Tutorial script for whole-cell MALDI-TOF analysis

Julien Textoris

June 19, 2013

Contents

Abstract

The present document is intended to be a tutorial to use R to analyze whole-cell MALDI-TOF data in the way it is described in the accompanying publication "Whole-cell MALDI-TOF mass spectrometry is an accurate and rapid method to analyze different modes of macrophage activation", published in the Journal of Visualized Experiments. The document will guide the reader to load, explore and analyze MALDI-TOF data.

The present tutorial will guide the reader through the analysis with R of MALDI-TOF data. To keep a reasonable computational time, this tutorial comes along with a reduced dataset of 48 spectra obtained from macrophages either unstimulated or stimulated with IFN gamma (to induce a M1 polarization) or IL-4 (to induce a M2 polarization).

1 Required libraries

In order to load and analyze MALDI-TOF data, you will need to install at least three specific libraries written by *Sebastian Gibb*: `readBrukerFlexData`, `MALDIquant`, and `MALDIquantForeign`.

```
## Loading libraries used in analysis
library(readBrukerFlexData)  # Allow to load Bruker raw data in R
## Contains most of pre-processing functions as well as
## alignment functions
library(MALDIquant)

library(MALDIquantForeign)
```

2 Data loading

Once you have extracted the archive, you should use this pdf tutorial, along with the script `.Rnw` that generated it, and a data folder: `data_18h`. This folder holds all raw MALDI-TOF data. These data were generated by a Bruker Autoflex II. First, you will set the directory to the current directory, so the script will find the data easily. This is done by the command `setwd()`. If you have extracted the archive in `/home/myuser/tutorialMALDI/`, you will use this as an argument for `setwd("/home/myuser/tutorialMALDI/")`. Then, raw data will automatically be imported from the data folder into an object we chose to name `spectra` by the command `importBrukerFlex()`. The `spectra` object is a list. You can find how many raw spectra it contains by executing the command `lenght(spectra)`. If you type `is(spectra[[1]])`, you'll see that it is a special `MALDIquant` object (see `MALDIquant` library doc for more details).

```
## set working directory to the root of the tree folders
## that contains raw data obtained from Bruker MALDI-TOF
setwd("/data/partage/MALDI/JoVE/data_18h/")

## Load raw data into MassSpectrum objects. All spectra are
## concatenated into a list
spectra = importBrukerFlex("./")


length(spectra)

## [1] 48
```

```
is(spectra[[1]])
```

```
## [1] "MassSpectrum"       "AbstractMassObject"
```

Sometimes, there is no spectra obtained from a given run. If you keep this "empty" spectra, it will end up with errors later. Empty spectra, if they exists, are then removed by the following code.

```
v.empty = lapply(spectra, function(y) {
    return(min(y@intensity) == max(y@intensity))
})
length(which(unlist(v.empty) == T))
```

```
## [1] 0
```

```
## If length of empty spectra is > 0, uncomment the
## following lines to remove the empty spectra spectra =
## spectra[-c(which(unlist(v.empty)==T))] length(spectra)
```

The following lines are used to compute "label" vectors for figures.

```
# sampleNames are the names of the main folders under the
# root
sampleNames = lapply(spectra, function(y) {
    y@metaData$sampleName
})
sampleNames = as.factor(unlist(sampleNames))
levels(sampleNames)
```

```
## [1] "macrophages_IFNg_18h"
"Macrophages_IL_4_18h"
## [3] "macrophages_NS_18h"
```

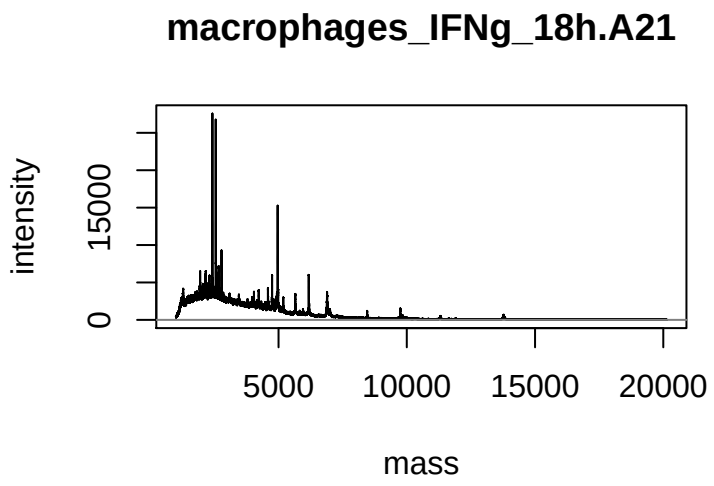
```
# all other vectors are computed from complete filename
# (with folder path)
group = lapply(spectra, function(y) {
    y@metaData$file
})
group[grep(" IFNg ", group)] = "IFNg"
group[grep("IL-4", group)] = "IL4"
group[grep("NS", group)] = "NS"
group = as.factor(unlist(group))
levels(group)
```

```
## [1] "IFNg" "IL4"  "NS"
```

