

flagme: Fragment-level analysis of GCMS-based metabolomics data

Mark Robinson
mrobinson@wehi.edu.au

October 15, 2013

1 Introduction

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

There are many aspects of data processing for GCMS 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 GCMS data

To run these examples, you must have the **gcspikelite** package installed. This data package contains several GCMS 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 filenames 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:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_468.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_474.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_475.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_485.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_493.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_496.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_470.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_471.CDF
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_479.CDF

> pd<-addAMDISPeaks(pd, eluFiles)

Reading retention time range: 7.500133 8.499917
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_468.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_474.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_475.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_485.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_493.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_496.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_470.ELU ... Done.
Reading D:/biocbld/bbs-2.13-bioc/R/library/gcspikelite/data/0709_471.ELU ... Done.
Reading D:/biocbld/bbs-2.13-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

```

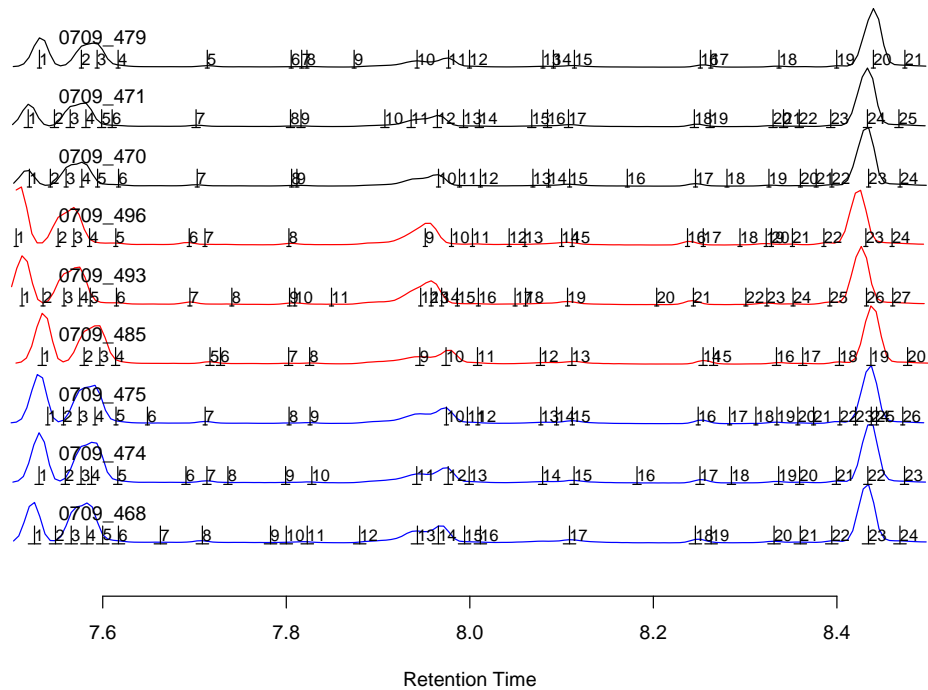
Here, we have added peaks from AMDIS, a well known and mature algorithm for deconvolution of GC-MS data. For GC-TOF-MS data, we have implemented a parser for the **ChromatOF** output (see the analogous **addChromatOF-Peaks** function). Support for XMCS or MzMine may be added in the future. Ask the author if another detection result format is desired as the parsers are generally easy to design.

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=.5, col=rep(c("blue",

