved from recent versions of Excel), providing two functions with the easiest to use interface is 'readxl'. Although we provide some functions for importing data from specific instruments and programs, there are many instruments and software that produce worksheets or workbooks. There also many users who do the initial data processing in Excel. We give here an example using data acquired with our own Ocean Optics spectrometer and stored in an 'Excel' workbook (Fig. 19.1).

We skip 17 rows containing other data, and we select the worksheet named 'front'.

```
my.wbk.file <- "inout/spectrum-ylianttila.xlsx"
excel_sheets(my.wbk.file)

## [1] "front"          "calc"
## [3] "figures"        "constants"
## [5] "slit correction"

my.df <- read_excel(my.wbk.file, sheet = "front", skip = 17)
head(my.df)

## # A tibble: 6 × 9
##   Wavelength   short   long   short    long
##        <dbl>   <dbl>  <dbl>   <dbl>   <dbl>
## 1     200.00 3042.28 3686.6 3543.98  8142.8
## 2     200.47 2964.72 2965.4 2985.35  2985.8
## 3     200.95 2964.15 2966.0 2983.10  2984.9
## 4     201.42 2960.85 2960.9 2981.88  2979.5
```
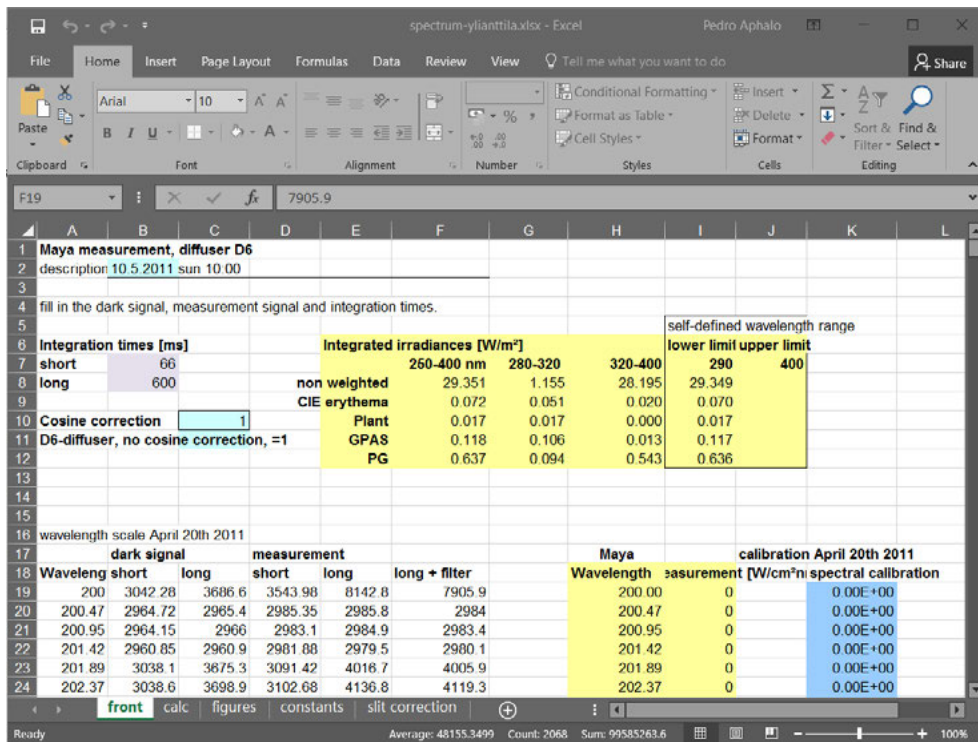
Figure 19.1: Screen capture showing the top of the 'Excel' worksheet containing data for the solar spectrum used in the example in section 19.3.2.

```
## # ... with 2 more rows, and 4 more variables:
## #   `long + filter` <dbl>, Wavelength <dbl>,
## #   `calibrated measurement [W/cm²nm]` <dbl>,
## #   `spectral calibration` <dbl>
```

Or selecting columns on the fly and then creating a `source_spct` object from this solar spectrum measured at ground level. As we do not really need the raw data, select all rows from columns 7 and 8. As Excel worksheets are read past the end of the data, we need to remove, the missing or NA (not available) values. We construct a `source_spct` object, but we multiply the spectral irradiance data by 10^4 to convert the units from $W\,cm^{-2}\,nm^{-1}$ to $W\,m^{-2}\,nm^{-1}$. Next we 'clip' the wavelengths outside the calibration range keeping data for the range 290–800 nm.

```
my.df <- read_excel(my.wbk.file, sheet = "front", skip = 17)[ , 7:8]
my.df <- na.omit(my.df)
my.spct <- source_spct(w.length = my.df[[1]], s.e.irrad = my.df[[2]] * 1e4)
my.spct <- clip_wl(my.spct, range = c(290,800))
```

