Introduction to spectrolab

Jose Eduardo Meireles, Anna K. Schweiger and Jeannine Cavender-Bares

The mission of `spectrolab` is to provide a set of standards as well as a unified, robust interface for spectroscopic analyses in ecology and plant biology in R that improves collaboration and reproducibility by facilitating sharing of data and analyses. `spectrolab` provides methods to read, process and visualize spectra and establishes a common interface other packages can build on. The package introduces a `spectra` S3 class and packs a ton of functionality:

- Read spectra from raw spectral files or matrices
- Access, aggregate, subset, split or combine spectra
- Seamlessly link and manipulate metadata (such as chemistry)
- Plot spectra or spectral quantiles, shade spectral regions (e.g. VIS)
- Scroll through and zoom in spectra interacively
- Perform tasks such as vector normalization, smoothing, resampling, and sensor overlap matching

The source code can be found on our GitHub repository. Please report any bugs and ask us your questions through the issue tracker.

1 Installing and loading `spectrolab`

The latest stable version of `spectrolab` is on CRAN. Install it with:

```
install.packages("spectrolab")
```

You can also install it directly from GitHub using the `devtools` package.

```
library("devtools")
install_github("meireles/spectrolab")
```

Assuming that everything went smoothly, you should be able to load `spectrolab` like any other package.

2 Reading spectra

There are two ways to read spectra into R: 1) converting a matrix or data.frame to a `spectra` object and 2) reading spectra from raw data files (e.g. Spectra Vista's `.sig`, Spectral Evolution's `.sed`, ASD's `.asd`, ENVI `.txt` files). Here are a couple examples:

2.1 Create spectra object from a matrix or data.frame

If you already have your spectra in a matrix or data frame (e.g. when you read your data from a .csv file), you can use the function `as.spectra()` to convert it to a `spectra` object. The matrix **must** have samples in rows and wavelengths in columns. The header of the wavelength columns must be numeric wavelength labels. You also need to declare which column holds the sample names (they are mandatory) using the `name_idx` argument and which columns contain metadata (they are optional) using the `meta_idxs` argument.

Here is an example using a dataset matrix named `spec_matrix_meta.csv` provided by the package.

```
dir_path = system.file("extdata/spec_matrix_meta.csv", package = "spectrolab")

# Read data from the CSV file. If you don't use `check.names` = FALSE when reading
```

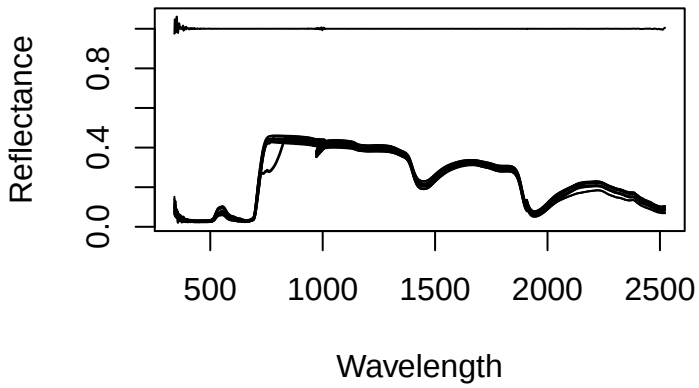
```r
# the csv, R will usually add a letter to the column names (e.g. 'X650') which will
# cause problems when converting the matrix to spectra.
spec_csv = read.csv(dir_path, check.names = FALSE)

# The sample names are in column 3. Columns 1 and 2 are metadata
achillea_spectra = as.spectra(spec_csv, name_idx = 3, meta_idxs = c(1,2) )

# And now you have a spectra object with sample names and metadata
achillea_spectra
```

```
## spectra object
## number of samples: 10
## wavelengths: 400 to 2400 (2001 bands)
## metadata (2): ident, ssp
```

2.2 Reading spectra from raw data: example with SVC's `.sig` files

The function `read_spectra()` reads raw spectra files. You can pass a vector of file names to `read_spectra()`, but it is usually easier to pass the path to the folder where your data are stored.

