`flagme`: Fragment-level analysis of GC-MS-based metabolomics data

Mark Robinson
mrobinson@wehi.edu.au
Riccardo Romoli
riccardo.romoli@unifi.it

May 19, 2021

1 Introduction

This document gives a brief introduction to the `flagme` package, which is designed to process, visualise and statistically analyze sets of GC-MS samples. The ideas discussed here were originally designed with GC-MS-based metabolomics in mind, but indeed some of the methods and visualizations could be useful for LC-MS data sets. The *fragment-level analysis* though, takes advantage of the rich fragmentation patterns observed from electron interaction (EI) ionization.

There are many aspects of data processing for GC-MS data. Generally, algorithms are run separately on each sample to detect features, or *peaks* (e.g. AMDIS). Due to retention time shifts from run-to-run, an alignment algorithm is employed to allow the matching of the same feature across multiple samples. Alternatively, if known standards are introduced to the samples, retention *indices* can be computed for each peak and used for alignment. After peaks are matched across all samples, further processing steps are employed to create a matrix of abundances, leading into detecting differences in abundance.

Many of these data processing steps are prone to errors and they often tend to be black boxes. But, with effective exploratory data analysis, many of the pitfalls can be avoided and any problems can be fixed before proceeding to the downstream statistical analysis. The package provides various visualizations to ensure the methods applied are not black boxes.

The `flagme` package gives a complete suite of methods to go through all common stages of data processing. In addition, R is especially well suited to the downstream data analysis tasks since it is very rich in analysis tools and has excellent visualization capabilities. In addition, it is freely available (`www.r-project.org`), extensible and there is a growing community of users and developers. For routine analyses, graphical user interfaces could be designed.

2 Reading and visualizing GC-MS data

To run these examples, you must have the `gcspikelite` package installed. This data package contains several GC-MS samples from a spike-in experiment we designed to interrogate data processing methods. So, first, we load the packages:

To load the data and corresponding peak detection results, we simply create vectors of the file-names and create a `peakDataset` object. Note that we can speed up the import time by setting the retention time range to a subset of the elution, as below:

```
> gcmsPath <- paste(find.package("gcspikelite"), "data", sep="/")
> data(targets)
> cdfFiles <- paste(gcmsPath, targets$FileName, sep="/")
```

```
> eluFiles <- gsub("CDF", "ELU", cdfFiles)
> pd <- peaksDataset(cdfFiles, mz=seq(50,550), rtrange=c(7.5,8.5))

 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_468.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_474.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_475.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_485.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_493.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_496.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_470.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_471.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_479.CDF

> pd <- addAMDISPeaks(pd, eluFiles)

Reading retention time range: 7.500133 8.499917
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_468.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_474.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_475.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_485.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_493.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_496.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_470.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_471.ELU ... Done.
Reading D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_479.ELU ... Done.

> pd

An object of class "peaksDataset"
9 samples: 0709_468 0709_474 0709_475 0709_485 0709_493 0709_496 0709_470 0709_471 0709_479
501 m/z bins - range: ( 50 550 )
scans: 175 175 175 175 175 174 175 175 175
peaks: 24 23 26 20 27 24 24 25 21
```

Here, we have added peaks from AMDIS, a well known and mature algorithm for deconvolution of GC-MS data. For GC-TOF-MS data, we have implemented a parser for the `ChromaTOF` output (see the analogous `addChromaTOF-Peaks` function). The `addXCMSPeaks` allows to use all the XCMS peak-picking algorithms; using this approach it is also possible to elaborate the raw data file from within R instead of using an external software. In particular the function reads the raw data using XCMS, group each extracted ion according to their retention time using CAMERA and attaches them to an already created `peaksDataset` object:

```
> pd.2 <- peaksDataset(cdfFiles[1:3], mz=seq(50,550), rtrange=c(7.5,8.5))

 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_468.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_474.CDF
 Reading  D:/biocbuild/bbs-3.13-bioc/R/library/gcspikelite/data/0709_475.CDF

> pd.2 <- addXCMSPeaks(cdfFiles[1:3], pd.2, peakPicking=c('mF'),
+                      snthresh=3, fwhm=4, step=1, steps=2, mzdiff=0.5)

Start grouping after retention time.
Created 33 pseudospectra.
Start grouping after correlation.
```

```
Calculating peak correlations in 33 Groups...
 % finished: 10  30  40  50  60  70  80  90  100

Calculating graph cross linking in 33 Groups...
 % finished: 10  30  40  50  60  70  80  90  100
New number of ps-groups:  42
xsAnnotate has now 42 groups, instead of 33
Start grouping after retention time.
Created 30 pseudospectra.
Start grouping after correlation.

Calculating peak correlations in 30 Groups...
 % finished: 10  20  50  60  70  80  90  100

Calculating graph cross linking in 30 Groups...
 % finished: 10  20  50  60  70  80  90  100
New number of ps-groups:  42
xsAnnotate has now 42 groups, instead of 30
Start grouping after retention time.
Created 36 pseudospectra.
Start grouping after correlation.

Calculating peak correlations in 36 Groups...
 % finished: 10  20  40  50  60  70  80  90  100

Calculating graph cross linking in 36 Groups...
 % finished: 10  20  40  50  60  70  80  90  100
New number of ps-groups:  43
xsAnnotate has now 43 groups, instead of 36
```

