Adaptive Processing of LC/MS Metabolomics data

Tianwei Yu

August 1, 2012

1. Introduction

The apLCMS package is designed for the processing of high resolution LC/MS data. The main characteristics include: (1) Model-based automatic searching for m/z and retention time tolerance levels from the data, (2) The use of run filter for noise reduction, (3) the use of non-parametric statistics for adaptive binning, (4) modeling of asymmetric peaks, (5) inference-based retention time deconvolution for peaks sharing m/z value, (6) hybrid feature detection that utilizes both de novo peak detection and knowledge on known features, and (7) parallel computing utilizing multiple cores in a computer.

The operation is file-based. Wrapper functions allow the processing of data using a single line of command. The package is capable of processing large amount of data in one batch (1320 spectra/30G tested so far).

2. Preparations for the analysis

1. The package apLCMS needs to be installed in R. It has a number of dependencies which are available from CRAN or Bioconductor. They include MASS, rgl, ncdf, splines, doSNOW, foreach, iterators, and snow.

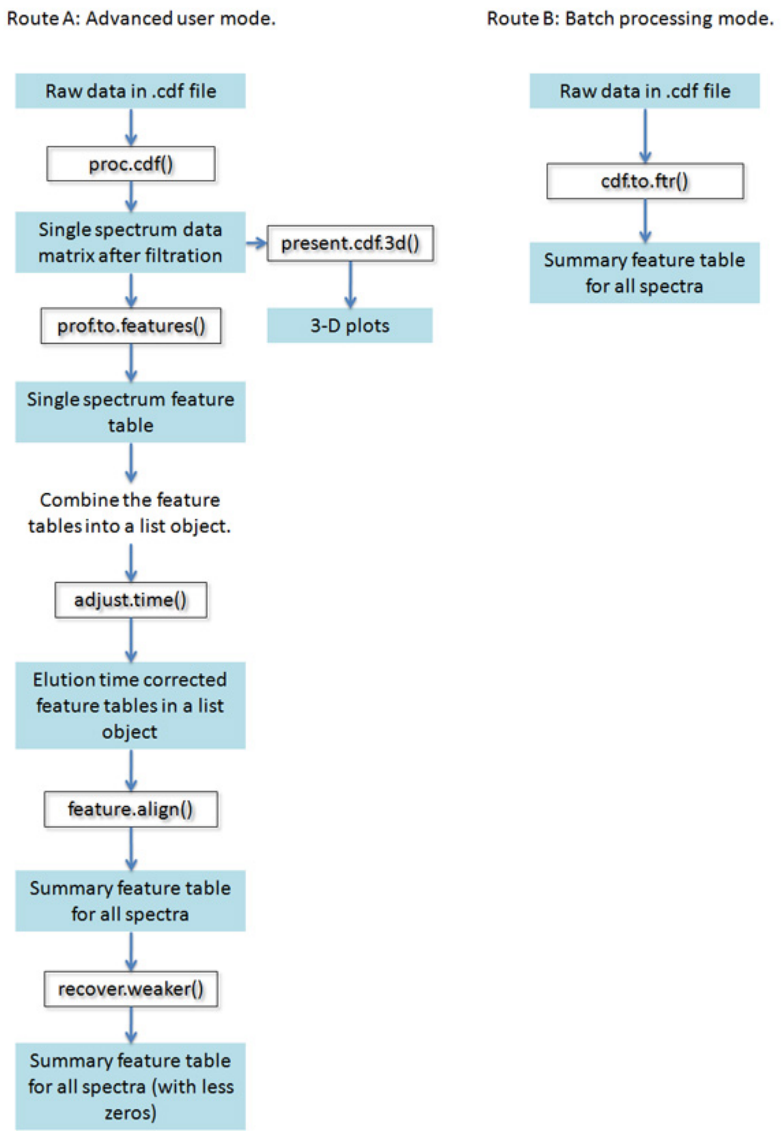
2. A folder for the project. All CDF files to be analyzed in a batch need to be stored in it. The analysis is file-based. So you don't need to worry about manipulating matrices etc. The apLCMS package will produce a number of intermediate files (don't worry, these files are much smaller than the CDF files) to speed up the process and to facilitate any re-analysis.

3. Major steps of unsupervised analysis

By "unsupervised" we mean analyzing the data by itself, not relying on any existing knowledge about metabolites. There are five major steps:

(1) From CDF format raw data, use run filter to reduce noise and select regions in the spectrum that correspond to peaks [proc.cdf()]
(2) Identify peaks from the spectrum and summarize peak location (m/z, retention time), spread and intensity [prof.to.features()]
(3) Retention time correction across spectra [adjust.time()]
(4) Peak alignment across spectra [feature.align()]
(5) Weak signal recovery [recover.weaker()]

For simplicity, there is a wrapper function cdf.to.ftr(). This function carries out the five steps above all the way in a single line of command. The figure below summarizes the functions involved in the analysis:


Route A: Advanced user mode.
Route B: Batch processing mode.
Raw data in .cdf file
proc.cdf()
Single spectrum data matrix after filtration
present.cdf.3d()
3-D plots
prof.to.features()
Single spectrum feature table
Combine the feature tables into a list object.
adjust.time()
Elution time corrected feature tables in a list object
feature.align()
Summary feature table for all spectra
recover.weaker()
Summary feature table for all spectra (with less zeros)
Raw data in .cdf file
cdf.to.ftr()
Summary feature table for all spectra

4. A demonstration of the unsupervised data analysis.

1. Download the package and install it in R.

2. Download the demonstration data and unzip it into a folder. I am using the folder "C:/apLCMS_demo" for this demonstration.

3. Open R. Load the package.

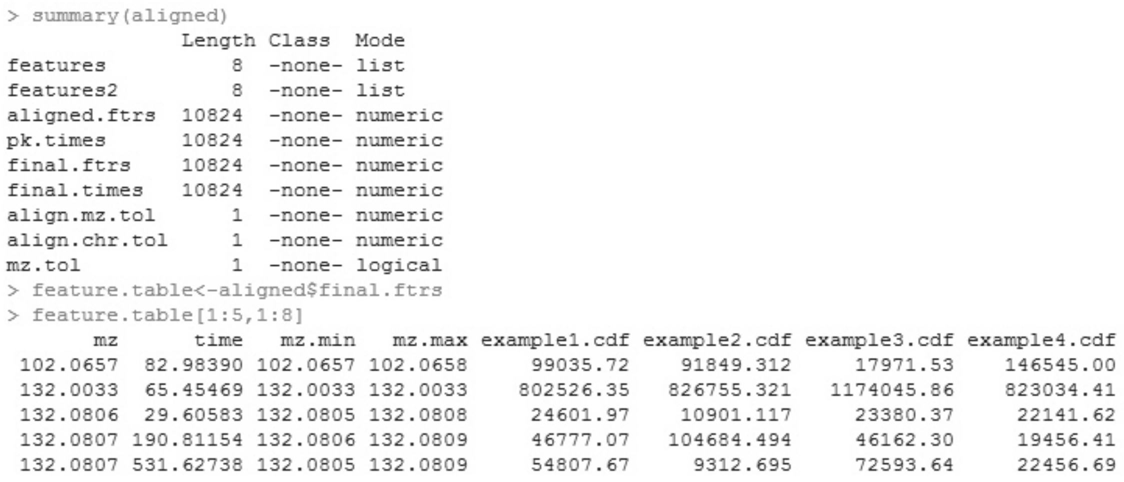
```
> library(apLCMS)
Loading required package: MASS
Loading required package: rgl
Loading required package: ncdf
Loading required package: splines
Loading required package: doSNOW
Loading required package: foreach
foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com
Loading required package: iterators
Loading required package: snow
```

