The structure and interaction of silica in plant cell walls: Data analysis report

Felipe Beltran

6/16/2021

Introduction

In this document we will save all data manipulation and analysis related to the infrared spectra from the project: **The structure and interaction of silica in plant cell walls**

Loading data into R

First, we create an object called `names`, which is a `list`, in which we will save the names of the files with the `.CSV` extension contained in this folder.

```
names <- list.files(pattern = '.CSV')
```

Then, we use the function `lapply` to apply the function `read.csv` to each element of the list `names` and then save each of the `data.frames` from the files in to R.

```
spectra.list <- lapply(names,
                       read.csv,
                       header = F)
```

Now, se can save the frequencies of incident light as a vector called `wavenumbers`. The code `spectra.list[[1]][1]` selects the first element of the list `spectra.list` using the index operator for lists `[[1]]`, which is a `data.frame`. Then, we select the first column of that `data.frame` using the index operator `[1]`:

```
wavenumbers <- unlist(spectra.list[[1]][1])
```

The object `wavenumbers` has 1 row and 1869 columns, each column is one of the frequencies used by the spectrometer in the experiment.

As we did previously, we can use again the function `lapply` to apply the index operator `[` and select ONLY the second column of each `data.frame` that has inside the absorbance at each frequency.

```
spectra.list2 <- lapply(spectra.list, '[', 2)
```

In order to make data easier to manipulate, we can transform the list of columns called `spectra.list2` into a `data.frame` as follows:

```
spectra.df <- as.data.frame(
  t(as.data.frame(spectra.list2))
  )
```

the object `spectra.df` has 318 rows, i.e. 318 spectra from different samples and 1869 columns, or infrared frequencies.

Since the function `t()` transforms `data.frame` class objects into `matrix` class objects, we have to assign again to `spectra.df` (a matrix) the corresponding column and row names:

```
rownames(spectra.df) <- names
colnames(spectra.df) <- wavenumbers
# gsub('.{6}$', '', names)
```

Tidying up the `data.frame`

Since the names of the sample right now have the following structure:

```
head(names[1:3])
```

```
## [1] "125-1.CSV" "125-2.CSV" "125-3.CSV"
```

We can get rid of the unnecessary characters in the names of the samples, such as the extension and other symbols such as `"` or `..` For that we can use the function `gsub()` and `regex` or regular expressions in order to replace these symbols for nothing, or `''`.

The regular expression for this replacement is built as follows:

- `.` matches any element that complies with the requirement, or applies the query to every element
- `{4}` replaces characters by `''` exactly 4 times
- `$` matches the end of each name

That is described before allows us to tell `R` that we want to replace the last 4 characters in each name by nothing, or `''`.

```
names2 <- gsub('.{4}$',
               '',
               rownames(spectra.df))
head(names2[1:4])
```

```
## [1] "125-1" "125-2" "125-3" "126-1"
```

There are some samples that have longer names, such as:

```
head(names2[292:318])
```

```
## [1] "P1-1_Thu May 09 07-26-09 2019 (GMT+02
## [2] "P1-2_Thu May 09 07-26-09 2019 (GMT+02
## [3] "P1-3_Thu May 09 07-26-09 2019 (GMT+02
## [4] "P2-1_Thu May 09 07-26-09 2019 (GMT+02
## [5] "P2-2_Thu May 09 07-26-09 2019 (GMT+02
## [6] "P2-3_Thu May 09 07-26-09 2019 (GMT+02
```

We can also save just the information that is useful for displaying in tables and plots:

```
names2[292:318] <- gsub('.{41}$',
                        '',
                        rownames(
                          spectra.df
                          )[292:318])
head(names2[292:318])[1:4]
```

```
## [1] "P1-1" "P1-2" "P1-3" "P2-1"
```

Finally, we assign the curated names to our data frame, `spectra.df`

```
rownames(spectra.df) <- names2
```

Plotting intial data

Since we have joined several spectra from different samples, we can try to visualize what we are dealing with.

Knowing that we have already the names of the samples, it would be handy if we used a factor that helped us to differentiate replicas of experiments:

```
cols <- factor(gsub('.{2}$', '', names2))
str(cols)
##  Factor w/ 106 levels "125","126","127",..: 1 1 1 2 2 2
```

If we use the factor `cols`, we can plot three replicas of one sample using the same color

Now, we can use a loop to plot each one of the rows of our `data.frame`:

```
for(i in  1:length(rownames(spectra.df))){

  plot(wavenumbers,
    spectra.df[i,],
    axes = F,
    xlab = '',
    ylab = '',
    xlim = c(4000, 400),
    ylim= c(0,0.2),
    type = 'l',
    col =cols[i]

  )
  par(new = T)
}

box()
axis(1)
axis(2)
title(main = '',
      xlab = expression(
        paste('Wave number (cm'^'-1',
              ')')
        ),
      ylab ='absorbance (a.u.)')
```

As we can see in Figure 8 In the range between 2500 and 2000 cm-1 variability due to experimental noise and background correction is present. In order to use multivariate methods and extract the chemical information from the dataset, we can select a region of interest (ROI) in which we take into account characteristic bands for molecules which concentration can be different between samples, such as silicon oxides, or lignin monomers.