3 Spectrum visualization and pre-processing

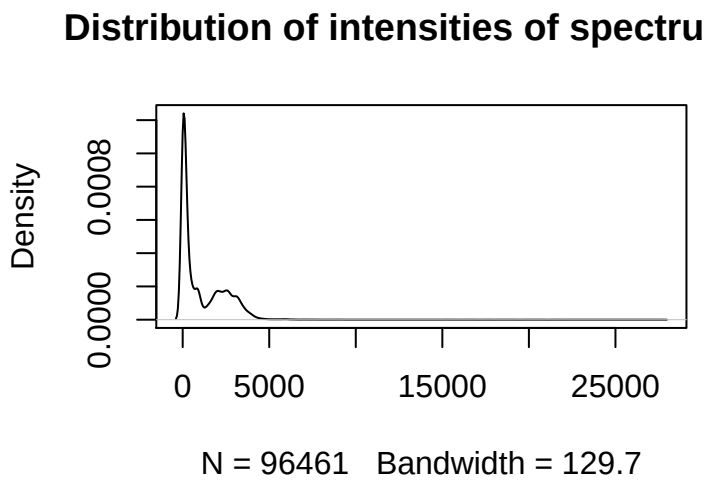
To get a graphical representation of a given spectrum, you can type `plot(spectra[[1]])`. You'll see one major peak close to 5000 m/z, and a few other peaks of much lower intensities. To improve visualization, you can transform the intensities (*i.e.* the abondance of each ion a a given mass) by applying a mathematical transformation such as log or square root. Square root transformation may also result in variance stabilization. Variance stabilization is used to overcome the dependancy of the variance from the mean. After square root transformation, the variance is nearly constant and the data are approximately Gaussian distributed. The later is important for various statistical analysis. Here, the distribution is still not Gaussian, but visualization of peaks with lower intensities is improved.

```
plot(spectra[[1]])
```

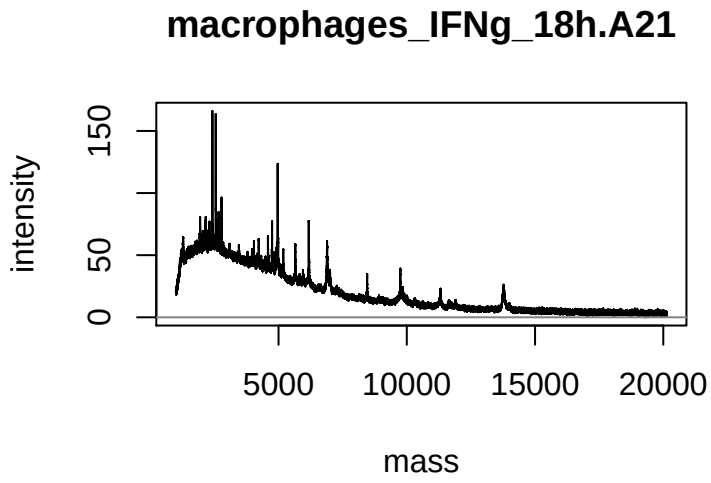


/data/partage/MALDI/JoVE/data_18h/macrophages IFNg 18h/0_A21/1/1

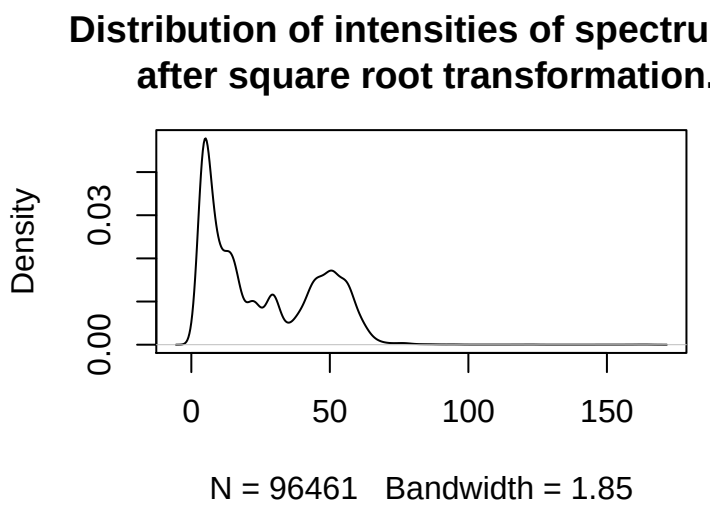
```
plot(density(intensity(spectra[[1]])),
main = "Distribution of intensities of spectrum 1.")
```




```
spectra = lapply(spectra, transformIntensity, fun = sqrt)
plot(spectra[[1]])
```




```
plot(density(intensity(spectra[[1]])),
main = "Distribution of intensities of spectrum 1\n
after square root transformation.")
```

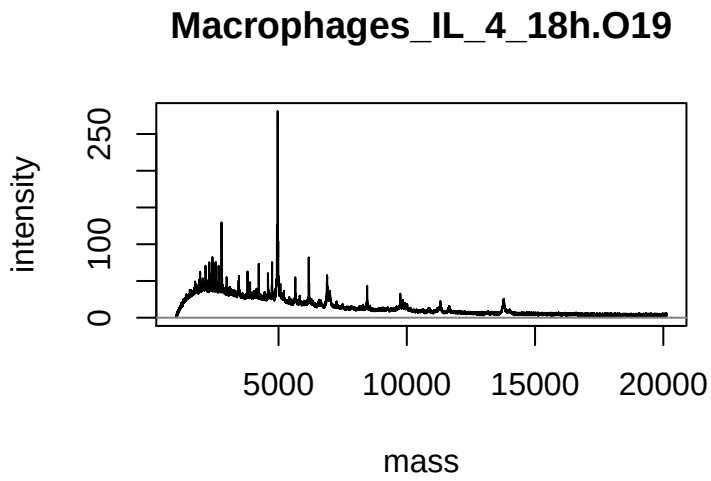


Then we apply a moving average with a window size of 5 m/z values to smooth the data.

