Preprocessing of untargeted (LC-MS) metabolomics data

Johannes Rainer [1]

[1]Institute for Biomedicine, Eurac Research, Bolzano, Italy

July 2019

Contents

1 Abstract

In this document we discuss mass spectrometry (MS) data handling and access using Bioconductor's *MSnbase* package (Gatto and Lilley 2012) and walk through the preprocessing of an (untargeted) LC-MS toy data set using the *xcms* package (Smith et al. 2006). The preprocessing comprises chromatographic peak detection, sample alignment and peak correspondence. Particular emphasis is given on defining data-set dependent values for the most important settings of popular preprocessing methods.

2 Introduction

Preprocessing of untargeted metabolomics data is the first step in the analysis of GC/LS-MS based untargeted metabolomics experiments. The aim of the preprocessing is the quantification of signals from ion species measured in a sample and matching of these entities across samples within an experiment. The resulting two-dimensional matrix with feature abundances in all samples can then be further processed, e.g. by normalizing the data to remove sampling differences, batch effects or injection order-dependent signal drifts. Another crucial step in untargeted metabolomics analysis is the annotation of the (m/z-retention time) features to the actual ions and metabolites they represent. Note that data normalization and annotation are not covered in this document.

People familiar with the concepts of mass spectrometry or LC-MS data analysis may jump directly to the next section.

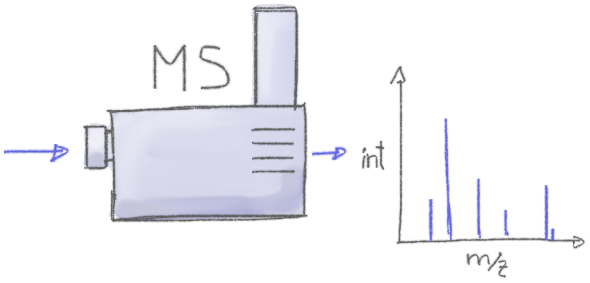
2.1 Prerequisites

The analysis in this document requires an R version >= 3.6.0 and recent versions of the `MSnbase` and `xcms` packages.

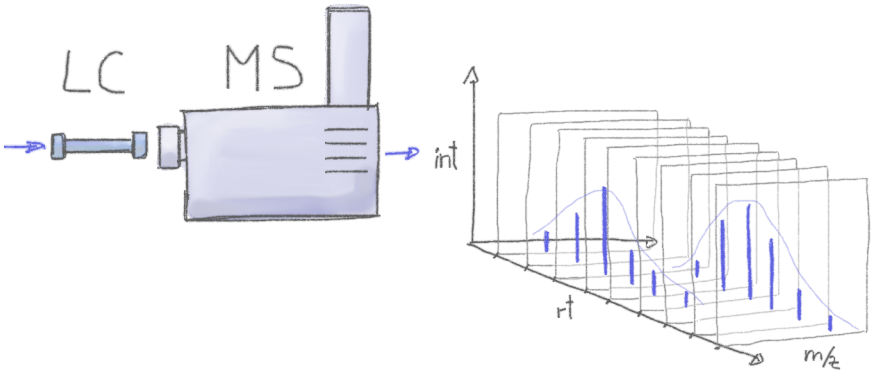
```
library(BiocManager)
BiocManager::install(c("xcms",
                       "MSnbase",
                       "msdata",
                       "magrittr",
                       "png"))
```

2.2 Mass spectrometry

Mass spectrometry allows to measure abundances of charged molecules (ions) in a sample. Abundances are determined as ion counts for a specific mass-to-charge ratio m/z. The measured signal is represented as a spectrum: intensities along m/z.



Many ions have the same or a very similar m/z making it difficult or impossible to discriminate them. MS is thus frequently coupled with a second technology to separate analytes based on other properties than their mass (or rather m/z). Common choices are gas chromatography (GC) or liquid chromatography (LC). Such an e.g. LC-MS setup performs scans at discrete time points resulting in a set of spectra for a given sample, with allows to separate compounds (ions) on m/z and on retention time dimension.



In such GC/LC-MS based untargeted metabolomics experiments the data is analyzed along the retention time dimension and *chromatographic* peaks (which are supposed to represent the signal from a ion species) are identified and quantified.

2.3 Definitions and common naming convention

Naming conventions and terms used in this document are:

- chromatographic peak: peak containing the signal from an ion in retention time dimension (different from a *mass* peak that represents the signal along the m/z dimension within a spectrum).
- chromatographic peak detection: process in which chromatographic peaks are identified within each file.
- alignment: process that adjusts for retention time differences between measurements/files.

- correspondence: grouping of chromatographic peaks (presumably from the same ion) across files.
- feature: chromatographic peaks grouped across files.

3 Workflow: preprocessing of untargeted metabolomics data

This workflow describes the basic data handling (I/O) of mass spectrometry data using the *MSnbase* package, and the LC/GC-MS data preprocessing using *xcms*. It showcases the new functionality and user interface functions of `xcms`, that re-use functionality from the `MSnbase` package. The first part of the workflow is focused on data import, access and visualization which is followed by the description of a simple data centroiding approach and finally the `xcms`-based LC-MS data preprocessing that comprises chromatographic peak detection, alignment and correspondence. The workflow does not cover data normalization procedures, compound identification and differential abundance analysis.

3.1 Data import and representation

The example data set of this workflow consists of two files in mzML format with signals from pooled human serum samples measured with a ultra high performance liquid chromatography (UHPLC) system (Agilent 1290) coupled with a Q-TOF MS (TripleTOF 5600+ AB Sciex) instrument. Chromatographic separation was based on hydrophilic interaction liquid chromatography (HILIC) separating metabolites depending on their polarity. The setup thus allows to measure small polar compounds and hence metabolites from the main metabolic pathways. The input files contain all signals measured by the MS instrument (so called *profile mode* data). To reduce file sizes, the data set was restricted to an m/z range from 105 to 134 and retention times from 0 to 260 seconds.

In the code block below we first load all required libraries and define the location of the mzML files, which are part of the `msdata` package. We also define a `data.frame` describing the samples/experiment and pass this to the `readMSData` function which imports the data. The option `mode = "onDisk"` tells the function to read only general metadata into memory. The m/z and intensity values are not imported but retrieved from the original files on demand, which enables also analyses of very large experiments.

```
library(xcms)
library(magrittr)

#' Define the file names.
fls <- dir(system.file("sciex", package = "msdata"), full.names = TRUE)

#' Define a data.frame with additional information on the files.
pd <- data.frame(file = basename(fls),
                 injection_idx = c(1, 19),
                 sample = c("POOL_1", "POOL_2"),
                 group = "POOL")
data <- readMSData(fls, pdata = new("NAnnotatedDataFrame", pd),
                   mode = "onDisk")
```

Next we set up parallel processing. This ensures that all required cores are registered and available from the beginning of the analysis. All data access and analysis functions of `xcms` and `MSnbase` are parallelized on a per-file basis and will use this setup by default.

```
#' Set up parallel processing using 2 cores
if (.Platform$OS.type == "unix") {
    register(bpstart(MulticoreParam(2)))
} else {
    register(bpstart(SnowParam(2)))
}
```

The MS experiment data is now represented as an `OnDiskMSnExp` object. Phenotype information can be retrieved with the `pData` function, single columns in the phenotype table using `$`. Below we access sample descriptions.

```
#' Access phenotype information
pData(data)
##                                file injection_idx sample group
## 1 20171016_POOL_POS_1_105-134.mzML             1 POOL_1  POOL
## 2 20171016_POOL_POS_3_105-134.mzML            19 POOL_2  POOL

#' Or individual columns directly using the $ operator
data$injection_idx
## [1]  1 19
```

General information on each spectrum in the experiment can be accessed with the `fData` function, that returns a `data.frame` each row with metadata information for one spectrum.

```
#' Access spectrum header information
head(fData(data), n = 3)
##         fileIdx spIdx smoothed seqNum acquisitionNum msLevel polarity
## F1.S001       1     1       NA      1              1       1        1
## F1.S002       1     2       NA      2              2       1        1
## F1.S003       1     3       NA      3              3       1        1
##         originalPeaksCount totIonCurrent retentionTime basePeakMZ
## F1.S001                578        898185         0.280   124.0860
## F1.S002               1529       1037012         0.559   124.0859
## F1.S003               1600       1094971         0.838   124.0859
##         basePeakIntensity collisionEnergy ionisationEnergy    lowMZ   highMZ
## F1.S001            154089               0                0 105.0435 133.9837
## F1.S002            182690               0                0 105.0275 133.9836
## F1.S003            196650               0                0 105.0376 133.9902
##         precursorScanNum precursorMZ precursorCharge precursorIntensity
## F1.S001                0           0               0                  0
## F1.S002                0           0               0                  0
## F1.S003                0           0               0                  0
##         mergedScan mergedResultScanNum mergedResultStartScanNum
## F1.S001          0                   0                        0
## F1.S002          0                   0                        0
## F1.S003          0                   0                        0
##         mergedResultEndScanNum injectionTime filterString
## F1.S001                      0             0         <NA>
```

```
## F1.S002                     0            0         <NA>
## F1.S003                     0            0         <NA>
##                                    spectrumId centroided
## F1.S001 sample=1 period=1 cycle=1 experiment=1      FALSE
## F1.S002 sample=1 period=1 cycle=2 experiment=1      FALSE
## F1.S003 sample=1 period=1 cycle=3 experiment=1      FALSE
##         ionMobilityDriftTime isolationWindowTargetMZ
## F1.S001                   NA                     NA
## F1.S002                   NA                     NA
## F1.S003                   NA                     NA
##         isolationWindowLowerOffset isolationWindowUpperOffset spectrum
## F1.S001                         NA                         NA        1
## F1.S002                         NA                         NA        2
## F1.S003                         NA                         NA        3
```

3.2 Basic data access and visualization

The MS data in an `OnDiskMSnExp` object is organized by spectrum (similar to *mzML* files), with `Spectrum` objects used as containers for the m/z and intensity values. General spectrum information can be retrieved using the `msLevel`, `centroided`, `rtime` or `polarity` functions that return the respective value for all spectra from all files. Here, the `fromFile` function can be helpful which returns for each spectrum the index of the file in which it was measured. This is shown in the code block below.

```
#' Get the retention time
head(rtime(data))
## F1.S001 F1.S002 F1.S003 F1.S004 F1.S005 F1.S006
##   0.280   0.559   0.838   1.117   1.396   1.675

#' Get the retention times splitted by file.
rts <- split(rtime(data), fromFile(data))

#' The result is a list of length 2. The number of spectra per file can
#' then be determined with
lengths(rts)
##   1   2
## 931 931
```

The `spectra` function can be used to retrieve the list of all spectra (from all files). This will load the full data from all raw files, which can take, depending on the size of the files and number of spectra, relatively long time and requires, depending on the experiment, a considerable amount of memory. In most cases we will however work with sub-sets of the data, and retrieving that can, in the case of indexed mzML, mzXML and CDF files, be very fast. Data objects can be subsetted using the filter functions: `filterFile`, `filterRtime`, `filterMz` or `filterMsLevel` that filter the data by file, retention time range, m/z range or MS level. To illustrate this we retrieve below all spectra measured between 180 and 181 seconds. These contain the signal from all compounds that eluted from the LC in that time window. Note that we use the pipe operator `%>%` from the `magrittr` package for better readability.

```
#' Get all spectra measured between 180 and 181 seconds
#' Use %>% to avoid nested function calls
sps <- data %>%
    filterRt(rt = c(180, 181)) %>%
    spectra
```

The result is a `list` of `Spectrum` objects. Below we determine the number of spectra we have got.

```
#' How many spectra?
length(sps)
## [1] 6
```

We can use the `fromFile` function to determine from which file/sample each spectrum is.

```
#' From which file?
sapply(sps, fromFile)
## F1.S646 F1.S647 F1.S648 F2.S646 F2.S647 F2.S648
##       1       1       1       2       2       2
```

We have thus 3 spectra per file. Next we plot the data from the last spectrum (i.e. the 3rd spectrum in the present retention time window from the second file).

```
plot(sps[[6]])
```

We can immediately spot several mass peaks in the spectrum, with the largest one at a m/z of about 130 and the second largest at about 106, which matches the expected mass to charge ratio for the [M+H]+ ion (adduct) of Serine. Serine in its natural state is not charged and can therefore not be measured directly with a MS instrument. Uncharged compounds, such as Serine, have thus to be first ionized to create charged molecules (e.g. using electrospray-ionization as in the present data set). Such ionization would create [M+H]+ ions of Serine, i.e. molecules that consist of Serine plus a hydrogen resulting in single charged ions (with a mass equal to sum of the masses of Serine and hydrogen).