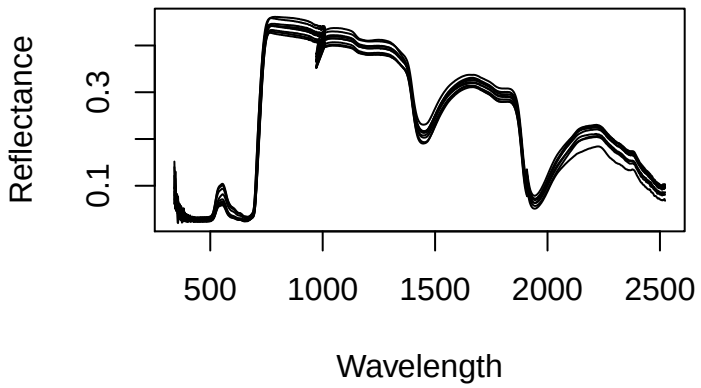
```r
# `dir_path` is the directory where our example datasets live
dir_path = system.file("extdata", "Acer_example", package = "spectrolab")

# Read .sig files
acer_spectra = read_spectra(path = dir_path, format = "sig")

# Plot the spectra
plot(acer_spectra)
```



In the example above you see one spectrum of a white reference panel (at ~100% reflectance) and two bad measurement (with dips at the red edge shoulder). It is generally good practice to quickly look at each spectrum during measurements and to flag the files of weird looking or otherwise special spectra. Here, we used the suffixes "_WR" and "_BAD" to denote white reference and bad measurements, respectively (you can pick whatever flag you want through). You can avoid importing such files by passing those flags to the argument `exclude_if_matches` in `read_spectra()`. Alternatively, you can read all spectra and use the flags for subsetting spectra based their name labels or passing the flags to a metadata argument (see section **Metadata** below).

```r
# Use the `exclude_if_matches` argument to excluded flagged files
acer_spectra_good = read_spectra(path = dir_path, format = "sig",
                                 exclude_if_matches = c("BAD","WR"))
```

```
# And plot the spectra
plot(acer_spectra_good)
```



Flagging unusual measurements during data collection speeds up data cleaning. However, you can also exclude bad spectra, outliers and unwanted samples by subsetting your spectra object, as shown in section **Subsetting spectra** below. Usually, it makes sense to check the data quality visually and to define exclusion criteria if needed. We give some examples of `spectrolab`'s plotting functions in section **Plotting spectra** below. But first, let's take a closer look at the structure of `spectra` objects.

3 Spectra objects

3.1 Inspecting and querying spectra objects

You can check out your spectra object in several ways. For instance, you may want to know how many samples and bands you have measured, retrieve file names and wavelengths.

```
# For an overview simply print the spectra object
achillea_spectra
```

```
## spectra object
## number of samples: 10
## wavelengths: 400 to 2400 (2001 bands)
## metadata (2): ident, ssp
```

```
# Get the dataset's dimensions
dim(achillea_spectra)
```

```
##     n_samples n_wavelengths
##            10          2001
```

`spectrolab` also lets you access the individual components of the `spectra`. This is done with the functions `names()` for sample names, `wavelengths()` for wavelength labels, `reflectance()` for the reflectance matrix, and `meta()` for the associated metadata (in case you have any).

```
# Vector of all sample names. Note: Duplicated sample names are permitted
names(achillea_spectra)
```

```
##  [1] "ACHMI_1"  "ACHMI_2"  "ACHMI_3"  "ACHMI_4"  "ACHMI_5"  "ACHMI_6"
##  [7] "ACHMI_7"  "ACHMI_8"  "ACHMI_9"  "ACHMI_10"
```

```
# Vector of wavelengths, inspect the first 20 wavelengths
w <- wavelengths(achillea_spectra)
w[1:20]
```

```
##  [1] 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416
## [18] 417 418 419
```

```
# Reflectance matrix, inspect reflectance values for the first 2 samples and
# the first 5 wavelengths
r <- reflectance(achillea_spectra)
r[1:2,1:5]
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,] 0.03734791 0.03698631 0.03804012 0.03948724 0.03952211
## [2,] 0.04608409 0.04536371 0.04436544 0.04355212 0.04415447
```

```
# Metadata, use simplify = TRUE to get a vector instead of a data.frame
meta(achillea_spectra[1:3], "ssp", simplify = TRUE)
```

```
## [1] Achillea millefolium  Achillea millefolium  Achillea millefolium
## Levels: Achillea millefolium
```