4. Run the analysis of the sample data using the wrapper function cdf.to.ftr(). The display from the wrapper function are mainly the names of the intermediate files being written into the folder and the processing time (in seconds). Summary and disgnostic plots (detailed below) are generated in every step. Below is a sample output.

```
> folder<-"C:/apLCMS_demo"
> setwd(folder)
> aligned<-cdf.to.ftr(folder, n.nodes=4, min.exp=5,min.run=12,min.pres=0.5)

**************************** prifiles --> feature lists ****************************
***************************** time correction
***************************************
**** performing time correction ****
m/z tolerance level:  1.14260717038009e-05
time tolerance level: 82.5881547056058
the template is sample 8
sample 1 using 354 ,sample 2 using 409 ,sample 3 using 552 ,
sample 4 using 874 ,sample 5 using 753 ,sample 6 using 1047 ,
sample 7 using 819 ,***** correcting time, CPU time (seconds) 1.566
***************************  aligning features
**************************************
**** performing feature alignment ****
m/z tolerance level:  1.14260717038009e-05
time tolerance level: 64.2255581174968
```

***** aligning features, CPU time (seconds): 8.571
*************************** recovering weaker signals
****

5. Examining the results. A list object is returned from the function. "Features" is a list of matrices, each of which is a peak table from a spectrum. "Features2" is of the same structure except the retention time is corrected. "Aligned.ftrs" is the matrix of aligned features across all the spectra. "Pk.times" is the matrix of peak retention times of the aligned features. "Final.features" is what's most important. It is the aligned feature table after weak signal recovery, i.e. the end product. "Final times" is the accompanying peak retention time matrix. A small section of the final.features table is shown. The first column of the table is the m/z value; the second column is the median retention time; from the third column on are the signal strength in each spectrum.



```
> summary(aligned)
              Length Class  Mode
features          8  -none- list
features2         8  -none- list
aligned.ftrs  10824  -none- numeric
pk.times      10824  -none- numeric
final.ftrs    10824  -none- numeric
final.times   10824  -none- numeric
align.mz.tol      1  -none- numeric
align.chr.tol     1  -none- numeric
mz.tol            1  -none- logical
> feature.table<-aligned$final.ftrs
> feature.table[1:5,1:8]
       mz      time   mz.min   mz.max example1.cdf example2.cdf example3.cdf example4.cdf
 102.0657  82.98390 102.0657 102.0658     99035.72    91849.312     17971.53    146545.00
 132.0033  65.45469 132.0033 132.0033    802526.35   826755.321   1174045.86    823034.41
 132.0806  29.60583 132.0805 132.0808     24601.97    10901.117     23380.37     22141.62
 132.0807 190.81154 132.0806 132.0809     46777.07   104684.494     46162.30     19456.41
 132.0807 531.62738 132.0805 132.0809     54807.67     9312.695     72593.64     22456.69
```

This feature table is the key output. If you prefer not to work in R for downstream analysis, you can simply output it to a tab-delimited text file, which can be read into excel and other statistical softwares easily.

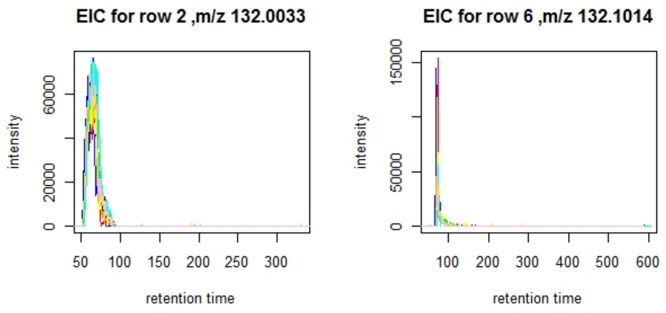
```
> write.table(feature.table,"result.txt",sep="\t",quote=F,col.names=T, row.names=F)
```

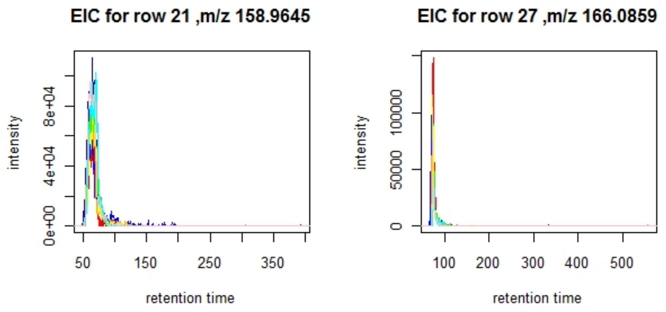
Now the table file called "result.txt" is in the working directory.

After finding some interesting features, you may want to examine the original extracted ion chromatograms of them. This can be done by using the function EIC.plot(). Here we demonstrate the usage of it by selecting some high intensity features that are present in all profiles. Notice "selection" is the rows of the feature table for which you wish to plot. It should be a vector like c(4,10,134,555).

```
> feature.table<-aligned$final.ftrs[,5:12]
> selection<-which(apply(feature.table!=0,1,sum)==8 & apply(feature.table,1,mean)>200000)
> EIC.plot(aligned, selection[1:4], min.run=12, min.pres=0.5)
the colors used are:
```

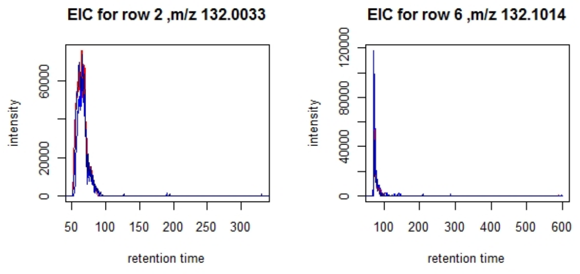
```
[1] "1: red" "2: blue" "3: dark blue" "4: orange" "5: green" "6: yellow"
[7] "7: cyan" "8: pink"
```

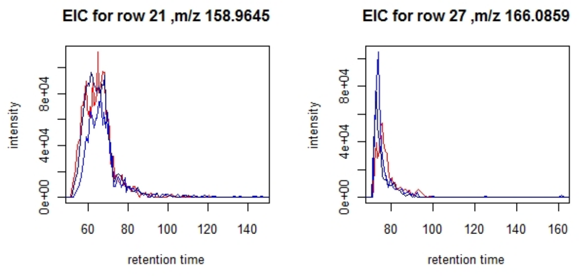




You can choose a subset of profiles for which to plot the EICs by setting a "subset" parameter. In the example below only EICs from samples 3, 6, and 8 are plotted.

```
> EIC.plot(aligned, selection[1:4],subset=c(3,6,8), min.run=12, min.pres=0.5)
the colors used are: [1] "3: red" "6: blue" "8: dark blue"
```