MS data is generally organized by spectrum, but in LC-MS experiments we analyze the data along the retention time axis and hence orthogonally to the spectral data representation. To extract data along the retention time dimension we can use the `chromatogram` function. This function aggregates intensities for each scan/retention time along the m/z axis (i.e. within each spectrum) and returns the retention time - intensity duplets in a `Chromatogram` object, one per file. The `Chromatogram` object supports, similar to the `Spectrum` object, the `rtime` and `intensity` functions to access the respective data. Below we use the `chromatogram` function to extract the total ion chromatogram (TIC) for each file and plot it. The TIC represents the sum of all measured signals per spectrum (i.e. per discrete time point) and provides thus a general information about compound separation by liquid chromatography.

```
#' Get chromatographic data (TIC) for an m/z slice
chr <- chromatogram(data)
chr
## Chromatograms with 1 row and 2 columns
##                   1             2
##      <Chromatogram> <Chromatogram>
## [1,]    length: 931    length: 931
## phenoData with 4 variables
```

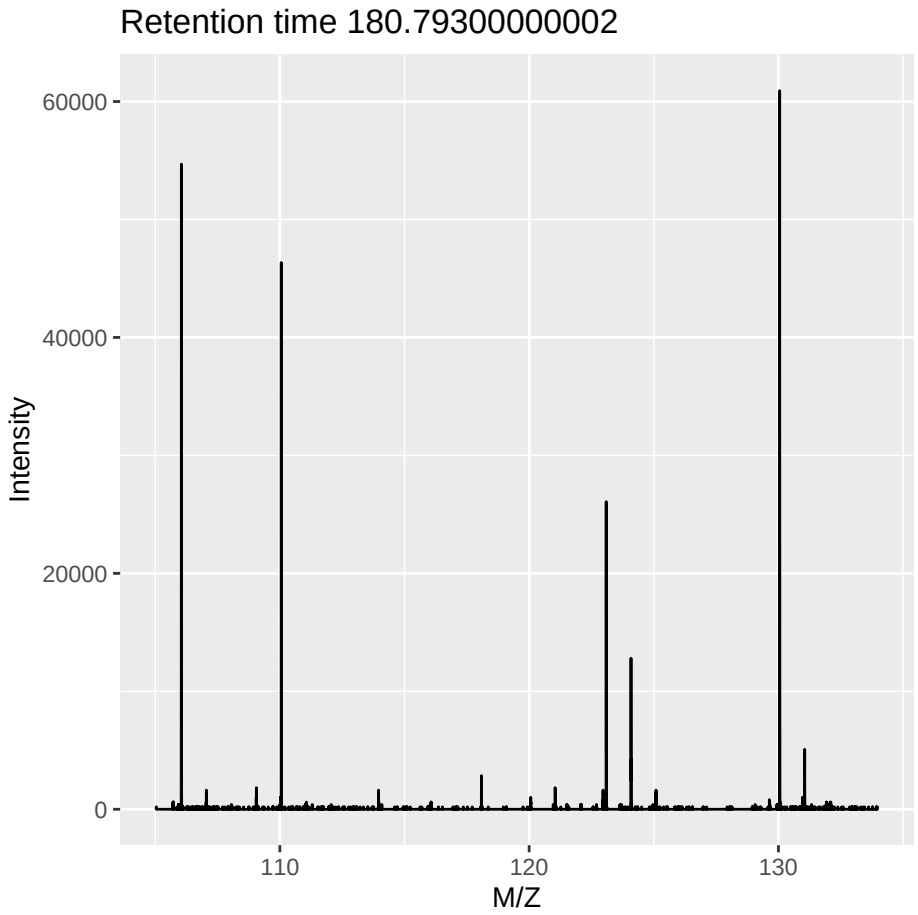


Figure 1: Spectrum at a retention time of about 180 seconds

```
## featureData with 1 variables

#' Plot the tic
plot(chr)
```

The object returned by the `chromatogram` function arranges the individual `Chromatogram` objects in a two-dimensional array, columns being samples (files) and rows data slices. Below we extract the (total ion) intensities from the TIC of the first file.

```
ints <- intensity(chr[1, 1])
head(ints)
## F1.S001 F1.S002 F1.S003 F1.S004 F1.S005 F1.S006
##  898185 1037012 1094971 1135015 1106233 1181489
```

The object contains also all phenotype information from the original `data` variable. This can be accessed in the same way than for `OnDiskMSnExp` objects (or most other data objects in Bioconductor).

```
#' Access the full phenotype data
pData(chr)
```

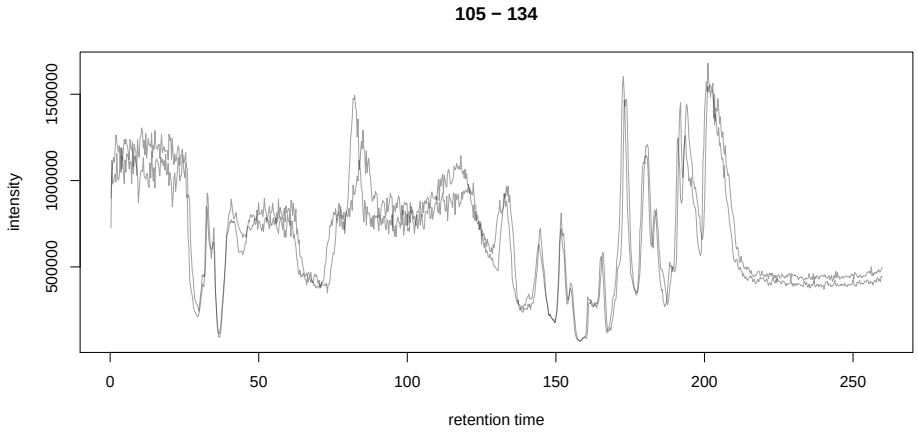


Figure 2: Total ion chromatogram

```
##                                file injection_idx sample group
## 1 20171016_POOL_POS_1_105-134.mzML             1 POOL_1  POOL
## 2 20171016_POOL_POS_3_105-134.mzML            19 POOL_2  POOL
```

Depending on the parameter `aggregationFun`, the function can produce total ion chromatograms (TIC, sum of the signal within a spectrum) with `aggregationFun = "sum"`, or base peak chromatograms (BPC, maximum signal per spectrum) with `aggregationFun = "max"`. The `chromatogram` function can also be used to generate extracted ion chromatograms (EIC), which contain the signal from a specific m/z range and retention time window, presumably representing the signal of a single ion species. Below we extract and plot the ion chromatogram for Serine after first filtering the data object to the retention time window and m/z range containing signal for this compound.

```
#' Extract and plot the XIC for Serine
data %>%
    filterRt(rt = c(175, 189)) %>%
    filterMz(mz = c(106.02, 106.07)) %>%
    chromatogram(aggregationFun = "max") %>%
    plot()
```

The area of such a chromatographic peak is supposed to be proportional to the amount of the corresponding ion in the respective sample and identification and quantification of such peaks is one of the goals of the GC/LC-MS data preprocessing.

3.3 Centroiding of profile MS data

MS instruments allow to export data in profile or centroid mode. Profile data contains the signal for all discrete m/z values (and retention times) for which the instrument collected data (Smith et al. 2014). MS instruments continuously sample and record signals and a mass peak for a single ion in one spectrum will thus consist of a multiple intensities at discrete m/z values. Centroiding is the process to reduce these mass peaks to a single representative signal, the centroid. This results in much smaller file sizes, without loosing too much information. `xcms`, specifically the *centWave* chromatographic peak detection algorithm, was designed for

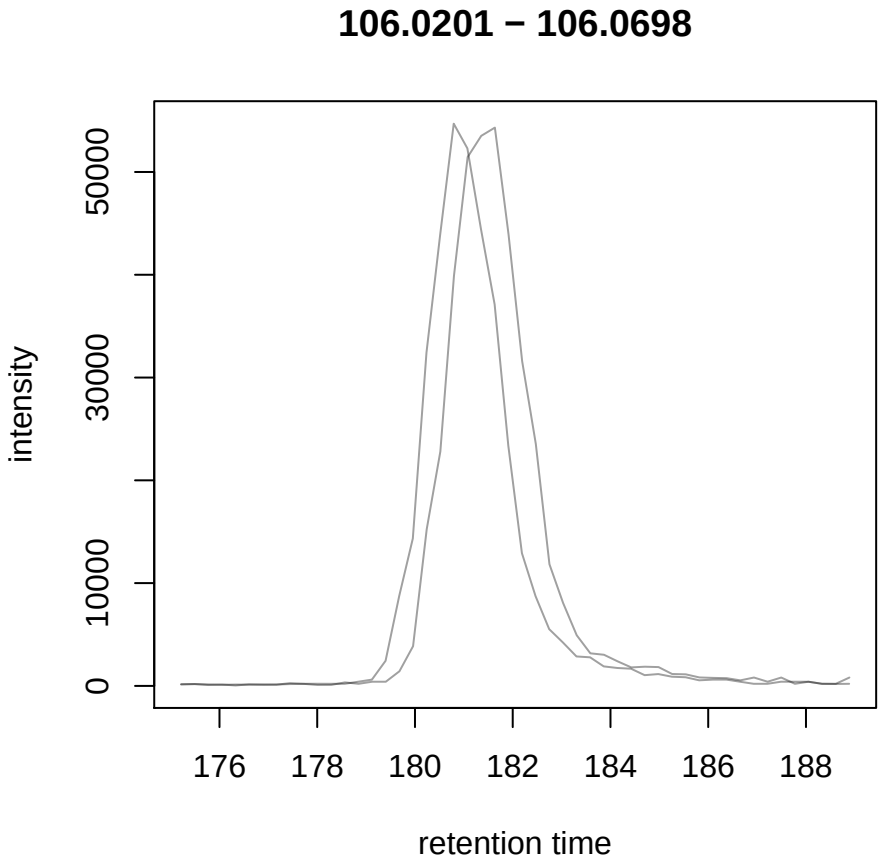


Figure 3: Extracted ion chromatogram for the Serine [M+H]+ ion in both files

centroided data, thus, prior to data analysis, profile data, such as the example data used here, should be centroided. The `MSnbase` package provides all tools to perform this centroiding (and data smoothing) in R: `pickPeaks` and `smooth`.