```



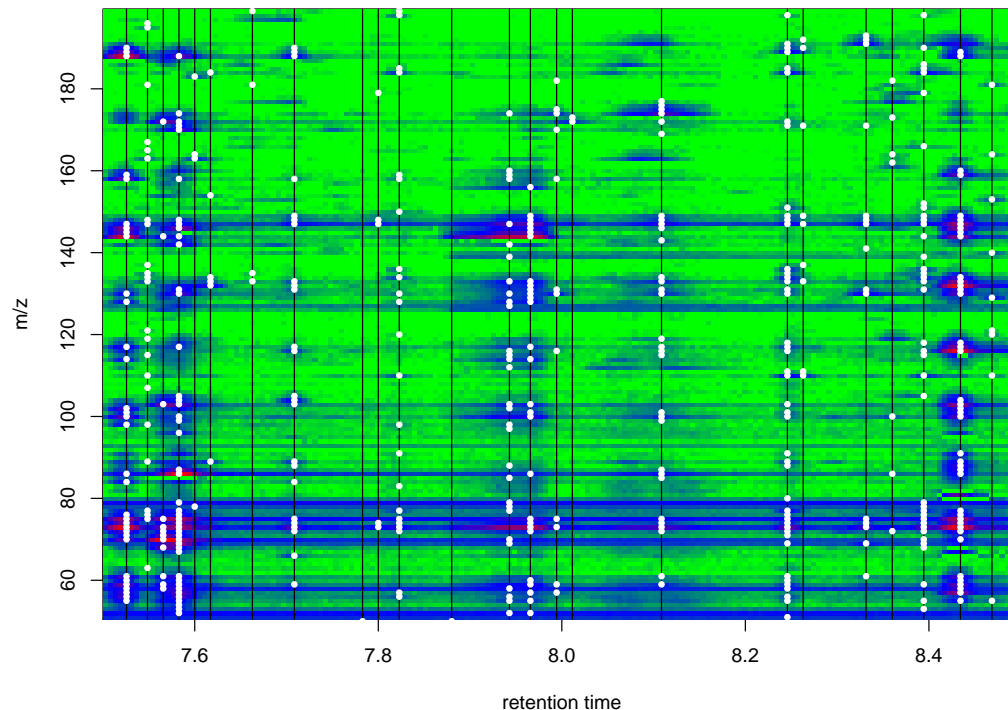
Note here the little *hashes* represent the detected peaks and are labeled with an integer index. One of the main challenges is to match these peak detections across several samples, given that they appear at slightly different times in different runs.

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 GCMS 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:

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

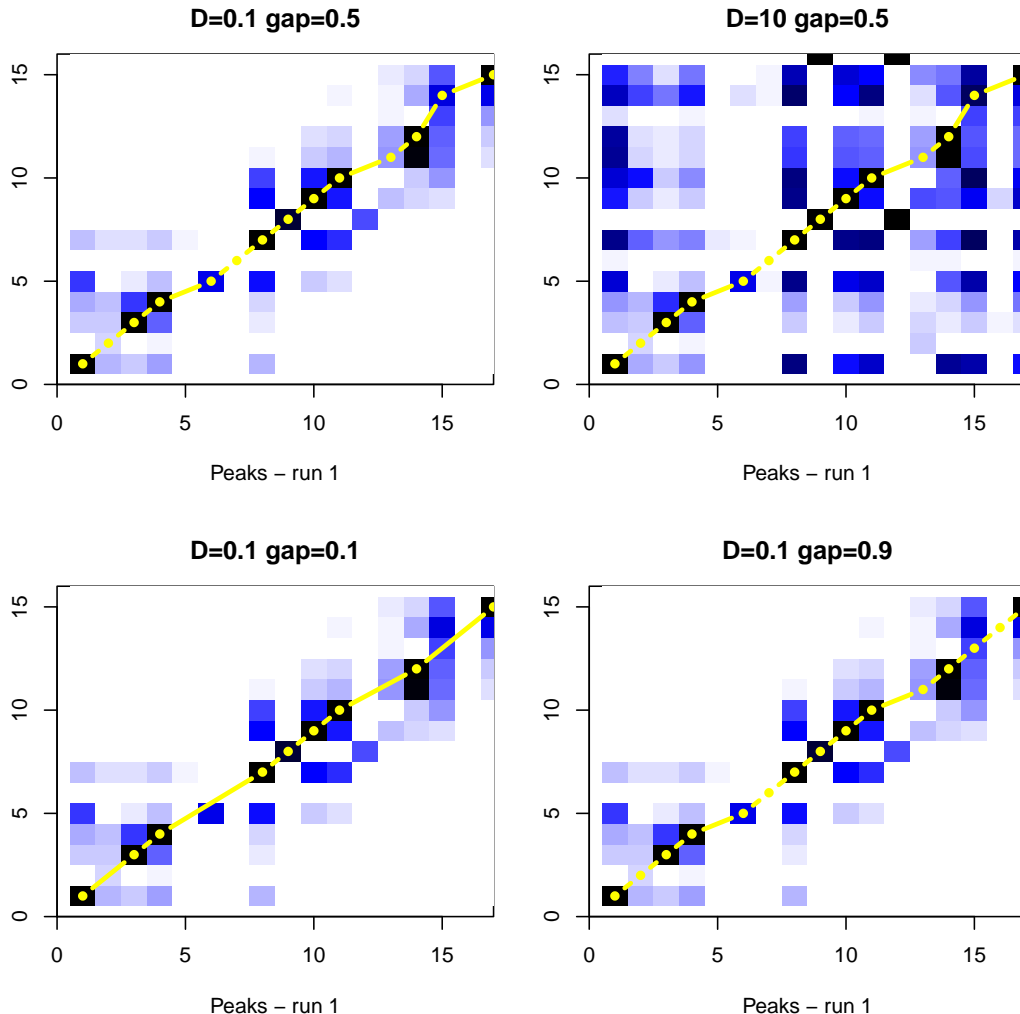
where i is the index of the peak in the first sample and j is the index of the peak in the second sample, \mathbf{x}_i and \mathbf{y}_j are the spectra vectors and t_i and t_j are their respective retention times. As you can see, there are two components to the similarity: spectra similarity (left term) 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 metrics 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 GCMS 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])
+   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= 0.1
[peaksAlignment] 21 matched. Similarity= 0.2634053
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.5 D= 10
[peaksAlignment] 21 matched. Similarity= 0.2458355
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.1 D= 0.1
[peaksAlignment] 14 matched. Similarity= 0.01748718
[peaksAlignment] Comparing 24 peaks to 23 peaks -- gap= 0.9 D= 0.1
[peaksAlignment] 22 matched. Similarity= 0.305486
```



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)
```