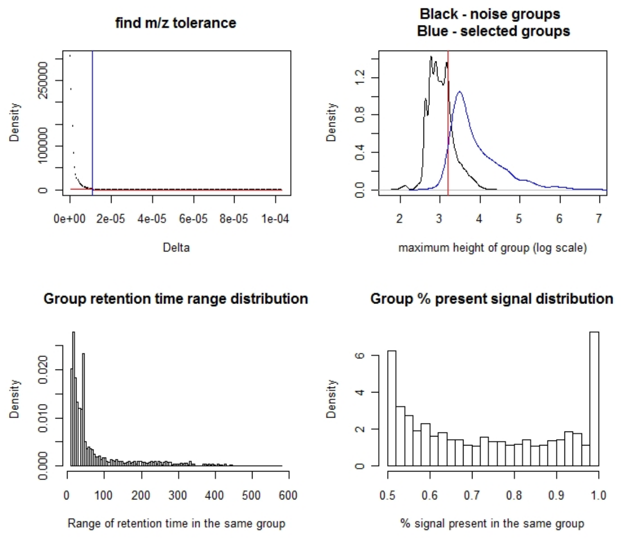


6. If you prefer not to use the wrapper, here is a demonstration of the 5 step processing.

```
> library(apLCMS)
> folder<-"C:/apLCMS_demo"
> setwd(folder)
> cdf.files<-dir(pattern=".cdf")
> cdf.files
[1] "example1.cdf" "example2.cdf" "example3.cdf" "example4.cdf" "example5.cdf"
[6] "example6.cdf" "example7.cdf" "example8.cdf"
```

6.1 Step (1), from cdf file to filtered data.

```
> this.spectrum<-proc.cdf(filename=cdf.files[4], min.pres = 0.5, min.run = 12, tol = 1e-5)
maximal height cut is automatically set at the 75th percentile of noise group heights: 1728.45823753139
```



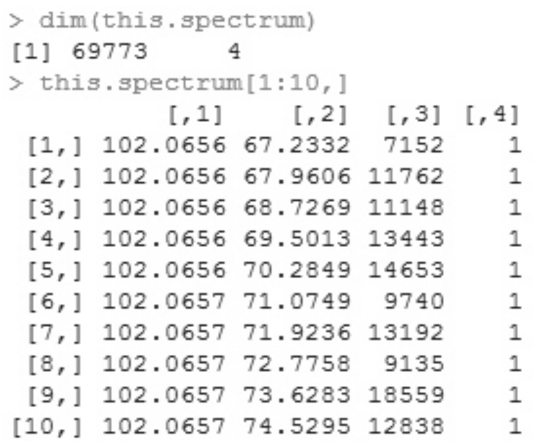
The upper-left plot is the search for the m/z tolerance.

The upper-right plot shows the distribution of the height of the signal groups. The black curve is from those groups with less intensity observations than the run filter requires (noise); the blue curve is from those passing the run filter (mostly signals). Some groups passing the run filter are eliminated because their intensities are low. The default cutoff is the redline-75th percentile from the black group. The user can input any cutoff using the parameter "baseline.correct".

The lower-left panel is the distribution of the groups' span in retention time. Because some metabolites share m/z, it is no surprise that a small number of groups have a very long span. Nonetheless the majority of the groups span <100 seconds.

The lower-right panel is the distribution of the proportion of time points where the group has an intensity observation. Because the run filter cut this value using the parameter min.pres, the minimum of the distribution is equal to min.pres. The maximum is certainly 1, where the group has no skipped time point.

Examining the matrix, there are four columns. The first is the median m/z value of the peak,the second is the time points where the m/z is observed, the third is the signal strength at the time point, and the fourth is the group number in case two peaks share the same m/z.

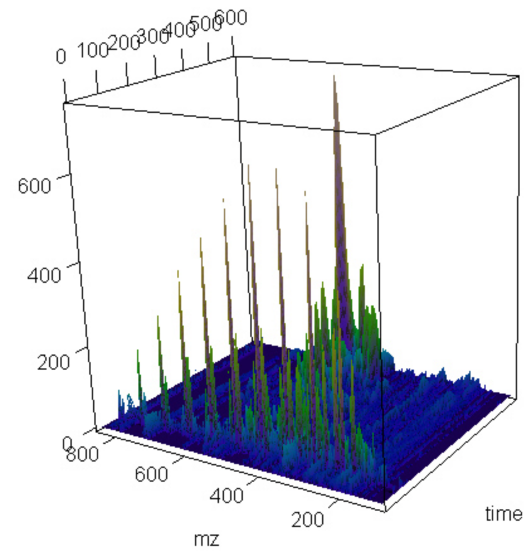


```
> dim(this.spectrum)
[1] 69773     4
> this.spectrum[1:10,]
          [,1]    [,2]  [,3] [,4]
 [1,] 102.0656 67.2332  7152    1
 [2,] 102.0656 67.9606 11762    1
 [3,] 102.0656 68.7269 11148    1
 [4,] 102.0656 69.5013 13443    1
 [5,] 102.0656 70.2849 14653    1
 [6,] 102.0657 71.0749  9740    1
 [7,] 102.0657 71.9236 13192    1
 [8,] 102.0657 72.7758  9135    1
 [9,] 102.0657 73.6283 18559    1
[10,] 102.0657 74.5295 12838    1
```

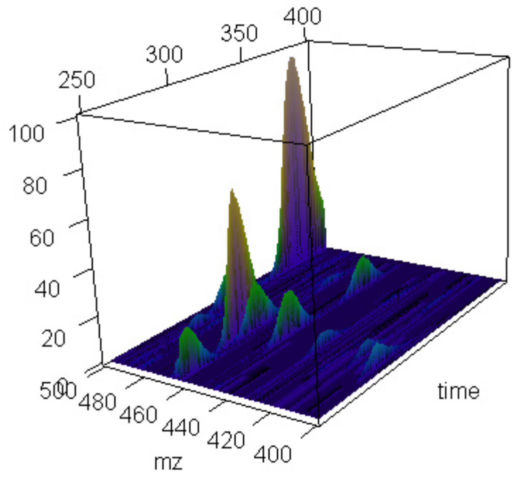
We can plot the spectrum or part of it.

> present.cdf.3d(this.spectrum,transform="sqrt")

Notice in these plots the intensity axis is of relative scale.