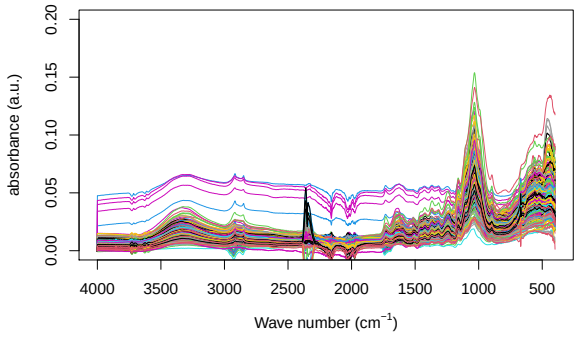


Figure 1: raw spectra of triplicated samples

range selection

Now, we can check for where is that ROI in our data frame. Since the frequencies are arranged along the columns, we can search for the columns that have column names between 1700 and 400 cm^{-1}

```
colnames(spectra.df)[c(1,676)]
```

```
## [1] "399.1989" "1700.935"
```

Once we know where the ROI is in the data frame, we can create a subset called `range1` to store the spectra of all samples within this range, and plot it:

```
range1 <- spectra.df[,c(1:676)]
wavenumbers1 <- wavenumbers[c(1:676)]
```

```
for(i in  1:length(rownames(range1))){

  plot(wavenumbers1,
    range1[i,],
    axes = F,
    xlab = '',
    ylab = '',
    xlim = c(1700, 400),
    ylim= c(0,0.2),
    type = 'l',
    col =cols[i]

  )
  par(new = T)
}

box()
axis(1)
axis(2)
title(main = 'raw spectra - ROI',
      xlab = expression(
        paste('Wave number (cm'^'-1',
              ')'
              )),
      ylab ='absorbance (a.u.)')
```

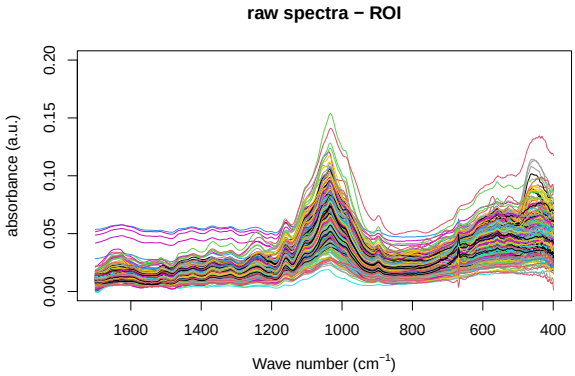


Figure 2: ROI spectra of triplicated samples

mean spectra calculation

Since each sample has three replicated spectra, we can calculate the mean of each three samples as follows:

- First, we create a vector to tell R which samples to look for. Using the function `unique()` we extract the unique values without taking their repetitions:

```
search.vector <- unique(unlist(cols))
head(search.vector)
```

```
## [1] 125 126 127 128 129 130
## 106 Levels: 125 126 127 128 129 130 132 133 134 135 136
```

Once we have the names for each sample, we can search for $\frac{318}{3} = 106$ triads of samples. Once we know where they are, we can calculate the mean for each three replicates.

First, we create a list in which we are going to save the position of each three spectra:

```
index <- list(106)
```

Now we can again use the help of regular expressions to search for every sample that matches with one single sample:

As an example:

The regular expression that we will use this time is `(?=.*<search>)` where `<search>` is replaced with the sample's name.

This will find any element for the vector `rownames(spectra.df)` that matches with the character in `search vector`. For example, if we search for the first sample, `search.vector[1]` which is sample `125`:

```
as.character(search.vector[1]) # the name of
```

```
## [1] "125"
```

```
#the first sample
```

The search will tell us the positions in `rownames(spectra.df)` where where we can find the character `'125'`

```
which(
  grepl(
    paste0('(?=.*',
           as.character(search.vector[1]),
           ')'
           ),
    rownames(spectra.df),
    perl=T
    )
  )
```

```
## [1] 1 2 3
```

Then we can confirm:

```
rownames(spectra.df)[c(1, 2, 3)]
```

```
## [1] "125-1" "125-2" "125-3"
```

This process can be automated using a `for` loop in R:

```
for (i in 1:106){

  index[[i]] <- which(
    grepl(
      paste0('(?=.*',
             as.character(search.vector[i])
             ,')'),
                       rownames(spectra.df),
                       perl=T
                       )
                   )
}
```

```
search.vector[2]
```

```
## [1] 126
## 106 Levels: 125 126 127 128 129 130 132 133 134 135 136
```

```
rownames(spectra.df)[c(index[[2]])]
```

```
## [1] "126-1" "126-2" "126-3"
```

Once we know where each triplicate is, we can proceed to calculate the means:

- First, we create a matrix to save the mean spectra that will result from the calculation.

```
mean <- matrix(ncol= ncol(range1),
               nrow = nrow(range1)/3)
```

*Then, we assign to this empty matrix, `mean`, its corresponding column and row names:

```
# mean <- as.data.frame(mean)

colnames(mean) <- colnames(range1)
rownames(mean) <- search.vector
```

Then we can loop through columns indexing each wave number, one after another, in each iteration of the loop, or each time `j` changes its value. Posibles values for `j` are defined when `in 1:length(ncol(range1))`. where `ncol` gives us the number of columns of `range1` or the number of wave numbers present along the ROI.

