Spectral Data Analysis Showcase

Jonas Anderegg jonas.anderegg@usys.ethz.ch

16 11 2020

Background

Processing and analysis of hyperspectral data is often performed using commercial software tools such as ENVI or the hyperspectral toolbox in MATLAB. Several R packages are also available for spectral data processing and analysis. Examples include the hyperSpec and hsdar packages. Compared to these, the set of functions presented here is limited in its functionality and therefore not a substitute for them. However, it allows users with minimal experience in spectral data analysis and R programming to implement a few standard steps in spectral data analysis such as signal filtering/pre-processing, visual inspection of the data including effects of treatments and measurement time point, calculation and scaling of spectral vegetation indices and multivariate outlier detection and removal. The functionalities are demonstrated in the following using a spectral dataset collected between 18 May 2020 and July 9 2020 at Agroscope Changins. The aim of the measurements was to evaluate the feasibility of fusarium head blight (FHB) detection and FHB severity quantification using a time series of spectral reflectance measurements on single lines of wheat plants.

Prepare environment and load data

The below code will prepare your R environment (i.e. install and load the required R packages)

```
# clean the environment
rm(list = ls())

 # set location where libraries are stored
.libPaths("T:/R3UserLibs")

# check whether required packages are installed
# install if necessary
list.of.packages <- c("tidyverse", "mvoutlier", "data.table", "prospectr", "ggsci")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages, dependencies = TRUE, repos = 'https://stat.ethz

# load packages to the current environment
library(tidyverse)
library(data.table)
library(mvoutlier)
library(prospectr)
library(ggsci)
```

Next, the paths to the data and the spectra processing functions are defined.

```r
# set paths
path_to_data <- "O:/Projects/KP0011/8/test/data/"
path_to_funcs <- "O:/Projects/KP0011/8/test/funcs/"

# load functions
source(paste0(path_to_funcs, "spectra_proc.R"))
```

All spetral data is then loaded, and saved as an .rds object (to enable faster re-loading from disk). The spectral data must be organised as follows: All measurements belonging to one measurement date must be stored in one folder, named with an eight digit number representing the measurment date (.e.g. "20200709"). All file names within this folder must start with a four-digit number representing the measurement year (e.g. 2020), and end with the extension ".sed" or ".asd". Loading the spectral data may take a while. The directory containing the subfolders (measurement dates) and the format of the spectral data must be passed to the function.

```r
# load all spectral data and save as .rds
# data <- load_spectra(dir = path_to_data, format = "sed")
# saveRDS(data, paste0(path_to_data, "alldat.rds"))

data <- readRDS(paste0(path_to_data, "alldat.rds"))
```

The filename is used as a measurement id, the measurement date taken from the subfolder name is also added. The spectral data is stored in a list column of data tables (column "rflt").