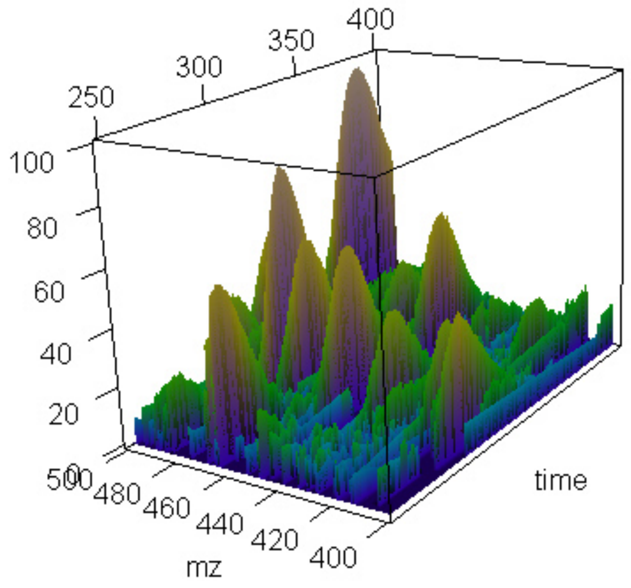


> present.cdf.3d(this.spectrum,time.lim=c(250,400), mz.lim=c(400,500),transform="none")
> ### (and this plot is shown)



Using cube-root transform helps bring out smaller peaks.

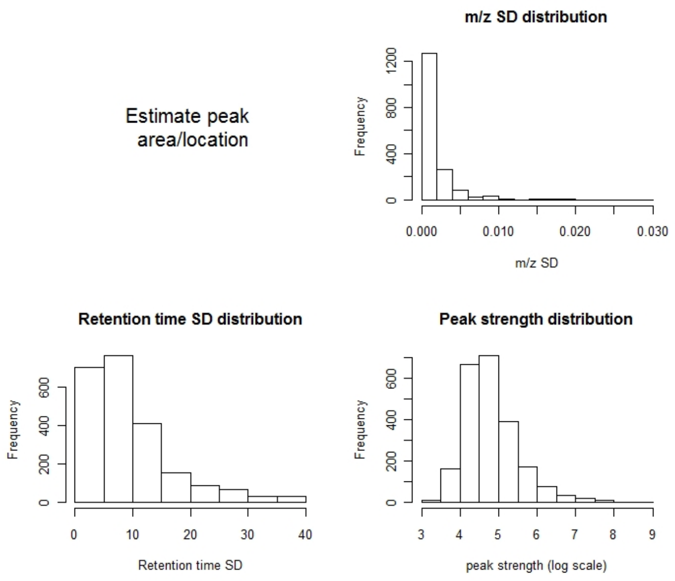
```
 > present.cdf.3d(this.spectrum,time.lim=c(250,400), mz.lim=c(400,500),transform="cuberoot")
                                              > ### (and this plot is shown)
```



6.2 Step (2), from the filtered data to feature table of the spectrum.

```
> this.peak.table<-prof.to.features(this.spectrum, bandwidth = 0.5, min.bw=10, max.bw=30)
```

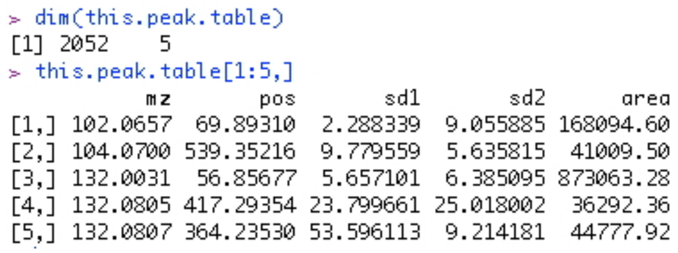
A plot is shown.



The upper-right panel is the distribution of the standard deviations of the m/z values in every feature. Notice we are using non-centroided data, which causes some spread around the true m/z. Compare this plot to the one generated by feature.align.

The lower-left panel is the distribution of the standard deviation of retention time in every feature.

The lower-right panel is the distribution of the log(peak area) of the features.


```
> dim(this.peak.table)
[1] 2052    5
> this.peak.table[1:5,]
           mz       pos       sd1       sd2      area
[1,] 102.0657  69.89310  2.288339  9.055885 168094.60
[2,] 104.0700 539.35216  9.779559  5.635815  41009.50
[3,] 132.0031  56.85677  5.657101  6.385095 873063.28
[4,] 132.0805 417.29354 23.799661 25.018002  36292.36
[5,] 132.0807 364.23530 53.596113  9.214181  44777.92
```

Every feature is represented by a row in this matrix. The first column is m/z, the second is retention time, the third is spread (standard deviation of the fitted normal curve), and the fourth is the estimated signal strength (peak area).

Now repeat the above step for every cdf file and obtain a list of all the peak tables. In the wrapper function this part is run in parallel. If you know how to use foreach(), replace the for loop with it, and this part is much faster.

```
> features<-new("list")
> for(i in 1:length(cdf.files))
+ {
```

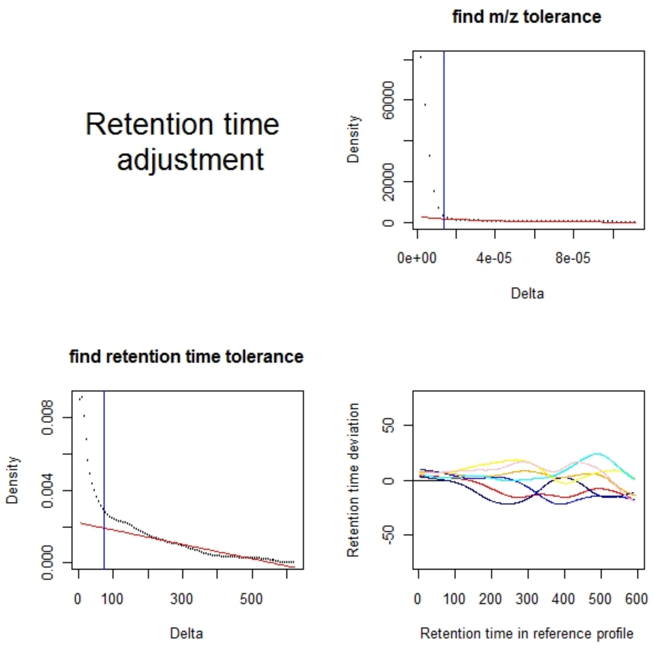
```
+ this.spectrum<-proc.cdf(filename=cdf.files[i], min.pres = 0.5, min.run = 10, tol = 1e-5)
+ this.peak.table<-prof.to.features(this.spectrum, bandwidth = 0.5, min.bw=10, max.bw=30)
+ features[[i]]<-this.peak.table
+ }
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1967.95020262201
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1566.88524222241
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1805.92912374766
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1679.73381566848
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights: 1503
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1529.99115330885
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1805.4999307671
m/z tolerance is:  1e-05
maximal height cut is automatically set at the 75th percentile of noise group heights:
1520.24846000009

> summary(features)
     Length Class  Mode
[1,] 10195  -none- numeric
[2,]  9955  -none- numeric
[3,] 11200  -none- numeric
[4,] 11750  -none- numeric
[5,] 11515  -none- numeric
[6,] 11450  -none- numeric
[7,] 11305  -none- numeric
[8,] 13475  -none- numeric
```

6.3 Step (3). Retention time adjustment. The procedure is based on non-parametric curve fitting. The number of features used for the curve-fitting is printed.

```
> features2<-adjust.time(features, mz.tol = NA, chr.tol = NA)
**** performing time correction ****
```

```
m/z tolerance level: 1.36718712335192e-05
time tolerance level: 75.532091457029
the template is sample 5
sample 1 using 566 ,sample 2 using 451 ,sample 3 using 847 ,
sample 4 using 1012 ,sample 6 using 935 ,
sample 7 using 1136 ,sample 8 using 872 ,
```

A plot is shown.



