Using XCMS and R to explore LC-MS data-sets

You will need R and XCMS installed on your PC. These are free, and can be obtained from http://www.stats.bri.ac.uk/R/ for R, and http://www.bioconductor.org/ for XCMS.

(1) File format conversion

XCMS needs its files in an open-access common format, not in the proprietary format of a manufacturer. For Thermo files I use ReadW which is available from http://sourceforge.net/projects/sashimi/files/ to do the conversion, but there are others (e.g. MSconvert). The best format for XCMS is mzXML, although it can use some others.

To convert, copy the readw.exe file into the same place as the data files (filename.raw) and use the dos "for" instruction, which allows you to repeat an instruction for a large number of files.

```
For %a in (*.raw) do readw %a
```

(2) Sort out the files into folders

XCMS was designed to compare two treatments, and expects the filename.mzXML files to be distributed between two folders to match the treatments. Even if you have multiple treatments it helps to make the two subfolders. If I have carried out QC runs using repeated injections of a mixed sample, I put these in one folder, and all the sample files in the other. Do not include blanks and standards (as these won't have peaks for XCMS to find and match).

(3) Peak finding with XCMS

Open R, and tell it that we would like to use the XCMS library, which you should already have installed (using the "Packages" menu). You only need to install it once, but every time you open R you need to tell it that you will be using XCMS:

```
library(xcms)
```

Note that XCMS is case-sensitive.
Using the File menu, "Change dir…", navigate to the directory that contains your two sub-directories of data.
Now create an object (you can use any name, e.g. "A"), and put in it the results of XCMS's function xcmsSet, which makes a peak-list for each individual file included in the subdirectories:

```
A <- xcmsSet()
```

You do not generally need to use any parameters with "xcmsSet" because the defaults are good enough. If you use data from UPLC, CE or GC, with narrower peak-widths than conventional LC, you will need to explore parameters. Typically you might have to specify the peak-width at half-height (fwhm) in seconds. If you expect many peaks of the same mass at different retention times (common in GC-MS, where many chemicals share fragments), you need to increase the default maximum number of peaks per selected mass chromatogram:

```
B <- xcmsSet(fwhm=5, max=30)
```

A is a very large object containing lots of information. If you want to investigate it, you might first like to tell R never to print out more than 50 lines of information about repetitive objects:

```
options(max.print = 50)
```

If you can't remember how to use an instruction, or what parameters it can take, there is a help-system.

```
help(options)
```

To find out more about A, or any other object you create, you can type:

```
attributes(A)
```

This tells you what sorts of things A contains, but not the actual values. If you want to see the value held by an object, just type its name

```
A
```

In the case of a huge object like A, you won't see every value, just some predefined information that the writers of xcmsSet thought might be helpful. Smaller, simpler objects just print their values. For example you could make an object B and assign it the value 3:

```
B <- 3
```

This will return "NULL" if you look at its attributes, because it is so small that R hasn't bothered remembering anything about it. If you type "B", you will get the result "3".

A is a collection of peak-lists, but at the moment each data-file has its own peak-list, and they are not cross-referenced. Peak 1 in file A is not necessarily the same as Peak 1 in file B. The next step is to let XCMS create a grouping of peaks such that all peaks of similar mass and retention time in all files are given a common number and recognised as a signal derived from one single chemical.