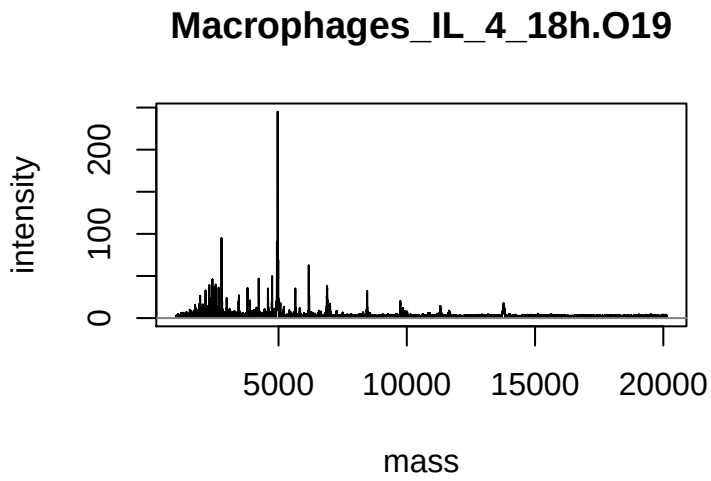
```
## This function perform a smoothing of the intensities by
## using a moving average window
movAvg = function(y) {
    return(filter(y, rep(1, 5)/5, sides = 2))
}
spectra = lapply(spectra, transformIntensity, fun = movAvg)
```

For some spectrum, there is a baseline deviation that we must remove before starting the analysis. This can be done by various algorithms (see MALDIquant documentation for details). Here we use SNIP algorithm.

```
plot(spectra[[27]])
```

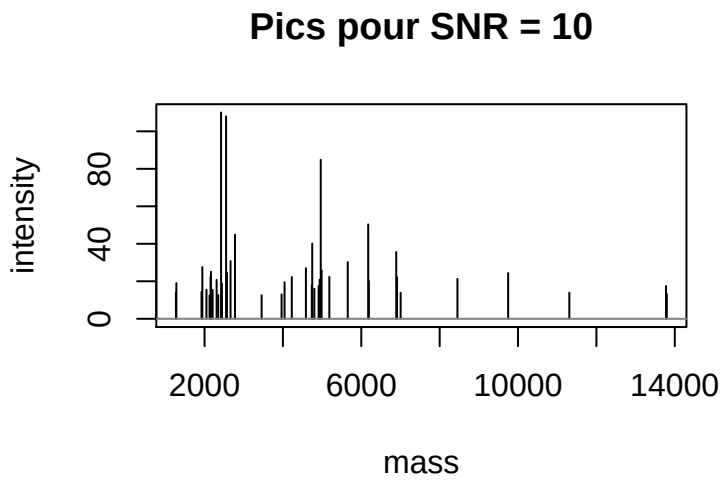



```
## Baseline correction (Best method is SNIP)
spectra = lapply(spectra, removeBaseline, method = "SNIP")

plot(spectra[[27]])
```



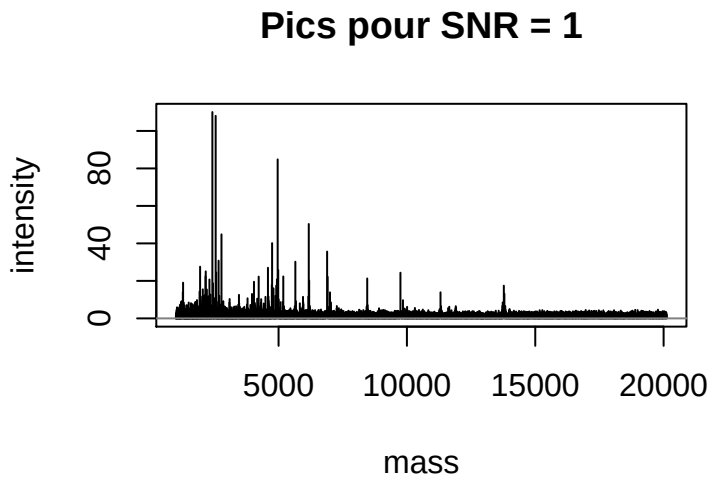
Then, we want to detect peaks in a given spectrum. For this, we have to choose a signal to noise ratio to discriminate between peaks and noise background. A SNR of 6 is usually a good compromise (between the number of peaks obtained and the specificity of these peaks). Peaks are stored in a second object. The plot function will now display only peaks. Many visualization possibilities are available within the MALDIquant library, for example to label the peaks with their respective m/z value.

```
pk = lapply(spectra, detectPeaks, SNR = 10, halfWindowSize = 20)

plot(pk[[1]], main = "Pics pour SNR = 10")
```