Finally we plot the spectrum.

```
plot(my.spct)
```
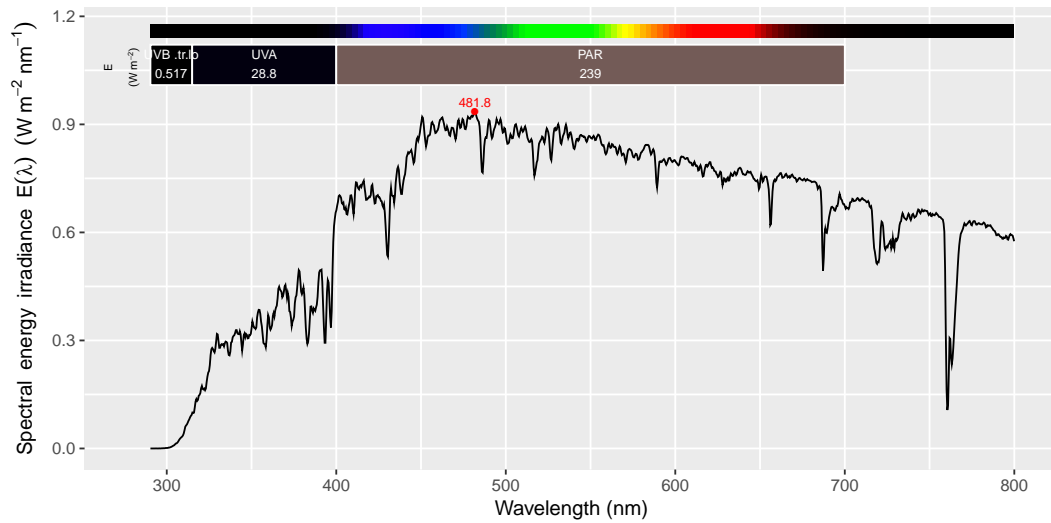

19.4 Reading instrument-output files

19.4.1 Task: Import data from Ocean Optics instruments and software

SpectraSuite

Reading spectral (energy) irradiance from a file saved in Ocean Optics SpectraSuite software (Fig. 19.2, now superseded by OceanView.

```
ooss.spct <- read_oo_ssirrad("inout/spectrum.SSIrrad")

## Parsed with column specification:
## cols(
##   w.length = col_double(),
##   s.e.irrad = col_double()
## )

ooss.spct

## Object: source_spct [1,044 x 2]
## Wavelength range 199.08 to 998.61 nm, step 0.72 to 0.81 nm
## Label: File: spectrum.SSIrrad
## Measured on 2013-05-06 15:13:40 UTC
## Time unit 1s
##
## # A tibble: 1,044 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
```

```
      SpectraSuite Data File
 +++++++++++++++++++++++++++++++++++++
 Date: Mon May 06 15:13:40 CEST 2013
 User: User
 Dark Spectrum Present: Yes
 Reference Spectrum Present: No
 Number of Sampled Component Spectra: 1
 Spectrometers: QEB1523
 Integration Time (usec): 100000 (QEB1523)
 Spectra Averaged: 1 (QEB1523)
 Boxcar Smoothing: 0 (QEB1523)
 Correct for Electrical Dark: No (QEB1523)
 Strobe/Lamp Enabled: No (QEB1523)
 Correct for Detector Non-linearity: No (QEB1523)
 Correct for Stray Light: Yes (QEB1523)
 Number of Pixels in Processed Spectrum: 1044
 >>>>>Begin Processed Spectral Data<<<<<
 199.08 0.0000E00
 199.89 0.0000E00
 200.70 0.0000E00
 201.50 1.3742E02
 202.31 1.2488E02
 .........
```

Figure 19.2: Top of text file `spectrum.SSirrad` written by Ocean Optics' 'SpectraSuite' software, and used in the example in section 19.4.1

```
## 1   199.08    0.0000
## 2   199.89    0.0000
## 3   200.70    0.0000
## 4   201.50    1.3742
## # ... with 1,040 more rows
```

The function accepts several optional arguments. Although the function by default attempts to read all information from the files, values like the date can be overridden and a geocode can be set.

```
ooss1.spct <- read_oo_ssirrad("inout/spectrum.SSIrrad",
                              date = now())

## Parsed with column specification:
## cols(
##  w.length = col_double(),
```

```
##   s.e.irrad = col_double()
## )

ooss1.spct
```

Jazz

Files saved by Ocean Optics *Jaz* spectrometers have a slightly different format (Fig. 19.3, and a function different function is to be used.

```
jaz.spct <- read_oo_jazirrad("inout/spectrum.JazIrrad")

## Parsed with column specification:
## cols(
##   W = col_double(),
##   D = col_double(),
##   S = col_double(),
##   P = col_double()
## )

jaz.spct

## Object: source_spct [2,048 x 2]
## Wavelength range 188.82523 to 1033.1483 nm, step 0.357056 to 0.459625 nm
## Label: File: spectrum.JazIrrad
## Measured on 2015-02-03 09:44:41 UTC
## Time unit 1s
##
## # A tibble: 2,048 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1 188.8252         0
## 2 189.2849         0
```

```
Jaz Absolute Irradiance File
++++++++++++++++++++++++++++++++++++
Date: Tue Feb 03 09:44:41 2015
User: jaz
Dark Spectrum Present: Yes
Processed Spectrum Present: Yes
Spectrometers: JAZA1065
Integration Time (usec): 193000 (JAZA1065)
Spectra Averaged: 3 (JAZA1065)
Boxcar Smoothing: 5 (JAZA1065)
Correct for Electrical Dark: Yes (JAZA1065)
Strobe/Lamp Enabled: No (JAZA1065)
Correct for Detector Non-linearity: Yes (JAZA1065)
Correct for Stray Light: No (JAZA1065)
Number of Pixels in Processed Spectrum: 2048
Fiber (micron): 3900
Collection Area: 0.119459
Int. Sphere: No
>>>>>Begin Processed Spectral Data<<<<<
W D S P
188.825226 0.000000     0.000000     0.000000
189.284851 0.000000     0.000000     0.000000
189.744415 -89.659378 -90.917900 -0.000000
190.203964 -106.165916 -96.419785 0.000000
.........
```