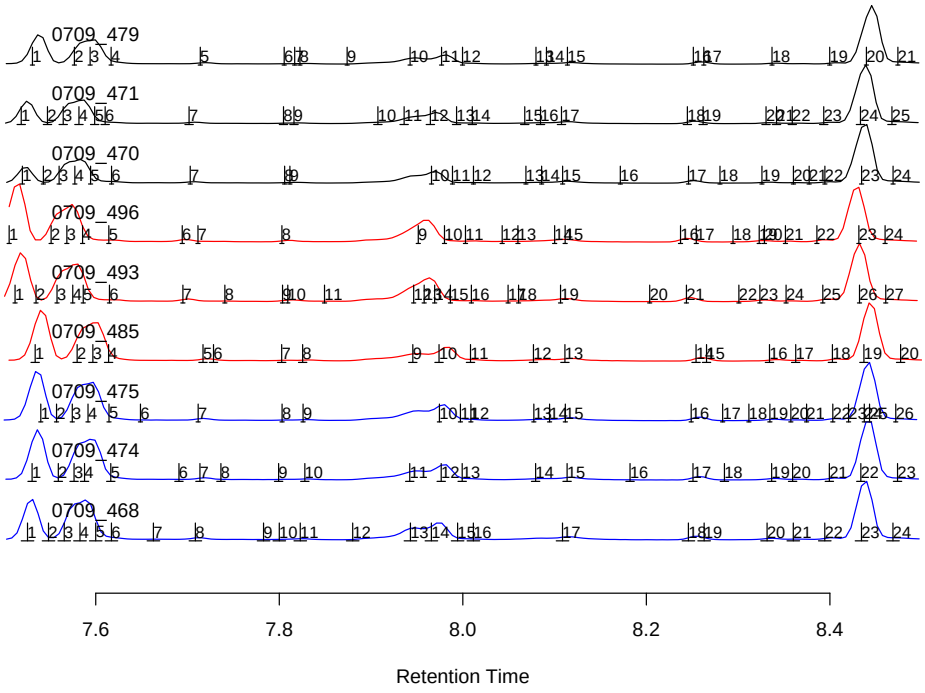
```
> pd.2
```

```
An object of class "peaksDataset"
3 samples: 0709_468 0709_474 0709_475
501 m/z bins - range: ( 50 550 )
scans: 175 175 175
peaks: 11 9 10
```

The possibility to work using computer cluster will be added in the future.

Regardless of platform and peak detection algorithm, a useful visualization of a set of samples is the set of total ion currents (TIC), or extracted ion currents (XICs). To view TICs, you can call:

```
> plot(pd, rtrange=c(7.5,8.5), plotPeaks=TRUE, plotPeakLabels=TRUE,
+      max.near=8, how.near=0.5, col=rep(c("blue","red","black"), each=3))
```



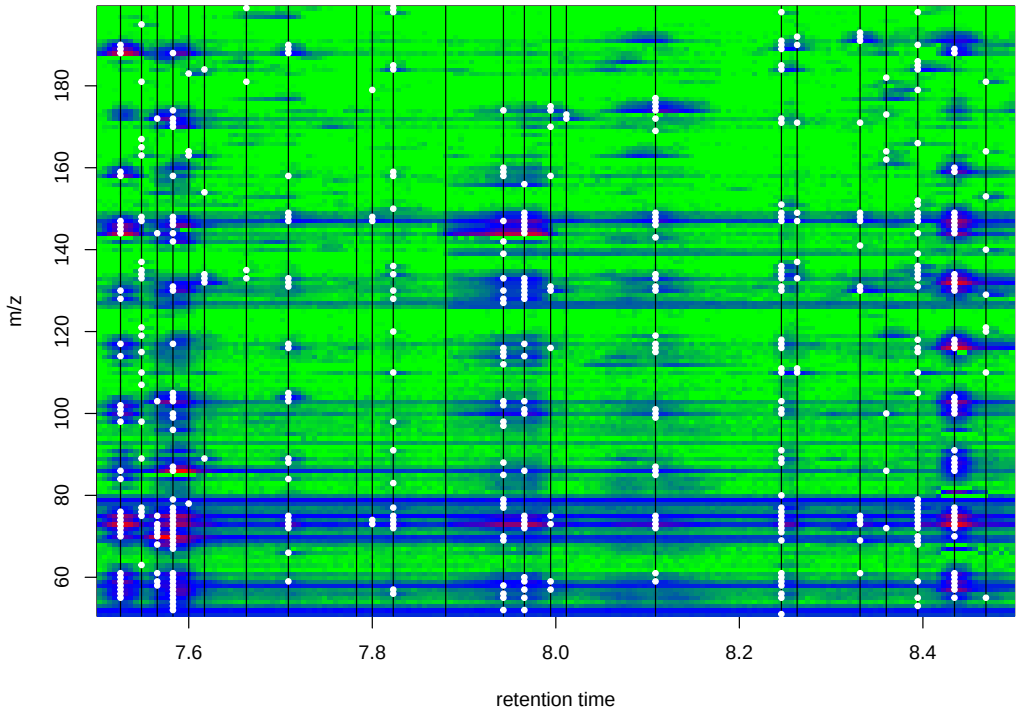
Note here the little *hashes* represent the detected peaks and are labelled with an integer index. One of the main challenges is to match these peak detections across several samples, given that the appear at slightly different times in different runs.