Below we inspect the profile data for the [M+H]+ ion adduct of Serine. We again subset the data to the m/z and retention time range containing signal from Serine and `plot` the data with the option `type = "XIC"`, that generates a combined chromatographic and *map* visualization of the data (i.e. a plot of the individual m/z, rt and intensity data tuples with data points colored by their intensity in the m/z - retention time space).

```
#' Filter the MS data to the signal from the Serine ion and plot it using
#' type = "XIC"
data %>%
    filterRt(rt = c(175, 189)) %>%
    filterMz(mz = c(106.02, 106.07)) %>%
    plot(type = "XIC")
```

The plot shows all data points measured by the instrument. Each *column* of data points in the lower panel represents the signal measured at one discrete time point, stored in one spectrum. We can see a distribution of the signal for serine in both retention time and also in m/z dimension.

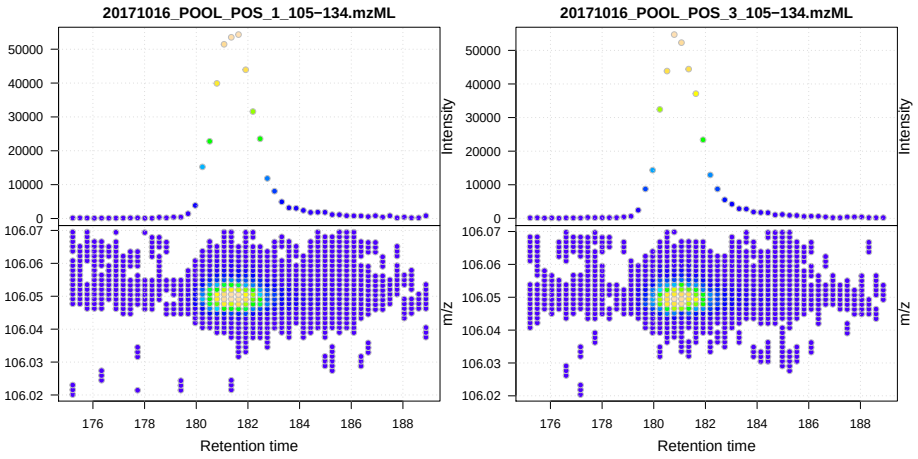


Figure 4: Profile data for Serine

Next we smooth the data in each spectrum using a Savitzky-Golay filter, which usually improves data quality by reducing noise. Subsequently we perform the centroiding based on a simple peak-picking strategy that reports the maximum signal for each mass peak in each spectrum.

```
#' Smooth the signal, then do a simple peak picking.
data_cent <- data %>%
    smooth(method = "SavitzkyGolay", halfWindowSize = 6) %>%
    pickPeaks()

#' Plot the centroided data for Serine
data_cent %>%
    filterRt(rt = c(175, 189)) %>%
    filterMz(mz = c(106.02, 106.07)) %>%
    plot(type = "XIC")
```

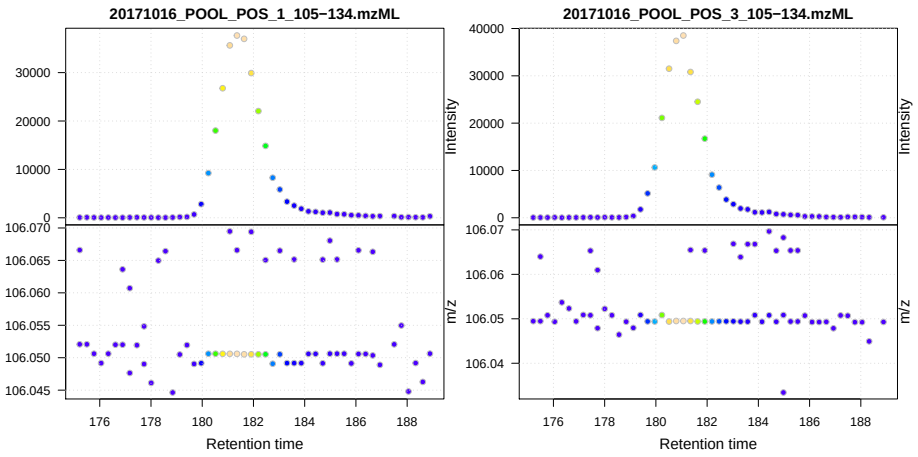


Figure 5: Centroided data for Serine

The centroiding reduced the data to a single data point for an ion in each spectrum. For more advanced centroiding options that can also fine-tune the m/z value of the reported centroid see the `pickPeaks` help or the centroiding vignette in `MSnbase`.

The raw data was imported using the *onDisk*-mode that does not read the full MS data into memory. Any data manipulation (such as the data smoothing or peak picking above) has thus to be applied *on-the-fly* to the data each time m/z or intensity values are retrieved. To make any data manipulation on an `OnDiskMSnExp` object *persistent* we need to export the data to mzML files and re-read the data again. Below we thus save the centroided data as mzML files and import it again.

```
#' Write the centroided data to files with the same names in the current
#' directory
fls_new <- basename(fileNames(data))
writeMSData(data_cent, file = fls_new)

#' Read the centroided data.
data_cent <- readMSData(fls_new, pdata = new("NAnnotatedDataFrame", pd),
                        mode = "onDisk")
```

3.4 Preprocessing of LC-MS data

Preprocessing of GC/LC-MS data in untargeted metabolomics experiments aims at quantifying the signal from individual ion species in a data set and consists of the 3 steps chromatographic peak detection, alignment (also called retention time correction) and correspondence (also called peak grouping). The resulting matrix of feature abundances can then be used as an input in downstream analyses including data normalization, identification of features of interest and annotation of features to metabolites.

3.4.1 Chromatographic peak detection

Chromatographic peak detection aims to identify peaks along the retention time axis that represent the signal from individual compounds' ions. This can be performed with the `findChromPeaks` function and one of the available algorithms that can be configures with the respective parameter object: passing a `MatchedFilterParam` to `findChromPeaks` performs peak detection as described in the original xcms article (Smith et al. 2006). With `CentWaveParam` a continuous wavelet transformation (CWT)-based peak detection is performed that can detect close-by and partially overlapping peaks with different (retention time) widths (Tautenhahn, Böttcher, and Neumann 2008). With `MassifquantParam` a Kalman filter-based peak detection can be performed (Conley et al. 2014). Additional peak detection algorithms for direct injection data are also available, but not discussed here.

We use the *centWave* algorithm that performs peak detection in two steps: first it identifies *regions of interest* in the m/z - retention time space and subsequently detects peaks in these regions using a continuous wavelet transform (see the original publication for more details). The algorithm can be configured with several parameters (see `?CentWaveParam`), the most important ones being `peakwidth` and `ppm`. `peakwidth` defines the minimal and maximal expected width of the peak in retention time dimension and depends thus on the setting of the employed LC-MS system making this parameter highly data set dependent. Appropriate values can be estimated based on extracted ion chromatograms of e.g. internal standards

or known compounds in the data. Below we extract the chromatographic data for Serine and perform a peak detection on the `Chromatogram` object using the default parameters for centWave.

```
#' Get the XIC for serine
srn_chr <- chromatogram(data_cent, rt = c(165, 200),
                        mz = c(106.03, 106.06),
                        aggregationFun = "max")
#' Plot the data
par(mfrow = c(1, 1), mar = c(4, 4.5, 1, 1))
plot(srn_chr)
```

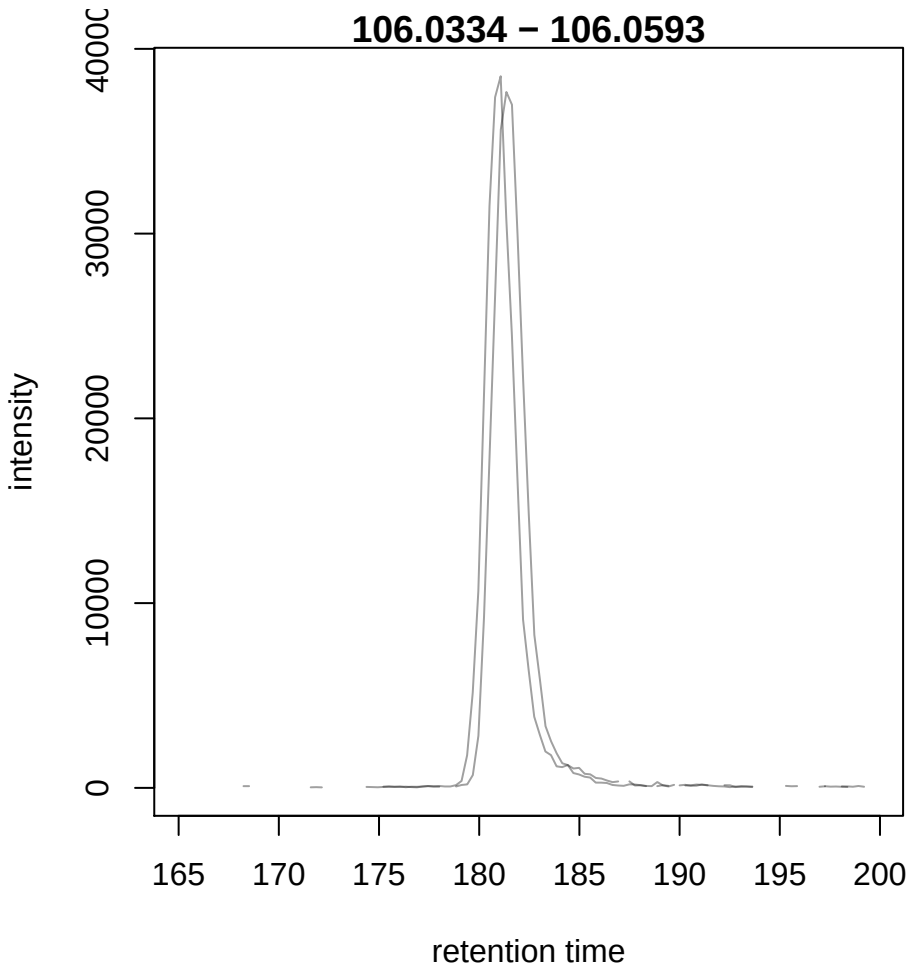


Figure 6: XIC for Serine

```
#' Get default centWave parameters
cwp <- CentWaveParam()

#' "dry-run" peak detection on the XIC.
findChromPeaks(srn_chr, param = cwp)
## XChromatograms with 1 row and 2 columns
```

```
##                   1              2
##      <XChromatogram> <XChromatogram>
## [1,]        peaks: 0        peaks: 0
## phenoData with 4 variables
## featureData with 5 variables
## - - - xcms preprocessing - - -
```