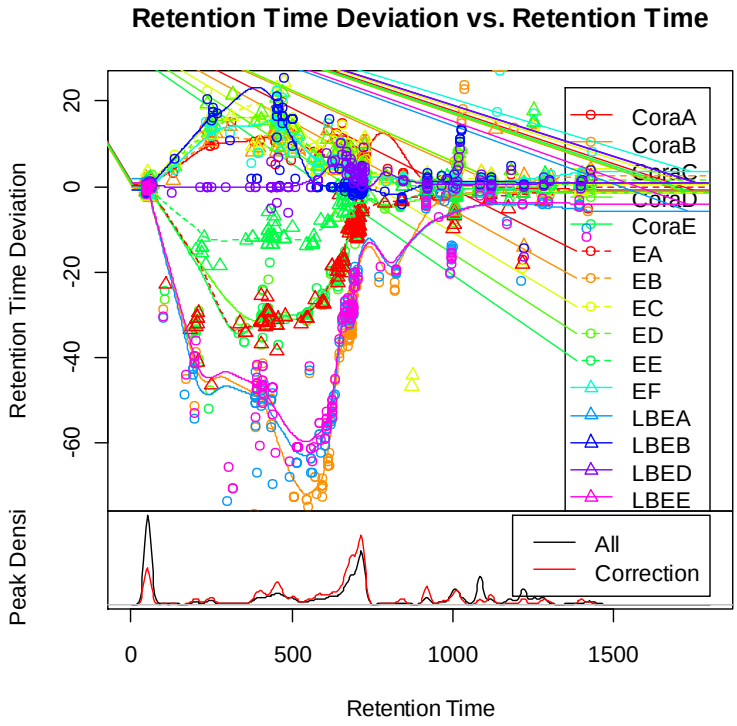
```
A <- group(A)
```

Using peaks that are common to all samples, and reasonably intense, XCMS can estimate how retention times have varied between runs. Using a running average of the "marker" peaks that it has chosen, it will recalibrate the time-axis of all the samples. This is done by retcor:

```
A2 <- retcor(A, family="s", plottype="m")
```

Note that we have used some parameters. Because R is case-sensitive, if you pass a parameter with the wrong capitalisation, it won't be recognised. R is object-orientated, which means each function is built on others, and parameters can be passed to function1 which are actually only used by subfunction2; as a result, function1 will simply pass anything it doesn't recognise to subfunction2 without complaint. In fact no function will complain about a spurious parameter, so if you type "Family" instead of "family" you will get no error, but the function won't do what you expect. Beware!
Some parameters need not be passed in full. "s" is short for "symmetric", which means a re-descending M estimator is used with Tukey's biweight function, allowing outlier removal (this is obviously exactly what we intended; if you don't know what a redescending M estimator is…)

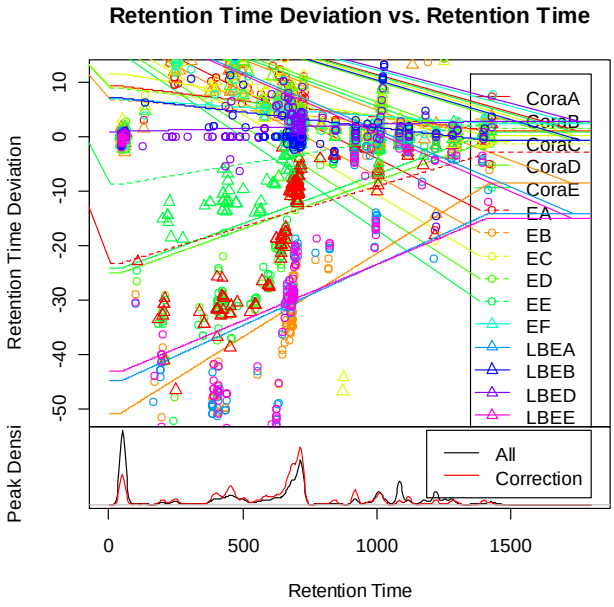
Retention time correction should produce a plot:



This shows the marker peaks as symbols, plotted at their corrected retention time on the x-axis, and with their deviation in each sample in seconds as the y-axis. The lines are a moving average that will be used to correct retention times of all remaining peaks, each sample having its own moving average.

Sometimes if there are not many marker peaks, or they are unevenly spaced, there may be wild digressions far beyond reasonable error. If this happens, you may prefer a linear best fit:

```
A2 <- retcor(A, family="s", plottype="m", method="linear")
```



We assign the results to a new object, A2, because we have actually changed the retention times of the peaks in the peak-list.

If retention time correction fails, it is possible to go ahead using the un-corrected data.