Figure 19.3: Top of text file `spectrum.JazIrrad` written by Ocean Optics' Jaz spectrometer, and used in the example in section 19.4.1

```
## 3 189.7444          0
## 4 190.2040          0
## # ... with 2,044 more rows
```

Function `read_oo_jazirrad` accepts the same arguments as function `read_oo_ssirrad`.

```
jaz1.spct <- read_oo_jazirrad("inout/spectrum.JazIrrad", date = now())

## Parsed with column specification:
## cols(
##   W = col_double(),
##   D = col_double(),
##   S = col_double(),
##   P = col_double()
```

19

```
## )
```

```
jaz1.spct
```

```
## Object: source_spct [2,048 x 2]
## Wavelength range 188.82523 to 1033.1483 nm, step 0.357056 to 0.459625 nm
## Label: File: spectrum.JazIrrad
## Measured on 2016-12-27 21:30:58 UTC
## Time unit 1s
##
## # A tibble: 2,048 × 2
##    w.length s.e.irrad
##       <dbl>     <dbl>
## 1 188.8252         0
## 2 189.2849         0
## 3 189.7444         0
## 4 190.2040         0
## # ... with 2,044 more rows
```

19.4.2 Task: Import data from Avantes instruments and software

```
avantes.spct <- read_avaspec_csv("inout/spectrum-avaspec.csv",
                                 date = now())
```

```
## Parsed with column specification:
## cols(
##   w.length = col_double(),
##   s.e.irrad = col_double()
## )
```

```
avantes.spct
```

```
## Object: source_spct [1,604 x 2]
## Wavelength range 172.485 to 1100.222 nm, step 0.544 to 0.607 nm
## Label: File: spectrum-avaspec.csv
## Measured on 2016-12-27 21:30:58 UTC
## Time unit 1s
##
## # A tibble: 1,604 × 2
##    w.length    s.e.irrad
##       <dbl>        <dbl>
## 1   172.485 9.27919e-12
## 2   173.091 1.10348e-11
## 3   173.697 1.04457e-11
## 4   174.303 8.93041e-12
## # ... with 1,600 more rows
```

```
@19/5/1997
@17:44:58
#No Title
 2.5000000000E+02
 0.0000000000E+00
 2.5100000000E+02
 0.0000000000E+00
 2.5200000000E+02
 0.0000000000E+00
 2.5300000000E+02
 0.0000000000E+00
.........
```

Figure 19.4: Top of text file `spectrum.DTA` written by the software of a Macam scanning spectrometer, and used in the example in section 19.4.3. In this case wavelengths and spectral irradiance are 'interlaced' in a single column of numbers and metadata is minimal.

19.4.3 Task: Import data from Macam instruments and software

The Macam PC-1900 spectroradiometer and its companion software save data in a simple text file. Data is always stored as spectral (energy) irradiance, so spectral data can be easily decoded. All the files we have tested had the name tag ".DTA". In Figure 19.4 the top of a file output by the Macam software is shown.

```
macam.spct <- read_macam_dta("inout/spectrum.DTA")
macam.spct

## Object: source_spct [151 x 2]
## Wavelength range 250 to 400 nm, step 1 nm
## Label: File: spectrum.DTA
## Measured on 1997-05-19 17:44:58 UTC
## Time unit 1s
##
## # A tibble: 151 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      250         0
## 2      251         0
## 3      252         0
## 4      253         0
## # ... with 147 more rows
```

Function `read_macam_dta` accepts the same arguments as function `read_ooss_file`.

315

```
macam1.spct <- read_macam_dta("inout/spectrum.DTA",
                              date = now())
macam1.spct

## Object: source_spct [151 x 2]
## Wavelength range 250 to 400 nm, step 1 nm
## Label: File: spectrum.DTA
## Measured on 2016-12-27 21:30:58 UTC
## Time unit 1s
##
## # A tibble: 151 × 2
##   w.length s.e.irrad
##      <dbl>     <dbl>
## 1      250         0
## 2      251         0
## 3      252         0
## 4      253         0
## # ... with 147 more rows
```