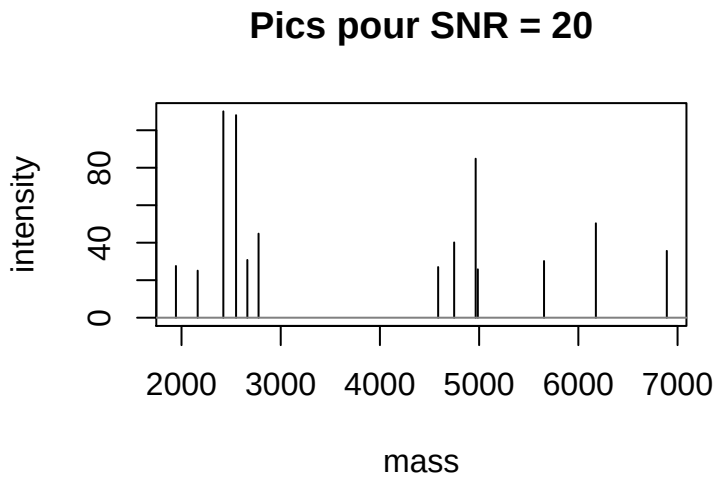
/data/partage/MALDI/JoVE/data_18h/macrophages IFNg 18h/0_A21/1/1

```
plot(lapply(spectra, detectPeaks, SNR = 1, halfWindowSize = 20)[[1]],
    main = "Pics pour SNR = 1")
```



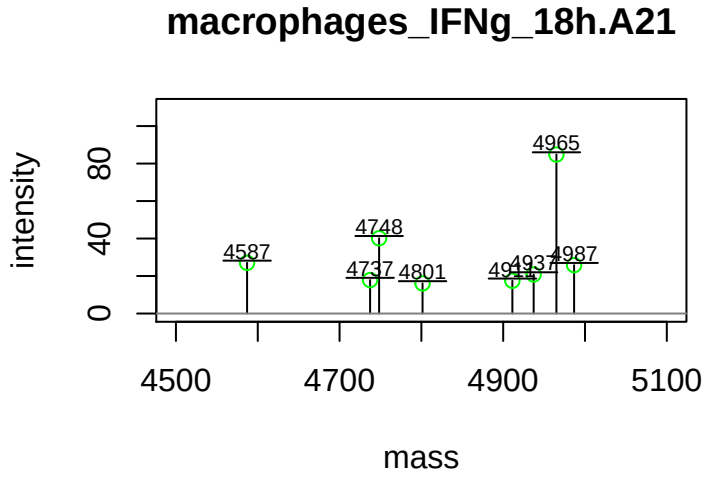
/data/partage/MALDI/JoVE/data_18h/macrophages IFNg 18h/0_A21/1/1

```
plot(lapply(spectra, detectPeaks, SNR = 20, halfWindowSize = 20)[[1]],
    main = "Pics pour SNR = 20")
```



/data/partage/MALDI/JoVE/data_18h/macrophages IFNg 18h/0_A21/1/1

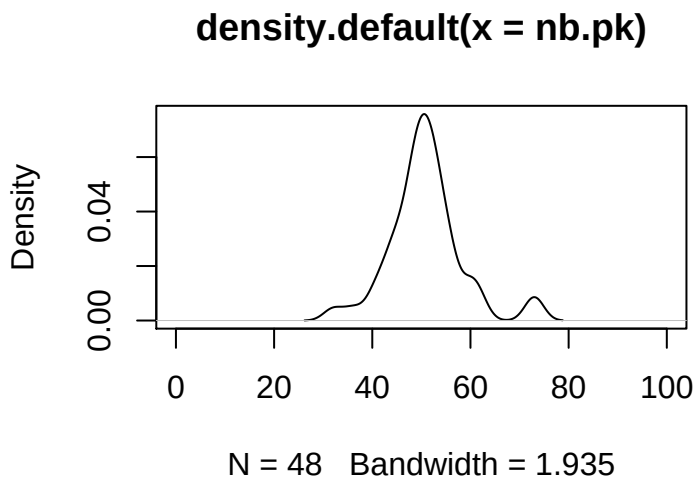
```
## To label peaks
plot(pk[[1]], xlim = c(4500, 5100))
points(pk[[1]], col = "green")
labelPeaks(pk[[1]])
```



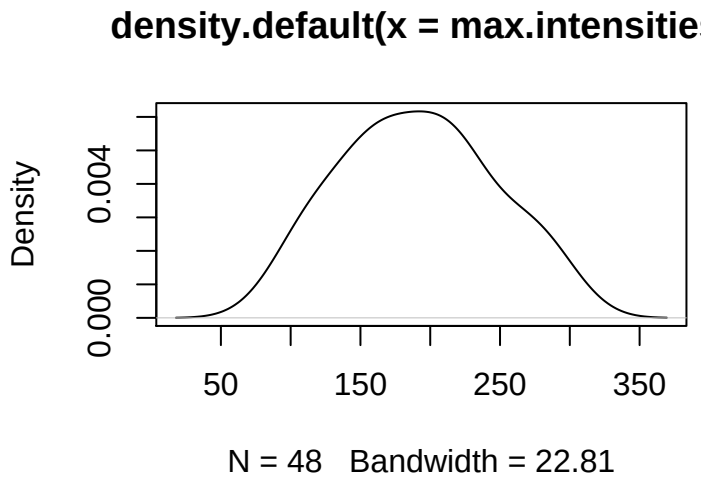
/data/partage/MALDI/JoVE/data_18h/macrophages IFNg 18h/0_A21/1/1

Here, we use peak detection to automatically check the quality of the samples. We want a given sample to have at least 50 peaks, and we want the maximal intensity of the main peak to be at least above 50. To determine these values, we plot the distribution of the number of peaks in each spectrum, and the distribution of maximal intensities.