3.2 Adding and manipulating metadata

You can easily add metadata to a `spectra` object. Here we add the data quality flag and species identifier as metadata. This information is stored in the file name, which we used as sample names for our `spectra` object before.

```
# Extract and display the names of your spectra object
(nam <- names(acer_spectra))
```

```
##  [1] "ACEPL_1_BAD.sig" "ACEPL_2.sig"     "ACEPL_3.sig"
##  [4] "ACEPL_4.sig"     "ACEPL_5.sig"     "ACEPL_6.sig"
##  [7] "ACEPL_7_WR.sig"  "ACESA_1_BAD.sig" "ACESA_2.sig"
## [10] "ACESA_3.sig"     "ACESA_4_BAD.sig" "ACESA_5.sig"
## [13] "ACESA_6.sig"
```

```
# Retieve measurement quality and species acronyms
bad <- grepl("BAD", nam)
ssp <- substr(nam, 1,5)

# And add them as metadata
meta(acer_spectra, label="is_BAD") <- bad
meta(acer_spectra, label="species") <- ssp

# Check the spectra object and look at the metadata of four samples
acer_spectra
```

```
## spectra object
## number of samples: 13
## wavelengths: 340.5 to 2522.8 (1024 bands, **overlap not matched**)
## metadata (2): is_BAD, species
```

```
meta(acer_spectra[c(1,4,8,10)])
```

```
##   is_BAD species
## 1   TRUE   ACEPL
## 2  FALSE   ACEPL
## 3   TRUE   ACESA
## 4  FALSE   ACESA
```

You can also edit or add new metadata to the `spectra` object.

```
# Here we add some random values for nitrogen content
n_content <- rnorm(n = nrow(acer_spectra_good), mean = 2, sd = 0.5)
meta(acer_spectra_good, label = "N_percent") <- n_content

# Let's look at the metadata
meta(acer_spectra_good, simplify = T)
```

```
## [1] 2.735241 2.283416 2.716700 1.828574 0.617091 2.083763 2.382402 1.030589
## [9] 1.642388
```

```
# If the metadata needs to be replaced
new_n_content <- rnorm (n = nrow(acer_spectra_good), mean = 2, sd = 0.5)
meta(acer_spectra_good, label = "N_percent") <- new_n_content
meta(acer_spectra_good, simplify = T)
```

```
## [1] 1.134284 2.410014 1.464025 1.972744 1.923803 2.120069 1.977918 1.555299
## [9] 1.181137
```

3.3 Changing samples names and wavelength labels

You may want to edit certain attributes of `spectra`. This is easily attainable in `spectrolab`.

```
spec_new = acer_spectra_good
meta(spec_new) <- NULL

# Replace names with lowercase letters
names(spec_new) = tolower(names(spec_new))

# Check the result
names(spec_new)[1:5]
```

```
## [1] "acepl_2.sig" "acepl_3.sig" "acepl_4.sig" "acepl_5.sig" "acepl_6.sig"
```

You can change the wavelength labels, too. This can be useful when your data has irregular spectral sampling intervals and you want to refer to certain band numbers. It is good practice to keep the orginal wavelengths saved in the metadata, though.

```
# Add original wavelengths as metadata to each element. If the `spectra` object does not
# contain any metadata, we fist need to create a dummy metadata column.
meta(spec_new, 1) <- rep(1,nrow(spec_new))

# Then we add list of wavelengths to each sample. The original wavelengths are stored in
# the first metadata column
meta(spec_new)[[1]] <- list(wavelengths(spec_new))

# Note that you will need to use list indexing to check the original wavelength.
# Here we check the first five original wavelenghts of the first and second sample,
# which are equal
meta(spec_new)[[1]][[1]][1:5]
```

```
## [1] 340.5 342.0 343.4 344.9 346.3
```

```
meta(spec_new)[[1]][[2]][1:5]
```

```
## [1] 340.5 342.0 343.4 344.9 346.3
```

```r
# Now we change the wavelength labels to band number and inspect the result
wavelengths(spec_new) <- 1:1024
tail(wavelengths(spec_new))
```

```
## [1] 1019 1020 1021 1022 1023 1024
```

4 Plotting spectra

The workhorse function for plotting `spectra` is `plot()`. You can also plot the quantiles of a `spectra` object with `plot_quantile()` and shade different spectral regions with `plot_regions()`. `spectrolab` also allows you to interactivelly plot spectra through a `shiny` app with the `plot_interactive()` function. Interactive plots are particularly useful for visually inspecting large datasets.