For XICs, you need to give the indices (of `pd@mz`, the grid of mass-to-charge values) that you want to plot through the `mzind` argument. This could be a single ion (e.g. `mzind=24`) or could be a range of indices if multiple ions are of interest (e.g. `mzind=c(24,25,98,99)`).

There are several other features within the `plot` command on `peaksDataset` objects that can be useful. See `?plot` (and select the flagme version) for full details.

Another useful visualization, at least for individual samples, is a 2D heatmap of intensity. Such plots can be enlightening, especially when peak detection results are overlaid. For example (with detected fragment peaks from AMDIS shown in white):

```
> r <- 1
> plotImage(pd, run=r, rtrange=c(7.5,8.5), main="")
> v <- which(pd@peaksdata[[r]] > 0, arr.ind=TRUE) # find detected peaks
> abline(v=pd@peaksrt[[r]])
> points(pd@peaksrt[[r]][v[,2]], pd@mz[v[,1]], pch=19, cex=.6, col="white")
```



3 Pairwise alignment with dynamic programming algorithm

One of the first challenges of GC-MS data is the matching of detected peaks (i.e. metabolites) across several samples. Although gas chromatography is quite robust, there can be some drift in the elution time of the same analyte from run to run. We have devised a strategy, based on dynamic programming, that takes into account both the similarity in spectrum (at the apex of the called peak) and the similarity in retention time, without requiring the identity of each peak; this matching uses the data alone. If each sample gives a 'peak list' of the detected peaks (such as that from AMDIS that we have attached to our `peaksDataset` object), the challenge is to introduce gaps into these lists such that they are best aligned. From this a matrix of retention times or a matrix of peak abundances can be extracted for further statistical analysis, visualization and interpretation. For this matching, we created a procedure analogous to a multiple *sequence* alignment.

To highlight the dynamic programming-based alignment strategy, we first illustrate a pairwise alignment of two peak lists. This example also illustrates the selection of parameters necessary for the alignment. From the data read in previously, let us consider the alignment of two samples, denoted `0709_468` and `0709_474`. First, a similarity matrix for two samples is calculated. This is calculated based on a scoring function and takes into account the similarity in retention time and in the similarity of the apex spectra, according to:

$$S_{ij}(D) = \frac{\sum_{k=1}^{K} x_{ik} y_{jk}}{\sqrt{\sum_{k=1}^{K} x_{ik}^2 \cdot \sum_{k=1}^{K} y_{jk}^2}} \cdot \exp\left(-\frac{1}{2}\frac{(t_i - t_j)^2}{D^2}\right)$$

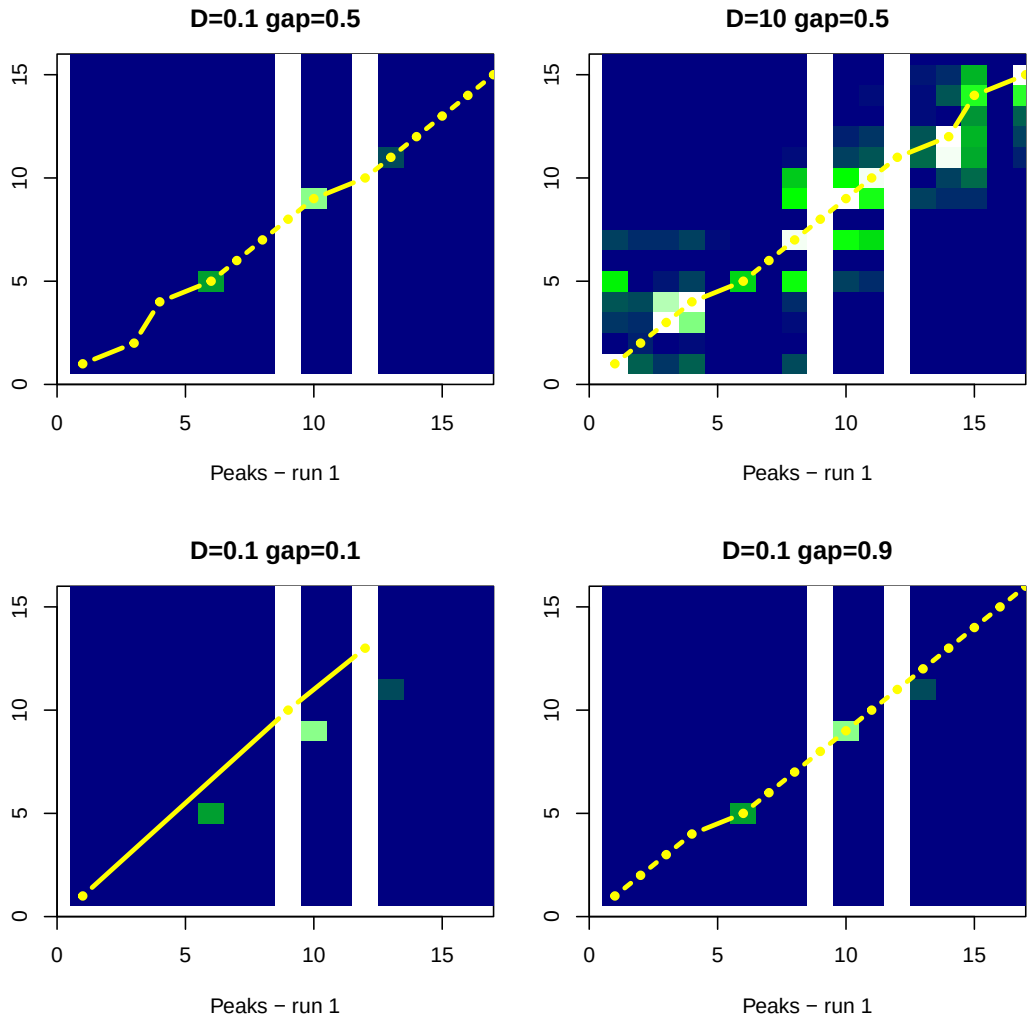
where i is the index of the peak in the first sample and j is the index of the peak in the second sample, $\mathbf{x}_i$ and $\mathbf{y}_j$ are the spectra vectors and t_i and t_j are their respective retention times. As you can see, there are two components to the similarity: spectra similarity (left term) and similarity in retention time (right term). Of course, other metrics