The first measurement can be accessed with the below code

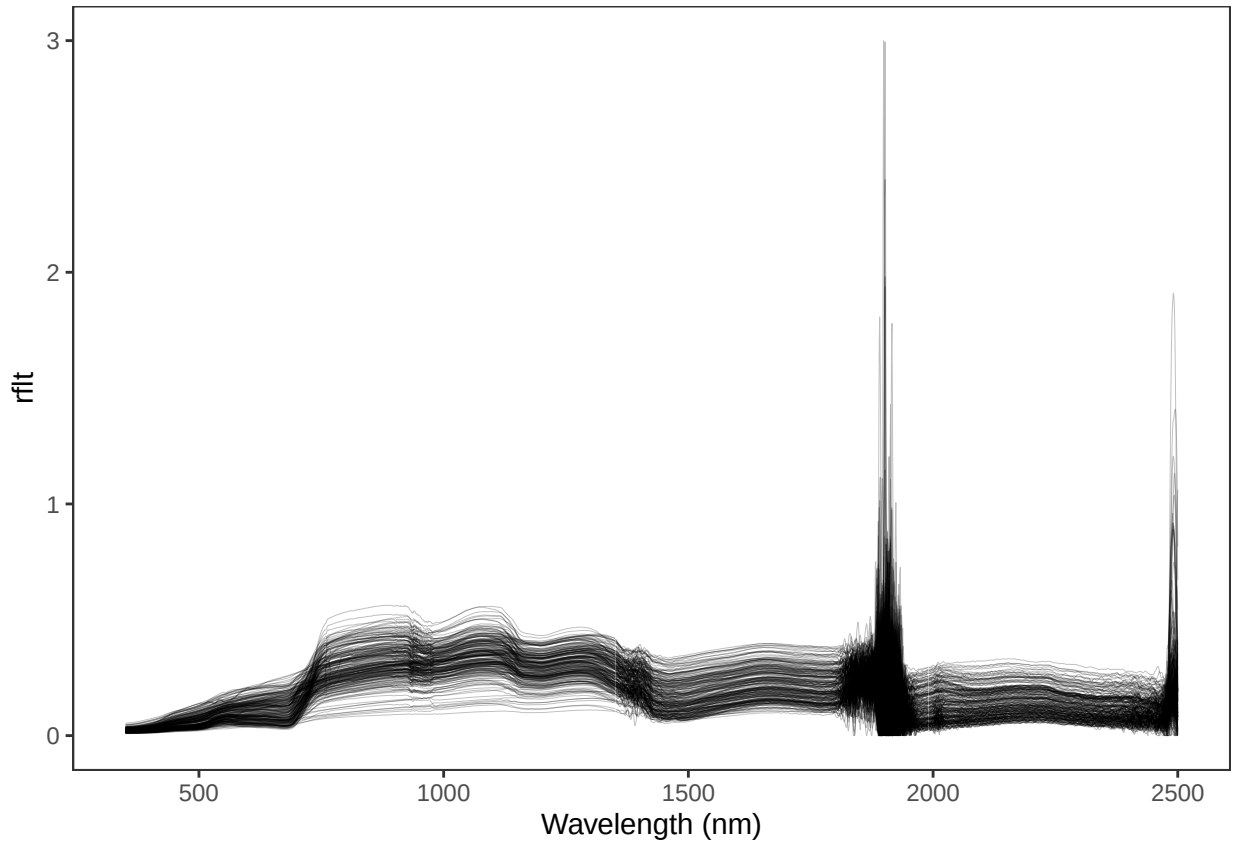
```r
# access the first measurement and display as a tibble
data$rflt[[1]] %>% as_tibble()
```

```
## # A tibble: 1 x 2,151
##    `350`  `351`  `352`  `353`  `354`  `355`  `356`  `357`  `358`  `359`  `360`
##    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 0.0186 0.0185 0.0185 0.0185 0.0185 0.0185 0.0184 0.0184 0.0184 0.0184 0.0183
## # ... with 2,140 more variables: `361` <dbl>, `362` <dbl>, `363` <dbl>,
## #   `364` <dbl>, `365` <dbl>, `366` <dbl>, `367` <dbl>, `368` <dbl>,
## #   `369` <dbl>, `370` <dbl>, `371` <dbl>, `372` <dbl>, `373` <dbl>,
## #   `374` <dbl>, `375` <dbl>, `376` <dbl>, `377` <dbl>, `378` <dbl>,
## #   `379` <dbl>, `380` <dbl>, `381` <dbl>, `382` <dbl>, `383` <dbl>,
## #   `384` <dbl>, `385` <dbl>, `386` <dbl>, `387` <dbl>, `388` <dbl>,
## #   `389` <dbl>, `390` <dbl>, `391` <dbl>, `392` <dbl>, `393` <dbl>,
## #   `394` <dbl>, `395` <dbl>, `396` <dbl>, `397` <dbl>, `398` <dbl>,
## #   `399` <dbl>, `400` <dbl>, `401` <dbl>, `402` <dbl>, `403` <dbl>,
## #   `404` <dbl>, `405` <dbl>, `406` <dbl>, `407` <dbl>, `408` <dbl>,
## #   `409` <dbl>, `410` <dbl>, `411` <dbl>, `412` <dbl>, `413` <dbl>,
## #   `414` <dbl>, `415` <dbl>, `416` <dbl>, `417` <dbl>, `418` <dbl>,
## #   `419` <dbl>, `420` <dbl>, `421` <dbl>, `422` <dbl>, `423` <dbl>,
## #   `424` <dbl>, `425` <dbl>, `426` <dbl>, `427` <dbl>, `428` <dbl>,
## #   `429` <dbl>, `430` <dbl>, `431` <dbl>, `432` <dbl>, `433` <dbl>,
## #   `434` <dbl>, `435` <dbl>, `436` <dbl>, `437` <dbl>, `438` <dbl>,
## #   `439` <dbl>, `440` <dbl>, `441` <dbl>, `442` <dbl>, `443` <dbl>,
## #   `444` <dbl>, `445` <dbl>, `446` <dbl>, `447` <dbl>, `448` <dbl>,
## #   `449` <dbl>, `450` <dbl>, `451` <dbl>, `452` <dbl>, `453` <dbl>,
## #   `454` <dbl>, `455` <dbl>, `456` <dbl>, `457` <dbl>, `458` <dbl>,
## #   `459` <dbl>, `460` <dbl>, ...
```