A2 is no longer grouped, because we've moved all the peaks, and some may no longer be in the right place to group with their original groupees, while others will have been moved closer to their equivalents in other files, and can now be grouped. We re-group, but with higher stringency:

```
A2 <- group(A2, bw=10)
```

Some peaks will not exist in some files. Rather than create a peak-list full of zero values, which can cause problems to downstream statistics, we want a best estimate of the real signal corresponding to that mass at that retention time in the samples where no peak was found. XCMS adds up the baseline in this case:

```
A3 <- fillPeaks(A2)
```

Finally, we create the table of peak-areas and other information. The original intent of XCMS was to look for differences, so this is a difference report.

```
Reporttab <- diffreport(A3, "SubdirectoryA", "SubdirectoryB", "myOutput", 5)
```

SubdirectoryA/B are where the original mzXML files were placed. MyOutput will become a tab-separated value file myOutput.tsv containing the peak-list. The number at the end is a strange thing:

Diffreport expects to compare two treatments, so it will carry out t-tests for each peak in the two subdirectory-groups, and it will order the results by decreasing significance. It expects that we will want to look at the most significant peaks to check they are real. Therefore it produces graphic files for the top "n", which we specify. This is the "5". Since our t-test comparison is meaningless if we have more than 2 treatments, we don't really need to produce these graphics, but if we type 1 or less, we get a strange error. Hence 5.

Diffreport does create a new object, Reporttab, but I do not recommend you try to use it. I find that columns of data within it that should be numerical become factors (non-numerical data), and I haven't found a way to make R treat them as numerical inside R. The best thing is to re-import the tab-separated value file, which (usually) seems to turn numerical values back into numbers.

(4) Trimming bad peaks

Some of the peaks that xcms has found are likely to be very small, or noise. It's possible to carry on and analyse data without removing these peaks, but they will make it harder to find the true differences. The best approach is to use a QC sample, a mixture of all the other samples (therefore containing all peaks) run repeatedly throughout the sequence. This should give the same area for every peak each time; peaks which vary are probably bad peaks. I show below how to do this in R. If a QC sample is not available, then we can look at the variability within replicates of a sample; I show this for Excel.

(4a) Using a QC sample

Read in the diffreport output into some convenient object:

```
Rtb <- read.table("myOutput.tsv")
```

Check what columns it contains, and find their numbers:

```
attributes(Rtb)
```

Rtb is a list of columns, each of which is itself a list of something else. R allows us to refer to just some of the columns. If our QC samples have their data in columns 14 to 17, we can select just these and put them in a separate table:

```
QCtb <- Rtb[14:17]
```

The measure of variability is the relative standard deviation. For this we first need the actual standard deviation of each *row*. Unfortunately R's instruction for calculating standard deviations does it for each *column*. Fortunately, we can transpose our table (turn rows into columns. But the transpose instruction (t) expects the data to be a matrix (2-dimensional grid of numbers) whereas what we actually have is a thing called a data-frame (a list of vectors; a list of columns that are all the same length, but not necessarily all containing the same sort of data). Fortunately since our data-frame has columns all of which are numerical, we can turn it into a matrix using as.matrix. These are all combined:

```
QCsd <- sd(t(as.matrix(QCtb)))
```

This produces a vector of numbers (list of numbers) containing a standard deviation for each peak. Now we need the means, which fortunately can be calculated directly from rows:

```
QCmean <- RowMeans(QCtb)
```

Finally we divide standard deviations by means to get the relative standard deviations:

```
RSD <- QCsd / QCmean
```

Note that this does them all in one go. This is the advantage of R.