for spectra similarity are feasible. Ask the author if you want to see other metrices implemented. We have some non-optimized code for a few alternative metrics.

The peak alignment algorithm, much like sequence alignments, requires a `gap` parameter to be set, here a number between 0 and 1. A high gap penalty discourages gaps when matching the two lists of peaks and a low gap penalty allows gaps at a very low *cost*. We find that a gap penalty in the middle range (0.4-0.6) works well for GC-MS peak matching. Another parameter, `D`, modulates the impact of the difference in retention time penalty. A large value for `D` essentially eliminates the effect. Generally, we set this parameter to be a bit larger than the average width of a peak, allowing a little flexibility for retention time shifts between samples. Keep in mind the `D` parameter should be set on the scale (i.e. seconds or minutes) of the `peaksrt` slot of the `peaksDataset` object. The next example shows the effect of the `gap` and `D` penalty on the matching of a small ranges of peaks.

```
> Ds <- c(0.1, 10, 0.1, 0.1)
> gaps <- c(0.5, 0.5, 0.1, 0.9)
> par(mfrow=c(2,2), mai=c(0.8466,0.4806,0.4806,0.1486))
> for(i in 1:4){
+   pa <- peaksAlignment(pd@peaksdata[[1]], pd@peaksdata[[2]],
+                        pd@peaksrt[[1]], pd@peaksrt[[2]], D=Ds[i],
+                        gap=gaps[i], metric=1, type=1)
+   plot(pa, xlim=c(0, 17), ylim=c(0, 16), matchCol="yellow",
+        main=paste("D=", Ds[i], " gap=", gaps[i], sep=""))
+ }

[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.5 D= 0.001 , metric= 1 , type= 1
[peaksAlignment]  21 matched.  Similarity= 0.7983038
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.5 D= 0.1 , metric= 1 , type= 1
[peaksAlignment]  21 matched.  Similarity= 0.2308905
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.1 D= 0.001 , metric= 1 , type= 1
[peaksAlignment]  3 matched.  Similarity= 0.333332
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.9 D= 0.001 , metric= 1 , type= 1
[peaksAlignment]  23 matched.  Similarity= 0.8227008
```



You might ask: is the flagme package useful without peak detection results? Possibly. There have been some developments in alignment (generally on LC-MS proteomics experiments) without peak/feature detection, such as Prince et al. 2006, where a very similar dynamic programming is used for a pairwise alignment. We have experimented with alignments without using the peaks, but do not have any convincing results. It does introduce a new set of challenges in terms of highlighting differentially abundant metabolites. However, in the `peaksAlignment` routine above (and those mentioned below), you can set `usePeaks=FALSE` in order to do *scan*-based alignments instead of *peak*-based alignments. In addition, the `flagme` package may be useful simply for its bare-bones dynamic programming algorithm.

3.1 Normalizing retention time score to drift estimates

In what is mentioned above for pairwise alignments, we are penalizing for differences in retention times that are non-zero. But, as you can see from the TICs, some differences in retention time are consistent. For example, all of the peaks from sample `0709_485` elute at later times than peaks from sample `0709_496`. We should be able to estimate this drift and normalize the time penalty to that estimate, instead of penalizing to zero. That is, we should replace $t_i - t_j$ with $t_i - t_j - \hat{d}_{ij}$ where $\hat{d}_{ij}$ is the expected drift between peak i of the first sample and peak j of the second sample.

More details coming soon.

3.2 Imputing location of undetected peaks

One goal of the alignment leading into downstream data analyses is the generation of a table of abundances for each metabolite across all samples. As you can see from the TICs above, there are some low intensity peaks that fall below detection in some but not all samples. Our view is that instead of inserting arbitrary low constants (such as 0 or half the detection limit) or imputing the intensities post-hoc or having missing data in the data matrices, it is best to return to the area of the where the peak should be and give some kind of abundance. The alignments themselves are rich in information with respect to the location of undetected peaks. We feel this is a more conservative and statistically valid approach than introducing arbitrary values.

More details coming soon.