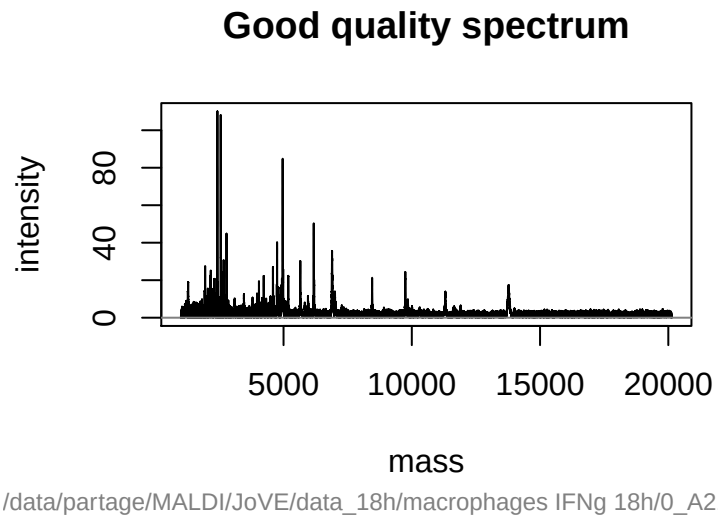
```
## The following steps check if spectra quality is
## sufficient for the analysis. We use two criteria : a
## sufficient number of peaks detected with a sufficient
## SNR (usually 50) the maximal intensity of all peaks is
## at least i = 40

nb.pk = unlist(lapply(pk, function(y) {
    length(y@mass)
}))
plot(density(nb.pk), xlim = c(0, 100))
```




```
length(which(nb.pk < 40))

## [1] 2


max.intensities = unlist(lapply(spectra, function(y) {
    as.numeric(y@intensity)[order(as.numeric(y@intensity), decreasing = T)[1]]
}))
plot(density(max.intensities))
```

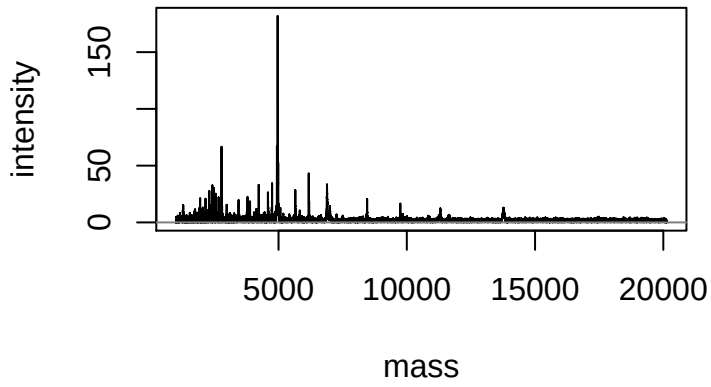



```
length(which(max.intensities < 50 | nb.pk < 40))

## [1] 2

# 2 spectrum will be removed for poor quality, this is how
# they look like :
which(max.intensities < 50 | nb.pk < 40)

## [1] 4 8


# good quality spectrum
plot(spectra[[1]], main = "Good quality spectrum")
```



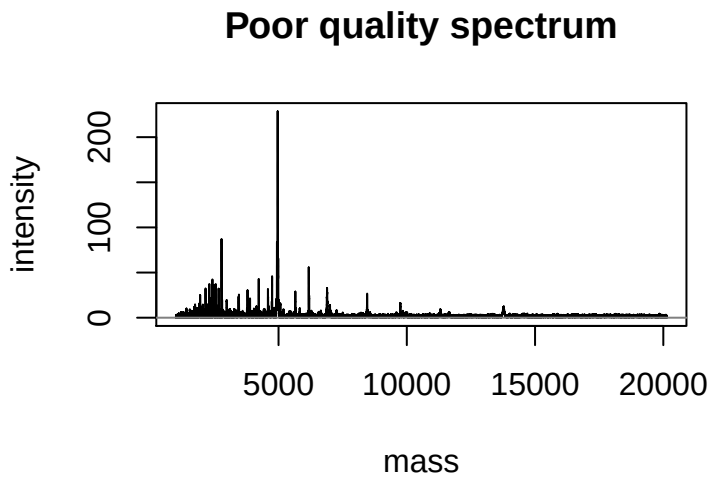
/data/partage/MALDI/JoVE/data_18h/macrophages IFNg 18h/0_A21/1/1

```
plot(spectra[[20]], main = "Poor quality spectrum")
```