The upper-right panel shows the selection of m/z tolerance.

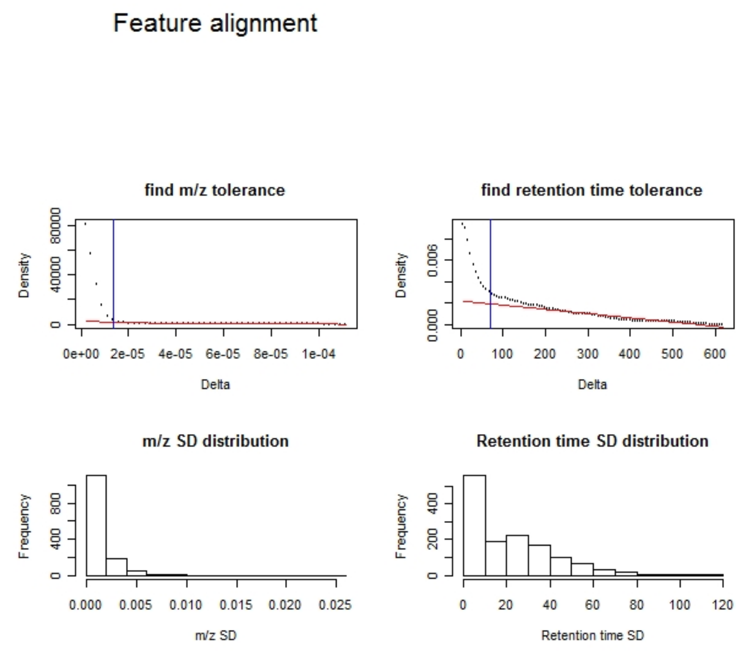
The lower-left panel shows the selection of retention time tolerance.

The lower-right panel shows the amount of adjustment for each of the profiles except the reference profile.

6.4 Step 4, feature alignment

```
> aligned<-feature.align(features2, min.exp = 4, mz.tol = NA, chr.tol = NA)
**** performing feature alignment ****
m/z tolerance level: 1.36718712335192e-05
time tolerance level: 68.811297036486
```

A plot is shown.

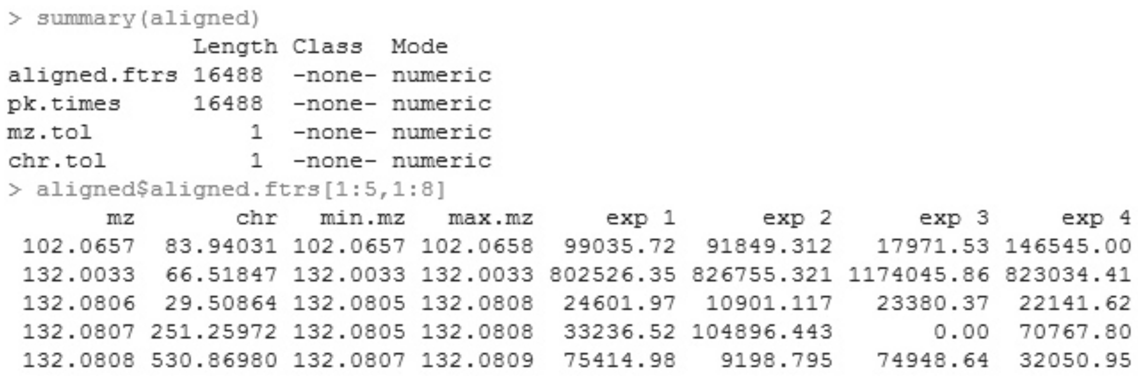


The upper-left panel shows the selection of m/z tolerance.

The upper-right panel shows the selection of retention time tolerance.

The lower-left panel shows the distribution of the standard deviation of m/z values of the features aligned.

The lower-right panel shows ths distribution of the standard deviation of retention times of the features aligned.



```
> summary(aligned)
             Length Class  Mode
aligned.ftrs 16488  -none- numeric
pk.times     16488  -none- numeric
mz.tol           1  -none- numeric
chr.tol          1  -none- numeric
> aligned$aligned.ftrs[1:5,1:8]
       mz       chr   min.mz   max.mz     exp 1      exp 2      exp 3     exp 4
 102.0657  83.94031 102.0657 102.0658  99035.72  91849.312   17971.53 146545.00
 132.0033  66.51847 132.0033 132.0033 802526.35 826755.321 1174045.86 823034.41
 132.0806  29.50864 132.0805 132.0808  24601.97  10901.117   23380.37  22141.62
 132.0807 251.25972 132.0805 132.0808  33236.52 104896.443       0.00  70767.80
 132.0808 530.86980 132.0807 132.0809  75414.98   9198.795   74948.64  32050.95
```

6.5 Step 5, weak signal recovery.

```
> post.recover<-aligned
> for(i in 1:length(cdf.files))
+ {
+     this.recovered<-recover.weaker(filename=cdf.files[i], loc=i,
aligned.ftrs=aligned$aligned.ftrs, pk.times=aligned$pk.times, align.mz.tol=aligned$mz.tol,
align.chr.tol=aligned$chr.tol, this.f1=features[[i]], this.f2=features2[[i]], orig.tol=1e-5)
+     post.recover$aligned.ftrs[,i+4]<-this.recovered$this.ftrs
+     post.recover$pk.times[,i+4]<-this.recovered$this.times
+     post.recover$features[[i]]<-this.recovered$this.f1
+     post.recover$f2[[i]]<-this.recovered$this.f2
+ }
>
>
> summary(post.recover)
             Length Class  Mode
aligned.ftrs 17028  -none- numeric
pk.times     17028  -none- numeric
mz.tol           1  -none- numeric
chr.tol          1  -none- numeric
features         8  -none- list
f2               8  -none- list
> post.recover$aligned.ftrs[1:5,1:8]
       mz       chr   min.mz   max.mz      exp 1     exp 2      exp 3     exp 4
 102.0658  71.98560 102.0657 102.0658 110441.458 114667.37   71112.78 168094.60
 132.0031  56.75482 132.0031 132.0032 926396.347 872208.26 1260537.10 873063.28
 132.1015  66.16977 132.1013 132.1017 484270.676 778707.78  313725.23 459622.23
 134.0447 237.98678 134.0447 134.0447  48146.521      0.00   59202.74  38482.00
 135.1015 329.44984 135.1015 135.1016   7664.599  88018.75   27330.81  52378.82
```

Comparatively, the feature matrix after recovery contains less zeros.

```
> sum(post.recover$aligned.ftrs==0)
[1] 2312
> sum(aligned$aligned.ftrs==0)
[1] 3625
```

The post.recover is the final result of the procedure. The central piece of information is the matrix post.recover$aligned.ftrs. Every row in this matrix is a feature. The first column is median m/z; the second column is median retention time; the third column is the minimum of the m/z in the profiles; the fourth column is the maximum of the m/z in the profiles; and from the fifth column on are the signal strength in each spectrum.