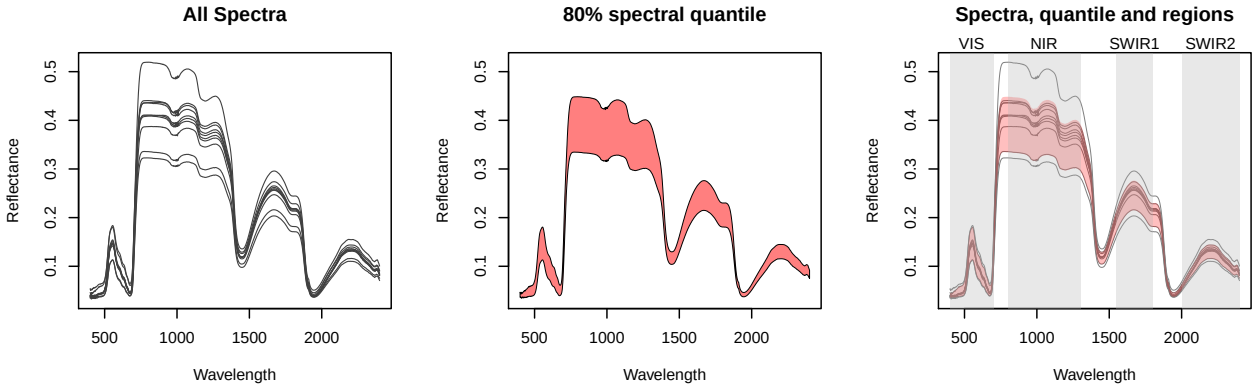
4.1 Plotting spectra, quantiles and spectral regions

Use `plot()` to jointly plot all spectra in a `spectra` object. You can also index spectra to only plot certain spectra and pass usual plot arguments, such as `col`, `ylab`, `lwd`, etc.

For plotting quantiles of a `spectra` object use `plot_quantile()`. Its second argument `total_prob`, is the total "mass" that the quantile encompasses. For instance, a `total_prob = 0.95` covers 95% of the variation in the `spectra` object, i.e. the `0.025 to 0.975` quantile. The quantile plot can stand alone or it can be added to a current plot if `add = TRUE`.

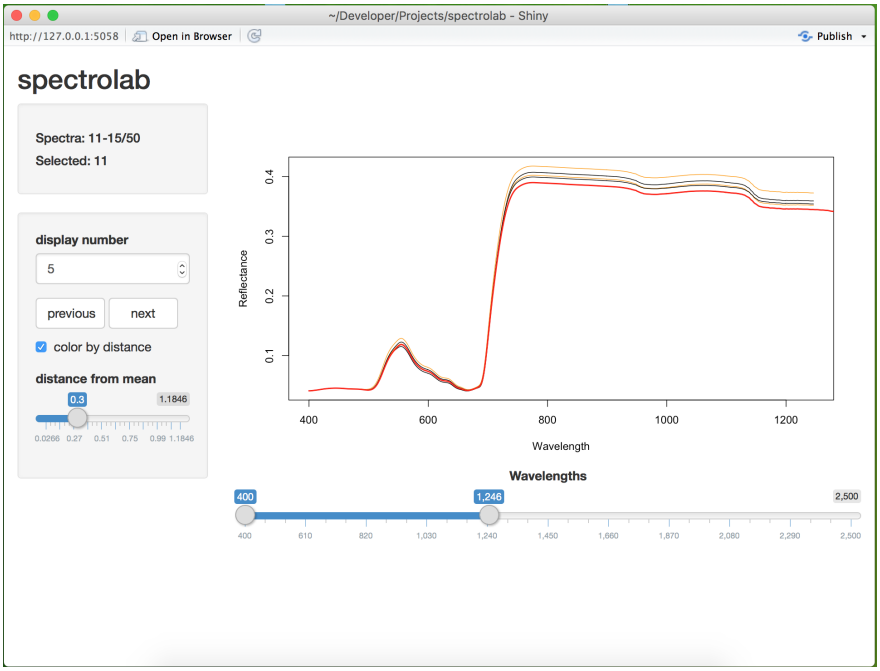
The function `plot_regions()` shades different spectral regions. `spectrolab` provides a `default_spec_regions()` matrix as an example, but you can customize it to your needs (see the help page for `plot_regions` for details).

```r
# Simple spectra plot
par(mfrow = c(1, 3))
plot(achillea_spectra, lwd = 0.75, lty = 1, col = "grey25", main = "All Spectra")

# Stand alone quantile plot
plot_quantile(achillea_spectra, total_prob = 0.8, col = rgb(1, 0, 0, 0.5), lwd = 0.5,
              border = TRUE)
title("80% spectral quantile")

# Combined individual spectra, quantiles and shade spectral regions
plot(achillea_spectra, lwd = 0.25, lty = 1, col = "grey50",
     main="Spectra, quantile and regions")
plot_quantile(achillea_spectra, total_prob = 0.8, col = rgb(1, 0, 0, 0.25),
              border = FALSE, add = TRUE)
plot_regions(achillea_spectra, regions = default_spec_regions(), add = TRUE)
```



4.2 Interactive plots

Interactive plots are particularly useful for visually inspecting large datasets. You can decide on the number of measurements to display, which makes it easy to search for bad measurements. You can also color spectra based on their distance from the mean to define outliers and limit the wavelength regions to display.