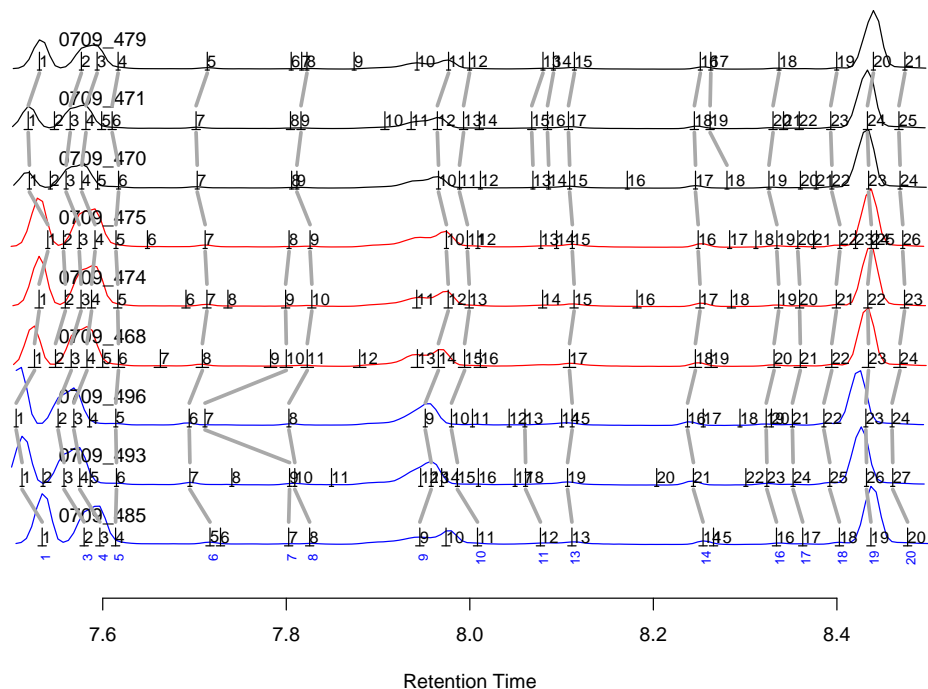
```
      FileName Group
1 0709_468.CDF  mmA
2 0709_474.CDF  mmA
3 0709_475.CDF  mmA
4 0709_485.CDF  mmC
5 0709_493.CDF  mmC
6 0709_496.CDF  mmC
7 0709_470.CDF  mmD
8 0709_471.CDF  mmD
9 0709_479.CDF  mmD
```

```
> ma<-multipleAlignment(pd,group=targets$Group,wn.gap=0.5,wn.D=.05,bw.gap=.6,bw.D=0.05,usePeaks=TRUE,fil
> ma
```

```
An object of class "multipleAlignment"
3 groups: 3 3 3 samples, respectively.
20 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,p
```