5. Major steps of hybrid analysis

By "hybrid" we mean incorporating the knowledge of known metabolites AND historically detected features on the same machinery to help detect and quantify lower-intensity peaks.

CAUTION: To use such information, especially historical data, you MUST keep using (1) the same chromatography system (otherwise the retention time will not match), and (2) the same type of samples with similar extraction technique, such as human serum.

First, it is necessary to put such information in a data frame that the program can use.

Second, **the first four steps** as in the unsupervised processing is conducted.

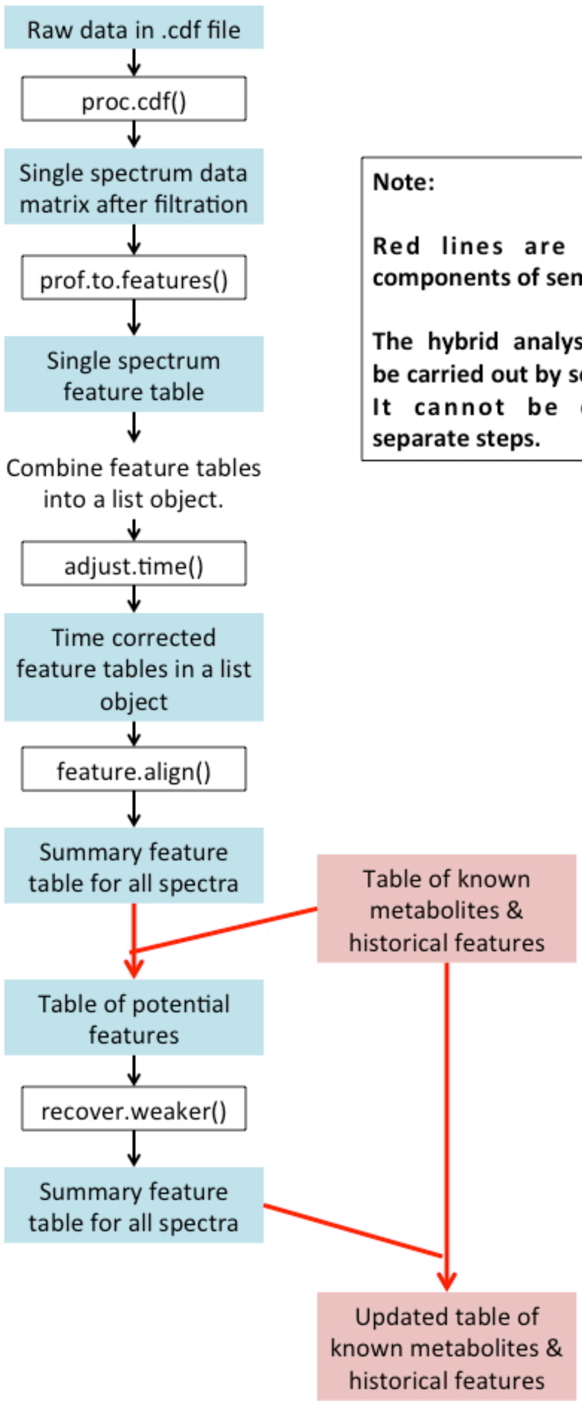
Third, the found features are merged with the data frame of known features.

Fourth, the weak signal recovery is performed using the merged feature table.

Fifth, the data frame of known metabolites and historical features is updated with the new information from the current data.

The hybrid approach is implemented in a wrapper function semi.sup(). This function carries out the above steps in a single line of command.

The figure summarizes the hybrid procedure:


Raw data in .cdf file
proc.cdf()
Single spectrum data matrix after filtration
prof.to.features()
Single spectrum feature table
Combine feature tables into a list object.
adjust.time()
Time corrected feature tables in a list object
feature.align()
Summary feature table for all spectra
Table of known metabolites & historical features
Table of potential features
recover.weaker()
Summary feature table for all spectra
Updated table of known metabolites & historical features

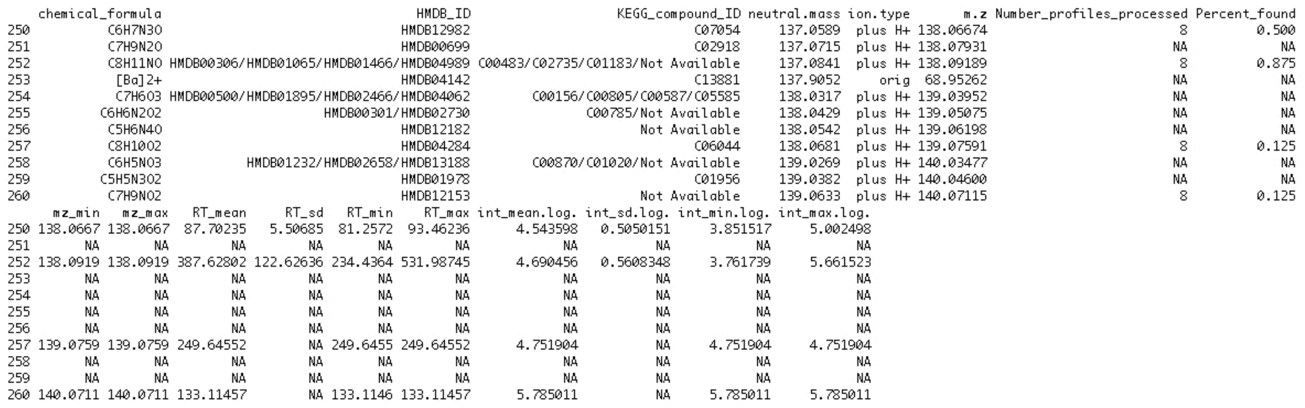
Note:

Red lines are internal components of semi.sup().

The hybrid analysis has to be carried out by semi.sup(). It cannot be done in separate steps.

The table of known metabolites and historical features needs to be provided to the subroutine semi.sup(). The data frame is like a matrix, but different column can be different variable types. We put an example data frame in the package. The measurement variability information is NA in that data frame, because it was built from the Human Metabolome Database alone. It only contains H+ derivatives of known metabolites. After running your first batch of samples, the database will be more populated. The provided database is mainly for demonstration purposes. You can build your own database using the metabolites and ion/isotope forms of your choice.