5 Subsetting spectra

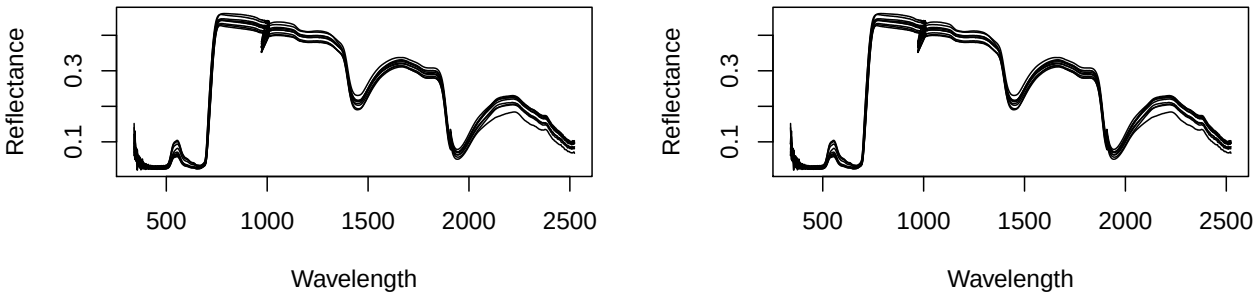
You can subset `spectra` using a notation *similar* to the `[i, j]` function used in matrices and data.frames. The first argument in `[i, ]` matches *sample names*, whereas the second argument `[ , j]` matches *wavelength names*. Here are some examples of how `[` works in `spectra`:

- `x[1:3, ]` will keep the first three samples of `x`, i.e. `1:3` are indices.
- `x["sp_1", ]` keeps **all** entries in `x` where sample names match `"sp_1"`
- `x[ ,800:900]` will keep wavelengths between `800` and `900`, given that reflectance values at exactly 800 nm and 900 nm are present in the dataset

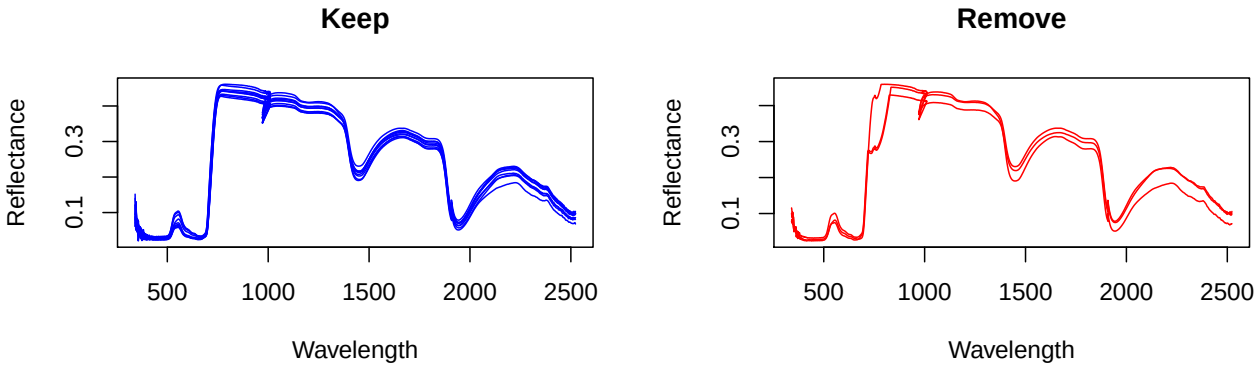
- `x[ ,wavelengths(x, 800, 900)]` same as above, but now wavelengths > 800 nm and < 900 nm are kept, irrespective if values at exactly 800 nm and 900 nm are present
- `x[ ,1:5]` will **fail**! *wavelengths **cannot** be subset by indices!*

To subset spectra to specific entries you can use e.g. sample names, sample indices or metadata attributes.

```
# Subset spectra to all entries where sample_name matches "ACHMI_7" or get
# the first three samples
spec_sub_byname <- achillea_spectra["ACHMI_7", ]
spec_sub_byidx  <- achillea_spectra[1:3, ]

# Subset spectra to white reference measurements
wr <- grepl("WR",names(acer_spectra))
acer_no_wr <- acer_spectra[!(wr),]

# Exlcude bad measurements based on sample name
out <- grepl("BAD",names(acer_no_wr))
spec_sub_good <- acer_no_wr[!(out),]

# Exlcude bad measurements based on metadata attributes
out2 <- meta(acer_no_wr,"is_BAD", simplify = T)
spec_sub_good2 <- acer_no_wr[!(out2),]

par(mfrow=c(1,2))
plot(spec_sub_good)
plot(spec_sub_good2)
```