4.1 Gathering results

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

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

```
[1] 4 5 6 1 2 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	NA	NA	2	2	2	NA	NA	NA
[3,]	2	3	2	3	3	3	3	3	2
[4,]	3	4	3	4	4	4	4	4	3
[5,]	4	6	5	6	5	5	6	6	4
[6,]	5	7	6	8	7	7	7	7	5
[7,]	7	9	7	10	9	8	NA	NA	NA
[8,]	8	10	8	11	10	9	9	9	8
[9,]	9	13	9	14	12	10	10	12	11
[10,]	11	15	10	15	13	11	11	13	12
[11,]	12	18	13	NA	NA	NA	13	15	13
[12,]	NA	NA	NA	NA	NA	NA	14	16	14
[13,]	13	19	15	17	15	15	15	17	15
[14,]	14	21	16	18	17	16	17	18	16

[15,]	NA	NA	NA	NA	NA	NA	18	19	17
[16,]	16	23	19	20	19	19	19	20	18
[17,]	17	24	21	21	20	20	NA	NA	NA
[18,]	18	25	22	22	21	22	22	23	19
[19,]	19	26	23	23	22	24	23	24	20
[20,]	20	27	24	24	23	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.1	mmA.2	mmA.3	mmD.7	mmD.8
7.825483	7.809283	7.802867	7.822867	7.827850	7.825917	7.811567	7.816033
mmD.9							
7.822517							

\$mz

[1]	56	58	69	73	74	75	77	110	130	134	184	198	228	229
-----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----

\$data

	mmC.4	mmC.5	mmC.6	mmA.1	mmA.2	mmA.3	mmD.7	mmD.8	mmD.9
[1,]	3505	5436	3792	3309	4131	3855	3602	3623	3456
[2,]	15137	16416	14540	9769	14673	13421	14715	13545	16356
[3,]	6464	9006	6990	5046	7130	5890	5724	6070	6364
[4,]	29784	51504	43816	29144	27656	30128	35288	36120	28600
[5,]	3622	5307	5044	3372	2898	3853	3580	3592	3008
[6,]	23032	27304	23688	18328	22496	22552	23912	23712	23096
[7,]	4956	7138	5878	4193	4678	5120	4664	5058	4768
[8,]	3267	6172	5300	3630	3494	4428	4186	4397	3360
[9,]	1463	2612	2802	1705	1616	1671	2162	2035	1590
[10,]	3484	7093	5305	4134	3864	4347	3479	4537	3634
[11,]	6220	9409	6999	5283	5727	6318	5404	6298	5176
[12,]	1841	2567	1721	1547	1716	1741	1423	1901	1601
[13,]	12839	23904	16009	14955	12906	14824	12637	16320	11330
[14,]	6294	11575	7713	7004	6519	8306	6792	7446	4945

```
> 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.1	mmA.2	mmA.3	mmD.7	mmD.8	mmD.9
1	7.534	7.512	7.506	7.526	7.531	7.540	7.520	7.519	7.531
2	NA	NA	NA	7.549	7.559	7.557	NA	NA	NA
3	7.580	7.558	7.551	7.566	7.576	7.574	7.560	7.565	7.577
4	7.597	7.575	7.569	7.583	7.588	7.592	7.577	7.582	7.594

5	7.614	7.615	7.614	7.617	7.616	7.614	7.617	7.610	7.617
6	7.717	7.695	7.694	7.709	7.714	7.712	7.703	7.702	7.714
7	7.803	7.804	7.711	7.800	7.799	7.803	NA	NA	NA
8	7.825	7.809	7.803	7.823	7.828	7.826	7.812	7.816	7.823
9	7.946	7.958	7.951	7.966	7.976	7.975	7.966	7.965	7.977
10	8.008	7.986	7.980	7.994	7.999	7.997	7.989	7.993	8.000
11	8.077	8.061	8.060	NA	NA	NA	8.069	8.068	8.080
12	NA	NA	NA	NA	NA	NA	8.086	8.085	8.091
13	8.111	8.107	8.111	8.109	8.114	8.112	8.109	8.108	8.114
14	8.254	8.244	8.237	8.246	8.251	8.249	8.246	8.245	8.251
15	NA	NA	NA	NA	NA	NA	8.280	8.262	8.263
16	8.334	8.324	8.323	8.332	8.337	8.335	8.326	8.330	8.337
17	8.363	8.352	8.352	8.360	8.359	8.357	NA	NA	NA
18	8.403	8.392	8.386	8.394	8.399	8.403	8.395	8.393	8.400
19	8.437	8.432	8.432	8.434	8.434	8.437	8.435	8.433	8.440
20	8.477	8.461	8.460	8.469	8.474	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 GCMS 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. XCMS, 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 3.0.2 (2013-09-25)
Platform: i386-w64-mingw32/i386 (32-bit)

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] parallel stats graphics grDevices utils datasets methods
[8] base

other attached packages:

[1] flagme_1.18.0 xcms_1.38.0 Biobase_2.22.0 BiocGenerics_0.8.0
[5] mzR_1.8.0 Rcpp_0.10.5 gcsplikelite_1.0.9

loaded via a namespace (and not attached):

[1] MASS_7.3-29 SparseM_1.03 codetools_0.2-8 gdata_2.13.2
[5] gplots_2.11.3 gtools_3.1.0 tools_3.0.2

> date()

[1] "Tue Oct 15 00:16:18 2013"