4 Multiple alignment of several experimental groups

Next, we discuss the multiple alignment of peaks across many samples. With replicates, we typically do an alignment within replicates, then combine these together into a summarized form. This cuts down on the computational cost. For example, consider 2 sets of samples, each with 5 replicates. Aligning first within replicates requires 10+10+1 total alignments whereas an all-pairwise alignment requires 45 pairwise alignments. In addition, this allows some flexibility in setting different gap and distance penalty parameters for the *within* alignment and *between* alignment. An example follows.

```
> print(targets)

      FileName Group
1 0709_468.CDF   mmA
2 0709_474.CDF   mmA
3 0709_475.CDF   mmA
4 0709_485.CDF   mmC
5 0709_493.CDF   mmC
6 0709_496.CDF   mmC
7 0709_470.CDF   mmD
8 0709_471.CDF   mmD
9 0709_479.CDF   mmD

> ma <- multipleAlignment(pd, group=targets$Group, wn.gap=0.5, wn.D=.05,
+                         bw.gap=.6, bw.D=0.05, usePeaks=TRUE, filterMin=1,
+                         df=50, verbose=FALSE, metric = 1, type = 1) # bug

[clusterAlignment] Aligning 0709_468 to 0709_474
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  22 matched.  Similarity= 0.8625793
[clusterAlignment] Aligning 0709_468 to 0709_475
[peaksAlignment] Comparing 24 peaks to 26 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  15 matched.  Similarity= 0.8
[clusterAlignment] Aligning 0709_474 to 0709_475
[peaksAlignment] Comparing 23 peaks to 26 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  20 matched.  Similarity= 0.899699
[progressiveAlignment] Doing merge -1 -3
[progressiveAlignment] left.runs: 1 , right.runs: 3
[progressiveAlignment] Doing merge -2 1
```

```
[progressiveAlignment] left.runs: 2 , right.runs: 1 3
[progressiveAlignment] (dot=50) going to 23 :
          used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  7742683 413.6   12915577 689.8 12915577 689.8
Vcells 14104986 107.7   39020654 297.8 39020618 297.8
[clusterAlignment] Aligning 0709_485 to 0709_493
[peaksAlignment] Comparing 20 peaks to 27 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  20 matched.  Similarity= 0.9354748
[clusterAlignment] Aligning 0709_485 to 0709_496
[peaksAlignment] Comparing 20 peaks to 24 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  20 matched.  Similarity= 0.9359244
[clusterAlignment] Aligning 0709_493 to 0709_496
[peaksAlignment] Comparing 27 peaks to 24 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  22 matched.  Similarity= 0.8515771
[progressiveAlignment] Doing merge -5 -6
[progressiveAlignment] left.runs: 5 , right.runs: 6
[progressiveAlignment] Doing merge -4 1
[progressiveAlignment] left.runs: 4 , right.runs: 5 6
[progressiveAlignment] (dot=50) going to 20 :
          used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  7742837 413.6   12915577 689.8 12915577 689.8
Vcells 14105539 107.7   39020654 297.8 39020618 297.8
[clusterAlignment] Aligning 0709_470 to 0709_471
[peaksAlignment] Comparing 24 peaks to 25 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  22 matched.  Similarity= 0.8879722
[clusterAlignment] Aligning 0709_470 to 0709_479
[peaksAlignment] Comparing 24 peaks to 21 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  21 matched.  Similarity= 0.8564714
[clusterAlignment] Aligning 0709_471 to 0709_479
[peaksAlignment] Comparing 25 peaks to 21 peaks -- gap= 0.5 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  19 matched.  Similarity= 0.8258931
[progressiveAlignment] Doing merge -8 -9
[progressiveAlignment] left.runs: 8 , right.runs: 9
[progressiveAlignment] Doing merge -7 1
[progressiveAlignment] left.runs: 7 , right.runs: 8 9
[progressiveAlignment] (dot=50) going to 24 :
          used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  7743015 413.6   12915577 689.8 12915577 689.8
Vcells 14106623 107.7   39020654 297.8 39020618 297.8
[clusterAlignment] Aligning to
[peaksAlignment] Comparing 36 peaks to 29 peaks -- gap= 0.6 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  25 matched.  Similarity= 0.9094807
[clusterAlignment] Aligning to
[peaksAlignment] Comparing 36 peaks to 29 peaks -- gap= 0.6 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  29 matched.  Similarity= 0.8798354
[clusterAlignment] Aligning to
[peaksAlignment] Comparing 29 peaks to 29 peaks -- gap= 0.6 D= 5e-04 , metric= 1 , type= 1
[peaksAlignment]  29 matched.  Similarity= 0.9655151
[progressiveAlignment] Doing merge -1 -3
[progressiveAlignment] left.runs: 1 , right.runs: 3
[progressiveAlignment] Doing merge -2 1
```