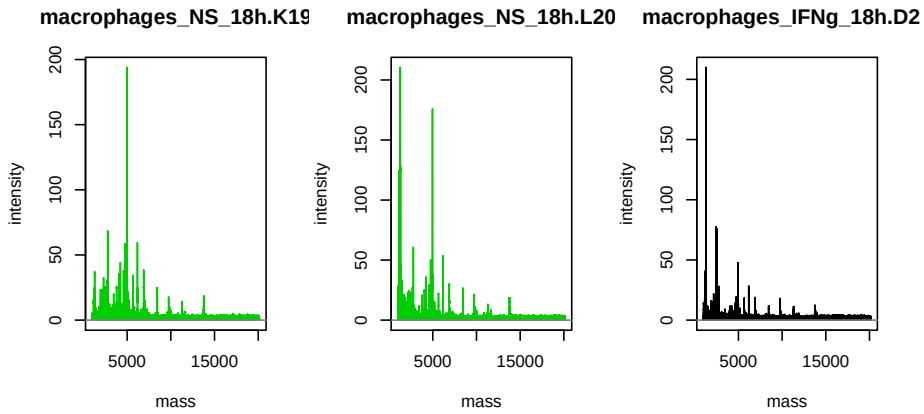
/data/partage/MALDI/JoVE/data_18h/Macrophages IL−4 18h/0_M20/1/1

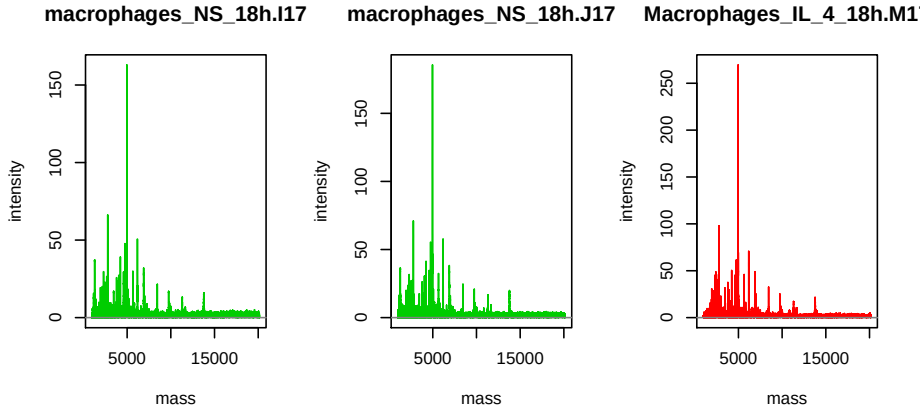
```
plot(spectra[[24]], main = "Poor quality spectrum")
```




```
## If you want to eliminate some spectra, uncomment the
## following lines
spectra = spectra[-c(which(max.intensities < 50 | nb.pk < 40))]
sampleNames = as.factor(as.vector(sampleNames[-c(which(max.intensities <
    50 | nb.pk < 40))]))
group = as.factor(as.vector(group[-c(which(max.intensities <
    50 | nb.pk < 40))]))
```

```
## Just to check visually some spectra sampled into the
## whole dataset
par(mfrow = c(2, 3))
ind = sample(1:length(spectra), 6)
for (i in ind) {
    plot(spectra[[i]], col = group[i])
}
```





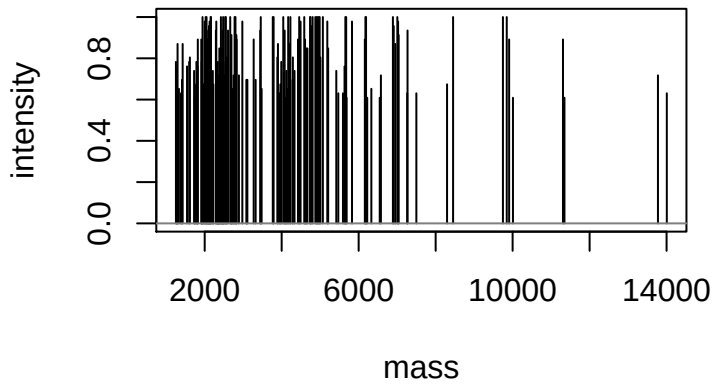
Then, we will normalize the dataset by "total ion current".

```
## Calibrate/Normalize intensity values by 'total ion
## current'
spectra <- standardizeTotalIonCurrent(spectra)
```