5 ## Spectra Pre-processing

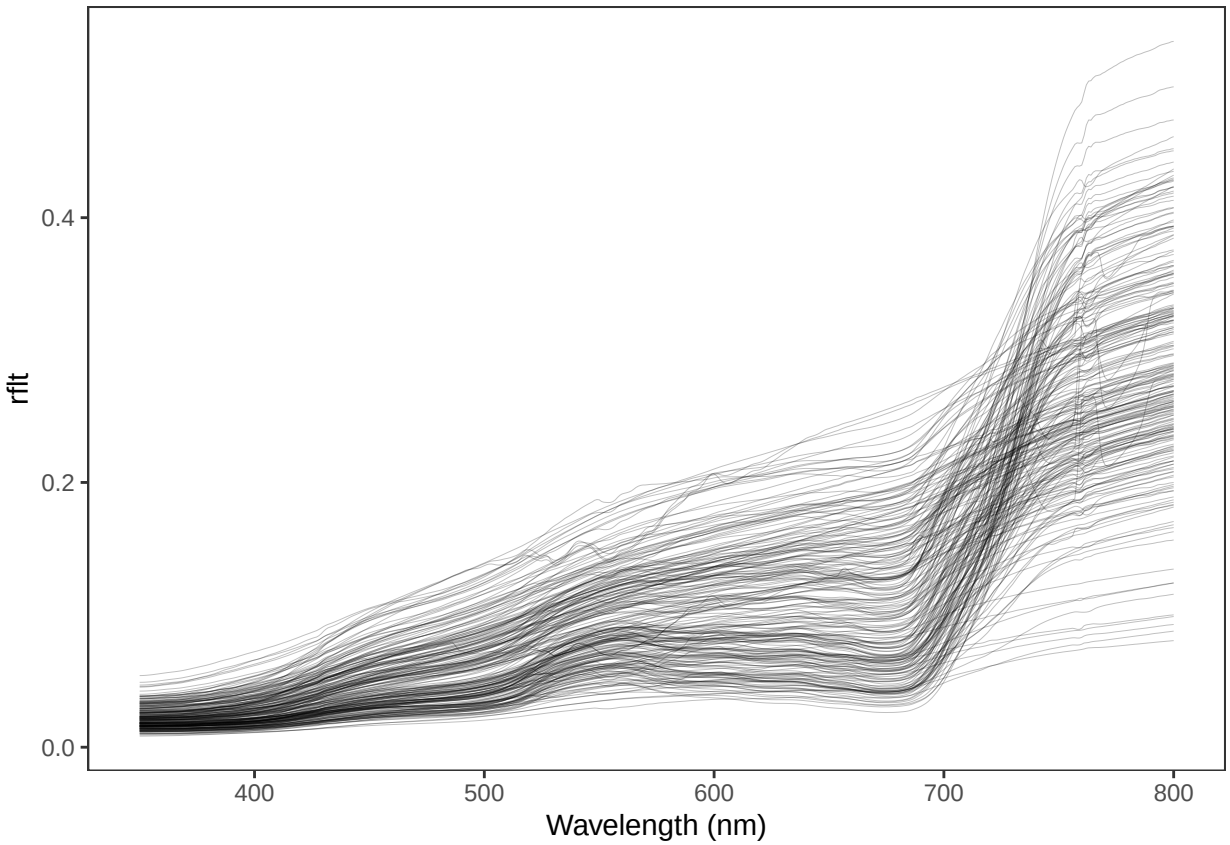
The function plot_spectra() can be used to visualize reflectance spectra. For this, the column "rflt" is specified as input data.

```
plot_spectra(data, col_in = "rflt")
```



Limiting the depicted range to the visible and near-infrared results in a better overview:

```
plot_spectra(data, col_in = "rflt", xlim = c(350, 800))
```



It appears that the signals are sometimes "noisy", i.e. the reflectance values fluctuate randomly around the true value, resulting in "wiggly" lines. Here, the averaging of many spectral readings internally by the measurement device has already resulted in a smoothing of the lines. When chosing lower integration times or averaging less measurements, noise in the signals can be substantial. Furthermore, some spectra appear to be very different from all others (e.g. between 750 nm and 800 nm), suggesting that measurement errors may have occured.

Different techniques can be used to reduce noise in the signal. These are usually summarised under the term "pre-processing" because they are applied before using the spectral data for modelling.

A popular method, Savitzky-Golay filtering, relies on fitting a local polynomial regression to the raw signal. The user specifies the order of the polynomial and the window size (which must be odd).

Several pre-processing techniques are implemented in the function preprocess_spc(). The below code performs Savitzky-Golay filtering on each measurement, fitting a third order polynomial in a window of 21 spectral bands to the raw signal. The processed signal is stored in a list column, renamed according to the procedure applied to process the raw reflectance values.

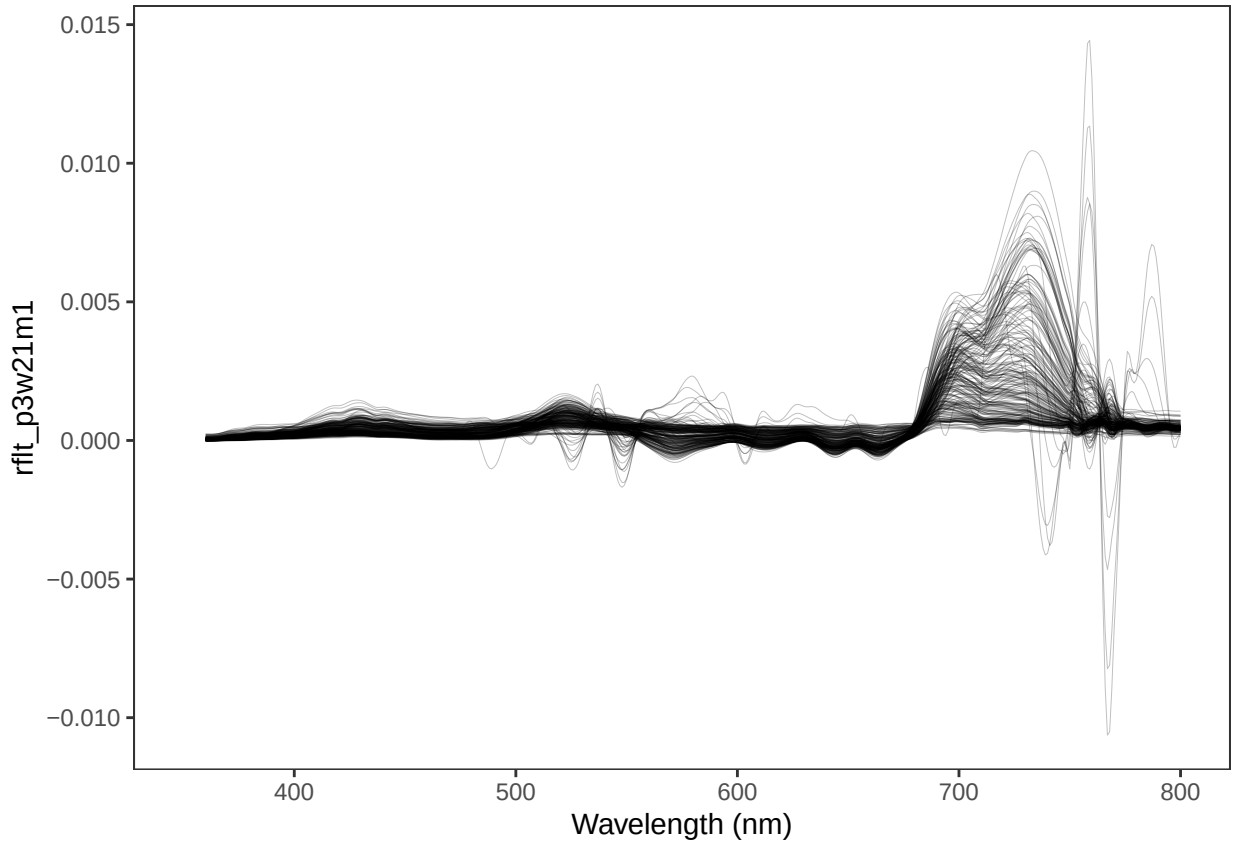
```
spc_pp <- data %>%
  preprocess_spc(w = 21, p = 3, new_col = TRUE)
```