No peaks were identified by the call above. Looking at the default values for the centWave parameters helps understanding why peak detection failed:

```
cwp
## Object of class:  CentWaveParam
## Parameters:
##  ppm: 25
##  peakwidth: 20, 50
##  snthresh: 10
##  prefilter: 3, 100
##  mzCenterFun: wMean
##  integrate: 1
##  mzdiff: -0.001
##  fitgauss: FALSE
##  noise: 0
##  verboseColumns: FALSE
##  roiList length: 0
##  firstBaselineCheck TRUE
##  roiScales length: 0
```

The default settings for `peakwidth` are 20 to 50 seconds, while from the plot above it is apparent that the chromatographic peak for Serine is about 4 seconds wide. We thus adapt the settings to accommodate peaks ranging from 2 to 10 seconds and re-run the peak detection. In general, it is advised to investigate peak widths for several ions in the data set to determine the most appropriate `peakwidth` setting.

```
cwp <- CentWaveParam(peakwidth = c(2, 10))

srn_chr <- findChromPeaks(srn_chr, param = cwp)

#' Plot the data and higlight identified peak area
plot(srn_chr)
```

With our data set-specific `peakwidth` we were able to detect the peak for Serine. The identified chromatographic peaks have been added to the result object `srn_chr` and can be extracted/inspected with the `chromPeaks` function.

```
chromPeaks(srn_chr)
##           rt   rtmin   rtmax     into     intb     maxo  sn row column
## [1,] 181.356 179.124 183.867 71660.11 70213.14 37664.94  63   1      1
## [2,] 181.072 178.840 183.304 67756.66 67576.23 38517.76 606   1      2
```

The `matrix` returned by `chromPeaks` contains the retention time and m/z range of the peak (`"rtmin"`, `"rtmax"`, `"mzmin"` and `"mzmax"` as well as the integrated peak area (`"into"`), the maximal signal (`"maxo"`) and the signal to noise ratio (`"sn"`).

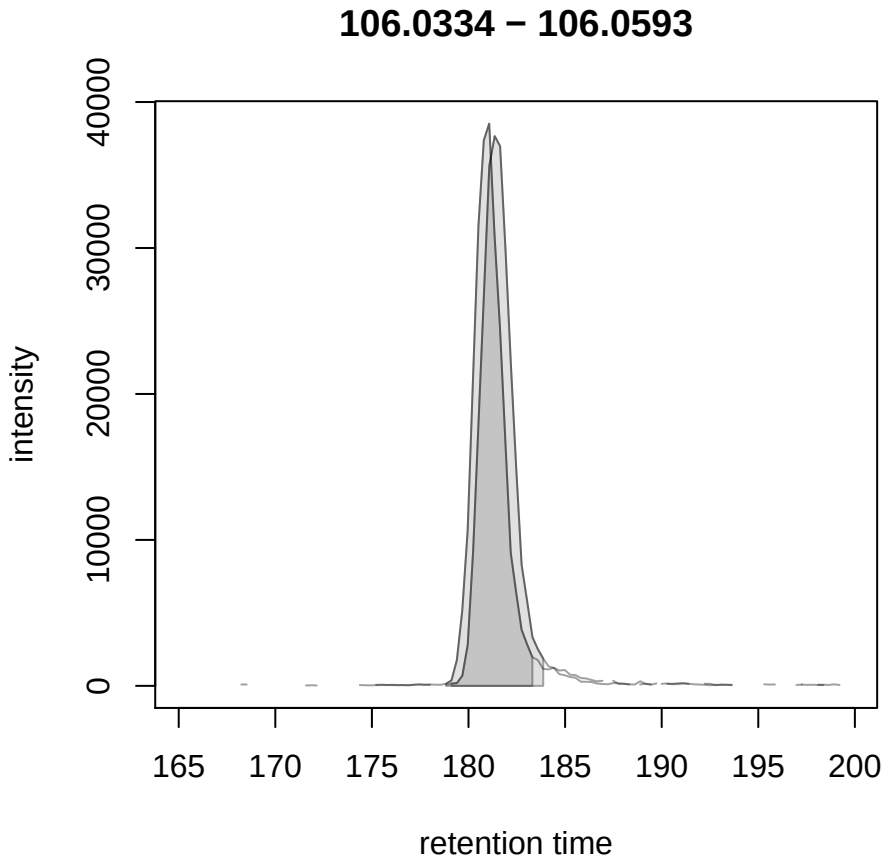
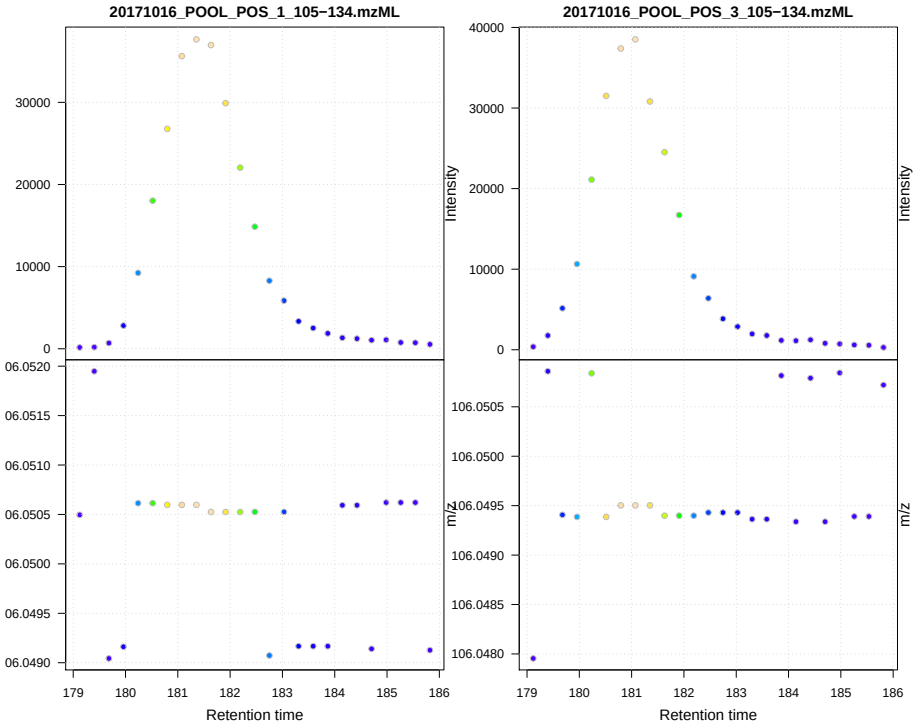


Figure 7: XIC for Serine with detected chromatographic peak (colored in grey)

Another important parameter for centWave is `ppm` which is used in the initial identification of the regions of interest. In contrast to random noise, the *real* signal from an ion is expected to yield stable m/z values in consecutive scans (the scattering of the m/z values around the *real* m/z value of the ion is supposed to be inversely related with its intensity). In centWave, all data points that differ by less than `ppm` in consecutive spectra are combined into a region of interest that is then subject to the CWT-based peak detection (same as performed above on the XIC). To illustrate this, we plot the data for Serine with the option `type = "XIC"`.

```
#' Restrict the data to signal from Serine
srn <- data_cent %>%
    filterRt(rt = c(179, 186)) %>%
    filterMz(mz = c(106.04, 106.06))

#' Plot the data
plot(srn, type = "XIC")
```



We can observe some scattering of the data points in m/z dimension (lower panel in the plot above), that decreases with increasing intensity of the signal. We next calculate the differences in m/z values between consecutive scans in this data subset.

```
#' Extract the Serine data for one file as a data.frame
srn_df <- as(filterFile(srn, 1), "data.frame")

#' The difference between m/z values from consecutive scans expressed
#' in ppm (parts per million)
diff(srn_df$mz) * 1e6 / mean(srn_df$mz)
##  [1]  13.695973646 -27.391665930   1.112565444  13.695804399   0.000000000
##  [6]  -0.158840806   0.000000000   0.000000000  -0.682098923   0.000000000
## [11]   0.000000000   0.007189239 -13.695795336  13.695795336 -12.807200180
## [16]   0.000000000   0.000000000  13.443799681   0.000000000 -13.695795190
## [21]  13.957010392   0.000000000   0.000000000 -14.085629933
```

The difference in m/z values for the Serine data is thus between 0 and 27 ppm. This should ideally be evaluated for several compounds and should be set to a value that allows to capture the full chromatographic peaks for most of the tested compounds. We can next perform the peak detection using our settings for the `ppm` and `peakwidth` parameters on the full data set.

```
#' Perform peak detection
cwp <- CentWaveParam(peakwidth = c(2, 10), ppm = 30)
data_cent <- findChromPeaks(data_cent, param = cwp)
```

The result from the `findChromPeaks` call is an `XCMSnExp` object which contains all preprocessing results and, by extending the `OnDiskMSnExp` object, inherits all of its functionality which was described so far. The results from the peak detection analysis can be accessed with the

`chromPeaks` function, that, with the optional `rt` and `mz` parameters, allows to extract identified chromatographic peaks from specific areas in the data. Below we extract all identified peaks for a certain m/z - rt area.

```
#' Access the peak detection results from a specific m/z - rt area
chromPeaks(data_cent, mz = c(106, 107), rt = c(150, 190))
##             mz    mzmin    mzmax      rt   rtmin   rtmax       into
## CP127 106.0625 106.0606 106.0636 173.264 171.869 175.217   567.5094
## CP157 106.0506 106.0505 106.0506 181.356 179.124 183.867 71660.1062
## CP456 106.0633 106.0609 106.0652 172.701 170.469 174.375   558.1327
## CP498 106.0496 106.0494 106.0508 181.072 178.840 183.304 67756.6562
##             intb       maxo  sn sample
## CP127   563.6306   426.6084  46      1
## CP157 71559.5369 37664.9371 751      1
## CP456   553.5792   381.6084  58      2
## CP498 67645.5242 38517.7622 898      2
```

For each identified peak the m/z and rt value of the apex is reported (columns `"mz"` and `"rt"`) as well as their ranges (`"mzmin"`, `"mzmax"`, `"rtmin"`, `"rtmax"`), the integrated signal of the peak (i.e. the peak area `"into"`), the maximal signal of the peak (`"maxo"`), the signal to noise ratio (`"sn"`) and the index of the sample in which the peak was detected (`"sample"`).

For quality assessment we could now calculate summary statistics on the identified peaks to e.g. identify samples with much less detected peaks. Also, we can use the `plotChromPeaks` function to provide some general information on the location of the identified chromatographic peaks in the m/z - rt space.