At the same time, `i` takes values at each one of the numbers present between `1` and `nrow(mean)`

at step 1 in both loops, we save in the position `mean[1,1]` the resulting value when we use the function `mean()` which as default calculates the arithmetic mean (when the argument trim = 0 is given as default, the function trims a fraction of 0 observations from each end of the three observations before the means is computed.)

In order to calculate this mean estimator from a sample of 3 replicates, we index the spectral matrix `range1` in three special positions for each sample, those found in the creation of the `index` list.

In this step of the calculation, the first value of absorbance for the sample 125 at wave number 399.1, is empty, as are all the other slots of this matrix.

```
mean[1,1]
```

```
## [1] NA
```

for example, for the first sample, and the first wave number, we will save the result of the calculation of the mean of three replicas of sample 125, and at a wave number of 399.1, `mean[1,1]`.In order to accomplish this, we select the first position of the list `index[[1]]`:

```
index[[1]]
```

```
## [1] 1 2 3
```

when we print the content of this position of the list, we can notice that the samples `125-1` `125-2` and `125-3` lie in the positions 1, 2, and 3 respectively of the data.frame `range1`.

So the calculation will be applied to `range1[index[[1]][1],1]`, `range1[index[[1]][2],1]` and `range1[index[[1]][2],1]`, or rows 1, 2, and 3.

the process is then repeated through all of the wave numbers (columns) and samples (rows) of the data set.

```
for(j in 1:ncol(range1)){

for(i in 1:nrow(mean)){

mean[i,j] <- mean(c(range1[index[[i]][1],j],
                    range1[index[[i]][2],j],
                    range1[index[[i]][3],j]
                                    ) )
}
}
```

then we turn this matrix into a data.frame:

```
mean <- as.data.frame(mean)
```

Once we have calculated the mean for each sample, we can plot the resulting mean spectra using the same factor used before, `cols` to color each sample of the same color of their triplicates.

```
cols.means <- as.factor(search.vector)

for(i in  1:length(rownames(mean))){

  plot(wavenumbers1,
    mean[i,],
    axes = F,
    xlab = '',
    ylab = '',
    xlim = c(1700, 400),
    ylim= c(0,0.135309),
    type = 'l',
    col =cols.means[i]

  )
  par(new = T)
}

box()
axis(1)
axis(2)
title(main = '',
      xlab = expression(
        paste('Wave number (cm'^'-1',
              ')')
        ),
      ylab ='absorbance (a.u.)')
```

Once we have calculated the mean estimator for each multivariate spectra, we can proceed to perform pretreatments in order to clean the data, and extract as much as chemical information as we can.

baseline correction

In this experiment we have performed an analogue to the the baseline correction described by beleites et al. 2020:

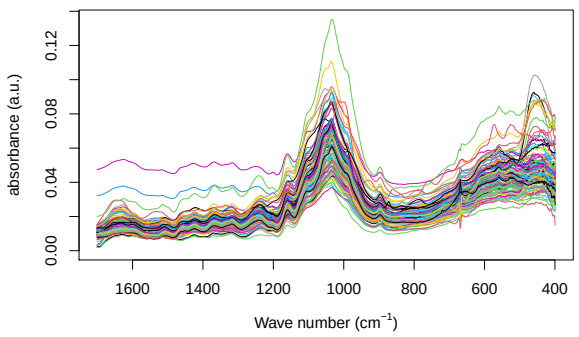


Figure 3: Calculated mean spectra

```
library(hyperSpec)
# hyperSpec package functions and
#data is charged in this session
```

We create an object that can be identified and manipulated by the functions of `hyperSpec` package.

```
spc <- new('hyperSpec', # The class of the ob
           spc= mean, # the spectra matrix
           wavelength = wavenumbers1
 # independent variable, whether wave number
#or wave length
           )
```

then in an object called `bend`, we can save the result of `wl.eval` applied to `spc`. This function generates a baseline 'reference spectra':

```
bend <- 0.1 * wl.eval(spc,
                      function (x)
                      x^6+x^5+x^4+x^3+x^2,
                      normalize.wl =
                        normalize01)
```

- `function (x) x^6+x^5+x^4+x^3+x^2` defines a polynomial that is used to fit the baselines to the supporting points in the lower region of the spectra contained in 'spc', using this functions the baseline 'reference spectra' is calculated
- `normalize.wl=normalize01` is a function used to transform the wave numbers before evaluating the polynomial. using `normalize01` we map the range of each wave number to the interval $[0, 1]$
- the multiplication by 0.1 is the normalization chosen for these spectra.

Now we can use the `rubberband` method to estimate the baseline. Setting the noise to $1 \cdot 10^{-4}$ allows us to take advantage of the low noise presented for this spectra. (a noise of `noise=4` sets an interval of 2 times standard deviations).

```
bl <- spc.rubberband(spc+bend,
                     noise = 1e-4,
                     df = 20)-bend
```

The base-line correction method selected for this work is one of many, and tuning the correction parameters can be subject to a design of experiments, using as a response variable a quality measure of the analysis that is going to be performed after the correction. For example Q^2 for PCA, or R_{adj} and $RMSEP_{CV}$ for multiple linear regression models as suggested by hovde 2010.

Now, we can visualize the results of these baseline corrections.

