

Advanced spectrolab for package developers and contributors

Jose Eduardo Meireles and Anna K. Schweiger

Style guide and conventions

General

- object names are lower case
- assignment is done with equals = instead of arrow <-
- only really obvious abbreviations may be used
- names should be separated by underscore _, unless you're overloading an R generic e.g. `as.matrix`.
- internal functions should use the `i_` prefix. e.g `i_find_spectra()`
- all functions must be documented with roxygen2 comments
- strive for small functions.
- try to fail gracefully

Specific

- **Do not reach inside the spectra object's guts.** If you're accessing the internal data structures directly, you're probably doing something wrong.
- If the only way you can implement something reasonably is gutting the `spectra` object, we did something wrong. Please report an issue and submit a pull request.
- **Do not** use a pattern of deconstructing and reconstructing the `spectra` object, even if doing so through the getters and setters.

```
library("spectrolab")

## spectrolab version: 0.0.6
##
## Please cite:
## Meireles J, Schweiger A, Cavender-Bares J (2018). spectrolab: Class
## and Methods for Hyperspectral Data. R package version 0.0.6, <URL:
## https://github.com/meireles/spectrolab>.
##
## Attaching package: 'spectrolab'
##
## The following objects are masked from 'package:stats':
##
## sd, smooth, var
```

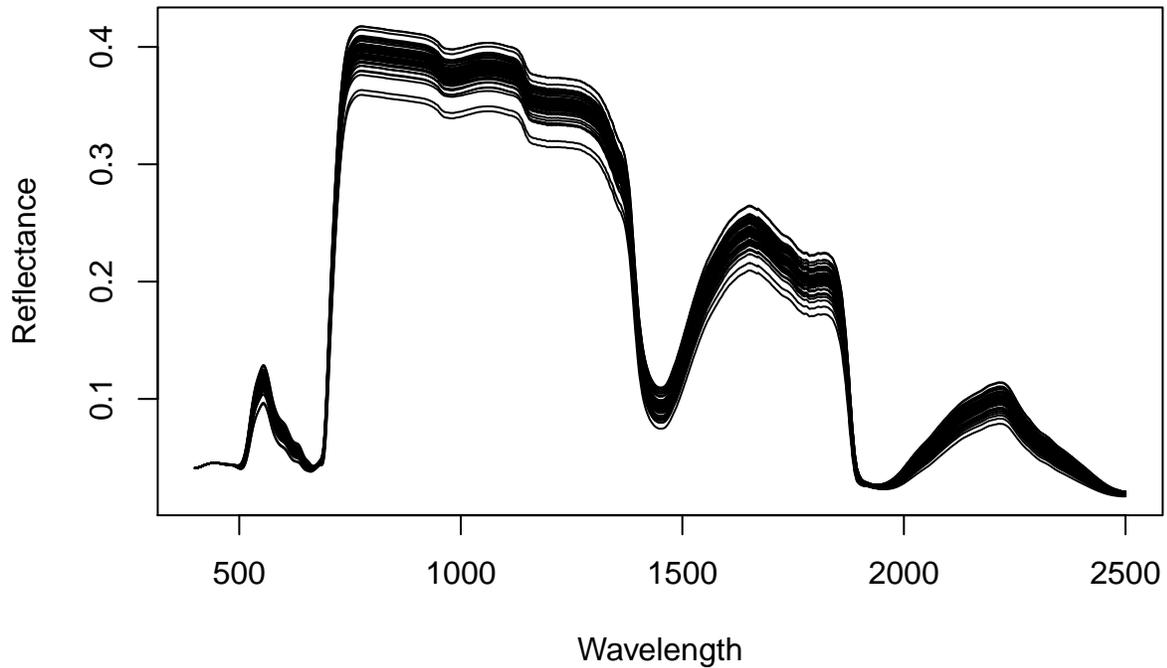
The spectra class

`spectrolab` defines a new S3 class called `spectra` that holds all of the different components of a spectral data. Without diving too much into its implementation, a `spectra` object holds the important information needed for most spectral datasets: reflectance, wavelengths, etc. The class has a bunch of requirements in terms of both format and values.

Constructing a spectra object “by hand”

In addition to `read_spectra()` and `as.spectra()`, you can create a `spectra` object “by hand” using the more flexible `spectra()` constructor, which takes at least arguments: (1) a reflectance matrix, (2) a vector of wavelengths and (3) the sample names.

```
# (1) Create a reflectance matrix.  
# In this case, by removing the first column that holds the species name  
rf = spec_matrix_example[ , -1]  
  
# (2) Create a vector with wavelength labels that match  
the reflectance matrix columns.  
wl = colnames(rf)  
  
# (3) Create a vector with sample labels that match  
the reflectance matrix rows.  
In this case, use the first column of spec_matrix_example  
sn = spec_matrix_example[ , 1]  
  
# Finally, construct the spectra object using the `spectra` constructor  
spec = spectra(reflectance = rf, wavelengths = wl, names = sn)  
  
# And hopefully this worked fine  
is_spectra(spec)  
  
## [1] TRUE  
  
plot(spec)
```



Getting and Setting

spectrolab gives you access to get and set functions for most spectra components. The `names()`, `wavelengths()` functions do both getting and setting. For example:

```
# Getters
names(spec) [1:4]
wavelengths(spec) [1:4]

# Setters
names(spec)      = toupper(names(spec))
wavelengths(spec) = wavelengths(spec) / 1000
```

Reflectances are set using the `[]` notation. For instance:

```
spec[1, 400:1200] = spec[1, 400:1200] * 2
plot(spec)
```