Light scattering effects or differences in the distance between sensor and the measured canopy can result in a shift of the spectral baselines (a "parallel shift" of the signal along the y-axis). Such effects can be reduced by calculating derivatives of the raw spectra. As these tend to be noisy, Savitzky-Golay filtering is most often applied. The below code calculates the first derivative (m = 1) and then applies the Savitzky-Golay smoothing filter, as described above, using the same parameters.

```
spc_pp <- spc_pp %>%
  preprocess_spc(w = 21, p = 3, m = 1, new_col = TRUE)
```

The plot_spectra() function can be used to depict the result:

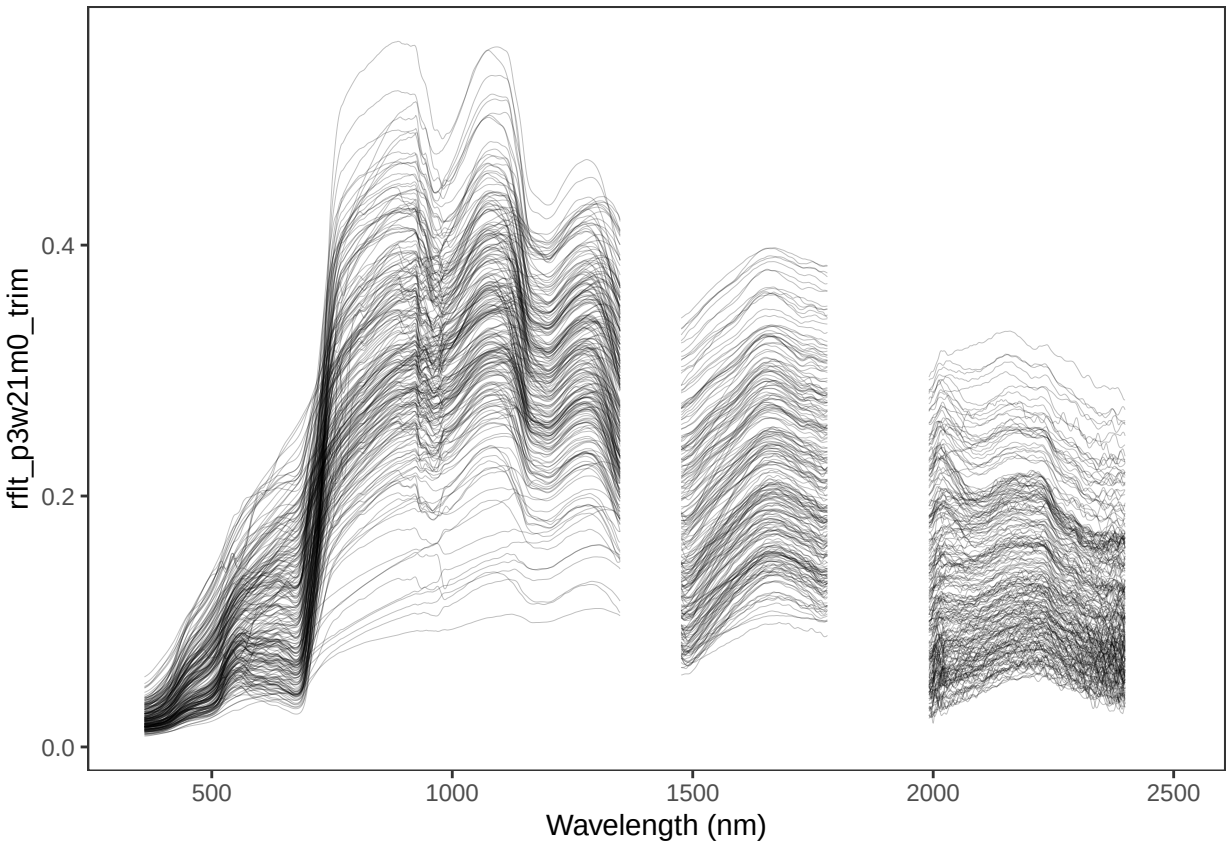
```
plot_spectra(spc_pp, col_in = "rflt_p3w21m1", xlim = c(350, 800))
```