19.4.4 Task: Import data from LI-COR instruments and software

The LI-COR LI-1800 spectroradiometer and its companion software can save data either as spectral photon irradiance or spectral (energy) irradiance. As files are labelled accordingly, our function, automatically detects the type of data being read. Be aware that the function is not able to decode the binary files ".DAT". Only ".PRN" as converted by LI-COR's PC1800 software can be decoded by our function. In Figure 19.5 the top of a file output by the Macam software is shown.

```
licor.spct <- read_licor_prn("inout/spectrum.PRN")
licor.spct

## Object: source_spct [601 x 2]
## Wavelength range 300 to 900 nm, step 1 nm
## Label: File: spectrum.PRN
## Measured on 0000-08-23 16:32:00 UTC
## Time unit 1s
##
## # A tibble: 601 × 2
##   w.length s.q.irrad
##      <dbl>     <dbl>
## 1      300 1.518e-10
## 2      301 3.355e-10
## 3      302 2.197e-10
## 4      303 3.240e-10
## # ... with 597 more rows
```

```
"FILE:FL2"
"REM: TLD 36W/865       (QNTM)"
"LIMS: 300– 900NM"
"INT:  1NM"
"DATE:08/23 16:32"
"MIN:  300NM  1.518E–04"
"MAX:  546NM  7.491E–01"
 300  1.518E–04
 301  3.355E–04
 302  2.197E–04
 303  3.240E–04
.........
```

Figure 19.5: Top of text file `spectrum.PRN` written by the software of a LI-COR LI-1800 scanning spectrometer, and used in the example in section 19.4.4. One limitation is that the year is missing from the date.

Function `read_licor_file` accepts the same arguments as function `read_ooss_file`.

```r
licor1.spct <- read_licor_prn("inout/spectrum.PRN",
                              date = now())
licor1.spct

## Object: source_spct [601 x 2]
## Wavelength range 300 to 900 nm, step 1 nm
## Label: File: spectrum.PRN
## Measured on 2016–12–27 21:30:58 UTC
## Time unit 1s
##
## # A tibble: 601 × 2
##   w.length s.q.irrad
##      <dbl>     <dbl>
## 1      300 1.518e–10
## 2      301 3.355e–10
## 3      302 2.197e–10
## 4      303 3.240e–10
## # ... with 597 more rows
```

19.4.5 Task: Import data from Bentham instruments and software

19

317

20

Data acquisition from within R

20.1 Introduction

The tasks described in this chapter concern the acquisition of spectral and other data directly from sensors, spectrometers and other pieces of equipment. We also touch again here the question of reading raw counts data from files, but only when the aim is further processing into other spectral quantities using the same functions as for directly acquired data. In the current version of the handbook and packages we discuss only spectrometers from Ocean Optics, as these are only ones with which we have several years of use experience.

20.2 Packages and other software used in this chapter

For the examples in this section to work, you will need to have Java and the 'OmniDriver' runtime installed. In addition examples as shown assume that an Ocean Optics spectrometer is connected. The output will depend on the model(s) and configuration(s) of the instrument(s) connected. The plural is correct, you can acquire spectra from more than one instrument, and from instruments with more than one channel.

Package 'rOmniDriver' is just a thin wrapper on the low-level access functions supplied by the driver. The names for functions in package 'rOmniDriver' are verbose, this is because we have respected the names used in the driver itself, written in Java. Thus was done so that information in the driver documentation can be found easily.For executing the examples listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)
library(photobiologyWavebands)
library(ggspectra)
library(rOmniDriver)
library(ooacquire)
#library(Yoctopuce)
library(lubridate)
```

20.3 Adcquiring spectra with Ocean Optics spectrometers

20.3.1 Task: Acquiring data from Ocean Optics instruments

In this first example we use low-level functions, mapping directly in most cases to the OmniDrive Java API.

First step is to load the package 'rOmniDriver' which is a low level wrapper on the driver supplied by Ocean Optics for their instruments. The runtime is free, and is all what you need for simple tasks as documentation is available both from Ocean Optics web site and as R help.

After physically connecting the spectrometer through USB, the data connection needs to be initiated and the instrument id obtained. This function returns a 'Java wrapper' object that will be used for all later operations and needs to be saved to variable. The second statement queries the number of spectrometers, or spectrometer modules in the case of the *Jaz*.

```
srs <- init_srs()
num_srs <- number_srs()
```

Indexing starts at zero, contrary to R's way, so the first spectrometer has index '0', the second index '1', etc.

We will now assume that only one spectrometer is attached to the computer, and just rely on the default index value of 0, which always points to the first available spectrometer. The next step, unless we always use the same instrument is to query for a description of the optical bench of the attached instrument.

```
get_name(srs)
get_serial_number(srs)
get_bench(srs)
```

If you are writing a script that should work with different instruments, you may need to query whether a certain function is available or not in the attached instrument. On the other hand, functions like those used for setting the integration time can be just assumed to be always available. Many functions come in pairs of `set` and `get` versions. The only thing to be careful with is that in some cases, the `set` functions can silently fail, ignoring the requested set operation. For this reason, scripts have to be written so that these functions are not assumed to always work. The most important case, setting the integration time, can be easily dealt with in two different ways: 1) being careful the set function is never passed as argument an off-range length of time value, or even more reliably, 2) always using the corresponding get function after each call to set, to obtain the value actually stored in the memory of

the spectrometer. Not following these steps can result in errors of any size, and render the data useless as the calculated counts-per-second values will be wrong.