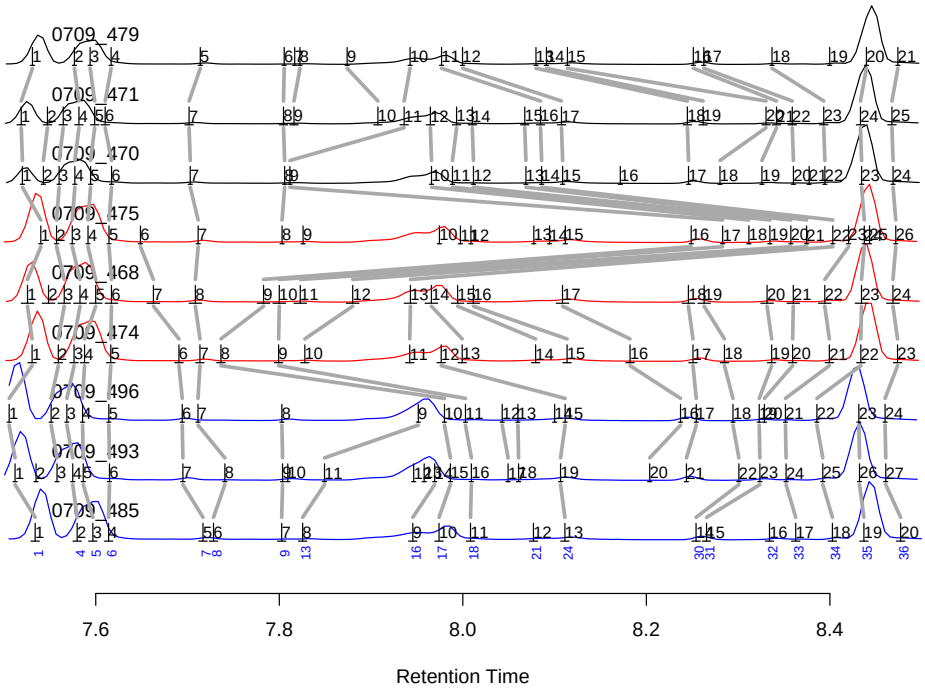
```
[progressiveAlignment] left.runs: 2 , right.runs: 1 3
[progressiveAlignment] (dot=50) going to 29 :
           used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  7744542 413.7   12915577 689.8 12915577 689.8
Vcells 14159169 108.1   39020654 297.8 39020618 297.8
```

```
> ma

An object of class "multipleAlignment"
3 groups: 3 3 3 samples, respectively.
36 merged peaks
```

If you set `verbose=TRUE`, many nitty-gritty details of the alignment procedure are given. Next, we can take the alignment results and overlay it onto the TICs, allowing a visual inspection.

```
> plot(pd, rtrange=c(7.5,8.5), runs=ma@betweenAlignment@runs,
+      mind=ma@betweenAlignment@ind, plotPeaks=TRUE,
+      plotPeakLabels=TRUE, max.near=8, how.near=.5,
+      col=rep(c("blue","red","black"), each=3))
```




```
> mp <- correlationAlignment(object=pd.2, thr=0.85, D=20, penality=0.2,
+                            normalize=TRUE, minFilter=1)
> mp
```

4.1 Gathering results

The alignment results can be extracted from the `multipleAlignment` object as:

```
> ma@betweenAlignment@runs
```

```
[1] 4 5 6 2 1 3 7 8 9
```

```
> ma@betweenAlignment@ind
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    1    1    1    1    1    1    1    1    1
 [2,]   NA    2   NA   NA    2   NA    2    2   NA
 [3,]   NA    3    2    2    3    2    3    3   NA
 [4,]    2    4    3    3    4    3    4    4    2
 [5,]    3    5    4    4    5    4    5    5    3
 [6,]    4    6    5    5    6    5    6    6    4
 [7,]    5    7    6    6    7    6   NA   NA   NA
 [8,]    6    8    7    7    8    7    7    7    5
 [9,]    7    9    8   NA   NA    8    8    8    6
[10,]   NA   NA   NA   NA   NA    9   NA   NA   NA
[11,]   NA   NA   NA   NA   NA   10   NA   NA   NA
[12,]   NA   10   NA   NA   NA   11   NA   NA   NA
[13,]    8   11    9   NA   NA   12   NA   NA   NA
[14,]   NA   12   NA   NA   NA   13   NA   NA   NA
[15,]   NA   13   NA   NA   NA   14   NA   NA    7
[16,]    9   14   NA   NA   NA   15   NA    9    8
[17,]   10   15   10    8    9   16   NA   10    9
[18,]   11   16   11    9   10   17    9   11   10
[19,]   NA   17   12   NA   NA   18   10   12   NA
[20,]   NA   18   13   NA   NA   19   11   13   NA
[21,]   12   NA   14   NA   11   20   12   14   NA
[22,]   NA   NA   NA   10   12   21   13   15   NA
[23,]   NA   NA   NA   11   13   22   14   16   11
[24,]   13   19   15   12   NA   NA   15   17   12
[25,]   NA   NA   NA   13   14   NA   16   NA   NA
[26,]   NA   NA   NA   14   15   NA   17   18   13
[27,]   NA   NA   NA   15   16   NA   NA   19   14
[28,]   NA   20   16   16   17   NA   18   20   15
[29,]   NA   21   17   17   18   NA   19   21   16
[30,]   14   22   18   18   19   NA   20   22   17
[31,]   15   23   19   19   20   NA   21   NA   NA
[32,]   16   NA   20   20   21   NA   22   23   18
[33,]   17   24   21   21   22   23   NA   NA   19
[34,]   18   25   22   22   23   24   23   24   20
[35,]   19   26   23   NA   NA   25   NA   NA   NA
[36,]   20   27   24   23   24   26   24   25   21
```

This table would suggest that matched peak `8` (see numbers below the TICs in the figure above) corresponds to detected peaks `9, 12, 11` in runs `4, 5, 6` and so on, same as shown in the above plot.