The function preprocess_spc() implements other pre-processing techniques, such as calculation of the standard normal variate and binning of the spectral signal. Wether these or other pre-processing techniques are required may be best determined by evaluating the effects on the association between spectra and reference measurements (e.g. through comparing performance metrics in predictive modelling).

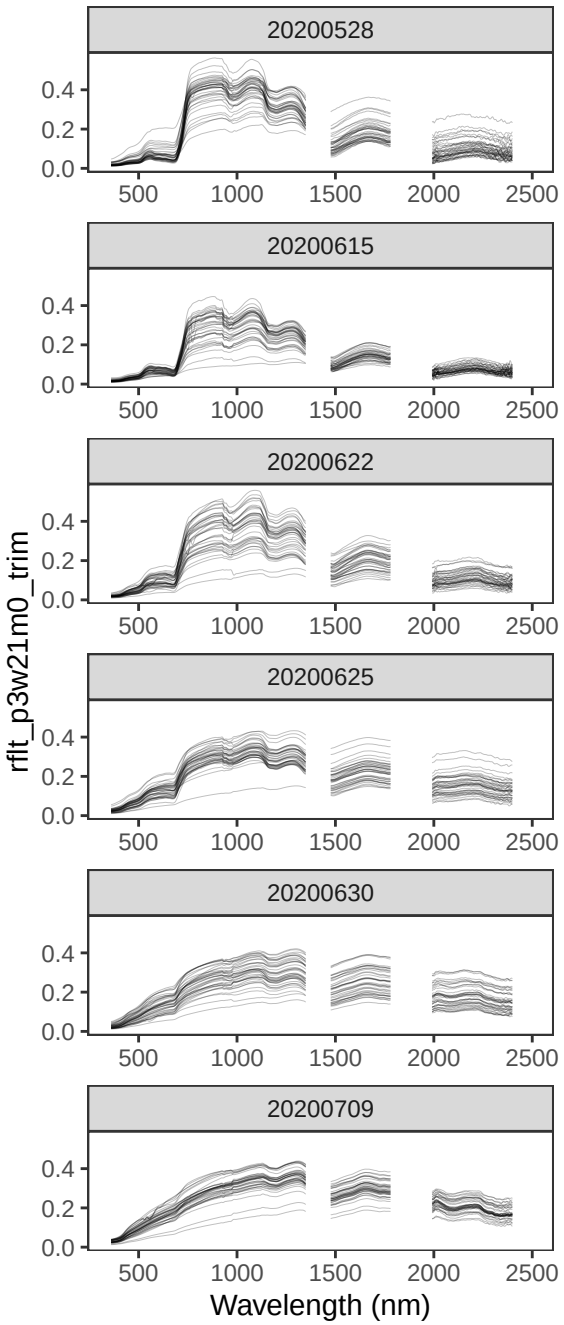
As seen before, there appear to be outliers in the measurement series. It may be desirable to identify these outliers before conducting further analyses (for example, calculation of spectral indices or predictive modelling). Before looking for outliers, however, we remove parts of the spectrum that are particularly noisy (even after pre-processing of the signal). These regions represent parts of the spectrum with a naturally very low signal to noise ratio due to atmospheric absoprtion of incoming radiation. In addition, the wavelengths above 2400 nm are also noisy and should be removed. This is done using the preprocess_spc() function:

```
spc_pp <- spc_pp %>%
  preprocess_spc(col_in = "rflt_p3w21m0", # smoothed spectra
                 p = 0, # Savitzky-Golay smoothing was done previously
                 trim = c(1350, 1475, 1781, 1990, 2400, 2500), # three spectral regions are removed
                 new_col = TRUE) # existing col_in is overwritten
plot_spectra(spc_pp, col_in = "rflt_p3w21m0_trim") # check result
```



Another consideration to be made before removing outliers concerns the structure of the data. In agronomy or breeding related applications of spectrometry, measurements are often repeated in time to track emerging phenomena or follow growth or developmental processes. In the present dataset, measurements were carried out through time in order to track the appearance and proliferation of a disease. Variation in spectral reflectance due to the time point of measurement can be very substantial. This can variation can be seen by plotting indivual measurement dates separatly.

```
plot_spectra(spc_pp,
             col_in = "rflt_p3w21m0_trim",
             facets = "meas_date") # specify the grouping variable as "facets" (passed to ggplot::facet
```



An average measurement taken on 7 July 2020 should probably be regarded as an outlier on 18 May 2020. Thus, the grouping structure must be considered when determining whether a spectral measurement is an outlier or not. This can be done using the function detect_outlier_spectra().

```
spc_pp <- spc_pp %>%
  detect_outlier_spectra(grouping = "meas_date", # specify the grouping variable
                         col_in = "rflt_p3w21m0_trim",
                         outliers_rm = NULL, # do not remove measurements detected as outliers
                         create_plot = F)
```

Detected multivariate outliers are marked in red. Which measurements are detected as outliers depends on

the type of data used. Using e.g. first derivatives will identify other measurements as outliers than the ones identified here.