The point of all this is that we now want to select peaks (rows in our original table) where the RSD is less than some suitable value (e.g. 30% = 0.3). R refers to anything in a matrix by its coordinates, [row, column]. Really cleverly, we can refer to rows by number, or by a condition:

```
RtbTrimmed <- Rtb[RSD < 0.3,]
```

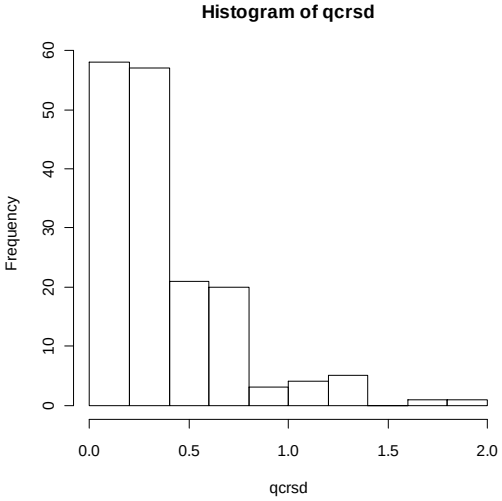
What is really happening here is that "RSD < 0.3" is creating a vector (list) of TRUE/FALSE values, and Rtb[###,] is selecting all rows where the vector contains the value TRUE. The comma is telling it that we're referring to rows, and we want all columns.

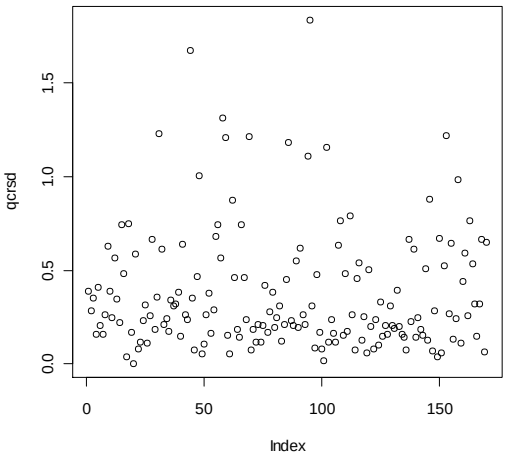
RtbTrimmed is now a shorter table containing only good peaks, and we can save it or use it for PCA. To save it:

```
write.csv(RtbTrimmed, "TrimmedOutput.csv")
```

Incidentally, you can also see the RSD values as a histogram or a plot, and view what sort of cut you are making:

```
hist(RSD)
plot(RSD)
```





(4b) Using Excel with no QC sample

In Excel, calculate the relative standard deviation of each treatment in a separate column using expressions such as:

=STDEV(T2:Y2)/AVERAGE(T2:Y2)

Now find the minimum RSD. This makes sense because a peak might be very clearly measured in treatment A, but completely absent from treatment B. In this case it will have a good RSD for A, but a terrible one in B. Make a new column for the minima:

=MIN(AD2:AF2)

Sort the table in order of increasing RSD, and take only the rows you want. Save the trimmed table as a comma-separated or tab-separated value file (R cannot read Excel workbooks).

(5) Multiple-univariate statistics

We now need to look which peaks have changed. It is possible to do multiple t-tests in Excel. The problem with this is that the more t-tests we do, the greater our risk of false positives (expect 1 per 100 rows at P<0.01). Classically the solution is Bonferroni correction, which is simply altering the P cut-off to 0.01/100 if working with 100 independent rows. This is over-cautious since many of our rows will be different ions derived from the same chemical, so we don't have 100 *independent* rows. It doesn't work for us. Multiple t-tests will cause you trouble with statistically-aware reviewers.

The next possibility is multiple Anova. I still feel this suffers from the same problem, but reviewers don't seem to distrust it as badly, and it is more appropriate where we have not only multiple peaks, but multiple (more than 2) treatments. It's hard to do in Excel because the Anova feature of Excel is part of the Analysis tool-pack, and returns a block of data, not a single number (and has to be used via a wizard).

Anova expects there to be equal numbers of replicates of each treatment. If you don't have this, the JIC statistics courses offer alternatives.

It may be possible to do multiple anova in Genstat. An alternative is to use R.

This is a little clumsy, but I provide a method below. I'm not sure it should be necessary to make a new function to carry out multiple Anova, but I did.