First, the spectra before correction and the estimated baseline 'reference spectra':

```
labels (spc, ".wavelength") <-
  expression(paste(
    'Wave number (cm'^'-1',
    ')'))
labels (spc, "spc") <-
  expression(paste('Absorbance (a.u.)'))

plot(spc, wl.reverse = TRUE)
plot(bl, add=TRUE, col=2,wl.reverse = TRUE)
```

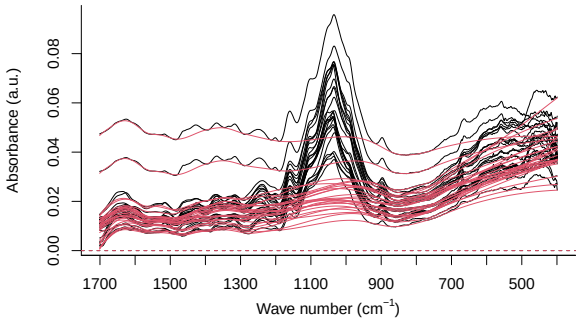


Figure 4: Mean spectra and rubberband baseline

And the heart of the rubberband method, the bending of the spectra to add support points in the convex part of the spectra:

```
sum <- spc+bend
plot(sum,wl.reverse = TRUE)
plot(bend, add=TRUE, col=2,wl.reverse = TRUE)
```

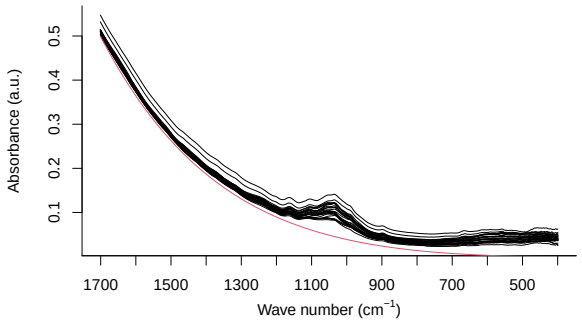


Figure 5: bent mean spectra and bent baseline

Then, the corrected spectra, which is calculated by substracting the baseline `bl` from the spectra `spc`:

```
spc3 <- spc - bl
spc3 <- spc3 + (min(spc3)*-1)
# We add the minimum value
#which is negative to have only positive
#values
plot(spc3,wl.reverse = TRUE)
```

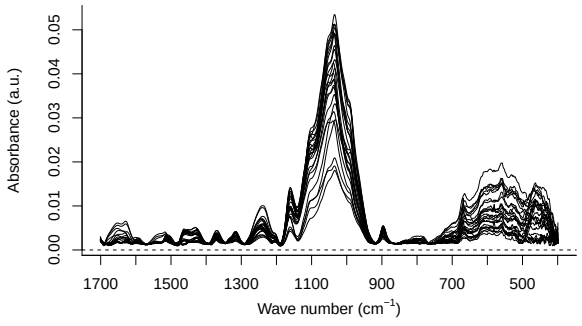


Figure 6: baseline corrected mean spectra

Now, we can extract the corrected spectra from the `hyperSpec` object and save it as a `data.frame` to handle the information in an easier way later.

```
corrected1  <- as.data.frame(spc3[1:106])
corrected <- as.data.frame(corrected1[,1])
corrected <- corrected + (min(corrected)*-1) # shifting up
```

Up to this point, we have calculated the mean spectra and corrected the base-line of each spectrum.

multiplicative scatter correction (MSC)

Since This is a popular pre-treatment (Rinnan 2009, Wu 2018) we tried to use it in this study.

```
library(pls)
correctedMSC <- msc(as.matrix(mean))
```

```
for (i in 1:length(rownames(correctedMSC))){

  plot(as.numeric(colnames(correctedMSC)),
     correctedMSC[i,],
     xlab = '',
     ylab = '',
     axes = F,
     type = 'l',
     xlim = c(1700,400),
     ylim = c(0,0.12),
     col = cols[i])
  par(new = T)
}
box()
axis(1)
axis(2)
title(main = '',
      xlab = expression(paste(
        'Wave number (cm'^'-1',
        ')')),
      ylab = 'Absorbance (a.u.)')
```

Since no noise reduction was identified, this option was dismissed for the moment.

metadata

First, we load the metadata table, created from the data table 'Datos_para_colombia.xlsx' available here:

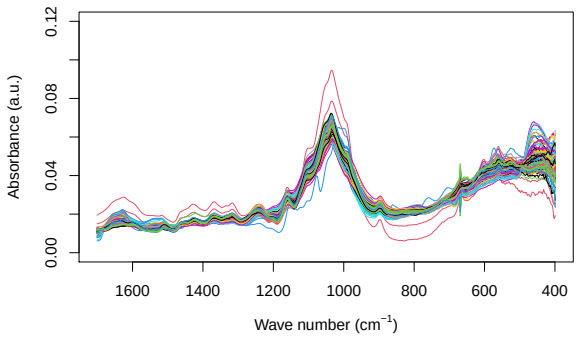


Figure 7: MSC correction of mean spectra

```
library(readxl)
metadata <- read_excel("metadata.xlsx")
```

The object called metadata is a data.frame and it has 88 rows and columns as it is. The rows vary with samples and the columns with variables. The variables in the columns are:

```
options(width = 30)
colnames(metadata)
## [1] "sample" "class"  "N %"
## [4] "C %"    "Si"     "Ca"
## [7] "Mg"     "P"
```

where:

- `sample`: Contains the alpha numeric ID given to each sample. *class=numeric*.
- `class`: The group of the experiment from where the sample comes from *class=character*.
- `N %`: The nitrogen percent quantified. *class=numeric*.
- `C %`: The nitrogen percent quantified. *class=numeric*.
- `Si`: Silicon quantified by ICP-OES *class=numeric*.
- `Ca`: Calcium quantified by ICP-OES *class=character*.
- `Mg`: magnesium quantified by ICP-OES *class=character*.
- `P`: Phosphorous quantified by ICP-OES *class=character*.