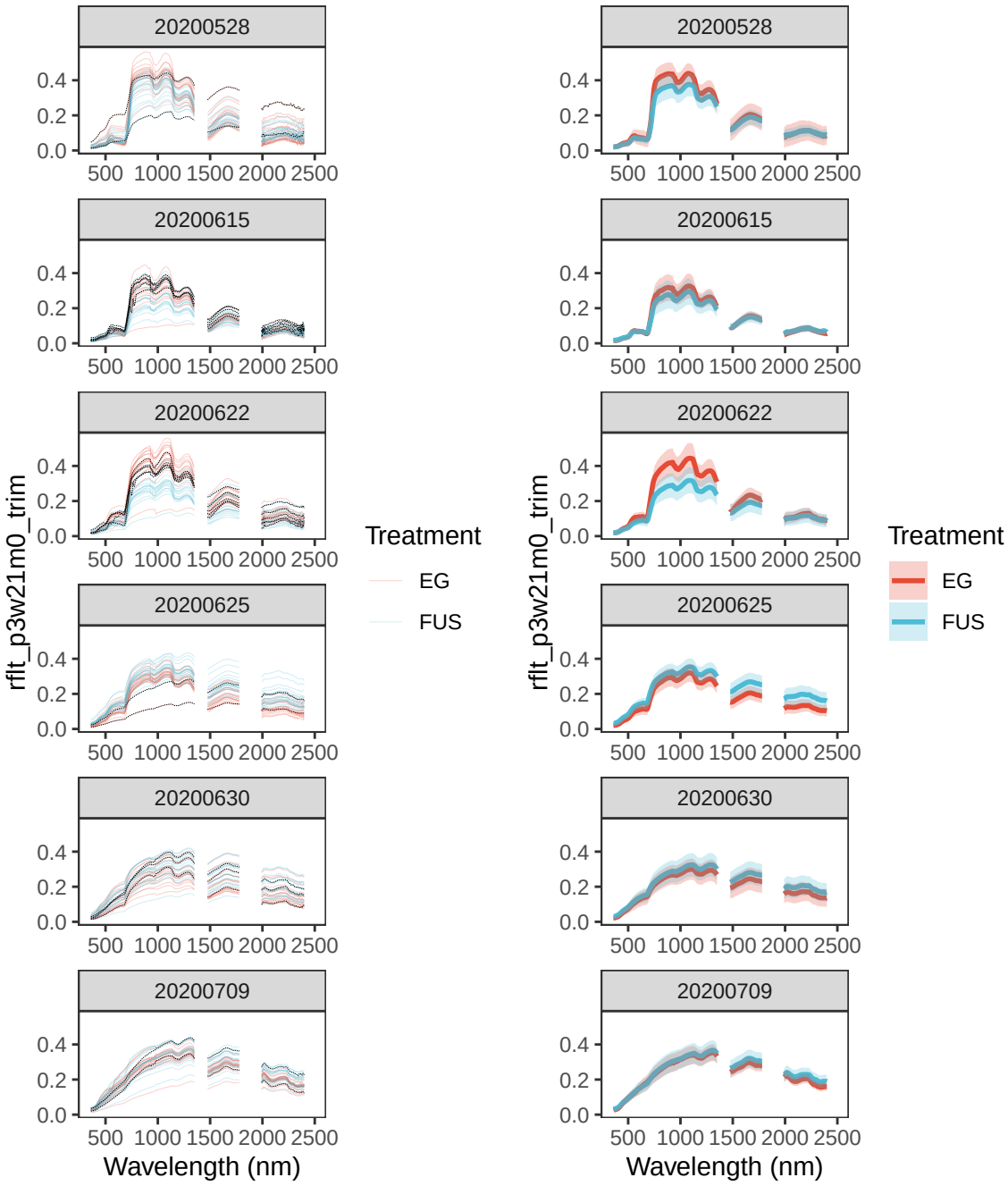
Apart from the above figure, the function detect_outlier_spectra() also returns a list, containing the cleaned or original dataset (depending on the outliers_rm parameter) as first list element and the measurement ids of the detected outliers as a second list element.

Treatment effects

A central question is often whether treatments of some kind can be distinguished using spectral reflectance. Such treatments can consist in e.g., different fertilizer application regimes. In this dataset, treatments consist in the presence or absence of artificial inoculation with FHB. To get a quick idea whether treatments have an effect on reflectance properties of the studied vegetation, the plot_spectra() can be used and the treatment variable specified. So far, treatments consisting of a factor combination must be encoded with a dummy variable with levels representing the full-factorial combination.

```
# add measurement meta data (including the treatment information)
meta <- read_csv(paste0(path_to_data, "spectral_files_fusarium_asign.csv"))
spc_ref <- right_join(meta, spc_pp[[1]], by = c("sed_new" = "meas_id")) %>%
  dplyr::rename("meas_id" = "sed_new")

# plot spectra
plot_spectra(spc_ref, facets = "meas_date", col_in = "rflt_p3w21m0_trim",
             treatment = "Treatment", mark_outliers = T)
```



Spectral Vegetation Indices

Most often, the first step in hyperspectral data analysis (after signal pre-processing and data exploration and quality control) consists in the calculation of spectral vegetation indices (SVI) and a correlation analysis with a reference measurement (here FHB disease severity). Spectral vegetation indices are mathematical combinations of reflectance values at specific wavelengths and therefore use only a small part of the full spectra. The goal of this analysis is to identify SVI that could be used as a surrogate for a reference measurement (here a visual scoring of disease symptoms).