```
set_integration_time(srs, time.usec = 100)
get_integration_time(srs)
```

We can similarly set the number of scans to average.

```
set_scans_to_avg(srs, 5)
get_scans_to_avg(srs)
```

To obtain data we use function `get_spectrum`

```
counts <- get_spectrum(srs)
```

```
srs_close(srs)
```

Users will rarely use these functions on a regular basis. They will either use the predefined high level functions from package 'ooacquire' or write similar functions themselves encapsulating the different steps of the acquisition and data processing.

20.3.2 Task: Acquiring spectral irradiance with Ocean Optics spectrometers

20.3.3 Task: Acquiring spectral transmittance with Ocean Optics spectrometers

20.3.4 Task: Acquiring spectral reflectance with Ocean Optics spectrometers

20.3.5 Task: Acquiring spectral absorptance with Ocean Optics spectrometers

20.4 sglux spectrometers and sensors

20.4.1 Task: Acquiring spectral data with sglux instrument

20.5 YoctoPuce modules

20.5.1 Task: Acquiring data with YoctoPuce modules and servers

20

21

Calibration

21.1 Task: Calibration of broadband sensors

21.2 Task: Correcting for non-linearity of sensor response

21.3 Task: Applying a spectral calibration to raw spectral data

21.4 Task: Wavelength calibration and peak fitting

22

Simulation

325

Part V

Catalogue of example data

23

Radiation sources

23.1 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(photobiologySun)
library(photobiologyLamps)
library(photobiologyLEDs)
```

23.2 Introduction

23.3 Data: extraterrestrial solar radiation spectra

23.4 Data: terrestrial solar radiation spectra

- Global irradiance consists of both direct and diffuse components:
  - Direct irradiance is solar radiation that comes in a straight line from the direction of the sun at its current position in the sky.
  - Diffuse irradiance is solar radiation that has been scattered by molecules and particles in the atmosphere or scattered and reflected from the surroundings.

- Two standardization parties commonly referred to, the American Society for Testing and Materials (ASTM) and the International Electrotechnical Commission (IEC) both use the Simple Model of the Atmospheric Radiative Transfer of Sunshine (SMARTS) program to generate terrestrial reference spectra for photovoltaic system performance evaluation and product comparison and hence their reference spectra are essentially the same.

- The standards define global irradiance where the receiving surface is defined in the standards as an inclined plane at 37° tilt toward the equator, facing the sun

(i.e., the surface normal points to the sun, at an elevation of 41.81° above the horizon), and expressed as $\mathrm{W\,m^{-2}\,nm^{-1}}$.

- American Society for Testing and Materials (ASTM) defines two standard terrestrial solar spectral irradiance distributions:
  - The AM1.5 Global irradiance spectrum
  - The AM1.5 Direct (+circumsolar) spectrum that is defined for solar concentrator work. It includes the the direct beam from the sun plus the circumsolar component, that results from scattered radiation appearing to come from around the solar disk.

- There is no standard sun spectrum specified separately for plant photobiology or horticulture.

- Plant photobiologists refer to these SMARTS derived spectrum as well.

- The global irradiance spectrum can be considered to be more relevant for plant photobiology and horticulture.

23.5 Data: radiation within plant canopies

23.6 Data: radiation in water bodies

23.7 Data: incandescent lamps

23.8 Data: discharge lamps

23.9 Data: LEDs

24

Optical properties of inanimate objects

24.1 Packages used in this chapter

For accessing the example data listed in this chapter you need first to load the following packages from the library:

```
library(photobiology)
library(photobiologyFilters)
library(photobiologyReflectors)
```

24.2 Introduction

24.3 Data: spectral transmittance of filters

24.4 Data: spectral reflectance of filters

24.5 Data: spectral transmittance of common materials

24.6 Data: spectral reflectance of common materials

25

Example data for organisms

25.1 Plants

25.1.1 Data: Surface properties of organs

25.1.2 Data: Photoreceptors

25.1.3 Data: Photosynthesis

25.1.4 Data: Damage

25.1.5 Data: Metabolites

25.2 Animals, including humans

25.2.1 Data: Surface properties of organs

25.2.2 Data: Photoreceptors

25.2.3 Data: Light driven synthesis

25.2.4 Data: Damage

25.2.5 Data: Metabolites

25.3 Microbes

25.3.1 Data: Photoreceptors

25.3.2 Data: Light driven synthesis

25.3.3 Data: Damage

25.3.4 Data: Metabolites

26

Further reading

26.1 Radiation physics

26.2 Photochemistry

26.3 Photobiology

26.4 Using R

26.5 Programming in 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 (cit. on p. 30).

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 (cit. on p. 14).

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. xxii, 3, 5, 49, 78).

Aphalo, P. J. (2016). *Learn R ...as you learnt your mother tongue*. Helsinki: Leanpub. eprint: http://leanpub.com/learnr (cit. on pp. xxii, 50, 180).