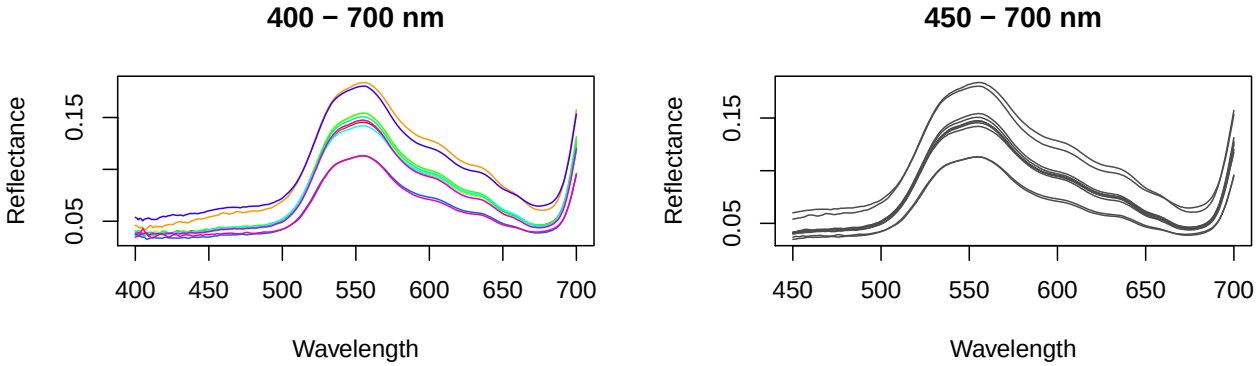



```
# You can also exclude spectra based on the shape of the spectrum when bad spectra
# are not flagged

keep <- names(acer_no_wr[(acer_no_wr[,wavelengths(acer_no_wr,800,801)]-
                            acer_no_wr[,wavelengths(acer_no_wr,770,771)])<0.01,])
par(mfrow=c(1,2))
plot(acer_no_wr[names(acer_no_wr) %in% keep, ], col="blue", main="Keep")
plot(acer_no_wr[!(names(acer_no_wr) %in% keep), ], col="red", main="Remove")
```



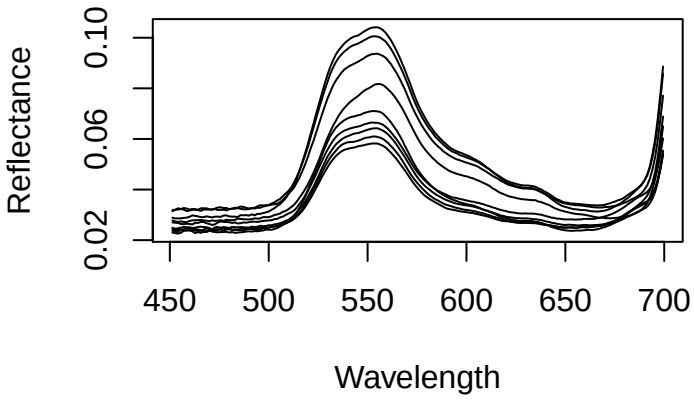
Subsetting by wavelength lets you, for instance, exclude noisy regions at the beginning and end of the spectrum.

```
# Subset wavelength region to VIS and remove first 50 nm
spec_sub_vis <- achillea_spectra[, 400:700]
par(mfrow=c(1,2))
plot(spec_sub_vis, col=palette(rainbow(10)), main="400 - 700 nm")
plot(spec_sub_vis[,450:700], col="grey30", main="450 - 700 nm")
```



When the wavelengths of your data are not sampled at or subsampled to a fixed interval, the spectral range can be subset using the `min` and `max` arguments for wavelengths:

```
acer_spectra_trim <- acer_spectra_good[ ,wavelengths(acer_spectra_good, 450, 700) ]
plot(acer_spectra_trim)
```



Note that you can 1) subset samples to specific entries using indices and 2) subset wavelengths using characters or numerics. As said before, you **cannot** use indices to subset wavelengths!

```
# Subsetting samples by indices works and so does subsetting wavelengths by numerics
# or characters.
spec_sub_byidx[1, "405"] == spec_sub_byidx[1, 405]
```

```
## ACHMI_1
##    TRUE
```

```
# Subsetting wavelengths by an index, like using 2 instead of 401 (i.e, referring to
# the 2nd band) will fail
spec_sub_byidx[ , 2]

`Error in i_match_ij_spectra(this = this, i = i, j = j) : Wavelength
subscript out of bounds. Use wavelength labels instead of raw indices.`
```

6 Processing spectra

`spectrolab` supports a suite of processing tasks, such as matching the overlap regions between sensors, interpolating wavelengths and applying functions to spectra.