In addition, you can gather a list of all the merged peaks with the `gatherInfo` function, giving elements for the retention times, the detected fragment ions and their intensities. The example below also shows the how to construct a table of retention times of the matched peaks (No attempt is made here to adjust retention times onto a common scale. Instead, the peaks are matched to each other on their original scale). For example:

```
> outList <- gatherInfo(pd,ma)
> outList[[8]]

$rt
   mmC.4    mmC.5    mmC.6    mmA.2    mmA.1    mmA.3    mmD.7    mmD.8
7.728317 7.740700 7.711417 7.713550 7.708567 7.711600 7.702967 7.701717
   mmD.9
7.713933

$mz
 [1]  52  59  66  70  72  73  74  75  79  89 104 116 133 147 148 188 204

$data
      mmC.4 mmC.5 mmC.6 mmA.2 mmA.1 mmA.3 mmD.7 mmD.8 mmD.9
 [1,]     0     0     0     0     0     0 26248     0     0
 [2,]     0     0     0  5113  4425  4994  4855  4557  4728
 [3,]     0     0     0  3926  5146  4876  4831  3354  4783
 [4,]     0     0     0 11568     0     0 10637     0     0
 [5,]     0     0     0  3680  3910  4492  4051  3427  3907
 [6,]     0     0     0 61816 65680 66768 65912 52848 61560
 [7,]     0     0     0  6705  6185  7400  6642  6235  7088
 [8,]     0     0 24160 26376 23328     0 28016 26304 27184
 [9,]     0     0     0     0     0     0 38712     0     0
[10,]     0     0     0  5617  5347  5702  5173  3946  5659
[11,]     0     0     0 13173 13808 13207 12852  9816 12492
[12,]     0     0     0  5417  5525  5912  5577  4504  5201
[13,]     0     0     0  3539  3730  2910  3599  2436  3893
[14,]     0     0 17904 21864 20016 22280 21904 17896 22400
[15,]     0     0     0  4413  3430  3890  3006  3335  3851
[16,]     0     0  7636 14433 14751 13765 14731 10680 14061
[17,]     0     0     0  6878  6667  7018  6935  5149  6830

> rtmat <- matrix(unlist(lapply(outList,.subset,"rt"), use.names=FALSE),
+                 nr=length(outList), byrow=TRUE)
> colnames(rtmat) <- names(outList[[1]]$rt); rownames(rtmat) <- 1:nrow(rtmat)
> round(rtmat, 3)

   mmC.4 mmC.5 mmC.6 mmA.2 mmA.1 mmA.3 mmD.7 mmD.8 mmD.9
1  7.534 7.512 7.506 7.531 7.526 7.540 7.520 7.519 7.531
2     NA 7.535    NA    NA 7.549    NA 7.543 7.547    NA
3     NA 7.558 7.551 7.559 7.566 7.557 7.560 7.565    NA
4  7.580 7.575 7.569 7.576 7.583 7.574 7.577 7.582 7.577
5  7.597 7.586 7.586 7.588 7.600 7.592 7.594 7.599 7.594
6  7.614 7.615 7.614 7.616 7.617 7.614 7.617 7.610 7.617
7  7.717 7.695 7.694 7.691 7.663 7.649    NA    NA    NA
8  7.728 7.741 7.711 7.714 7.709 7.712 7.703 7.702 7.714
9  7.803 7.804 7.803    NA    NA 7.803 7.806 7.805 7.805
10    NA    NA    NA    NA    NA 7.826    NA    NA    NA
11    NA    NA    NA    NA    NA 7.975    NA    NA    NA
12    NA 7.809    NA    NA    NA 7.997    NA    NA    NA
13 7.825 7.849 7.951    NA    NA 8.009    NA    NA    NA
14    NA 7.946    NA    NA    NA 8.077    NA    NA    NA
```

```
15    NA 7.958    NA    NA    NA 8.095    NA    NA 7.817
16 7.946 7.969    NA    NA    NA 8.112    NA 7.816 7.823
17 7.974 7.986 7.980 7.736 7.783 8.249    NA 7.907 7.874
18 8.008 8.009 8.003 7.799 7.800 8.283 7.812 7.936 7.943
19    NA 8.049 8.043    NA    NA 8.312 7.966 7.965    NA
20    NA 8.061 8.060    NA    NA 8.335 7.989 7.993    NA
21 8.077    NA 8.100    NA 7.823 8.357 8.012 8.010    NA
22    NA    NA    NA 7.828 7.880 8.375 8.069 8.068    NA
23    NA    NA    NA 7.942 7.943 8.403 8.086 8.085 7.977
24 8.111 8.107 8.111 7.976    NA    NA 8.109 8.108 8.000
25    NA    NA    NA 7.999 7.966    NA 8.172    NA    NA
26    NA    NA    NA 8.079 7.994    NA 8.246 8.245 8.080
27    NA    NA    NA 8.114 8.011    NA    NA 8.262 8.091
28    NA 8.204 8.237 8.182 8.109    NA 8.280 8.330 8.114
29    NA 8.244 8.254 8.251 8.246    NA 8.326 8.342 8.251
30 8.254 8.301 8.294 8.285 8.263    NA 8.360 8.359 8.263
31 8.266 8.324 8.323 8.337 8.332    NA 8.377    NA    NA
32 8.334    NA 8.329 8.359 8.360    NA 8.395 8.393 8.337
33 8.363 8.352 8.352 8.399 8.394 8.420    NA    NA 8.400
34 8.403 8.392 8.386 8.434 8.434 8.437 8.435 8.433 8.440
35 8.437 8.432 8.432    NA    NA 8.443    NA    NA    NA
36 8.477 8.461 8.460 8.474 8.469 8.472 8.469 8.468 8.474
```

5 Future improvements and extension

There are many procedures that we have implemented in our investigation of GC-MS data, but have not made part of the package just yet. Some of the most useful procedures will be released, such as:

1. Parsers for other peak detection algorithms (e.g. MzMine) and parsers for other alignment procedures (e.g. SpectConnect) and perhaps retention indices procedures.
2. More convenient access to the alignment information and abundance table.
3. Statistical analysis of differential metabolite abundance.
4. Fragment-level analysis, an alternative method to summarize abundance across all detected fragments of a metabolite peak.