Since the raw data is not debugged (as the majority of experimental raw data sometimes are) we can tidy up this table as far as we can using `R`.

First we can search if there is any row of the `metadata` that has its label or ID missing using the column `sample`:

```
which(is.na(metadata$sample))
```

```
## [1] 17
```

The row 17 has an NA or blank space in the column `sample`. we can proceed to remove this row:

```
metadata <- metadata[-c(17),]
which(is.na(metadata$sample))
```

```
## integer(0)
```

`integer(0)` means that once we have deleted row 17, there are no blank spaces whatsoever.

matching samples that have metadata with samples with spectra

We can now try to find where in the rows of `corrected` are the samples listed in `metadata$sample`. Doing this we will know which sample that has quantification and classification data, also has spectra available and where it is.

```
positions <- vector('list', 87) # the same sizeas metadata

for (i in 1:87){

positions[[i]] <- which(
                  grepl(
                  paste0('(?=.*',
                  as.character(
                    metadata$sample[i]),
                        ')'),
                  rownames(corrected),
                 perl=T
                             )
                       )
}
```

In `positions`, an object of *class=list* we save in each position where in the data frame `corrected` the current sample is. For instance:

```
metadata$sample[1]
```

```
## [1] 125
```

```
positions[[1]]
```

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

```
rownames(corrected)[positions[[1]]]
```

```
## [1] "125"
```

In the last chunk we print the name of the spectra, `rownames(corrected)`, that is selected by the indexing list `positions[[1]]` which is `125`, the same sample at the first row of `metadata`.

This seems a little bit obvious but in the raw data there are several spectra rows that have not matching quantification data, or vice versa, there are several samples that have quantification information but have no spectra available.

Since now whe know where in the spectral matrix is each sample of the `metadata` data set, we can add a new column that relates where is each sample of `metadata` in the `corrected` matrix:

```
for(i in 1:length(metadata$sample)) {

  if(length(positions[[i]]) == 1){

    metadata$spectra[i] <-
      rownames(corrected)[positions[[i]]]
  }else{
    metadata$spectra[i] <- NA
  }

}
```

```
## Warning: Unknown or
## uninitialised column:
## 'spectra'.
```

Now we can compare if the indexing tool `positions` is doing its job. Creating the `data.frame` called `compare` we list side to side the names of `metadata` samples and the names of `corrected samples`

```
compare <- data.frame(metadata =
                        metadata$sample,
                      spectra =
                        metadata$spectra)
head(compare)
```

```
##   metadata spectra
## 1      125     125
## 2      126     126
## 3      127     127
## 4      128     128
## 5      129     129
## 6      130     130
```

Since now we know where in `corrected` is each spectra for each sample from `metadata`, we can add this information to the `metadata` table:

```
for(i in 1:length(metadata$sample)) {

  if(length(positions[[i]]) == 1){

    metadata$spectra.index[i] <- positions[[i]]
  }else{
    metadata$spectra.index[i] <- NA
  }

}
```

```
## Warning: Unknown or
## uninitialised column:
## 'spectra.index'.
```

This is an analogue process to that used for saving the names of the spectra contained in `corrected`. The difference is that now in the column `metadata$spectra.index` we have exactly where the sample of the current row, is in the rows of the `corrected` matrix.

Exploratory Data analysis

Hierarchical clustering

Hierarchical clustering can be used to measure multidimensional distances between objects, or samples. If the objective is to use this analysis we have to first tidy up the data and prepare it for the analysis.

First, we can search for missing values in the classification column:

```
which(is.na(metadata$class))
```

```
## [1]  7 26
```

Now, once is known that samples in rows 7 and 26 have no classification, we can create a new data table without these

```
metadata.class <- metadata[-c( which(is.na(me

which(is.na(metadata.class$class))
```

```
## integer(0)
```

Once the samples that have no classification information have been removed, it is useful to check if there is any missing spectrum for the remaining samples:

```
which(is.na(metadata.class$spectra.index))
```

```
## [1] 24 26 29 32 35
```

Since these samples have no spectra, they can be also removed as follows:

```
metadata.class2 <- metadata.class[-c(which(is.na(metadata.class$spectra.index)) ),]
which(is.na(metadata.class2$spectra.index))
```

```
## integer(0)
```

The next thing needed for the analysis is the spectra. It is important that each classification matches each spectra:

```
spectra.class <- corrected[metadata.class2$spectra.index,]
```

Then we can compare if we have the same samples in both tables:

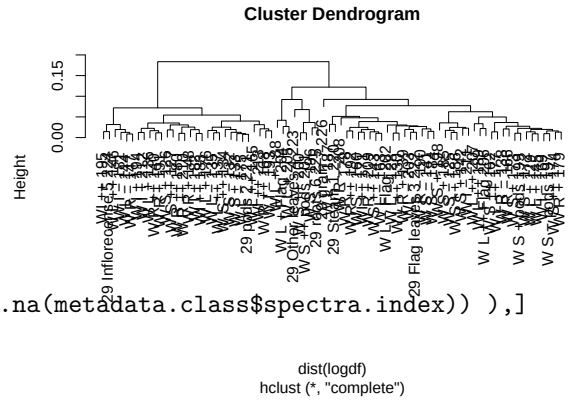
```
compare.class <- data.frame(classification = m
                            spectra= rownames(
head(compare.class)
```

```
##   classification spectra
## 1            125     125
## 2            126     126
## 3            127     127
## 4            128     128
## 5            129     129
## 6            130     130
```

```
df1 <- spectra.class

names.class <- paste(metadata.class2$class, rownames(spect

rownames(df1) <- names.class
logdf <- log10(df1[,1:676] + 1)
rownames(logdf) <- names.class
library(factoextra)
```

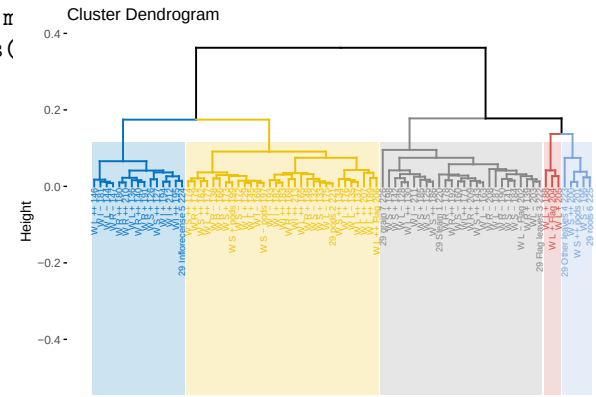
```
## Welcome! Want to learn more? See two factoextra-related
```

```
hClust1 <- hclust(dist(logdf))
plot(hClust1)
```

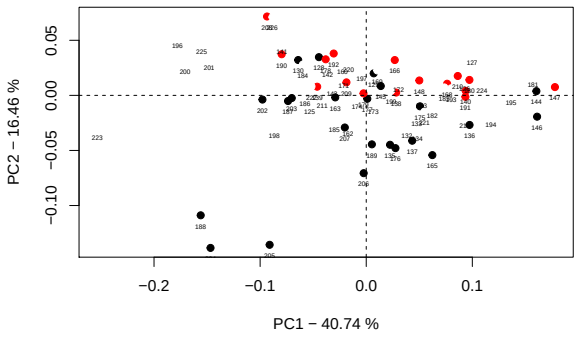



```
res.hk <-hkmeans(logdf,
                 5,
                 hc.metric =  'euclidean')

fviz_dend(res.hk, cex = 0.5, palette = "jco",
          rect = TRUE, rect_border = "jco", rect_fill = TR
```

```
## Warning: `guides(<scale> =
## FALSE)` is deprecated. Please
## use `guides(<scale> = "none")`
## instead.
```



```
colspca <- vector('character', nrow(df1))



for(i in 1:nrow(df1)){
  if( grepl(
    paste0('(?=.*',
           'L',
           ')'),
    metadata.class2$class[i],perl = T)){colspca[i] <- 'black'}else{
      if(grepl(
    paste0('(?=.*',
           'R',
           ')'),
    metadata.class2$class[i],perl = T)){colspca[i] <- 'red'}else{colspca[i] <- 'white'}
    }
}

pcaall <- prcomp(spectra.class)

vp <- (pcaall$sdev)^2

variance <- round(vp/sum(vp)*100,2)

coord <- pcaall$x

plot(coord[,1],
     coord[,2],
     col=colspca,
     xlab="PC1 - 40.74 %",
     ylab= "PC2 - 16.46 %",

     pch=19
     )
abline(v=0, h=0, lty=2)
 text(coord[,1],coord[,2], rownames(coord), cex=0.4, pos=1)
```



Leaves selection

Selection of samples

prior work (PCA applied to the same spectra, with different pre-treatment) indicates that leave samples presented more variability. thus, it is worth it to subset a table with these samples:

First, we ask R where can we find leave samples in metadata:

Again, regular expressions can be used. This time, the function `grepl` will find every element in the column `metadata$class` that has the letter L, i.e. every sample from wheat leaves.

We create, as before, an object that has the information of where the samples that match the query (leave samples) are in the `metadata` table:

```
leaves.index <- which(
  grepl(
    paste0('(?=.*',
           'L',
           ')'),
    metadata$class,perl = T))
leaves.index
```

```
##  [1]  4  5  6 11 12 13 20 21
##  [9] 22 30 31 32 42 43 44 54
## [17] 55 56 69 70 71 72 73 74
```

Since now we know exactly where the leaves samples are, we can subset the `metadata` table into a new `data.frame`:

```
metadata.leaves <- metadata[leaves.index,]
```

Silicon quantification in leaves samples

Since information about silicon content is needed, those rows that have no silicon content can be deleted.

```
which(is.na(metadata.leaves$Si))
```

```
## [1] 11
```

```
which(is.na(metadata.leaves$spectra))
```

```
## [1] 11
```

position 11 has NA both in silicon content and in index of spectra:

```
metadata.leaves$sample[11]
```

```
## [1] 164
```

Since sample 164 neither has silicon content nor spectra, it can be deleted:

```
metadata.leaves.Si <- metadata.leaves[-c(11),
```

