Processing spectral data for biodiversity detection

Jesús N. Pinto-Ledezma and Jeannine Cavender-Bares

In this lab, we will explore some aspects of **Spectral data**. To do so, we will use the R package **spectrolab** and the data stored in it. **spectrolab** is a R package that process and visualize data from portable spectrometers and establishes a common interface to spectra information. As you will see, working with spectral information/data is similar to work with any other information (e.g., species, communities) and consequently it can be used to calculate any metric of diversity, a biodiversity dimension called **Spectral diversity**.

Basically, this practice is based on the vignette of the spectrolab package (https://cran.r-project.org/web/packages/spectrolab/vignettes/introduction_to_spectrolab.pdf), although Jesús made some modifications and additions. Thanks to Dudu (José Eduardo Meireles) to make spectra data easy to handle.

Set up your data and your working directory

Set up a working directory and put the two data files in that directory. Tell R that this is the directory you will be using, and read in your data:

```
setwd("path/for/your/directory")
```

Install and load the following packages.

```
install.packages("spectrolab", dependencies = TRUE)
library(spectrolab)
```

Prepare spectral data

There are two ways to get spectra into R: 1) converting a matrix or data.frame to spectra; 2) reading spectra from raw data files (formats: SVC's sig, Spectral Evolution's sed and ASD's asd). Here are a couple examples:

Here is an example using a dataset matrix named spec_matrix_meta.csv provided by the spectrolab package. ### Create spectra from a matrix or data.frame

```
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
# 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)
```

Inspect the data loaded.

```
head(spec_csv)
```

Once you have your spectra in a matrix or data frame in R, 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 wavelengths columns must be (numeric) wavelength labels. You also should declare which column has the sample names (which are mandatory) using the name_idx argument. If other columns are present (other than sample name and reflectances), their indices must be passed to as.spectra as the meta_idxs argument.

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

```r
# And now you have a spectra object with sample names and metadata...
achillea_spec
```

Reading spectra: example with SVC's .sig files

To read spectra files, you should use the function **read_spectra()**.

```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")
```

Inspecting and querying spectra

You can check out your spectra object in several ways. For instance, You may want to know how many spectra and how many bands are in there, retrieve the file names, etc. Of course you will need to plot the data, but that topic gets its own section further down.

```r
# Simply print the object
acer_spectra
```

```r
# Get the dataset's dimensions
dim(acer_spectra)
```

You can access to 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).

```r
# Vector of all sample names. Note: Duplicated sample names are permitted
n <- names(achillea_spec)
# Vector of wavelengths
w <- wavelengths(achillea_spec)
# Reflectance matrix
r <- reflectance(achillea_spec)
# Metadata. Use simplify = TRUE to get a vector instead of a data.frame
m <- meta(achillea_spec, "ssp", simplify = TRUE)
```

Subsetting spectra

You can subset the spectra information using a notation similar to the [i, j] function used in matrices and data.frames. The first argument in [i,] matches sample names (rows), whereas the second argument [, j] matches the wavelength names (columns). 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 indexes.
- x["sp_1",] keeps all entries in x where sample names match "sp_1".
- x[, 800:900] will keep wavelengths between 800 and 900.
- x[, 1:5] will fail!. wavelengths cannot be subset by indexes!.

Subsetting lets you, for instance, exclude noisy regions at the beginning and end of the spectrum or limit the data to specific entries.

```
# Subset wavelength regions.
spec_sub_vis <- achillea_spec[ , 400:700 ]
# Subset spectra to all entries where sample_name matches "ACHMI_7" or get the first three samples
spec_sub_byname <- achillea_spec["ACHMI_7", ]
spec_sub_byidx <- achillea_spec[ 1:3, ]
```

The resolution of some spectra may be different from 1nm, as is the case with SVC. In those cases, the best way to subset spectra is using the min and max arguments for wavelengths:

```
acer_spectra_trim <-  acer_spectra[ , wavelengths(acer_spectra, 400, 2400) ]
```

Note that you can (1) subset samples using indexes and (2) use characters or numerics to subset wavelengths. As said before, you cannot use indexes to subset wavelengths though.

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

As you see until now, working with spectra is similar to work with any data.frame or matrix. Now, lets see how spectra is...

Plotting

The main function for plotting spectra is **plot()**. It will jointly plot each spectrum in the spectra object. You should be able to pass the usual plot arguments to it, such as col, ylab, lwd, etc. Lets see...

```
# Simple spectra plot
plot(achillea_spec, lwd = 0.75, lty = 1, col = "grey25", main = "All Spectra")
```

You can also plot the quantiles of a spectra object with **plot_quantile()**. It's 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. it is the 0.025 to 0.975 quantile. The quantile plot can stand alone or be added to a current plot if add = TRUE.

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

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

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

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.

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

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

Ok, enough of exploring and plotting spectra. Now we will use spectra to calculate some descriptive statistics and to calculate some diversity metrics.

Using spectra

```
VIS <- c(400:700)
NIR <- c(800:1300)
SWIR1 <- c(1550:1800)
SWIR2 <- c(2000:2400)
```

```
achillea_spec_VIS <- achillea_spec[, VIS]
achillea_spec_VIS
```

```
achillea_spec_VIS_df <- as.data.frame(achillea_spec_VIS, fix_names = "none", metadata = TRUE)
achillea_spec_VIS_df[1:10, 1:5]
```

Drop the first 3 columns for further analyses

```
spec_as_df_clean <- spec_as_df[, 4:2004]
spec_as_df_clean[1:10, 1:5]
```

```
achillea_spec_VIS_df_clean <- achillea_spec_VIS_df[, 4:304]
achillea_spec_VIS_df_clean[1:10, 1:5]
```

```
spec_as_df_clean_ind1 <- spec_as_df_clean[1, ]
spec_as_df_clean_ind1
```

```
hist(as.numeric(spec_as_df_clean_ind1), col = "black")
abline(v = mean(as.numeric(spec_as_df_clean_ind1)), col = "red", lwd = 3)
```

```
source("https://raw.githubusercontent.com/jesusNPL/BiodiversityScience/master/RFunctions/mixR.R")
```

```
Descriptives(as.numeric(spec_as_df_clean_ind1))
```

```
achillea_spec_VIS_df_clean_ind1 <- achillea_spec_VIS_df_clean[1, ]
achillea_spec_VIS_df_clean_ind1
```

```
hist(as.numeric(achillea_spec_VIS_df_clean_ind1), col = "black")
abline(v = mean(as.numeric(achillea_spec_VIS_df_clean_ind1)), col = "red", lwd = 3)
```

```
Descriptives(as.numeric(achillea_spec_VIS_df_clean_ind1))
```

What about all individuals...

```
descript_all_individuals <- apply(spec_as_df_clean, MARGIN = 1, FUN = Descriptives)
descript_all_individuals[[10]]
```

```
diversities(spec_as_df_clean)
```

```
diversities(achillea_spec_VIS_df_clean)
```

Exercises

Please respond each question based on the practice.

References

Cavender-Bares, J., Meireles, J., Couture, J., Kaproth, M., Kingdon, C., Singh, A., ... Townsend, P. (2016). Associations of Leaf Spectra with Genetic and Phylogenetic Variation in Oaks: Prospects for Remote Detection of Biodiversity. Remote Sensing, 8(3), 221. doi:10.3390/rs8030221