First create a function to do the job:

```
MultipleAnova <- function(indata, inlevels) {}
```

This is currently empty. Edit it, putting the edited version back into the original object (the original function).

```
MultipleAnova <- edit(MultipleAnova)
```

This will open a text editor, in which you can put the following code:

```
function(indata, inlevels)
## carries out repeated one-way anova
{
Pvector <- c()
indata <- as.matrix(indata)
qrows <- attr(indata, "dim")[1]
for (q in 1:qrows)
    {
    subdata <- indata[q,]
    submodel <- lm(subdata~inlevels)
    subanova <- anova(submodel)
    Pvector <- c(Pvector, subanova$"Pr(>F)"[1])
    }
Pvector
}
```

Refer to section 4a on how to read in your peak-areas, and select from the whole table a sub-table containing the columns you want. Create a sub-table containing just the peak-areas from the samples to be analysed. Your actions might look a bit like this:

```
dta <- read.csv("SLloyd_1841_3Nov2012_Export.csv")
options(max.print = 50)
attr(dta)
attributes(dta)
adta <- dta[,6:17]
attributes(adta)
```

Note that we checked what columns we had using attributes.

Now we create a model, a description of our experiment, for R. We should have grouped the replicates of the treatment together. For example, if we have three treatments, 4 replicates each, we'd create a model dtalevels using gl:

```
dtalevels <- gl(3, 4, 12, labels=c("A", "B", "C"))
dtalevels
```

The second line just checks what the model looks like. Now we run the multiple anovas:

```
tempPvec <- multipleanova(adta, dtalevels)
write.csv(tempPvec, "multanova.csv")
```

The results can be combined with the full peak-list in Excel, taking care that we don't combine two peak-lists sorted differently!

(6) Multivariate statistics

There are many different approaches, the commonest being PCA and PLS-DA. PCA is very user-friendly and safe; if you wish to use PLS-DA, it's a good idea to discuss it with experts (e.g. Kate Kemsley and her group at IFR).

Both PLS-DA and PCA are based on the idea of reducing a 300-dimensional data-set to something we can see.

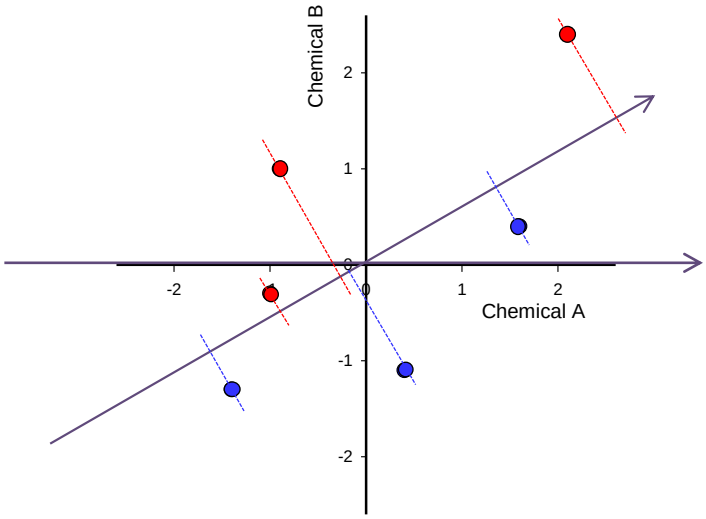
Introduction to PCA

Please skip this section if you are already familiar with the method.

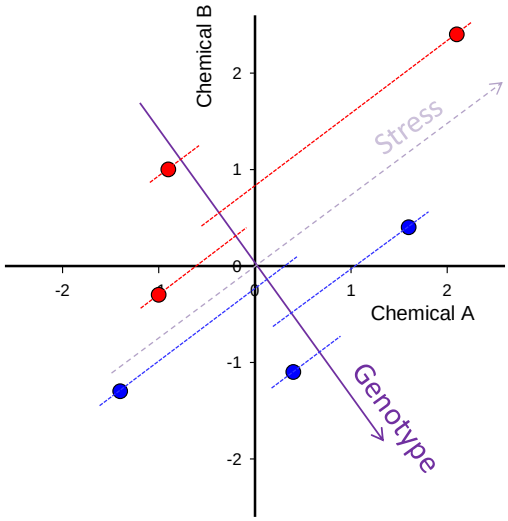
PCA is a traditional approach to looking at the differences between two (or more) sets of samples. The idea is that each sample gives a set of measurements, each corresponding to a different chemical. If a sample only had two chemicals, we could plot it in 2-dimensional space, each chemical being one axis of our plot, and each sample giving one dot on the plot. In real examples we might have hundreds, even thousands of values (though probably many less chemicals, as one chemical will create multiple ions). We therefore need a plot with more than 3 dimensions (in fact hundreds or thousands), which cannot be imagined. PCA provides a way to reduce this plot to two dimensions, on paper.