ASTM (2012). *Tables for Reference Solar Spectral Irradiances: Direct Normal and Hemispherical on 37 Tilted Surface. ASTM G173-03(2012)*. West Conshohocken: ASTM International. DOI: 10.1520/g0173-03r12. URL: http://dx.doi.org/10.1520/G0173-03R12 (cit. on p. 18).

Björn, L. O. (2015). *Photobiology: The Science of Light and Life*. 3rd ed. New York: Springer. ISBN: 9781493914685. URL: https://books.google.fi/books?id=VDgLBgAAQBAJ (cit. on p. xxii).

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 (cit. on p. 30).

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 pp. xxvi, 136).

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 (cit. on p. 30).

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 (cit. on p. 30).

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 (cit. on p. 22).

Eichler, H.-. J., A. Fleischner, J. Kross, M. Krystek, H. Lang, H. Niedrig, H. Rauch, G. Schmahl, H. Schoenebeck, E. Sedlmayr, et al. (1993). *Bergmann, Schaefer: Lehrbuch der Experimentalphysik Band 3: Optik*. Ed. by H. Niedrig. Verlag Walter de Gruyter Berlin/New York (cit. on p. 6).

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 (cit. on p. 22).

Fröhlich, C. and J. Lean (2004). "Solar radiative output and its variability: evidence and mechanisms". In: *The Astronomy and Astrophysics Review* 12, pp. 273–320. DOI: 10.1007/s00159-004-0024-1 (cit. on p. 15).

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 (cit. on p. 20).

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 (cit. on p. 30).

– (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 (cit. on p. 30).

– (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 (cit. on p. 30).

– (2004). "UV Radiation Penetration in Plant Canopies". In: *Encyclopedia of Plant and Crop Science*, pp. 1261–1264. DOI: 10.1081/E-EPCS-120010624 (cit. on p. 30).

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 pp. xxvi, 136).

Gueymard, C. A. (2004). "The sun's total and spectral irradiance for solar energy applications and solar radiation models". In: *Solar Energy* 76.4, pp. 423–453. DOI:

10.1016/j.solener.2003.08.039. URL: http://dx.doi.org/10.1016/j.solener.2003.08.039 (cit. on p. 16).

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 (cit. on p. 22).

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 (cit. on p. 22).

Hillebrand, J. and M. H. Nierhoff (2015). *Mastering RStudio - Develop, Communicate, and Collaborate with R*. Packt Publishing. 348 Seiten. URL: http://www.ebook.de/de/product/25599227/julian_hillebrand_maximilian_h_nierhoff_mastering_rstudio_develop_communicate_and_collaborate_with_r.html (cit. on p. 51).

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 (cit. on p. 29).

Horton, N. (2015). *Using R and RStudio for data management, statistical analysis, and graphics*. Boca Raton: CRC Press. ISBN: 9781482237368. URL: https://www.crcpress.com/Using-R-and-RStudio-for-Data-Management-Statistical-Analysis-and-Graphics/Horton-Kleinman/9781482237368 (cit. on p. 50).

Horton, N. J., R. Pruim, and D. T. Kaplan (2015). *A Student's Guide to R*. 1.2. 119 pp. URL: https://github.com/ProjectMOSAIC/LittleBooks/blob/master/README.md (visited on 07/17/2016) (cit. on p. xxii).

Iqbal, M. (1983). *An introduction to solar radiation*. Academic Press Canada (cit. on p. 15).

ISO (2007). *Space environment (natural and artificial) - Process for determining solar irradiances. ISO Standard 21348*. Tech. rep. Geneva.: ISO (cit. on p. 3).

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 (cit. on p. 15).

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 (cit. on p. 20).

Lindfors, A. V. and L. Ylianttila (2015). "Visualizing Rayleigh Scattering through UV Photography". In: *Bulletin of the American Meteorological Society*, e-View. DOI: 10.1175/BAMS-D-14-00260.1 (cit. on p. 17).

26

Loo, M. P. van der and E. de Jonge (2012). *Learning RStudio for R Statistical Computing: : : Amazon.com: Books*. 1st ed. Birmingham: Packt Publishing, p. 126. ISBN: 9781782160601. URL: http : / / www . amazon . com / Learning – RStudio – R – Statistical – Computing / dp / 1782160604 / ref=sr%5C_1%5C_2?s=books% 5C&ie=UTF8%5C&qid=1409482367%5C&sr=1–2%5C&keywords=RStudio (cit. on p. 51).

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 (cit. on p. 20).

Meeus, J. (1998). *Astronomical Algorithms*. Willmann-Bell. ISBN: 0943396611 (cit. on pp. 40, 155).

Mobley, C. D. (1994). *Light and water - radiative transfer in natural waters*. San Diego: Academic Press. URL: http://www.curtismobley.com/lightandwater.zip (cit. on pp. 8, 15).

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 (cit. on p. 29).

Monteith, J. and M. Unsworth (2008). *Principles of Environmental Physics*. 3rd ed. Academic Press. 440 pp. ISBN: 0125051034 (cit. on p. 30).