Then, we can extract the silicon content to a vector called `leavesSi`:

```
LeavesSi <- cbind(metadata.leaves.Si$sample,m
```

And we can create a table with corrected mean spectra for this samples:

```
leavesSiSpectra <- corrected[metadata.leaves.
leavesSiSpectraRaw <- mean[metadata.leaves.Si
```

```
par(mfrow = c(1,2))

for (i in 1:length(
                    rownames(leavesSiSpectraR
                    )
    ){

  plot(as.numeric(
                  colnames(leavesSiSpectraRaw )
                  ),
    leavesSiSpectraRaw [i,],
    xlab = '',
    ylab = '',
    axes = F,
    type = 'l',
    xlim = c(1700,400),
    ylim = c(0,0.1))
  par(new = T)
}
box()
axis(1)
axis(2)
title(main = '',
      xlab = expression(
                        paste(
                       'Wave number (cm'^'-1',
                              ')'
                              )
                        ),
      ylab = 'Absorbance (a.u.)')

par(new = F)

for (i in 1:length(
                    rownames(leavesSiSpectra)
                    )
    ){

  plot(as.numeric(
                  colnames(leavesSiSpectra)
                  ),
    leavesSiSpectra[i,],
    xlab = '',
    ylab = '',
    axes = F,
    type = 'l',
    xlim = c(1700,400),
    ylim = c(0,0.1))
  par(new = T)
}
box()
axis(1)
axis(2)
title(main = '',
      xlab = expression(
                        paste(
                       'Wave number (cm'^'-1',
                              ')'
                              )
                        ),
      ylab = 'Absorbance (a.u.)')
```

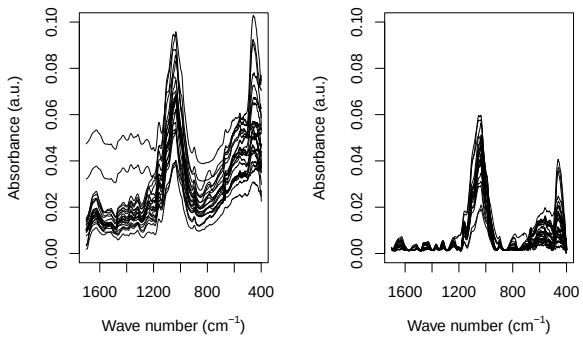


Figure 8: Mean and corrected spectra for leaves samples

Silicon concentration

we have to create a `data.frame` to use the `plsr` function available in the `pls` package.

```
leavesSiSpectra <- as.matrix(leavesSiSpectra)
SiTable <- data.frame(Si = I(metadata.leaves.Si$Si), spectra = I(leavesSiSpectra) )
newRange <-   leavesSiSpectra[,c(261:676)]
colnames( leavesSiSpectra[,c(261,676)])
```

```
## [1] "900.608" "1700.94"
```

modelling

The package `mdatools` can be used to perform the pls modelling with this data.

```
library(mdatools)
```

```
##
## Attaching package: 'mdatools'

## The following object is masked from 'package:pls':
##
##     crossval
```

```
LeavesPLS <- pls(leavesSiSpectra,
                 SiTable$Si,
                 10,

                 cv =1)
```

Then, we can check for

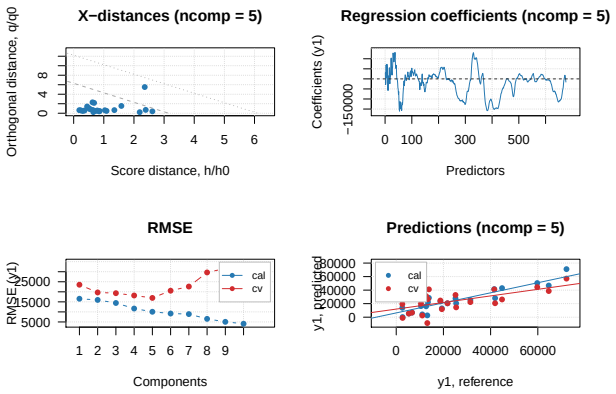
```
summary(LeavesPLS)
```

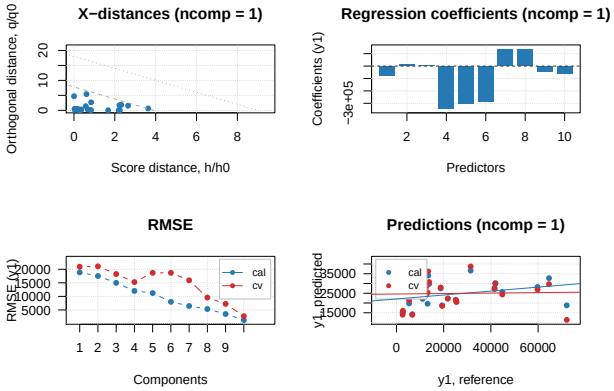
```
##
## PLS model (class pls) summary
## ------------------------------
## Info:
## Number of selected components: 5
## Cross-validation: full (leave one out)
##
##     X cumexpvar Y cumexpvar
## Cal    97.86202    74.37723
## Cv           NA          NA
##        R2     RMSE Slope
## Cal 0.744 10063.72 0.744
## Cv  0.273 16956.23 0.487
##         Bias  RPD
## Cal   0.0000 2.02
## Cv  641.3282 1.20
```

```
summary(LeavesPLS$res$cal)
```