Here's a small portion of such a table.



```
    chemical_formula                                 HMDB_ID                   KEGG_compound_ID neutral.mass ion.type       m.z Number_profiles_processed Percent_found
250          C6H7N3O                               HMDB12982                             C07054     137.0589  plus H+ 138.06674                         8         0.500
251          C7H9N2O                               HMDB00699                             C02918     137.0715  plus H+ 138.07931                        NA            NA
252          C8H11NO HMDB00306/HMDB01065/HMDB01466/HMDB04989 C00483/C02735/C01183/Not Available     137.0841  plus H+ 138.09189                         8         0.875
253           [Ba]2+                               HMDB04142                             C13881     137.9052     orig  68.95262                        NA            NA
254           C7H6O3 HMDB00500/HMDB01895/HMDB02466/HMDB04062        C00156/C00805/C00587/C05585     138.0317  plus H+ 139.03952                        NA            NA
255         C6H6N2O2                     HMDB00301/HMDB02730               C00785/Not Available     138.0429  plus H+ 139.05075                        NA            NA
256          C5H6N4O                               HMDB12182                      Not Available     138.0542  plus H+ 139.06198                        NA            NA
257          C8H10O2                               HMDB04284                             C06044     138.0681  plus H+ 139.07591                         8         0.125
258          C6H5NO3           HMDB01232/HMDB02658/HMDB13188        C00870/C01020/Not Available     139.0269  plus H+ 140.03477                        NA            NA
259         C5H5N3O2                               HMDB01978                             C01956     139.0382  plus H+ 140.04600                        NA            NA
260          C7H9NO2                               HMDB12153                      Not Available     139.0633  plus H+ 140.07115                         8         0.125
      mz_min   mz_max   RT_mean     RT_sd   RT_min    RT_max int_mean.log. int_sd.log. int_min.log. int_max.log.
250 138.0667 138.0667  87.70235   5.50685  81.2572  93.46236      4.543598   0.5050151     3.851517     5.002498
251       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
252 138.0919 138.0919 387.62802 122.62636 234.4364 531.98745      4.690456   0.5608348     3.761739     5.661523
253       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
254       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
255       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
256       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
257 139.0759 139.0759 249.64552        NA 249.6455 249.64552      4.751904          NA     4.751904     4.751904
258       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
259       NA       NA        NA        NA       NA        NA            NA          NA           NA           NA
260 140.0711 140.0711 133.11457        NA 133.1146 133.11457      5.785011          NA     5.785011     5.785011
```

The columns are:
"chemical_formula": the chemical formula if knonw;
"HMDB_ID": HMDB ID if known;
"KEGG_compound_ID": KEGG compound ID if known;
"neutral.mass": the neutral mass if known:
"ion.type": the ion form, such as H+, Na+, ..., if known;
"m.z": m/z value, either theoretical for known metabolites, or mean observed value for unknown but previously found features;
"Number_profiles_processed": the total number of LC/MS profiles that were used to build this database;
"Percent_found": in what percentage was this feature found historically amount all data processed in building this database;
"mz_min": the minimum m/z value observed for this feature;
"mz_max": the maximum m/z value observed for this feature;
"RT_mean": the mean retention time observed for this feature;
"RT_sd": the standard deviation of retention time observed for this feature;
"RT_min": the minimum retention time observed for this feature;
"RT_max": the maximum retention time observed for this feature;
"int_mean.log.": the mean log intensity observed for this feature;
"int_sd.log.": the standard deviation of log intensity observed for this feature;
"int_min.log.": the minimum log intensity observed for this feature;
"int_max.log.": the maximum log intensity observed for this feature.

The following is a demonstration with the demo dataset. All the material copied from R are in black, and all other comments are in blue.

1. Download the package and install it in R.

2. Download the data and unzip it into a folder. I am using the folder "C:/apLCMS_demo" for this demonstration.

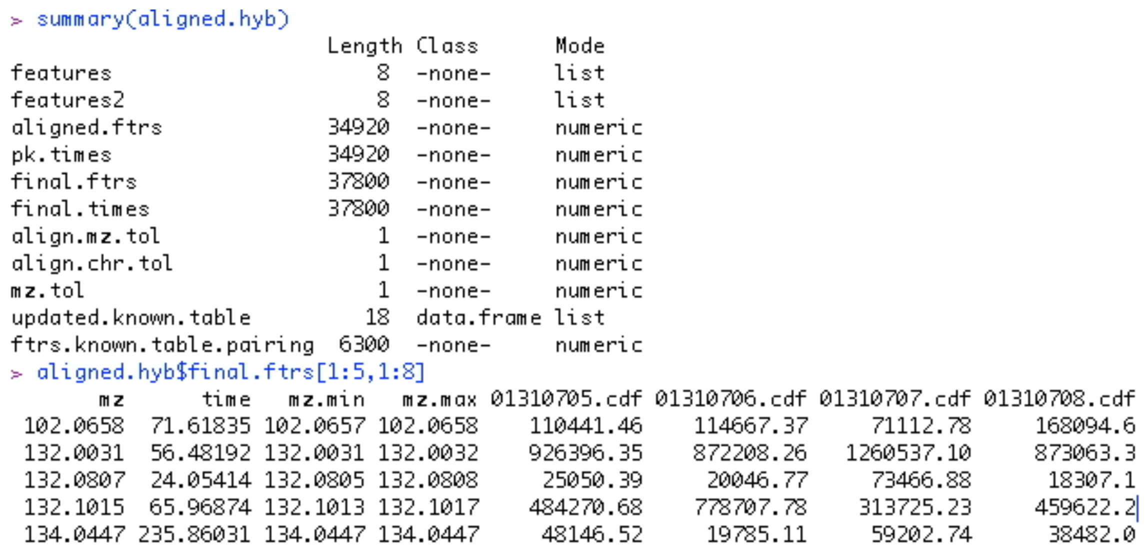
3. Open R. Load the package.

```
> library(apLCMS)
Loading required package: MASS
Loading required package: rgl
Loading required package: ncdf
Loading required package: splines
Loading required package: doSNOW
Loading required package: foreach
foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com
Loading required package: iterators
Loading required package: snow
```

4. Run the analysis of the sample data using the wrapper function semi.sup().

```
> folder<-"C:/apLCMS_demo"
> setwd(folder)
> data(known.table.hplus)
> aligned.hyb<-semi.sup(folder, known.table=known.table.hplus, n.nodes=4, min.pres=0.5,
min.run=12, mz.tol=1e-5, new.feature.min.count=4)

**************************** prifiles --> feature lists ****************************
***************************** time correction
****************************************
**** performing time correction ****
m/z tolerance level:  1.14260717038009e-05
time tolerance level: 82.5881547056058
the template is sample 8
sample 1 using 352 ,sample 2 using 410 ,sample 3 using 554 ,
sample 4 using 876 ,sample 5 using 757 ,sample 6 using 1047 ,
sample 7 using 843 ,***** correcting time, CPU time (seconds) 1.71
****************************  aligning features
**************************************
**** performing feature alignment ****
m/z tolerance level:  1.14260717038009e-05
time tolerance level: 70.6474780161716
***** aligning features, CPU time (seconds): 19.913
merging to known peak table
*************************** recovering weaker signals
*****************************
```