6.1 Matching overlap regions and resampling

Raw spectra are characterised by overlapping wavelengths and/or jumps between different sensors, i.e. between the VIS and NIR, and the NIR and SWIR regions, respectively. If you do not see sensor overlaps and/or jumps, they were already internally corrected by the instrument's software. For some instruments it is possible to change the corretion settings and to preserve the raw data, for other instruments it is not. When you are working with raw spectra, `spectrolab`'s `match_overlap` function makes it easy to correct `spectra` at the overlap regions. Essentially, `match_overlap` splices `spectra` at the overlap regions, removes duplicated wavelengths and corrects the jumps by multiplying reflecance values from adjacent sensors by a factor. The second sensor (covering the NIR region) is held constant by default, but you can overwrite this behavior and define your own matching function (see Vignette **Match sensors**). A common processing task after overlap matching is resampling wavelengths. Often a common interval is used, but you can also manually define new wavelengths.

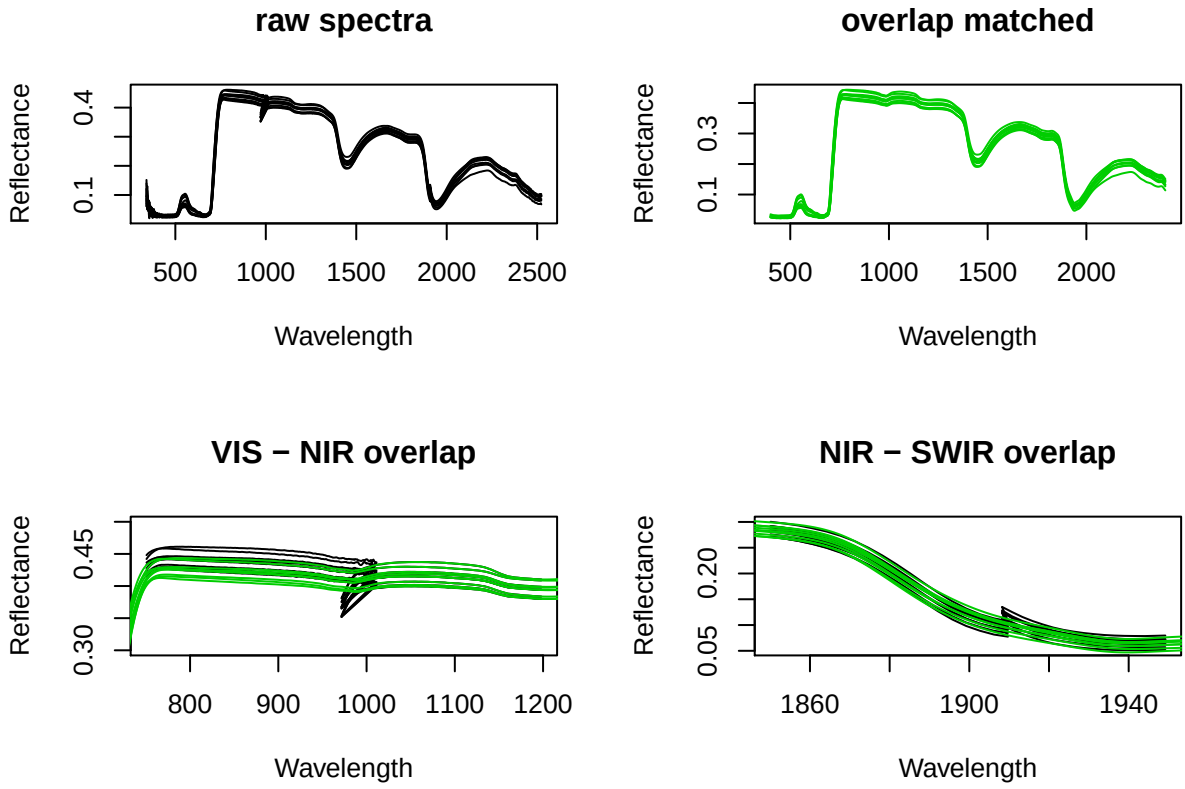
```
# Match overlap at 990 nm and 1900 nm, keeping the second (NIR) sensor fixed (default)
acer_matched <- match_sensors(acer_spectra_good, splice_at = c(990,1900))

# Resample spectra to a common interval of 1 nm and limiting the wavelength
# range to 400 nm - 2400 nm
acer_res <- resample(acer_matched,new_wvls = 400:2400)
```

```
## Using spline to predict reflectance at new wavelengths...
```

```
## Beware the spectra are now partially smoothed.
```

```
# Plot result
par(mfrow=c(2,2), oma=c(0,0,0,0))
plot(acer_spectra_good, main="raw spectra")
plot(acer_res,col="green3", main= "overlap matched")

plot(acer_spectra_good[,wavelengths(acer_spectra_good,750,1200)],
     main="VIS - NIR overlap", ylim=c(0.3,0.5))
plot(acer_res, add=T, col="green3")
```

```
plot(acer_spectra_good[,wavelengths(acer_spectra_good,1850,1950)],
     main="NIR - SWIR overlap")
plot(acer_res, add=T, col="green3")
```