The idea is that the experiment will produce a cloud of spots. Hopefully the spots aren't scattered randomly, but treated samples will contain different metabolites than controls. Treated samples will be found on one side of the cloud while the controls are the other. PCA merely turns the axes round so that one of the axes points along the direction in which the cloud is most stretched. Ideally the treated samples are a long way from the controls, so the cloud is longest in that direction! In this case, PCA finds the most obvious differences between controls and treated samples. It progresses to put the second axis at right-angles to the first, but in the second-most-stretched direction in the cloud, and so on.

The diagram below shows two treatments, red and blue, with just two chemicals. In this experiment there is some other cause of scatter, and the cloud is long in the "wrong" direction. PCA will draw its first axis in this direction. We could re-plot the data as points moved onto this axis, reducing the number of dimensions, but it wouldn't tell us what we want to know:



On the other hand, the second axis at right-angles to the first, does contain the right information:



In the same way that we can move points sideways from the two-dimensional plot and create a one-dimensional plot in which the treatments are separated, PCA can be used to reduce a very large number of dimensions (100's to 1000's) to just two, which is can be drawn on paper.

Note that PCA rotates the axes through the origin, which means the points should be scattered around the origin. For this reason the data should be centred, i.e. the mean is subtracted from the data. Some metabolites are more abundant than others, so the cloud of points will be more stretched along their axes. For this reason, PCA is biased towards abundant chemicals. This can be avoided by scaling each chemical (dividing all the measurements of each chemical by the standard deviation of the measurements for that chemical).

PCA is closely related to partial least squares regression (PLS) in which the axes still exist, but are rotated so that they distinguish most clearly the different treatments. This is a very sensitive method, but certain to produce false positives. It is only safe when used in

conjunction with carefully planned controls (typically by rotating the axes using some data, and testing the validity of the result with other data, for which reason PLS is even more dependent than PCA on having lots of replicate samples).

Doing it!

First import the data from the trimmed peak-list in Excel (or R directly if you trimmed using R and a QC sample, see section 4a).

```
Rtab <- read.csv("TrimmedData.csv")
```

Using attributes, find the columns that contain sample data:

```
attributes(Rtab)
samptab <- Rtab[,15:38]
```

This selects, as an example, all rows (before comma) from columns 15-38 (after comma). The R function that does PCA expects the data in rows rather than columns, so we need to transpose:

```
samptabT <- t(samptab)
```

Create a list of labels corresponding to the treatments of each column. These should be in the right order, and should be single letters as they will be used for plotting. Lists can be created by concatenating, using "c":

```
sampLabels <- c("A", "W", "b", "A", "A", "b", "W", "W", "b")
```