Morgan, D. and H. Smith (1981). "Control of development in Chenopodium album L. by shadelight: the effect of light quantity (total fluence rate) and light quality (red:far-red ratio)". In: *New Phytologist* 88.2, pp. 239–248. DOI: 10.1111/j.1469– 8137.1981.tb01720.x (cit. on p. 127).

Paradis, E. (2005). *R for Beginners*. Montpellier. 76 pp. URL: https : / / cran . r – project.org/doc/contrib/Paradis–rdebuts_en.pdf (visited on 07/17/2016) (cit. on pp. xxii, 50).

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 (cit. on p. 30).

Peng, R. D. (2016). *R Programming for Data Science*. Leanpub. 182 pp. URL: https: //leanpub.com/rprogramming (visited on 08/07/2016) (cit. on pp. xxii, 50).

Prahl, S. A., M. Keijzer, S. L. Jacques, and A. J. Welch (1989). "A Monte Carlo Model of Light Propagation in Tissue". In: *SPIE Proceedings of Dosimetry of Laser Radiation in Medicine and Biology*. Ed. by G. J. Müller and D. H. Sliney. Vol. IS 5, pp. 102–111 (cit. on p. 8).

Schwiegerling, J. (2004). *Field guide to visual and ophthalmic optics*. SPIE Press, Bellingham, WA (cit. on p. 10).

Sellaro, R., M. Crepy, S. A. Trupkin, E. Karayekov, A. S. Buchovsky, C. Rossi, and J. J. Casal (2010). "Cryptochrome as a sensor of the blue / green ratio of natural radiation in Arabidopsis." eng. In: *Plant Physiology* 154.1, pp. 401–409. DOI: 10.1104/pp.110.160820 (cit. on p. 5).

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. xxv).

Smith, H. (1981). *Plants and the Daylight Spectrum*. London: Academic Press (cit. on p. 5).

Somasundaram, R. (2013). *Git*. Packt Publishing. 180 Seiten. ISBN: 1849517525. URL: http://www.ebook.de/de/product/20403701/ravishankar_somasundaram_git.html (cit. on p. 51).

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 (cit. on p. 22).

Swicegood, T. (2010). *Pragmatic Guide to Git*. eBook Pedro: Pragmatic Programmers, LLC. ISBN: 978-1-934356-72-2 (cit. on p. 51).

Technical Committee ISO/TC 20, A., S. s. space vehicles Subcommittee SC 14, and operations (2007). *Space environment (natural and artificial) – Process for determining solar irradiances*. English. Satandard 21348:2007. ISO (the International Organization for Standardization). 12 pp. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=39911 (cit. on p. 5).

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. xxvi).

Wang, L.-. H., S. L. Jacques, and L.-. Q. Zheng (1995). "MCML - Monte Carlo modeling of photon transport in multi-layered tissues". In: *Computer Methods and Programs in Biomedicine* 47, pp. 131–146. DOI: 10.1016/0169-2607(95)01640-F (cit. on p. 8).

Wehrli, C. (1985). *Extraterrestrial solar spectrum*. PMOD/WRC Publication 615. Physikalisch-Meteorologisches Observatorium und World Radiation Center Davos Dorf, Switzerland (cit. on p. 17).

Wickham, H. (2014). "Tidy Data". In: *Journal of Statistical Software* 59.10. ISSN: 1548-7660. URL: http://www.jstatsoft.org/v59/i10 (cit. on p. 56).

26

Glossary

absorbance $A = \log E_0/E_1$, where E_0 is the incident irradiance, and E_1 is the transmitted irradiance. xxv

absorptance radiation that is absorbed by an object, as a fraction of the incident irradiance: $\alpha = E_{abs}/E_0$, where E_0 is the incident irradiance and E_{abs} is the absorbed irradiance. xxv

biological spectral weighting function a function used to estimate the biological effect of radiation. It is convoluted—i.e. multiplied wavelength by wavelength— with the spectral irradiance of a source of UV radiation to obtain a biologically effective irradiance. xxv

CRAN, Comprehensive R Archive Network A network of software and documentation repositories for R packages and R itself. 57

direct radiation solar radiation that arrives directly at the ground level, without being scattered by gases and particles of the atmosphere. 17, 30

global radiation total solar radiation arriving at ground level. It is the sum of direct and diffuse radiation. 17, *see* direct radiation

isotropic radiation is isotropic when it arrives equally from all directions, e.g. it is completely diffuse. *see* scattered or 'diffuse' radiation

photosynthetic photon flux density another name for 'PAR photon irradiance'. xxvii

photosynthetically active radiation radiation driving photosynthesis in higher plants, it describes a wavelength range—i.e. λ =400–700 nm—but does not define whether an energy or photon quantity is being used. xxvi

proportional-integral-derivative a *proportional integral derivative* controller (PID controller) is a control loop feedback mechanism. A PID controller calculates an "error" value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting

the process control inputs. A well tuned PID controller (with correct parameters) minimizes overshoot and transient deviations, by adjusting, for example, the dimming in a modulated system based on the size of the error and the response characteristics of the controlled system. xxvii

radiation amplification factor gives the percent change in biologically effective UV irradiance for a 1% change in stratospheric ozone column thickness. Its value varies with the BSWF used in the calculation. xxvii

reflectance radiation that is reflected by an object, as a fraction of the incident irradiance: $\rho = E_{\mathrm{rfl}}/E_0$, where E_0 is the incident irradiance and E_{rfl} is the reflected irradiance. xxv

scattered or 'diffuse' radiation solar radiation that arrives at ground level after being scattered by gases and particles of the atmosphere, also called 'diffuse radiation'. 17

transmittance radiation that is transmitted by an object, as a fraction of the incident irradiance: $\tau = E_{\mathrm{trs}}/E_0$, where E_0 is the incident irradiance and E_{trs} is the transmitted irradiance. xxv