Finally, as we will compare spectrum for presence/absence of a given peak, we must first align the spectra.

```
## Alignment.  To perform alignement, we first create a
## reference spectra for whole dataset with a low SNR, and
## a low minimal frequency This reference spectra is a
## MassPeak object, and will contain a list of peaks used
## to aligned all spectra together
pk = lapply(spectra, detectPeaks, SNR = 4, halfWindowSize = 20)

refPeaks <- referencePeaks(pk, "strict", 0.6, 0.002)

## Check that the reference spectra contains sufficient
## peaks and that these peaks are distributed across the
```

```
## whole range of m/z values to obtain a good alignment
par(mfrow = c(1, 1))
plot(refPeaks)
```




```
## If necessary, recompute PeakLists with the SNR wanted
## for the analysis
pk = lapply(spectra, detectPeaks, SNR = 6, halfWindowSize = 20)

## This function from MALDIquant computes all warping
## functions for the alignment
warpingFunctions <- determineWarpingFunctions(pk, reference = refPeaks)

pk.aligned <- warpMassPeaks(pk, warpingFunctions)
sp.aligned = warpMassSpectra(spectra, warpingFunctions)


## As aligned spectra may have different minimum and
## maximum masses we computed the max of the minimal values
## and the min of maximal values This gives us the range
## that is common for all aligned spectra
mins = unlist(lapply(sp.aligned, function(y) {
    min(y@mass)
}))
maxs = unlist(lapply(sp.aligned, function(y) {
    max(y@mass)
}))
lim1 = round(max(mins, na.rm = T), 0) + 1
lim2 = round(min(maxs, na.rm = T), 0) - 1
lim1
```

```
## [1] 1002

lim2

## [1] 20132
```

4 Analysis and comparison of spectra

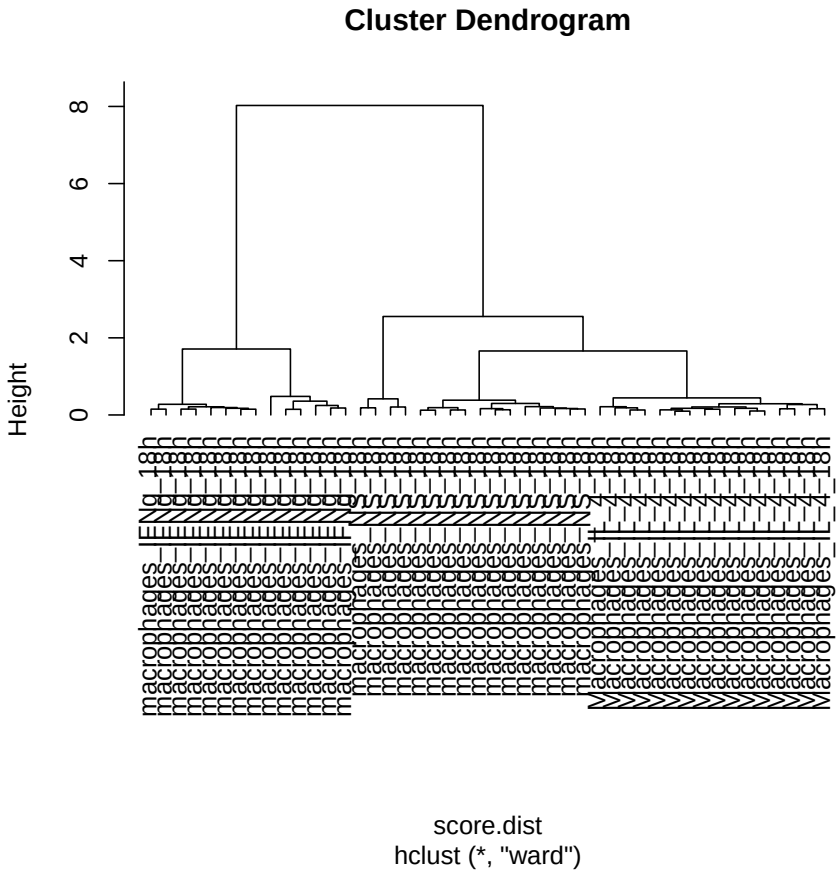
See below the functions that we will use to compute a score to compare spectra. The idea of that is to use two criteria : the presence / absence of a given peak in the two spectra that are compared (as it is a boolean criteria, we will compute the Jaccard Index), and then a correlation coefficient to adjust for the intensities of a given peak in both spectra. The second criteria is less important and may be avoided.

```
### Some functions we will need to compute scores (modified
### by S. Gibb)
as.binary.matrix <- function(x) {
    return(ifelse(is.na(x), 0, 1))
}

jaccard <- function(x) {
    n11 <- tcrossprod(x)
    n01 <- tcrossprod(1 - x, x)
    n10 <- tcrossprod(x, 1 - x)
    # return(n11/(n01+n10+n11))
    return(2 * n11/(n01 + n10 + 2 * n11))
}

## This function computes the 'Score'.  This version
## computes a score based on the Jaccard indice and a
## pearson correlation coefficient based on the intensities
## of common peaks between the two spectra : S = jac*cor ;
## jac is Jaccard indice (between 0-1) and cor is Pearson
## correlation coefficient (between 0-1).
computeModJacScoreOnPeaks <- function(p, tolerance = 0.002, range = c(0,
    20000)) {
    trimmedPeaks <- trim(p, range[1], range[2])
    binnedPeaks <- binPeaks(trimmedPeaks, method = "relaxed",
        tolerance = tolerance)
    ## remove peaks occuring only once
    filteredPeaks <- filterPeaks(binnedPeaks, minFrequency = 2/length(binnedPeaks))
    ## to run a groupwise filtering use filteredPeaks <-
    ## filterPeaks(binnedPeaks, labels=group, minFrequency=2/3)

    peakMatrix <- intensityMatrix(filteredPeaks)

    ja <- jaccard(as.binary.matrix(peakMatrix))
```

