Introduction to casper

Jose Eduardo Meireles

2016-11-11

The goal of `casper` is first and foremost to provide a `spectra` class for `R` that exposes a standard interface and patafrom that allows other `R` packages to build on. The package will provide very basic IO, plotting and conversion functionality, but that is about it. `casper` is implemented with ease of use in mind, but shouldn't slow you down.

Installing and loading `casper`

The best way to get `casper` is to install it directly from the *github repository*. You will need the `devtools` package to do it though.

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

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

```
library("casper")
```

Reading spectra and creating a `spectra` object

First, explore the example dataset `spec_matrix_example`

As already stated, `casper` comes with limited IO capabilities. To illustrate how to create a `spectra` object, we will use an example dataset called `spec_matrix_example`. Samples are in rows, and wavelengths in columns, and the first column is the sample name (in this case, a species name). The column names match wavelength labels. I tried to format `spec_matrix_example` to typical result of a `read.csv` command.

```
# Example spectral dataset in matrix format.
spec_matrix_example[1:4, 1:3]
```

```
##      species     400                 401
## [1,] "species_7" "0.0409992869075924" "0.0410386116912567"
## [2,] "species_9" "0.0410052470437898" "0.041044764176962"
## [3,] "species_8" "0.0410016756838812" "0.041040951774663"
## [4,] "species_7" "0.0409985571758403" "0.0410377949963602"
```

```
# Note that this is NOT a spectra object.
# You can verify this by asking what class `spec_example` is.
class(spec_matrix_example)
```

```
## [1] "matrix"
```

```
# An alternative is to use casper's `is_spectra()` function.
is_spectra(spec_matrix_example)
```

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

Constructing a spectra object

The `spectra` class holds the essential information used in spectral dataset: reflectance, wavelengths, etc. The class has a bunch of requirements in terms of both format and values, for instance, relative reflectance must be between 0 and 1.

If your data is in a matrix with the same format as `spec_matrix_example` (check above for details), you can construct a `spectra` object by calling the `as.spectra()` function.

```
# Make a spectra object if you have a matrix in the right format
spec = casper::as.spectra(spec_matrix_example)

# Did it work?
is_spectra(spec)
```

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

Alternatively, you can create a `spectra` object using the more flexible `spectra()` constructor, which takes three arguments: (1) a reflectance matrix, (2) the wavelength numbers and (3) the sample names.

```
# (1) Create a reflectance matrix.
#     In this case, by removing the species column
rf = spec_matrix_example[ , -1 ]

# Check the result
rf[1:4, 1:3]
```

```
##      400                 401                 402
## [1,] "0.0409992869075924" "0.0410386116912567" "0.0410649647002033"
## [2,] "0.0410052470437898" "0.041044764176962"  "0.0410712937865739"
## [3,] "0.0410016756838812" "0.041040951774663"  "0.0410672386447536"
## [4,] "0.0409985571758403" "0.0410377949963602" "0.0410640566176718"
```

```
# (2) Create a vector with wavelength labels that match
#     the reflectance matrix columns.
wl = colnames(rf)

# Check the result
wl[1:6]
```

```
## [1] "400" "401" "402" "403" "404" "405"
```

```
# (3) Create a vector with sample labels that match
#     the reflectance matrix rows.
#     In this case, use the first colum of spec_example
sn = spec_matrix_example[ , 1]

# Check the result
sn[1:6]
```

```
## [1] "species_7"  "species_9"  "species_8"  "species_7"  "species_8"
## [6] "species_10"
```

```
# 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
```

Converting a spectra object into a matrix

It is possible to convert a `spectra` object to a matrix format, using the `as.matrix()` function. `casper` will (1) place wavelength in columns, assigning wavelength labels to `colnames`, and (2) samples in rows, assigning sample names to `rownames`. Since `R` imposes strict on column name formats and sometimes on row names too, `as.matrix()` will try to fix potential dimname issues if `fix_names != "none"`.