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

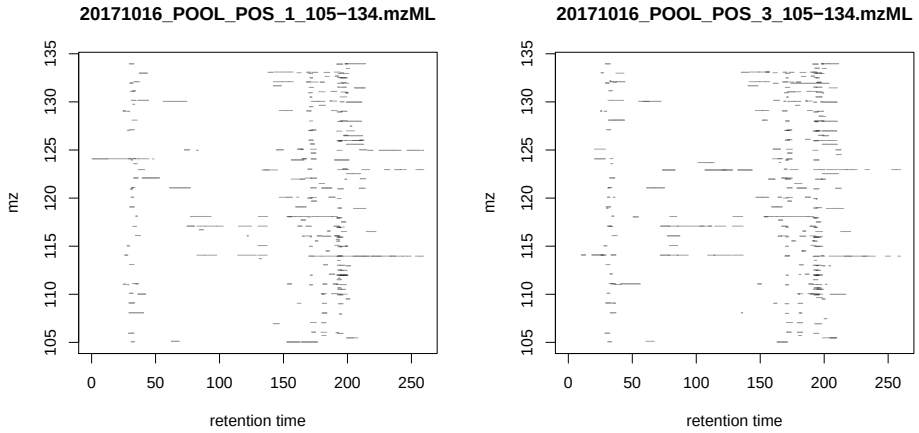


Figure 8: Location of the identified chromatographic peaks in the m/z - rt space

3.4.2 Alignment

While chromatography helps to discriminate better between analytes it is also affected by variances that can lead to shifts in retention times between measurement runs. The alignment step aims to adjust these retention time differences between samples within an

experiment. Below we plot the base peak chromatograms of both files of our toy data set to visualize these differences. Note that with `peakType = "none"` we disable plotting of identified chromatographic peaks that would be drawn by default on chromatograms extracted from an object containing peak detection results.

```
#' Extract base peak chromatograms
bpc_raw <- chromatogram(data_cent, aggregationFun = "max")
plot(bpc_raw, peakType = "none")
```

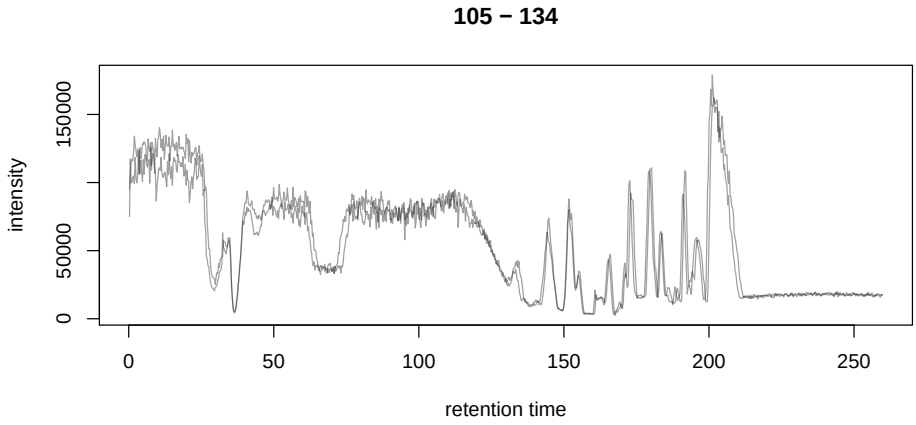


Figure 9: BPC of all files

While both samples were measured with the same setup on the same day the two chromatograms are slightly shifted.

Alignment can be performed in `xcms` with the `adjustRtime` function that supports the *peakGroups* (Smith et al. 2006) and the *obiwarp* (Prince and Marcotte 2006) method. The settings for the algorithms can be defined with the `PeakGroupsParam` and the `ObiwarpParam` parameter objects, respectively.

For our example we use the peakGroups method that aligns samples based on the retention times of *hook peaks*, which should be present in most samples and which, because they are supposed to represent signal from the same ion species, can be used to estimate retention time shifts between samples. Prior to the alignment we thus have to group peaks across samples, which is accomplished by the *peakDensity* correspondence analysis method. Details about this method and explanations on the choices of its parameters are provided in the next section. After having performed this initial correspondence analysis, we perform the alignment using settings `minFraction = 1` and `span = 0.6`. `minFraction` defines the proportion of samples in which a candidate hook peak has to be detected/present. A value of 0.9 would e.g. require for a hook peak to be detected in in 90% of all samples of the experiment. Our data represents replicated measurements of the same sample pool and we can therefore require hook peaks to be present in each file. The parameter `span` defines the degree of smoothing of the loess function that is used to allow different regions along the retention time axis to be adjusted by a different factor. A value of 0 will most likely cause overfitting, while 1 would perform a constant, linear shift. Values between 0.4 and 0.6 seem to be reasonable for most experiments.

```
#' Define the settings for the initial peak grouping - details for
#' choices in the next section.
pdp <- PeakDensityParam(sampleGroups = data_cent$group, bw = 1.8,
                        minFraction = 1, binSize = 0.02)
```

```
data_cent <- groupChromPeaks(data_cent, pdp)

#' Define settings for the alignment
pgp <- PeakGroupsParam(minFraction = 1, span = 0.6)
data_cent <- adjustRtime(data_cent, param = pgp)
```

Adjusted retention times are stored, along with the raw retention times, within the result object. Any function accessing retention times (such as `rtime`) will by default return adjusted retention times from an `XCMSnExp` object, if present. Note that also the retention times of the identified chromatographic peaks were adjusted by the `adjustRtime` call. After alignment it is suggested to evaluate alignment results e.g. by inspecting differences between raw and adjusted retention times.

```
#' Plot the difference between raw and adjusted retention times
plotAdjustedRtime(data_cent)
```

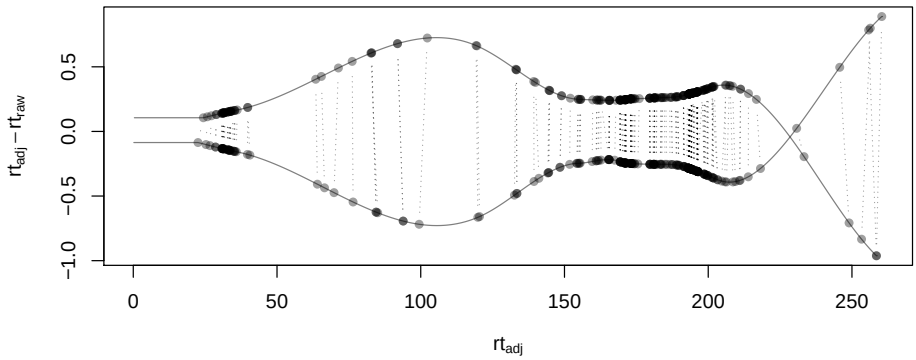


Figure 10: Alignment results
Shown is the difference between raw and adjusted retention times and the hook peaks that were used for the alignment (shown as points).

The difference between raw and adjusted retention time should be reasonable. In our example it is mostly below one second, which is OK since the samples were measured within a short time period and differences are thus expected to be small. Also, hook peaks should ideally be present along the full retention time range. Next we plot the base peak chromatograms before and after alignment.

```
par(mfrow = c(2, 1))
#' Plot the raw base peak chromatogram
plot(bpc_raw, peakType = "none")
#' Plot the BPC after alignment
plot(chromatogram(data_cent, aggregationFun = "max"), peakType = "none")
```

The base peak chromatograms are nicely aligned after retention time adjustment. The impact of the alignment should also be evaluated on known compounds or internal standards. We thus plot below the XIC for Serine before and after alignment.

```
#' Use adjustedRtime parameter to access raw/adjusted retention times
par(mfrow = c(1, 2), mar = c(4, 4.5, 1, 0.5))
plot(chromatogram(data_cent, mz = c(106.04, 106.06),
                  rt = c(179, 186), adjustedRtime = FALSE))
```

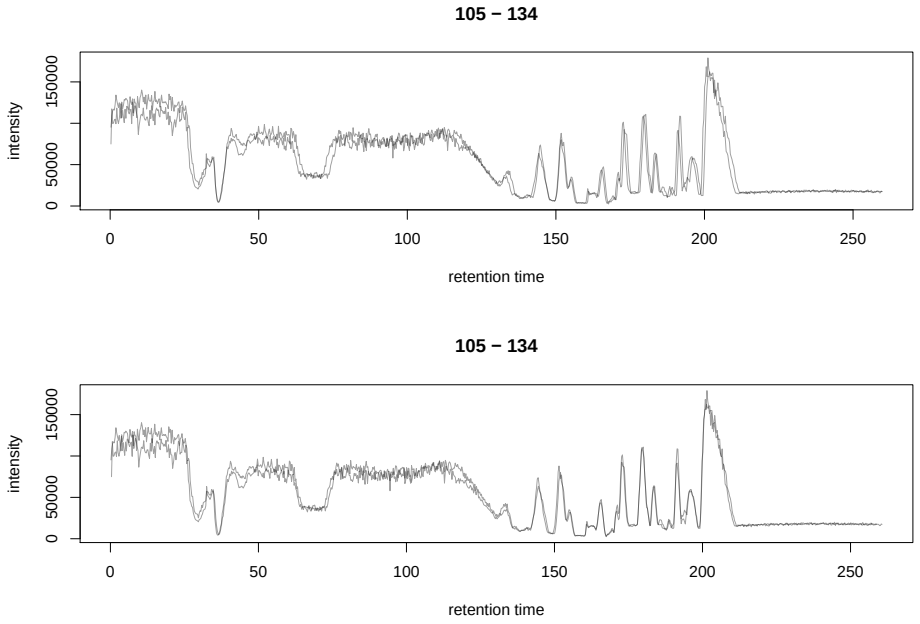


Figure 11: BPC before (top) and after (bottom) alignment

```
plot(chromatogram(data_cent, mz = c(106.04, 106.06),
                  rt = c(179, 186)))
```

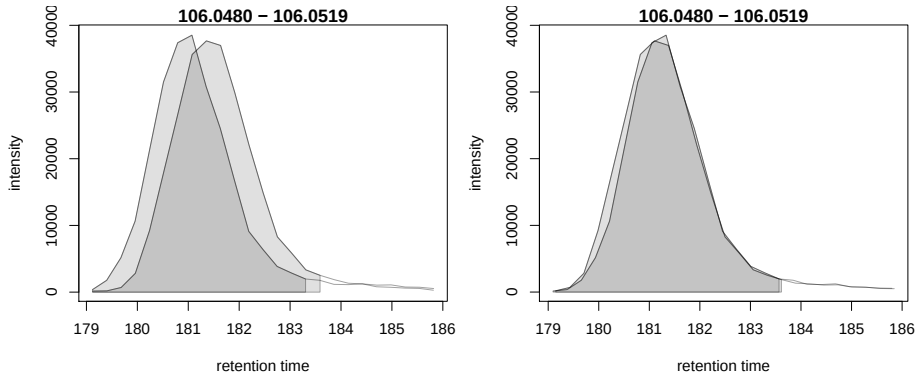


Figure 12: XIC for Serine before (left) and after (right) alignment

The Serine peaks are also nicely aligned after adjustment. Note that if we were not happy with the alignment results we could simply retry with different settings after removing old results with the `dropAdjustedRtime` function. This function restores also the original retention times of the identified chromatographic peaks.