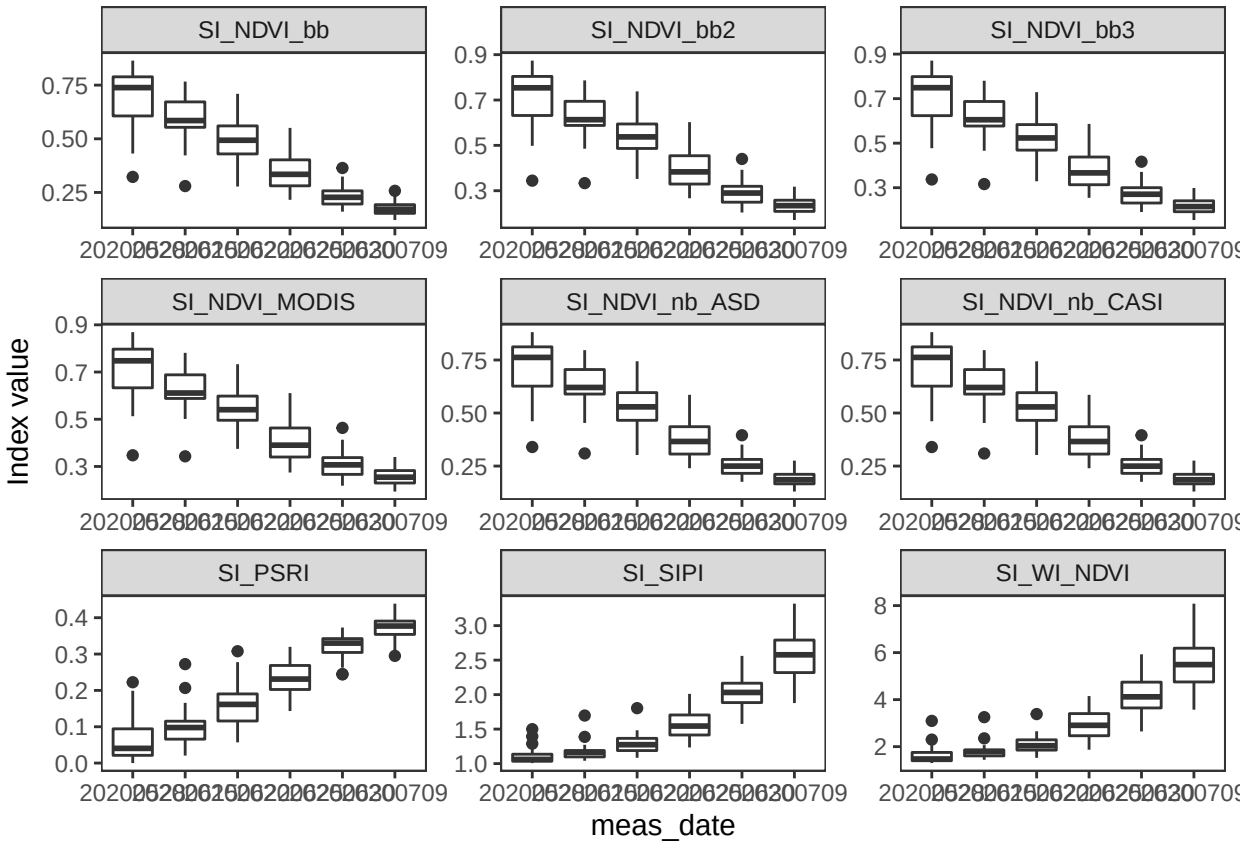
A large number of SVI have been developed for different purposes in different types of vegetation, ranging

from the estimation of ground cover to disease severity estimation. The function calculate_SVI() calculates reflectance-based SVI. Other SVI use differently pre-processed spectra (e.g., the first derivative).

```
SVI <- spc_ref %>%
  calculate_SVI(col_in = "rflt_p3w21m0")
```

The function plot_SVI() can be used to visualize results. Through the argument "svi" the a seletion of indices to visualize can be made. The argument "x" is used to specify the

```
plot_SVI(SVI, svi = c("SIPI", "PSRI", "NDVI"),
         col_in = "SVI",
         x = "meas_date")
```