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

```
##                  400        401        402
## species_7 0.04099929 0.04103861 0.04106496
## species_9 0.04100525 0.04104476 0.04107129
## species_8 0.04100168 0.04104095 0.04106724
## species_7 0.04099856 0.04103779 0.04106406
```

Exploring `spectra` object

`casper` exposes a few ways to plot and query spectral data in `spectra` format.

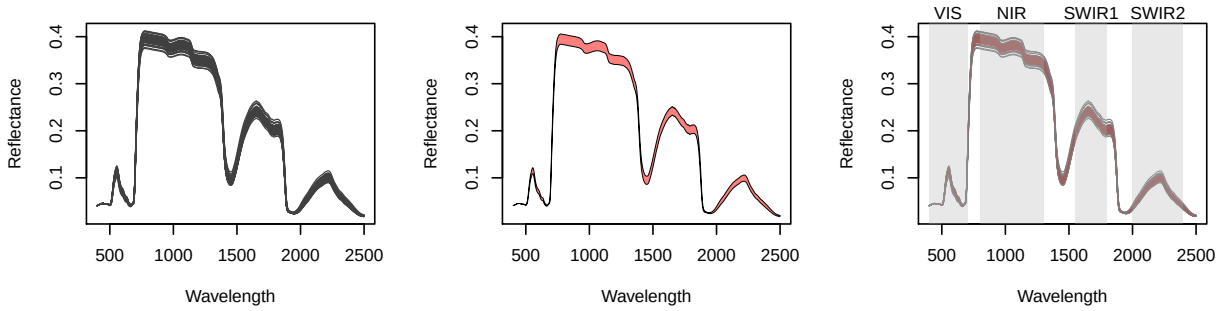
Plotting

The workhorse 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`, etc.

You can also plot the quantile 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`.

Last but not least, you can also shade spectral regions with the `plot_spec_regions()` function. `casper` provides a `default_spec_regions` matrix as an example, but you obviously can cistomize it for your needs.

```
par(mfrow = c(1, 3))

# Simple spectra plot
plot(spec, lwd = 0.75, lty = 1, col = "grey25")

# Stand along quantile plot
plot_quantile(spec, total_prob = 0.8,
              col = rgb(1, 0, 0, 0.5), lwd = 0.5, border = TRUE)

# Combined quantile and individual spctra plot
# With an added bonus of shading 4 spectral regions
plot(spec, lwd = 0.25, lty = 1, col = "grey50")
plot_quantile(spec, total_prob = 0.8,
              col = rgb(1, 0, 0, 0.25), add = TRUE, border = FALSE)
plot_spec_regions(spec, regions = default_spec_regions, add = TRUE)
```



Querying

`casper` lets you query the `spectra` object and get summary infomation. You can easly get sample names with `names()` and wavelength labels with `wavelengths()`. It is also possible to recover the

```
# Get the vector of all sample names
# Note that duplicate sample names are permitted
n = names(spec)
n[1:5]
```

```
## [1] "species_7" "species_9" "species_8" "species_7" "species_8"
```

```
# Or get the vector of wavelengths
w = wavelengths(spec)
w[1:5]
```

```
## [1] 400 401 402 403 404
```

```
# You can also get the dimensions of your `spectra` object
dim(spec)
```

```
##     n_samples n_wavelengths
##            50          2101
```

If you really need the raw reclectance, you can retrieve it with the `reflectance()` function. This is not recomended though.

Subsetting spectra

You can subset the `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 the *wavelength names*. Here are some important differences between how `[` works in matrices and in `specrta`:

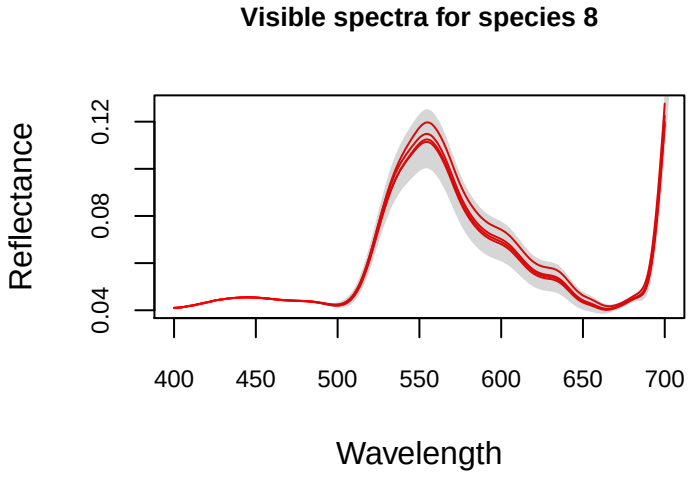
- `x[ 1:3 , ]` will keep the first three samples of `x`.
- `x[ "sp_1" , ]` keeps **all** entries in `x` where sample names match `"sp_1"`
- `x[ , 800:900 ]` will keep wavelengths between 800 and 900.
- `x[ , 1:5 ]` will **fail**!. wavelengths cannot be subset by index!

```
# Subset spectra to all entries where sample_name matches "species_8"
spec_sp8 = spec[ "species_8", ]

# And maybe further subset to the visible wavelengths only
```

```r
spec_sp8 = spec_sp8[ , 400:700 ]

dim(spec_sp8)
```

```
##     n_samples n_wavelengths
##             4           301
```

```r
# This subset should still plot just fine
plot(spec_sp8, col = "red", main = "Visible spectra for species 8",
     cex.main = 0.8, cex.axis = 0.75)

plot_quantile(spec, total_prob = 1.0, add = TRUE,
              col = rgb(0.2, 0.2, 0.2, 0.2), border = FALSE)
```




```r
# Note that you can subset by wavelength using numerics or characters.
reflectance(spec_sp8[ 1 , "405"]) == reflectance(spec_sp8[ 1 , 405])
```

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

```r
# But you CANNOT use indexes to subset wavelengths!
# Something that is obvioulsy an index will fail. For instance, using 2 instead of 401
spec_sp8[ , 2 ]
```

```
## Error in `[.spectra`(spec_sp8, , 2): Wavelength subscript out of bounds. Use wavelength labels instea
```

```r
# However, if you use 2000:2001 you will NOT get the two last bands, but instead
# wavelengths "2000" and "2001".Bottomline, be careful not to use indexes!
```

Manipulating samples and wavelength labels

You may want to edit certain simple attributes of `spectra`, such as make all sample names uppercase. This is easlily attainable in `casper`

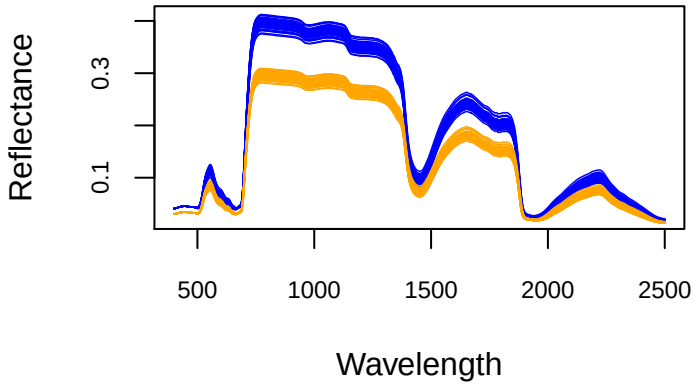
```r
spec_new = spec

# Replace names with an uppercase version
```

```
names(spec_new) = toupper(names(spec_new))

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

```
## [1] "SPECIES_7" "SPECIES_9" "SPECIES_8" "SPECIES_7" "SPECIES_8"
```

You may want to fiddle with the reflectance itself. This is easy to do, but there are some constraints. For example. `casper` will not allow you to have negative reflectance values or values greater than 1.

```
# Scale reflectance by 0.75
spec_new[] = reflectance(spec_new) * 0.75

# Plot the results
plot(spec, col = "blue", lwd = 0.75, cex.axis = 0.75)
plot(spec_new, col = "orange", lwd = 0.75, add = TRUE)
```



However, `casper` will throw an error if you try to perform an illegal operation to reflectance, for instance

```
# Trying to add 1.0 to all reflectance values will fail.
spec_new[] = reflectance(spec_new) + 1.0
```

```
## Error in i_reflectance(value): Reflectance values must be between 0 and 1
```