3.4.3 Correspondence

The final step of the LC-MS preprocessing with `xcms` is the correspondence analysis, in which chromatographic peaks from the same ion are grouped across samples to form a *feature*. `xcms` implements two methods for this purpose: *peak density* (Smith et al. 2006) and *nearest* (Katajamaa, Miettinen, and Oresic 2006) that can be configured by passing either

a `PeakDensityParam` or a `NearestPeaksParam` object to the `groupChromPeaks` function. For our example we use the *peak density* method that iterates through m/z slices in the data and groups chromatographic peaks to features in each slice (within the same or across samples) depending on their retention time and the distribution of chromatographic peaks along the retention time axis. Peaks representing signal from the same ion are expected to have a similar retention time and, if found in many samples, this should also be reflected by a higher peak density at the respective retention time. To illustrate this we extract below an m/z slice containing the Serine peak and use the `plotChromPeakDensity` function to visualize the distribution of peaks along the retention time axis and to *simulate* a correspondence analysis based on the provided settings.

```
#' Extract an ion chromatogram containing the signal from serine
chr <- chromatogram(data_cent, mz = c(106.04, 106.06),
                    aggregationFun = "max")

#' Get default parameters for the grouping
pdp <- PeakDensityParam(sampleGroups = data_cent$group)

#' Dry-run correspondence and show the results.
plotChromPeakDensity(chr, param = pdp)
```

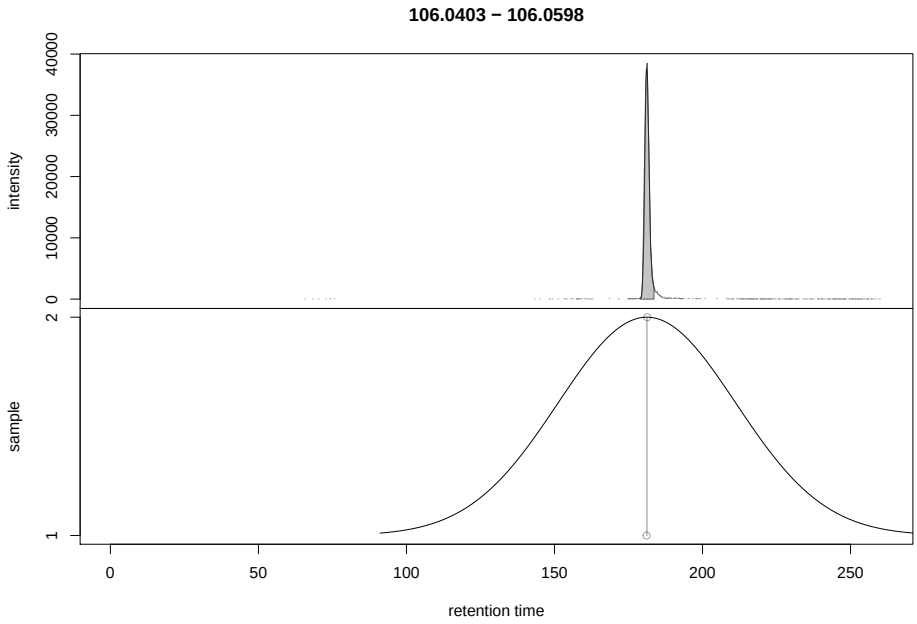


Figure 13: BPC for a m/z slice and defined features within this slice based on default settings

The upper panel in the plot above shows the chromatographic data with the identified peaks. The lower panel shows the retention time of identified peaks (x-axis) per sample (y-axis) with the black solid line representing their distribution along the x-axis. Peak groups (features) are indicated with grey rectangles. The peak density correspondence method groups all chromatographic peaks under the same *density peak* into a feature. With the default settings we were able to group the Serine peak of each sample into a feature. The parameters for the peak density correspondence analysis are:

- `binSize`: m/z width of the bin/slice of data in which peaks are grouped.
- `bw` defines the smoothness of the density function.

- `maxFeatures`: maximum number of features to be defined in one bin.
- `minFraction`: minimum proportion of samples (of one group!) for which a peak has to be present.
- `minSamples`: minimum number of samples a peak has to be present.

The parameters `minFraction` and `minSamples` depend on the experimental layout and should be set accordingly. `binSize` should be set to a small enough value to avoid peaks from different ions, but with similar m/z and retention time, being grouped together. The most important parameter however is `bw` and, while its default value of 30 was able to correctly group the Serine peaks, it should always be evaluated on other, more complicated, signals too. Below we evaluate the performance of the default parameters on an m/z slice that contains signal from multiple ions with the same m/z, including isomers Betaine and Valine ([M+H]+ m/z 118.08625).

```
#' Plot the chromatogram for an m/z slice containing Betaine and Valine
mzr <- 118.08625 + c(-0.01, 0.01)
chr <- chromatogram(data_cent, mz = mzr, aggregationFun = "max")

#' Correspondence in that slice using default settings
pdp <- PeakDensityParam(sampleGroups = data_cent$group)
plotChromPeakDensity(chr, param = pdp)
```

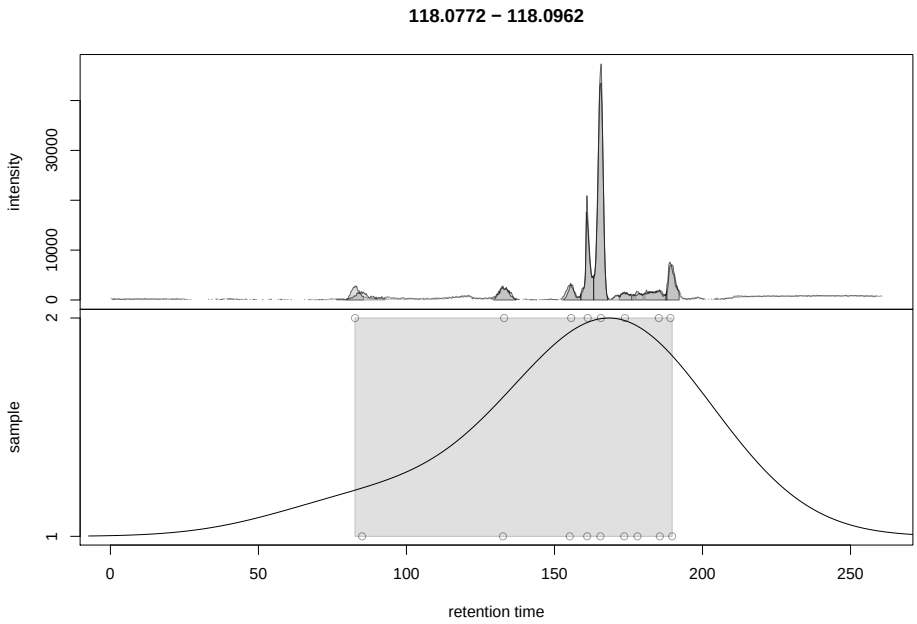


Figure 14: Correspondence analysis with default settings on a m/z slice containing signal from multiple ions

With default settings all chromatographic peaks present in the m/z slice were grouped into the same feature. Signal from different ions would thus be treated as a single entity. Below we repeat the analysis with a strongly reduced value for `bw`.

```
#' Reducing the bandwidth
pdp <- PeakDensityParam(sampleGroups = data_cent$group, bw = 1.8)
plotChromPeakDensity(chr, param = pdp)
```

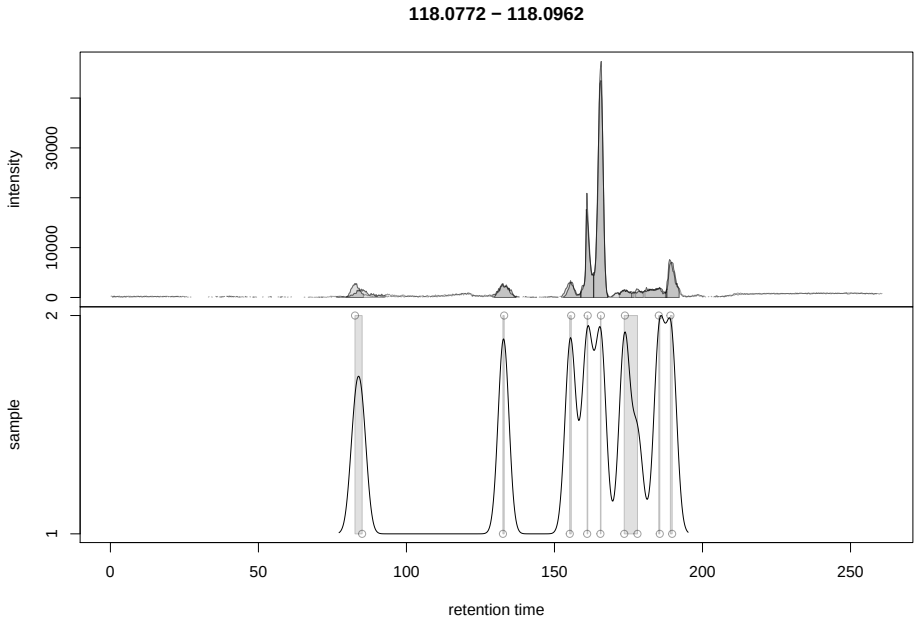


Figure 15: Correspondence analysis with reduced bw setting on a m/z slice containing signal from multiple ions

With a `bw` of `1.8` we successfully grouped the peaks into different features. We can now use these settings for the correspondence analysis on the full data set.

```
pdp <- PeakDensityParam(sampleGroups = data_cent$group, bw = 1.8,
                        minFraction = 0.4, binSize = 0.02)

#' Perform the correspondence analysis
data_cent <- groupChromPeaks(data_cent, param = pdp)
```

Next we evaluate the results from the correspondence analysis on a different m/z slice containing isomers Leucine and Isoleucine ([M+H]+ m/z 132.10191). Setting `simulate = FALSE` in `plotChromPeakDensity` will show the actual results from the correspondence analysis.

```
#' Plot the chromatogram for an m/z slice containing Leucine and Isoleucine
mzr <- 132.10191 + c(-0.01, 0.01)
chr <- chromatogram(data_cent, aggregationFun = "max", mz = mzr)
plotChromPeakDensity(chr, simulate = FALSE)
```

Despite being very close, chromatographic peaks of isomers were successfully grouped into separate features.