5. Examining the results. A list object is returned from the function. "Features" is a list of matrices, each of which is a peak table from a spectrum. "Features2" is of the same structure except the retention time is corrected. "Aligned.ftrs" is the matrix of aligned features across all the spectra. "Pk.times" is the matrix of peak retention times of the aligned features. "Final.features" is what's most important. It is the aligned feature table after weak signal recovery, i.e. the end product. "Final times" is the accompanying peak retention time matrix. A small section of the final.features table is shown. The first column of the table is the m/z value; the second column is the median retention time; from the third column on are the signal strength in each spectrum.



```
> summary(aligned.hyb)
                        Length Class      Mode
features                    8  -none-     list
features2                   8  -none-     list
aligned.ftrs            34920  -none-     numeric
pk.times                34920  -none-     numeric
final.ftrs              37800  -none-     numeric
final.times             37800  -none-     numeric
align.mz.tol                1  -none-     numeric
align.chr.tol               1  -none-     numeric
mz.tol                      1  -none-     numeric
updated.known.table        18  data.frame list
ftrs.known.table.pairing  6300 -none-     numeric
> aligned.hyb$final.ftrs[1:5,1:8]
       mz      time   mz.min   mz.max 01310705.cdf 01310706.cdf 01310707.cdf 01310708.cdf
 102.0658  71.61835 102.0657 102.0658    110441.46    114667.37     71112.78     168094.6
 132.0031  56.48192 132.0031 132.0032    926396.35    872208.26   1260537.10     873063.3
 132.0807  24.05414 132.0805 132.0808     25050.39     20046.77     73466.88      18307.1
 132.1015  65.96874 132.1013 132.1017    484270.68    778707.78    313725.23     459622.2
 134.0447 235.86031 134.0447 134.0447     48146.52     19785.11     59202.74      38482.0
```

This feature table is the key output. If you prefer not to work in R for downstream analysis, you can simply output it to a tab-delimited text file, which can be read into excel and other statistical softwares easily.

```
> write.table(aligned.hyb$final.ftrs, "result.txt",sep="\t",quote=F,col.names=T, row.names=F)
```

Now the table file called "result.txt" is in the working directory.

The item "aligned.hyb$updated.known.table" is the updated known feature table. For future data analysis, you should use the updated table.

```
> known.table.new<-aligned.hyb$updated.known.table
> save(known.table.new, file="updated_table.bin")
```

In the processing of next batch of data,

```
> load("updated_table.bin")
> aligned.hyb<-semi.sup(......, known.table=known.table.new, .......)
```

6. Choice of parameters

These are parameters of the two major wrapper functions cdf.to.ftr() or semi.sup(). If you are savvy in R and choose to do it in a stepwise manner, you can easily see the parameters with the same names are passed to the individual functions by the wrapper. Thus you can use the same guidelines here.

The default parameters of apLCMS were optimized for the high-resolution LC-FTMS. We received multiple requests on parameter optimization for data from other machinery. Here we provide some comments/guidelines.

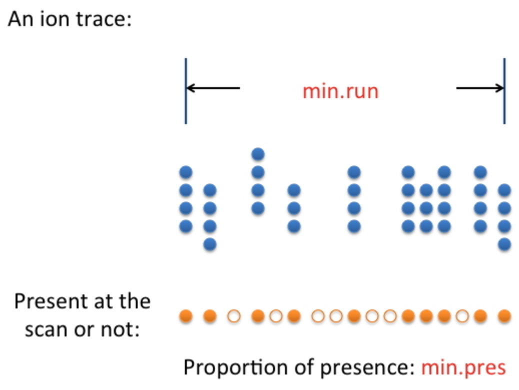
Critical parameters for computation:

n.nodes: The number of cores to use in a multi core machine. See the [computation] section below for guidelines.

Critical parameters for peak detection in a single spectrum:

(1) ***min.run:*** the minimum footprint of a real feature in terms of retention time. It is closely related to the chromatography method used. How long should the retention time be for a real feature? Use that value or a value that's slightly smaller.

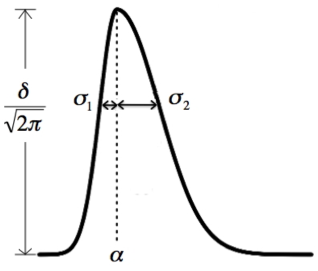
(2) ***min.pres:*** Sometimes the signal from a real feature isn't present 100% of the time along the feature's retention time. The parameter min.pres sets the threshold for an ion trace to be considered a feature. This parameter is best determined by examining the raw data.



(3) ***mz.tol*** and ***align.mz.tol:*** these are m/z tolerance levels. For un-centroided data from FTMS machinery, these values are automaticly determined based on point distribution patterns in the data. However, for some lower-resolution data, especially when centroided, the automatic procedure may not yield the optimal result. Based on the machine's characteristics or examining the raw data - how far along m/z do two points need to be separated for them to be considered

different? Use this value for mz.tol and align.mz.tol. An easy way is to use the machine's nominal resolution.

(4) ***sd.cut:*** this is a similar situation as min.run. It depends on how fast the elution of each peak is. This parameter (a vector of two numbers) sets the limits of the minimum and maximum standard deviation along rention time for an ion trace to be a real feature. Essentially four times the smaller value is the shortest retention time that is allowed, and four times the larger value is the longest retention time that's allowed. The picture below shows the bi-Gaussian model. It can be skewed to the other direction too. The sd.cut parameter sets the minimal and maximal value allow for either sigma.1 and sigma.2 in the picture.



To aid parameter selection, we provide the function plot.2d() for the user to examine the detected features overlaid on the raw data. Set a small m/z and retention time range to zoom in to examine the pattern.

Critical parameters for determining features across multiple spectra in a project:

min.exp: If a feature is to be included in the final feature table, it must be present in at least this number of spectra.

Critical parameters for hybrid feature detection:

match.tol.ppm: The ppm tolerance to match identified features to known metabolites/features. Similar to mz.tol, this depends on the mass accuracy of your machine.

new.feature.min.count: The number of spectra a new feature must be present for it to be added to the database. We recommend setting this parameter in a stringent manner.

recover.min.count: The minimum time point count for a series of point in the EIC for it to be considered a true feature in supervised detection.

7. Computation

The computation time depends on the information contained in the spectra and the parameters. At the default setting, from one CDF file to its feature table, a laptop computer with a core 2

DUO CPU at 2.2 GHz takes about two minutes when the CDF file is about 30Mb. Retention time correction and feature alignment don't take too much time. Weaker signal recovery may take longer, as every spectrum is re-examined.

There are built-in mechanisms to speed up compuation.

(1) You can use multiple cores/machines to simultaneously perform steps 1 and 2 (from CDF file to per-spectrum feature table). The function cdf.to.ftr() has a pre-processing mode, in which it can work on only a subset of the CDF files, and save partially processed information in binary files. Later on, simply re-run the function cdf.to.ftr() or semi.sup() and it will look for the partially processed information in binary files and skip the first two steps if the files are present.

(2) Both and   can utilize multiple cores on a machine. It is controlled by the ***n.nodes*** parameter in the wrapper functions. How many nodes you can use depends on two things: (1) the availability of cores in the machine, and (2) the size of RAM. A rule of thumb is to allow 2Gb RAM for each process.

8. References

apLCMS--adaptive processing of high-resolution LC/MS data.
Yu T, Park Y, Johnson JM, Jones DP. Bioinformatics. 2009 Aug 1;25(15):1930-6.

Quantification and deconvolution of asymmetric LC-MS peaks using the bi-Gaussian mixture model and statistical model selection.
Yu T, Peng H. BMC Bioinformatics. 2010 Nov 12;11:559.

A practical approach to detect unique metabolic patterns for personalized medicine.
Johnson JM, Yu T, Strobel FH, Jones DP. Analyst. 2010 Nov;135(11):2864-70.