Index

26

Part VI

Appendix

A

Build information

```
Sys.info()

##        sysname         release         version
##      "Windows"        "10 x64"  "build 14393"
##       nodename         machine           login
##        "MUSTI"        "x86-64"        "aphalo"
##           user  effective_user
##       "aphalo"        "aphalo"
```

```
sessionInfo()

## R version 3.3.2 (2016-10-31)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## 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] splines   grid      methods   stats
## [5] graphics  grDevices utils     datasets
## [9] base
##
## other attached packages:
##  [1] ooacquire_0.1.1
##  [2] rOmniDriver_0.1.11.9000
##  [3] rJava_0.9-8
##  [4] lubridate_1.6.0
##  [5] ggspectra_0.2.0
##  [6] photobiologyInOut_0.4.12.9000
##  [7] photobiologyWavebands_0.4.1.9000
##  [8] photobiology_0.9.13.9000
##  [9] readxl_0.1.1
```

```
## [10] readr_1.0.0
## [11] rpart_4.1-10
## [12] mgcv_1.8-15
## [13] nlme_3.1-128
## [14] MASS_7.3-45
## [15] Matrix_1.2-7.1
## [16] ggrepel_0.6.5
## [17] caret_6.0-73
## [18] ggplot2_2.2.0
## [19] lattice_0.20-34
## [20] signal_0.7-6
## [21] rootSolve_1.7
## [22] rgdal_1.2-5
## [23] raster_2.5-8
## [24] sp_1.2-4
## [25] polynom_1.3-9
## [26] knitr_1.15.1
##
## loaded via a namespace (and not attached):
##  [1] pavo_1.0.0
##  [2] pbkrtest_0.4-6
##  [3] RColorBrewer_1.1-2
##  [4] latex2exp_0.4.0
##  [5] tensorA_0.36
##  [6] tools_3.3.2
##  [7] R6_2.2.0
##  [8] DBI_0.5-1
##  [9] photobiologySun_0.3.7
## [10] lazyeval_0.2.0
## [11] colorspace_1.3-2
## [12] nnet_7.3-12
## [13] gridExtra_2.2.1
## [14] bayesm_3.0-2
## [15] compositions_1.40-1
## [16] quantreg_5.29
## [17] photobiologyPlants_0.4.0.9001
## [18] photobiologyReflectors_0.3.2
## [19] photobiologyLamps_0.4.1
## [20] SparseM_1.74
## [21] labeling_0.3
## [22] photobiologyFilters_0.4.3.9000
## [23] scales_0.4.1
## [24] DEoptimR_1.0-8
## [25] robustbase_0.92-7
## [26] stringr_1.1.0
## [27] digest_0.6.10
## [28] colorSpec_0.5-3
## [29] minqa_1.2.4
```

```
## [30] photobiologyLEDs_0.4.2
## [31] fda.usc_1.3.0
## [32] jpeg_0.1-8
## [33] lme4_1.1-12
## [34] highr_0.6
## [35] maps_3.1.1
## [36] svUnit_0.7-12
## [37] energy_1.7-0
## [38] dplyr_0.5.0
## [39] ModelMetrics_1.1.0
## [40] car_2.1-4
## [41] magrittr_1.5
## [42] geosphere_1.5-5
## [43] Rcpp_0.12.8
## [44] munsell_0.4.3
## [45] proto_1.0.0
## [46] scatterplot3d_0.3-37
## [47] stringi_1.1.2
## [48] plyr_1.8.4
## [49] hsdar_0.5.1
## [50] parallel_3.3.2
## [51] mapproj_1.2-4
## [52] rcdd_1.1-10
## [53] boot_1.3-18
## [54] rjson_0.2.15
## [55] fda_2.4.4
## [56] reshape2_1.4.2
## [57] codetools_0.2-15
## [58] hyperSpec_0.98-20161118
## [59] stats4_3.3.2
## [60] magic_1.5-6
## [61] evaluate_0.10
## [62] latticeExtra_0.6-28
## [63] splus2R_1.2-2
## [64] png_0.1-7
## [65] nloptr_1.0.4
## [66] foreach_1.4.3
## [67] RgoogleMaps_1.4.1
## [68] MatrixModels_0.4-1
## [69] gtable_0.2.0
## [70] tidyr_0.6.0
## [71] assertthat_0.1
## [72] geometry_0.3-6
## [73] tibble_1.2
## [74] iterators_1.0.8
## [75] ggtern_2.2.0
## [76] ggmap_2.6.1
```

A