Results from the correspondence analysis can be accessed with the `featureDefinition` function. This function returns a data frame with the retention time and m/z ranges of the apex positions from the peaks assigned to the feature and their respective indices in the `chromPeaks` matrix.

```
#' Definition of the features
featureDefinitions(data_cent)
## DataFrame with 373 rows and 9 columns
##                  mzmed           mzmin           mzmax           rtmed
```

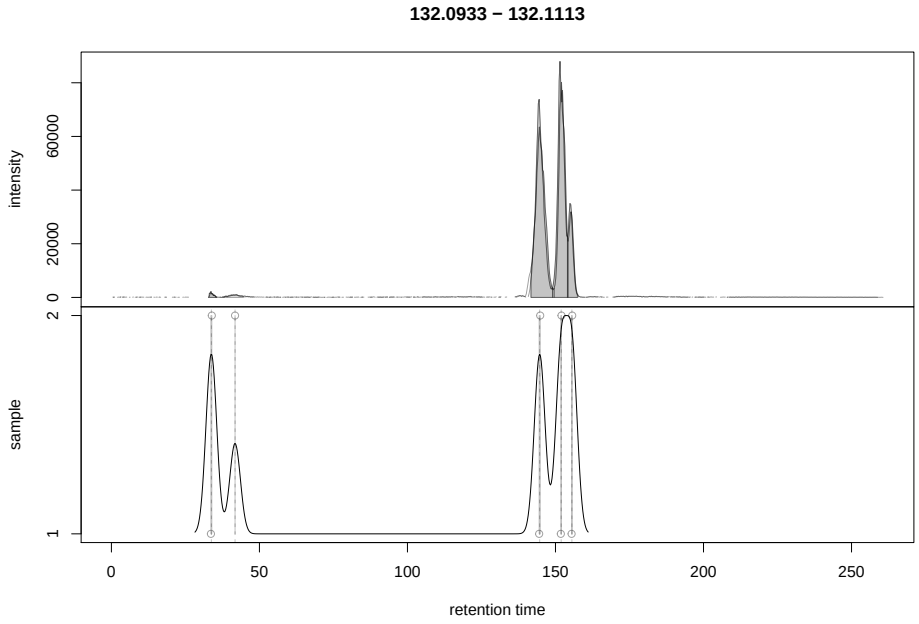


Figure 16: Result of correspondence on a slice containing the isomers Leucine and Isoleucine

```
##             <numeric>       <numeric>       <numeric>       <numeric>
## FT001 105.041765759115   105.0417381237  105.04179339453 167.690828441715
## FT002 105.041534700049 105.041534700049 105.041534700049 157.678493020291
## ...               ...             ...             ...             ...
## FT372 133.972794276215 133.972754801636 133.972833750795  206.86375484864
## FT373 133.974012650623 133.974012650623 133.974012650623 201.504155299324
##                 rtmin           rtmax    npeaks      POOL     peakidx
##             <numeric>       <numeric> <numeric> <numeric>      <list>
## FT001 167.461570484448 167.920086398982         2         2 c(125, 431)
## FT002 157.678493020291 157.678493020291         1         1         124
## ...               ...             ...       ...       ...         ...
## FT372 206.353905400883 207.373604296396         2         2 c(283, 639)
## FT373 201.504155299324 201.504155299324         1         1         640
```

Also, we can calculate simple per-feature summary statistic with the `featureSummary` function. This function reports for each feature the total number and the percentage of samples in which a peak was detected and the total numbers and percentage of these samples in which more than one peak was assigned to the feature.

```
#' Per-feature summary.
head(featureSummary(data_cent))
##       count perc multi_count multi_perc        rsd
## FT001     2  100           0          0 0.01492685
## FT002     1   50           0          0         NA
## FT003     2  100           0          0 0.22934418
## FT004     2  100           0          0 0.17149504
## FT005     2  100           0          0 0.04142419
## FT006     2  100           0          0 0.33421678
```

The final result from the LC-MS data preprocessing is a matrix with feature abundances, rows being features, columns samples. Such a matrix can be extracted with the `featureValues` function from the result object. The function takes two additional parameters `value` and `method`: `value` defines the column in the `chromPeaks` table that should be reported in the matrix, and `method` the approach to handle cases in which more than one peak in a sample is assigned to the feature. Below we set `value = "into"` (the default) to extract the total integrated peak area and `method = "maxint"` to report the peak area of the peak with the largest intensity for features with multiple peaks in a sample.

```
#' feature intensity matrix
fmat <- featureValues(data_cent, value = "into", method = "maxint")
head(fmat)
##       20171016_POOL_POS_1_105-134.mzML 20171016_POOL_POS_3_105-134.mzML
## FT001                        3159.7569                         3093.752
## FT002                        4762.3987                               NA
## FT003                         744.8752                         1033.232
## FT004                       20211.2634                        15839.550
## FT005                       10220.8762                        10837.710
## FT006                       19653.1073                        31816.844
```

While we do have abundances reported for most features, we might also have missing values for some, like for feature *FT002* in the second sample above. Such `NA`s occur if no chromatographic peak was assigned to a feature, either because peak detection failed, or because the corresponding ion is absent in the respective sample. One possibility to deal with such missing values is data imputation. With the `fillChromPeaks` function, `xcms` provides however an alternative approach that integrates the signal measured at the m/z - retention time region of the feature in the original files of samples for which an `NA` was reported hence *filling-in* missing peak data. The region from which signal is recovered is defined by the columns `"mzmin"`, `"mzmax"`, `"rtmin"` and `"rtmax"` in the `featureDefinitions` data frame, which represent the minimal and maximal positions of the apexes of all chromatographic peaks assigned to the feature. Because only peak apex positions are considered, this region might not be representative of the actual chromatographic peaks. The feature region can however be increased in m/z and/or retention time retention: parameter `fixedRt` enables for example the expansion of the feature area in retention time dimension by a constant value. In the example below we fill-in missing peak data expanding the feature region by the median width of all chromatographic peaks in the data.

```
#' Number of missing values
sum(is.na(fmat))
## [1] 137

#' Determine the median retention time width of detected peaks
rt_med <- median(chromPeaks(data_cent)[, "rtmax"] -
                 chromPeaks(data_cent)[, "rtmin"])

fpp <- FillChromPeaksParam(fixedRt = rt_med / 2)
data_cent <- fillChromPeaks(data_cent, param = fpp)

#' How many missing values after
sum(is.na(featureValues(data_cent)))
## [1] 12
```

```
fmat_fld <- featureValues(data_cent, value = "into", method = "maxint")
head(fmat_fld)
##       20171016_POOL_POS_1_105-134.mzML 20171016_POOL_POS_3_105-134.mzML
## FT001                        3159.7569                         3093.752
## FT002                        4762.3987                         5234.356
## FT003                         744.8752                         1033.232
## FT004                       20211.2634                        15839.550
## FT005                       10220.8762                        10837.710
## FT006                       19653.1073                        31816.844
```

With `fillChromPeaks` we could *rescue* signal for all but 14 features with missing values. Note that filled-in peak information can also be removed any time with the `dropFilledChromPeaks` function. Also, setting `filled = FALSE` in the `featureValues` function would return only data from detected peaks.

The data analysis would now continue with the feature matrix and could comprise normalization of the abundances, identification of the compounds and differential abundance analysis.

One final thing worth mentioning is that `XCMSnExp` objects keep, next to the preprocessing results, also a history of all processing steps and all parameter objects used during the analysis. The process history can be accessed with the `processHistory` function.

```
#' Overview of the performed processings
processHistory(data_cent)
## [[1]]
## Object of class "XProcessHistory"
##  type: Peak detection
##  date: Tue Jul 23 19:27:25 2019
##  info:
##  fileIndex: 1,2
##  Parameter class: CentWaveParam
##  MS level(s) 1
##
## [[2]]
## Object of class "XProcessHistory"
##  type: Peak grouping
##  date: Tue Jul 23 19:27:30 2019
##  info:
##  fileIndex: 1,2
##  Parameter class: PeakDensityParam
##  MS level(s) 1
##
## [[3]]
## Object of class "XProcessHistory"
##  type: Retention time correction
##  date: Tue Jul 23 19:27:31 2019
##  info:
##  fileIndex: 1,2
##  Parameter class: PeakGroupsParam
##  MS level(s) 1
##
## [[4]]
```

```
## Object of class "XProcessHistory"
##  type: Peak grouping
##  date: Tue Jul 23 19:27:36 2019
##  info:
##  fileIndex: 1,2
##  Parameter class: PeakDensityParam
##  MS level(s) 1
##
## [[5]]
## Object of class "XProcessHistory"
##  type: Missing peak filling
##  date: Tue Jul 23 19:27:38 2019
##  info:
##  fileIndex: 1,2
##  Parameter class: FillChromPeaksParam
##  MS level(s) 1
```

The parameter object for one analysis step can be accessed with `processParam`:

```
#' Access the parameter class for a processing step
processParam(processHistory(data_cent)[[1]])
## Object of class:  CentWaveParam
## Parameters:
##  ppm: 30
##  peakwidth: 2, 10
##  snthresh: 10
##  prefilter: 3, 100
##  mzCenterFun: wMean
##  integrate: 1
##  mzdiff: -0.001
##  fitgauss: FALSE
##  noise: 0
##  verboseColumns: FALSE
##  roiList length: 0
##  firstBaselineCheck TRUE
##  roiScales length: 0
```

4 Bonus material - peak detection fun

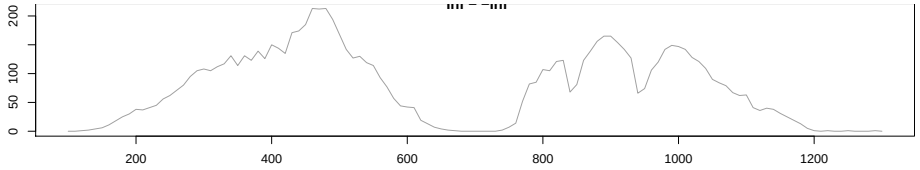
In this section we apply the lessons learned from previous sections, in particular how to adapt peak detection setting on a rather noisy *chromatographic* data. Below we load the example data from a text file.

```
data <- read.table("data/Chromatogram.txt", sep = "\t", header = TRUE)
head(data)
##    rt intensity
## 1 100         0
## 2 110         0
## 3 120         1
```

```
## 4 130         2
## 5 140         4
## 6 150         6
```

Our data has two columns, one with *retention times* and one with *intensities*. We can now create a `Chromatogram` object from that and plot the data.

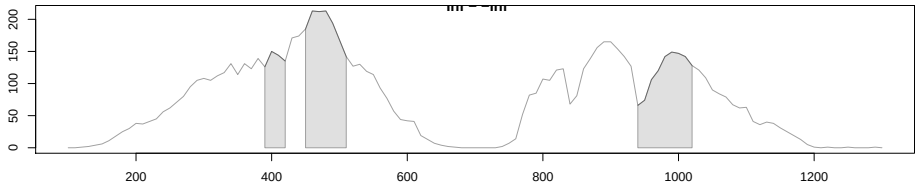
```
chr <- Chromatogram(rtime = data$rt, intensity = data$intensity)
par(mar = c(2, 2, 0, 0))
plot(chr)
```



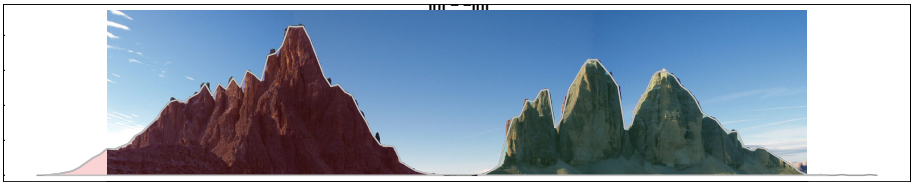
There are two peaks present in the data, with the signal from the latter being particularly noisy. The goal is now to perform the peak detection and to identify the two peaks. A first try with the default settings for *centWave* clearly shows that we have to tune the parameters (note that the setting of `sn = 0` is required for the present data set as there are not enough *background* data points for the algorithm to estimate the noise level properly).