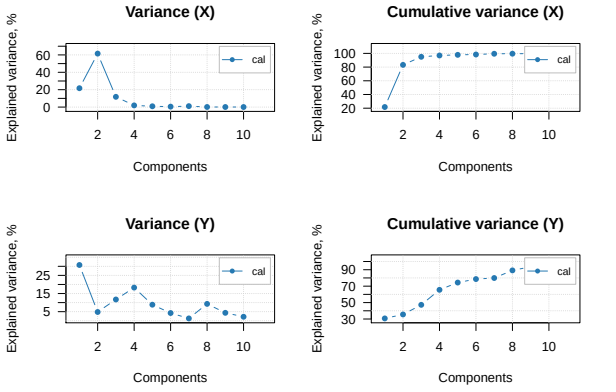
```
##
## PLS regression results (class plsres) summary
## Info: calibration results
## Number of selected components: 5
##
## Response variable y1:
##          X expvar X cumexpvar
## Comp 1     21.702      21.702
## Comp 2     61.508      83.210
## Comp 3     11.791      95.002
## Comp 4      1.932      96.933
## Comp 5      0.929      97.862
## Comp 6      0.503      98.365
## Comp 7      1.076      99.440
## Comp 8      0.096      99.536
## Comp 9      0.081      99.617
## Comp 10     0.062      99.679
##          Y expvar Y cumexpvar
## Comp 1     30.723      30.723
## Comp 2      4.827      35.550
## Comp 3     11.719      47.269
## Comp 4     18.265      65.534
## Comp 5      8.844      74.377
## Comp 6      4.250      78.627
## Comp 7      1.280      79.907
## Comp 8      9.286      89.193
## Comp 9      4.363      93.556
## Comp 10     2.196      95.753
##             R2      RMSE Slope
## Comp 1   0.307 16547.737 0.307
## Comp 2   0.356 15960.838 0.356
## Comp 3   0.473 14437.054 0.473
## Comp 4   0.655 11671.935 0.655
## Comp 5   0.744 10063.719 0.744
## Comp 6   0.786  9191.378 0.786
## Comp 7   0.799  8911.810 0.799
## Comp 8   0.892  6535.688 0.892
## Comp 9   0.936  5046.742 0.936
## Comp 10  0.958  4097.286 0.958
##          Bias  RPD
## Comp 1      0 1.23
## Comp 2      0 1.27
## Comp 3      0 1.41
## Comp 4      0 1.74
## Comp 5      0 2.02
## Comp 6      0 2.21
## Comp 7      0 2.28
## Comp 8      0 3.11
## Comp 9      0 4.03
## Comp 10     0 4.96
```

```
plot(LeavesPLS)
```





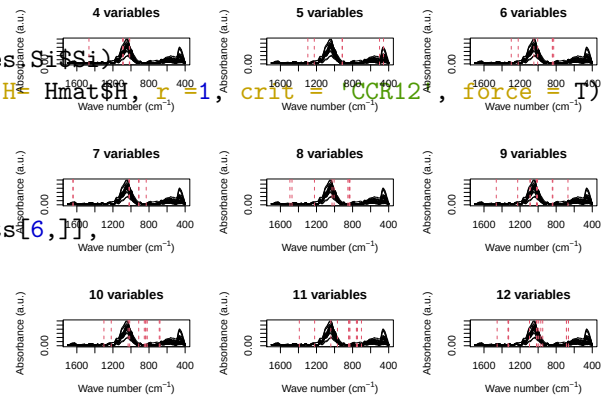

```
par(mfrow=c(2,2))
plotXVariance(LeavesPLS)
plotXCumVariance(LeavesPLS)
plotYVariance(LeavesPLS)
plotYCumVariance(LeavesPLS)
```



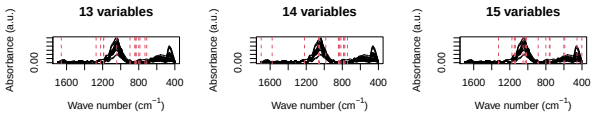

```
win.graph()
par(mfrow=c(3,3))
for(j in 1:nrow(gen$bestsets)){
for (i in 1:length(rownames(leavesSiSpectra))){
plot(as.numeric(colnames(leavesSiSpectra)),
leavesSiSpectra[i,],
xlab = '',
ylab = '',
axes = F,
type = 'l',
xlim = c(1700,400),
ylim = c(0,0.06))
par(new = T)
}
box()
axis(1)
axis(2)
title(main = paste(as.character(c(4:15)[j]),'variables'),
xlab = expression(paste('Wave number (cm'^'-1',')')),
ylab = 'Absorbance (a.u.)')
abline(v = as.numeric(colnames(leavesSiSpectra)[gen$bestse
col = 2,
lty = 2)
}
```

Variable selection

```
library(subselect)
Hmat <- lmHmat(leavesSiSpectra,metadata.leaves$Si)
gen <- genetic(Hmat$mat, kmin =5, kmax = 16, H= Hmat$H, r =1, crit = "CCR12", force = T)
```




```
library(mdatools)

LeavesPLS <- pls(leavesSiSpectra[,gen$bestsets[16,]],
                 SiTable$Si,
                 10,
                 cv =1)

plot(LeavesPLS)
```


13 variables
14 variables
15 variables
Absorbance (a.u.)
0.00
1600
1200
800
400
Wave number (cm^{-1})