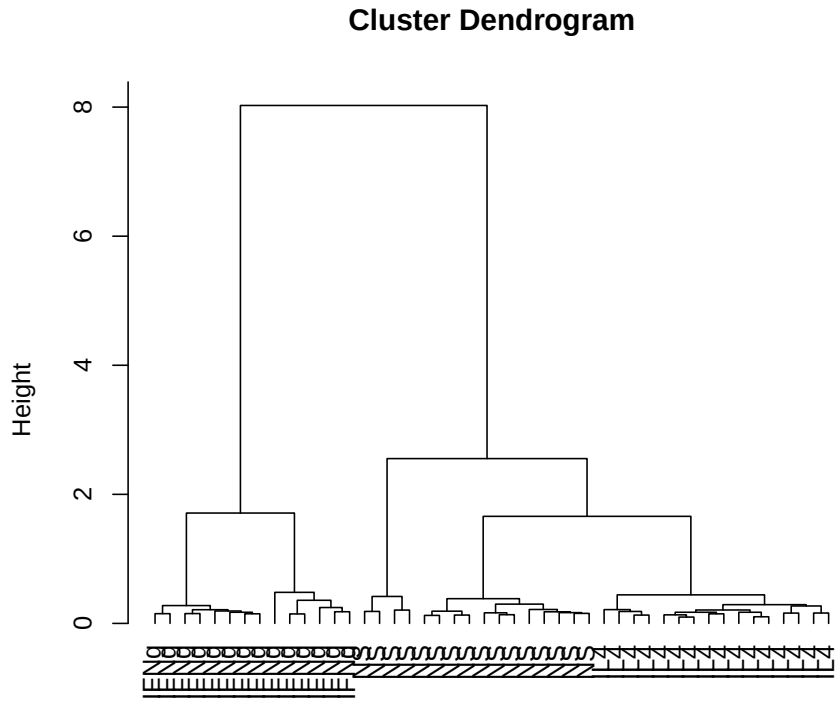
```
    co <- cor(t(peakMatrix), method = "pearson", use = "pairwise.complete.obs")

    return(ja * co)
}
```

In order to draw a virtual gelview of the dataset, the code below will compute a matrix with a given resolution for m/z, and for each value, we will keep a median intensity value. We will end with a numerical data matrix which will be plot as a heatmap. In the heatmap, m/z value order will be conserved, but samples will be reorganized according to a clustering based on the score that compare the spectra.

```
## To build a virtual gel-view representation of the
## spectra, we first summarize the dataset by keeping only
## one value for each m/z value We take the median
## intensity of the n values in a given m/z value
system.time({
    m3 <- trim(sp.aligned, lim1, lim2)

    m3 <- lapply(m3, function(x) {
        m <- round(mass(x))
        i <- unlist(lapply(split(intensity(x), m), median))
        x@intensity <- i
        x@mass <- unique(m)
        return(x)
    })

    m3 <- intensityMatrix(m3)
})

## Check that you have 1 on the diagonal
score.mat <- computeModJacScoreOnPeaks(pk.aligned)

# Computes distance matrix to draw dendrogram
score.dist = as.dist(1 - score.mat)

# Plot dendrogram, label leafs with various vectors
# (sampleNames, group)
plot(hclust(score.dist, method = "ward"), labels = sampleNames,
    hang = -1)
```




```
plot(hclust(score.dist, method = "ward"), labels = group, hang = -1)
```



Of note, it is important to understand that clustering implies an agglomerative method to display as a dendrogram the distance matrix. The distances presented in the dendrogram are not the true distances of the initial matrix (`score.dist`). They may differ depending on the agglomerative function used (textttaverage / complete / ward ...). Here, we chose to use the Ward algorithm. It favorize clusters of equal number of samples by variance stabilization. However, the heigh displayed along the dendrogram is not comprised between 0 and 1, because it does not represent the distance, but the Ward's crierion for agglomeration. To illustrate the difference between the intra-class distance (which is low) and the inter class distance (which is high), we proide below a chunk of code which comptes the mean distance of each class.

```
# Mean intra-class distance is low
mean(as.vector(as.matrix(score.dist)[which(group == "IFNg"),
    which(group == "IFNg")]))
mean(as.vector(as.matrix(score.dist)[which(group == "IL4"), which(group ==
    "IL4")]))
mean(as.vector(as.matrix(score.dist)[which(group == "NS"), which(group ==
    "NS")]))
# Mean inter-class distance is high
mean(as.vector(as.matrix(score.dist)[which(group == "IFNg"),
```

```
    which(group == "IL4")]))
mean(as.vector(as.matrix(score.dist)[which(group == "IFNg"),
    which(group == "NS")]))
mean(as.vector(as.matrix(score.dist)[which(group == "NS"), which(group ==
    "IL4")]))
```

```
## To draw the gel-view representation : As score.dist is
## used to compute the spectra dendrogram, you have to
## compute score.mat and score.dist first
heatmap(as.matrix(t(m3)), Rowv = NA, scale = "col",
Colv = as.dendrogram(hclust(score.dist,
method = "ward")), col = colorRampPalette(c("white",
"blue", "blue"))(1000), labCol = group, labRow = FALSE, ylab =
"Mass m(Da)/z", xlab = "Samples")
```