Which parameter would you now adapt to the data? What would be your choices? Go ahead and try different settings or setting combination to see if you can succeed in detecting the two peaks. Eventually you might even try a different peak detection algorithm (e.g. `MatchedFilterParam`).

```
xchr <- findChromPeaks(chr, param = CentWaveParam(sn = 0))
par(mar = c(2, 2, 0, 0))
plot(xchr)
```



With the default parameters centWave clearly failed to identify the two large peaks, defining only smaller fragments of them as potential peaks. Especially the second peak with its peculiar tri-forked shape seems to cause troubles. This would be even for a hydrophilic liquid interaction chromatography (HILIC), known to potentially result in noisy odd-shaped peaks, a rather unusual peak shape. In fact, the signal we were analyzing here is not of chromatographic origin:



Our example data represents a panorama picture featuring mountains from the Dolomites, the *Paternkofel* (left peak, colored red) and the famous *Drei Zinnen* (right tri-forked peak colored green).

5 Session information

```
devtools::session_info()
## - Session info -------------------------------------------------------------
##  setting  value
##  version  R version 3.6.1 (2019-07-05)
##  os       macOS Mojave 10.14.6
##  system   x86_64, darwin18.7.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       CET
##  date     2019-07-23
##
## - Packages -----------------------------------------------------------------
##  package         * version   date       lib source
##  affy              1.62.0    2019-05-02 [1] Bioconductor
##  affyio            1.54.0    2019-05-02 [1] Bioconductor
##  assertthat        0.2.1     2019-03-21 [1] CRAN (R 3.6.1)
##  backports         1.1.4     2019-04-10 [1] CRAN (R 3.6.1)
##  Biobase         * 2.44.0    2019-05-02 [1] Bioconductor
##  BiocGenerics    * 0.30.0    2019-05-02 [1] Bioconductor
##  BiocManager       1.30.4    2018-11-13 [1] CRAN (R 3.6.1)
##  BiocParallel    * 1.18.0    2019-05-03 [1] Bioconductor
##  BiocStyle       * 2.12.0    2019-05-02 [1] Bioconductor
##  bookdown          0.12      2019-07-11 [1] CRAN (R 3.6.1)
##  callr             3.3.1     2019-07-18 [1] CRAN (R 3.6.1)
##  cli               1.1.0     2019-03-19 [1] CRAN (R 3.6.1)
##  codetools         0.2-16    2018-12-24 [1] CRAN (R 3.6.1)
##  colorspace        1.4-1     2019-03-18 [1] CRAN (R 3.6.1)
##  crayon            1.3.4     2017-09-16 [1] CRAN (R 3.6.1)
##  DEoptimR          1.0-8     2016-11-19 [1] CRAN (R 3.6.1)
##  desc              1.2.0     2018-05-01 [1] CRAN (R 3.6.1)
##  devtools          2.1.0     2019-07-06 [1] CRAN (R 3.6.1)
##  digest            0.6.20    2019-07-04 [1] CRAN (R 3.6.1)
##  doParallel        1.0.14    2018-09-24 [1] CRAN (R 3.6.1)
##  dplyr             0.8.3     2019-07-04 [1] CRAN (R 3.6.1)
##  evaluate          0.14      2019-05-28 [1] CRAN (R 3.6.1)
##  foreach           1.4.4     2017-12-12 [1] CRAN (R 3.6.1)
##  fs                1.3.1     2019-05-06 [1] CRAN (R 3.6.1)
##  ggplot2           3.2.0     2019-06-16 [1] CRAN (R 3.6.1)
##  glue              1.3.1     2019-03-12 [1] CRAN (R 3.6.1)
##  gtable            0.3.0     2019-03-25 [1] CRAN (R 3.6.1)
##  htmltools         0.3.6     2017-04-28 [1] CRAN (R 3.6.1)
```

```
##  impute            1.58.0    2019-05-02 [1] Bioconductor
##  IRanges           2.18.1    2019-05-31 [1] Bioconductor
##  iterators         1.0.10    2018-07-13 [1] CRAN (R 3.6.1)
##  knitr           * 1.23      2019-05-18 [1] CRAN (R 3.6.1)
##  labeling          0.3       2014-08-23 [1] CRAN (R 3.6.1)
##  lattice           0.20-38   2018-11-04 [1] CRAN (R 3.6.1)
##  lazyeval          0.2.2     2019-03-15 [1] CRAN (R 3.6.1)
##  limma             3.40.2    2019-05-17 [1] Bioconductor
##  magrittr        * 1.5       2014-11-22 [1] CRAN (R 3.6.1)
##  MALDIquant        1.19.3    2019-05-12 [1] CRAN (R 3.6.1)
##  MASS              7.3-51.4  2019-03-31 [1] CRAN (R 3.6.1)
##  MassSpecWavelet   1.50.0    2019-05-02 [1] Bioconductor
##  Matrix            1.2-17    2019-03-22 [1] CRAN (R 3.6.1)
##  memoise           1.1.0     2017-04-21 [1] CRAN (R 3.6.1)
##  MSnbase         * 2.10.1    2019-05-31 [1] Bioconductor
##  multtest          2.40.0    2019-05-02 [1] Bioconductor
##  munsell           0.5.0     2018-06-12 [1] CRAN (R 3.6.1)
##  mzID              1.22.0    2019-05-02 [1] Bioconductor
##  mzR             * 2.18.0    2019-05-02 [1] Bioconductor
##  ncdf4             1.16.1    2019-03-11 [1] CRAN (R 3.6.1)
##  pcaMethods        1.76.0    2019-05-02 [1] Bioconductor
##  pillar            1.4.2     2019-06-29 [1] CRAN (R 3.6.1)
##  pkgbuild          1.0.3     2019-03-20 [1] CRAN (R 3.6.1)
##  pkgconfig         2.0.2     2018-08-16 [1] CRAN (R 3.6.1)
##  pkgload           1.0.2     2018-10-29 [1] CRAN (R 3.6.1)
##  plyr              1.8.4     2016-06-08 [1] CRAN (R 3.6.1)
##  png             * 0.1-7     2013-12-03 [1] CRAN (R 3.6.1)
##  preprocessCore    1.46.0    2019-05-02 [1] Bioconductor
##  prettyunits       1.0.2     2015-07-13 [1] CRAN (R 3.6.1)
##  processx          3.4.1     2019-07-18 [1] CRAN (R 3.6.1)
##  ProtGenerics    * 1.16.0    2019-05-02 [1] Bioconductor
##  ps                1.3.0     2018-12-21 [1] CRAN (R 3.6.1)
##  purrr             0.3.2     2019-03-15 [1] CRAN (R 3.6.1)
##  R6                2.4.0     2019-02-14 [1] CRAN (R 3.6.1)
##  RANN              2.6.1     2019-01-08 [1] CRAN (R 3.6.1)
##  RColorBrewer    * 1.1-2     2014-12-07 [1] CRAN (R 3.6.1)
##  Rcpp            * 1.0.1     2019-03-17 [1] CRAN (R 3.6.1)
##  remotes           2.1.0     2019-06-24 [1] CRAN (R 3.6.1)
##  rlang             0.4.0     2019-06-25 [1] CRAN (R 3.6.1)
##  rmarkdown       * 1.14      2019-07-12 [1] CRAN (R 3.6.1)
##  robustbase        0.93-5    2019-05-12 [1] CRAN (R 3.6.1)
##  rprojroot         1.3-2     2018-01-03 [1] CRAN (R 3.6.1)
##  S4Vectors       * 0.22.0    2019-05-02 [1] Bioconductor
##  scales            1.0.0     2018-08-09 [1] CRAN (R 3.6.1)
##  sessioninfo       1.1.1     2018-11-05 [1] CRAN (R 3.6.1)
##  stringi           1.4.3     2019-03-12 [1] CRAN (R 3.6.1)
##  stringr           1.4.0     2019-02-10 [1] CRAN (R 3.6.1)
##  survival          2.44-1.1  2019-04-01 [1] CRAN (R 3.6.1)
##  testthat          2.1.1     2019-04-23 [1] CRAN (R 3.6.1)
##  tibble            2.1.3     2019-06-06 [1] CRAN (R 3.6.1)
##  tidyselect        0.2.5     2018-10-11 [1] CRAN (R 3.6.1)
```

```
##  tinytex          0.14      2019-06-25 [1] CRAN (R 3.6.1)
##  usethis          1.5.1     2019-07-04 [1] CRAN (R 3.6.1)
##  vsn              3.52.0    2019-05-02 [1] Bioconductor
##  withr            2.1.2     2018-03-15 [1] CRAN (R 3.6.1)
##  xcms           * 3.6.1     2019-05-16 [1] Bioconductor
##  xfun             0.8       2019-06-25 [1] CRAN (R 3.6.1)
##  XML              3.98-1.20 2019-06-06 [1] CRAN (R 3.6.1)
##  yaml             2.2.0     2018-07-25 [1] CRAN (R 3.6.1)
##  zlibbioc         1.30.0    2019-05-02 [1] Bioconductor
##
## [1] /Users/jo/R/2019-07/CSAMA2019/lib/R/library
```

References

Conley, Christopher J, Rob Smith, Ralf J O Torgrip, Ryan M Taylor, Ralf Tautenhahn, and John T Prince. 2014. "Massifquant: open-source Kalman filter-based XC-MS isotope trace feature detection." *Bioinformatics* 30 (18): 2636–43.

Gatto, Laurent, and Kathryn S Lilley. 2012. "MSnbase-an R/Bioconductor package for isobaric tagged mass spectrometry data visualization, processing and quantitation." *Bioinformatics* 28 (2): 288–89.

Katajamaa, Mikko, Jarkko Miettinen, and Matej Oresic. 2006. "MZmine: toolbox for processing and visualization of mass spectrometry based molecular profile data." *Bioinformatics* 22 (5): 634–36.

Prince, John T, and Edward M Marcotte. 2006. "Chromatographic alignment of ESI-LC-MS proteomics data sets by ordered bijective interpolated warping." *Analytical Chemistry* 78 (17): 6140–52.

Smith, Colin A, Elizabeth J Want, Grace O'Maille, Ruben Abagyan, and Gary Siuzdak. 2006. "XCMS: processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching, and identification." *Analytical Chemistry* 78 (3): 779–87.

Smith, Rob, Andrew D Mathis, Dan Ventura, and John T Prince. 2014. "Proteomics, lipidomics, metabolomics: a mass spectrometry tutorial from a computer scientist's point of view." *BMC Bioinformatics* 15 Suppl 7 (Suppl 7): S9.

Tautenhahn, Ralf, Christoph Böttcher, and Steffen Neumann. 2008. "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics* 9 (1): 504.