6.2 Combine spectra and apply functions

You can combine two or more `spectra` objects using `combine_spectra`. The wavelength intervals need to be the same in all datasets, but different metadata are acceptable.

```
# Combine two spectra objects and check metadata
spec_combi <- combine(acer_res, achillea_spectra)
meta(spec_combi)
```

```
##    N_percent ident                    ssp
## 1   1.134284  <NA>                   <NA>
## 2   2.410014  <NA>                   <NA>
## 3   1.464025  <NA>                   <NA>
## 4   1.972744  <NA>                   <NA>
## 5   1.923803  <NA>                   <NA>
## 6   2.120069  <NA>                   <NA>
## 7   1.977918  <NA>                   <NA>
## 8   1.555299  <NA>                   <NA>
## 9   1.181137  <NA>                   <NA>
## 10        NA 10526 Achillea millefolium
## 11        NA 10527 Achillea millefolium
## 12        NA 10528 Achillea millefolium
## 13        NA 10529 Achillea millefolium
## 14        NA 10530 Achillea millefolium
```

```
## 15        NA 10531 Achillea millefolium
## 16        NA 10532 Achillea millefolium
## 17        NA 10533 Achillea millefolium
## 18        NA 10534 Achillea millefolium
## 19        NA 10535 Achillea millefolium
```

```
# Use species abbreviation as metadata
meta(spec_combi,"ssp") <- substr(names(spec_combi),1,5)

# Calculate mean spectra per species. try_keep_txt() preserves unique text in the
# metadata, while numeric values are by default aggregated using the same function as
# applied to the reflectance values.

spec_means <- aggregate(spec_combi,by = meta(spec_combi,"ssp"), FUN = mean,
                        try_keep_txt(mean))
meta(spec_means,c(1,3,2))
```

```
##   N_percent   ssp
## 1  1.780974 ACEPL
## 2  1.708606 ACESA
## 3        NA ACHMI
##                                                                    ident
## 1                                                                     NA
## 2                                                                     NA
## 3 10526, 10527, 10528, 10529, 10530, 10531, 10532, 10533, 10534, 10535
```

7 Converting a spectra object into a matrix or data.frame

It is also possible to convert a `spectra` object to a matrix or data.frame using the `as.matrix()` or `as.data.frame()` functions. This is useful if you want to export your data in a particular format, such as csv.

If you're converting spectra to a matrix, `spectrolab` will (1) place wavelengths in columns, assigning wavelength labels to `colnames`, and (2) samples in rows, assigning sample names to `rownames`. Since `R` imposes strict rules on column name formats and sometimes on row names, `as.matrix()` will try to fix potential dimname issues if `fix_names != "none"`. Note that `as.matrix()` will not keep metadata.

Conversion to data.frame is similar, but keeps the metadata by default (unless you set the `metadata` agument to `FALSE`).

```
# Make a matrix from a `spectra` object
spec_as_mat = as.matrix(achillea_spectra, fix_names = "none")
spec_as_mat[1:4, 1:3]
```

```
##                400        401        402
## ACHMI_1 0.03734791 0.03698631 0.03804012
## ACHMI_2 0.04608409 0.04536371 0.04436544
## ACHMI_3 0.04058113 0.04025678 0.03958125
## ACHMI_4 0.04063730 0.03993420 0.03793455
```

```
# Make a data.frame from a `spectra` object
spec_as_df = as.data.frame(achillea_spectra, fix_names = "none", metadata = TRUE)
spec_as_df[1:4, 1:5]
```

```
##   sample_name ident                   ssp        400        401
## 1     ACHMI_1 10526 Achillea millefolium  0.03734791 0.03698631
```

```
## 2     ACHMI_2 10527 Achillea millefolium  0.04608409 0.04536371
## 3     ACHMI_3 10528 Achillea millefolium  0.04058113 0.04025678
## 4     ACHMI_4 10529 Achillea millefolium  0.04063730 0.03993420
```