6 References

See the following for further details:

1. Robinson MD. *Methods for the analysis of gas chromatography - mass spectrometry data.* **Ph.D. Thesis**. October 2008. Department of Medical Biology (Walter and Eliza Hall Institute of Medical Research), University of Melbourne.
2. Robinson MD, De Souza DP, Keen WW, Saunders EC, McConville MJ, Speed TP, Likić VA. (2007) *A dynamic programming approach for the alignment of signal peaks in multiple gas chromatography-mass spectrometry experiments.* **BMC Bioinformatics**. 8:419.
3. Prince JT, Marcotte EM (2006) *Chromatographic alignment of ESI-LC-MS proteomics data sets by ordered bijective interpolated warping.* **Anal Chem**. 78(17):6140-52.

7 This vignette was built with/at ...

```
> sessionInfo()

R version 4.1.0 RC (2021-05-10 r80283)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows Server x64 (build 17763)

Matrix products: default

locale:
[1] LC_COLLATE=C
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] flagme_1.48.0       CAMERA_1.48.0       xcms_3.14.0
 [4] MSnbase_2.18.0      ProtGenerics_1.24.0 S4Vectors_0.30.0
 [7] mzR_2.26.0          Rcpp_1.0.6          Biobase_2.52.0
[10] BiocGenerics_0.38.0 BiocParallel_1.26.0 gcspikelite_1.29.0

loaded via a namespace (and not attached):
 [1] bitops_1.0-7                matrixStats_0.58.0
 [3] doParallel_1.0.16           RColorBrewer_1.1-2
 [5] GenomeInfoDb_1.28.0         backports_1.2.1
 [7] tools_4.1.0                 utf8_1.2.1
 [9] R6_2.5.0                    affyio_1.62.0
[11] KernSmooth_2.23-20          rpart_4.1-15
[13] Hmisc_4.5-0                 DBI_1.1.1
[15] colorspace_2.0-1            nnet_7.3-16
[17] tidyselect_1.1.1            gridExtra_2.3
[19] compiler_4.1.0              MassSpecWavelet_1.58.0
[21] preprocessCore_1.54.0       graph_1.70.0
[23] htmlTable_2.2.1             SparseM_1.81
[25] DelayedArray_0.18.0         caTools_1.18.2
[27] checkmate_2.0.0             scales_1.1.1
[29] DEoptimR_1.0-8              robustbase_0.93-7
[31] affy_1.70.0                 RBGL_1.68.0
[33] stringr_1.4.0               digest_0.6.27
[35] foreign_0.8-81              XVector_0.32.0
[37] htmltools_0.5.1.1           base64enc_0.1-3
[39] jpeg_0.1-8.1                pkgconfig_2.0.3
[41] MatrixGenerics_1.4.0        limma_3.48.0
[43] htmlwidgets_1.5.3           rlang_0.4.11
[45] rstudioapi_0.13             impute_1.66.0
```

```
[47] generics_0.1.0              gtools_3.8.2
[49] mzID_1.30.0                 dplyr_1.0.6
[51] RCurl_1.98-1.3              magrittr_2.0.1
[53] GenomeInfoDbData_1.2.6      Formula_1.2-4
[55] MALDIquant_1.19.3           Matrix_1.3-3
[57] munsell_0.5.0               fansi_0.4.2
[59] MsCoreUtils_1.4.0           lifecycle_1.0.0
[61] vsn_3.60.0                  stringi_1.6.2
[63] MASS_7.3-54                 SummarizedExperiment_1.22.0
[65] zlibbioc_1.38.0             gplots_3.1.1
[67] plyr_1.8.6                  grid_4.1.0
[69] crayon_1.4.1                lattice_0.20-44
[71] splines_4.1.0               knitr_1.33
[73] pillar_1.6.1                igraph_1.2.6
[75] GenomicRanges_1.44.0        codetools_0.2-18
[77] XML_3.99-0.6                glue_1.4.2
[79] latticeExtra_0.6-29         data.table_1.14.0
[81] pcaMethods_1.84.0           BiocManager_1.30.15
[83] vctrs_0.3.8                 png_0.1-7
[85] foreach_1.5.1               gtable_0.3.0
[87] RANN_2.6.1                  purrr_0.3.4
[89] clue_0.3-59                 assertthat_0.2.1
[91] ggplot2_3.3.3               xfun_0.23
[93] ncdf4_1.17                  survival_3.2-11
[95] tibble_3.1.2                iterators_1.0.13
[97] IRanges_2.26.0              cluster_2.1.2
[99] ellipsis_0.3.2
```

```
> date()
```

```
[1] "Wed May 19 20:05:35 2021"
```