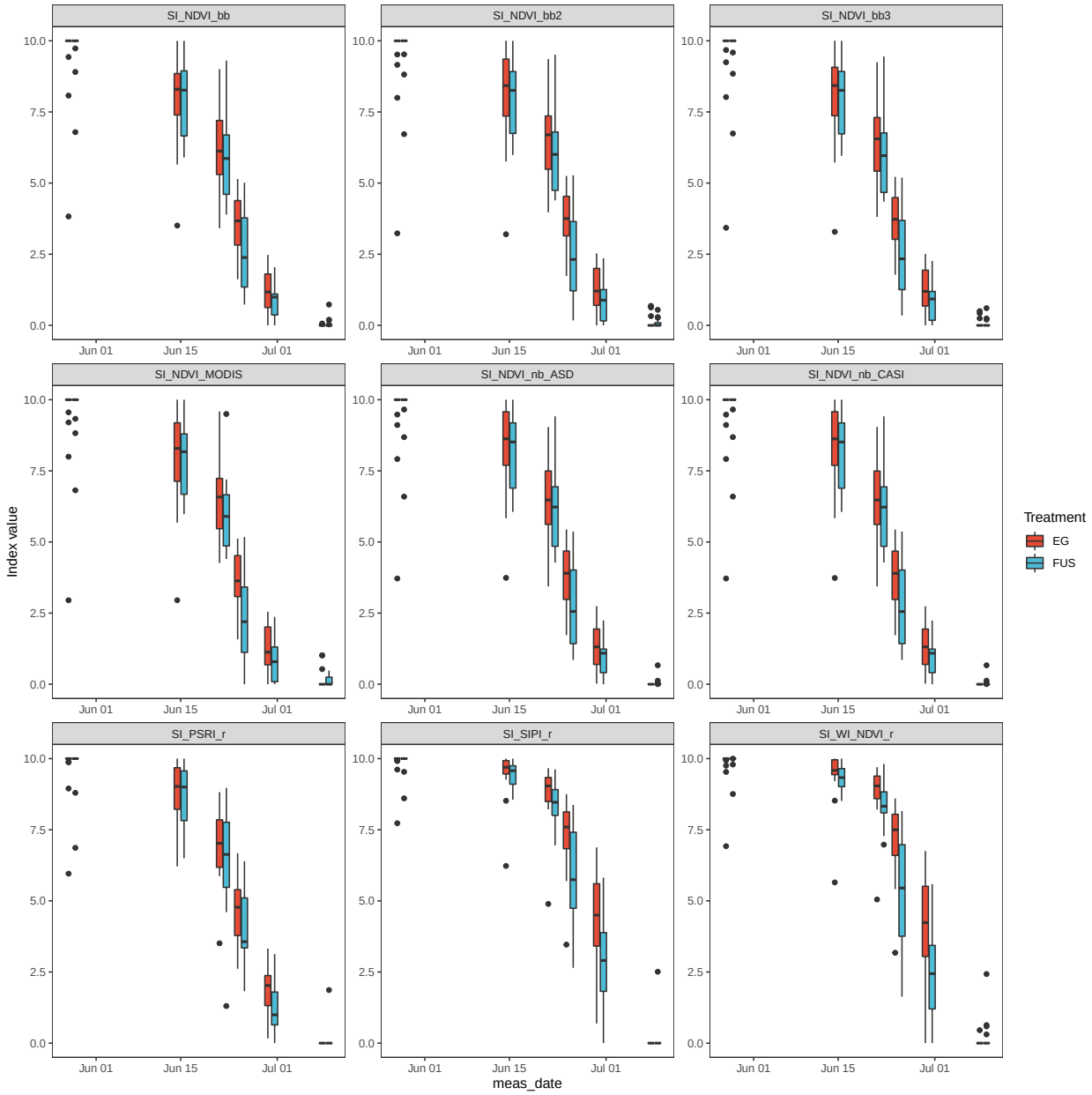


It can be seen that the scale for different SVIs is different, and some increase over time, whereas others decrease. It would be helpful for direct comparisons and for downstream analysis to homogenize scales for different SVI. This can be done using the function scale_SVI().

```
SVI <- scale_SVI(SVI, plot = F)
```

Furthermore, in the plot above, it would be instructive to differentiate between treatments, and to scale the x axis to represent measurment dates, rather than just factor levels. For this, the call to plot_SVI() is modified as follows:

```
plot_SVI(SVI, svi = c("SIPI", "PSRI", "NDVI"),
         col_in = "SVI_sc",
         x = "meas_date",
         x_is_date = TRUE,
         groups = "Treatment")
```




```
parameters <- get_svi_dynamics(data = SVI, timevar = "dafm", plot = F)
parameters
```

```
## # A tibble: 5,320 x 8
## # Groups:   Plot_ID, Treatment, variable [5,320]
##    Plot_ID           Treatment variable     t80   t50   t20  dur1  dur2
##    <chr>             <chr>     <fct>      <dbl> <dbl> <dbl> <dbl> <dbl>
```

```
##  1 EG - PL34 - 1 - A  EG       SI_760_730     9    21    28    12    19
##  2 EG - PL34 - 1 - E  EG       SI_760_730    21    24    29     3     8
##  3 EG - PL34 - 10 - E EG       SI_760_730    NA    NA    NA    NA    NA
##  4 EG - PL34 - 12 - A EG       SI_760_730     9    21    29    12    20
##  5 EG - PL34 - 13 - B EG       SI_760_730    13    23    30    10    17
##  6 EG - PL34 - 13 - D EG       SI_760_730    NA    NA    NA    NA    NA
##  7 EG - PL34 - 15 - E EG       SI_760_730     7    18    29    11    22
##  8 EG - PL34 - 16 - B EG       SI_760_730     6    15    25     9    19
##  9 EG - PL34 - 17 - E EG       SI_760_730     6    15    25     9    19
## 10 EG - PL34 - 18 - B EG       SI_760_730     7    16    27     9    20
## # ... with 5,310 more rows
```