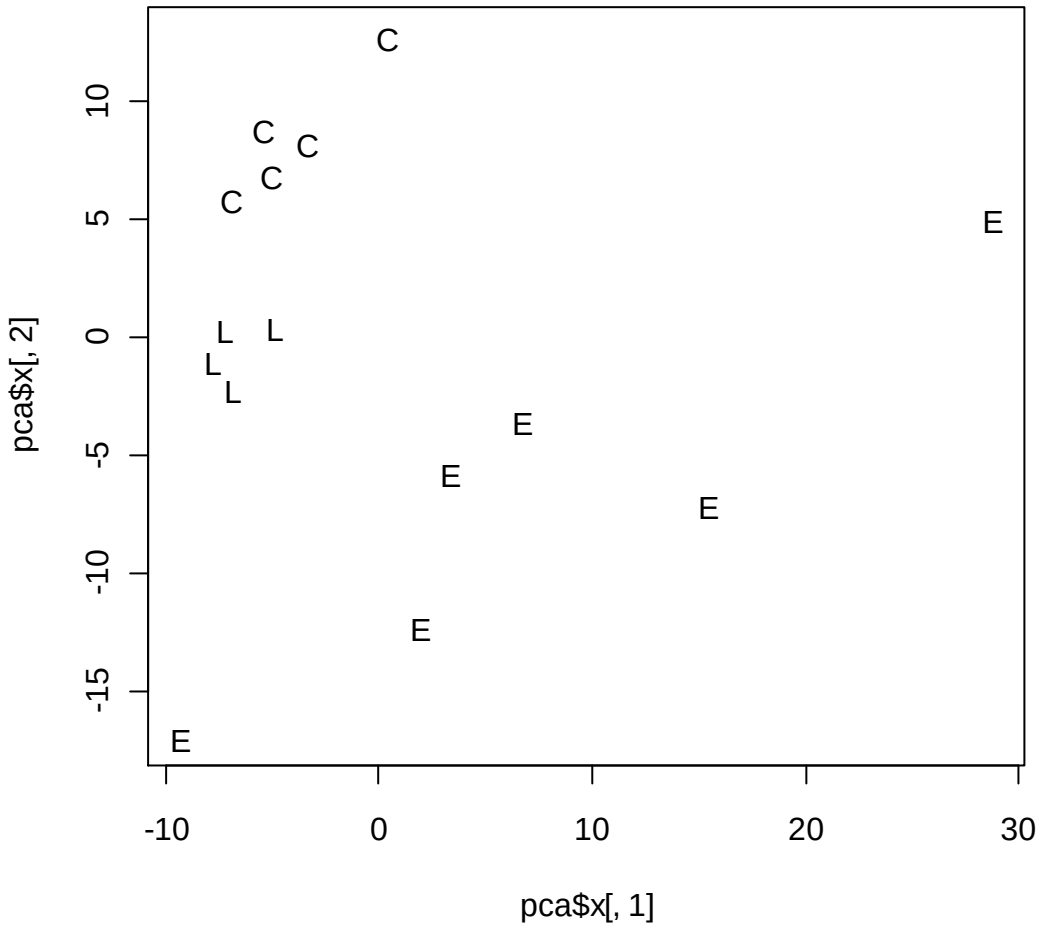
Now carry out the principle component analysis, putting the results in a new object:

```
pca <- prcomp(samptabT, center=TRUE, scale=TRUE, xret=TRUE)
```

This automatically centres the peak areas on zero (subtracts the mean) and scales them such that they all have a standard deviation of 1. If you don't scale, you will probably find that the greatest variability (PC1) is always the most abundant chemical, even if it doesn't tell you anything about your experiment. Xret ensures that the actual data are put in the object so we can plot our PCA plots.

To plot PC1 versus PC2,:

```
Plot(pca$x[,1], pca$x[,2], pch=sampLabels)
```



Having seen that PC2 in this case explains the difference between "L" and "C", while PC1 contains the difference between "L" and "E", we want to know which peaks contribute to each PC. The contributions are in the loadings, which R refers to as rotations. We can export these, for the first three PCs:

```
Write.csv(pca$rotation[,1:3], "rotation_values.csv")
```

We would then attach them to the Excel peak-list, and add a column containing the absolute values (i.e. ignoring the sign). The sign of the rotation tells you whether a chemical went down or up on treatment, and is arbitrary (different software packages will reverse the axis). Big negative values are just as important as big positive ones. In Excel:

```
=ABS(AH2)
```

Sort by this column (largest to smallest) and then explore what the peaks are. You should expect to find multiple features (rows) at nearly the same retention time, meaning that an ion has changed, and so has its isotope peak, and some related fragments…