# 1 Aims

So the point of this is multi-faceted:

- To provide a tutorial/introduction and access to the use of some of the crappy R code I wrote in the process of analysing peaklist MALDI imaging data during my thesis.

- To provide some example code on how (not) to use R, `knitr`, and LaTeX, including referencing using `bibtex`.

# 2 Disclaimer

This code is a steaming pile of crap. Use it with extreme skeptisism – as you should use any software, and do any analysis, skeptisism is the lifeblood of a scientist, embrace it. When I get some free time I plan on re-writing all the plotting functions from scratch to make them nicer. Which should make this steaming pile of crap slightly more bearable.

Also, I really ought to have wrapped my code up in a package, but I am shit and haven't, so for the time being all the relevant functions can be loaded into the  workspace by calling

```
source('localFunctions.R')
```

the fact that this is here may prompt me to getting around to cleaning it up and wrapping it up in package form at some point in the future, but probably not in the immediate future.

# 3 Setup and Reading Peaklists

I set the current `dataset_name` to a variable,

```
dataset_name <- "A1"
```

and then call

```
pl_all <- readPeaklists(dataset_name)
```

This reads in peaklist files from

```
<parent_folder_name>/<dataset_name>/<peaklist_folder_name>
```

and returns a combined peaklist `data.frame`  object after writing the three files:

```
<dataset_name>_comprehensive_peaklist.txt
<dataset_name>_fExists.txt
<dataset_name>_LXY.txt
```

to

./<data_folder>

dataset_name is required, but the other arguments are optional and default to:

- peaklist_folder_name = "peaklists"
- parent_folder_name = "."
- data_folder = "./data"

Note that the function readPeaklists reads peaklists in batches of 1000 at a time, and at the end of each batch prints the name of the last peaklist file to inform the user of progress, finally it prints the total number of empty peaklists read in.

Once created the files written to data_folder can be read in easily by calling

```
pl_all  <- load_peaklist(dataset_name)
LXY     <- load_LXY(dataset_name)
fExists <- load_fExists(dataset_name)
```

respectively. These load_* functions also accept an optional data_folder argument if an alternative location is used to store these files.

# 4 Peak Grouping and Peaklist Subsets

There are three functions included for assigning 'peakgroup' labels to peaks:

## 4.1 Mass Matching

Peaks can be matched to known masses by mass-error. In these data for example, there are some internal calibrants of known mass, as described by [2].

```
cal_df = data.frame(m.z = c(1296.685,
                            1570.677,
                            2147.199,
                            2932.588
                    ),
                    name = c('Angiotensin I',
                             '[Glu1]-Fibrinopeptide B',
                             'Dynorphin A',
                             'ACTH fragment (124)'))
```

The function `mzMatch` extracts peaks from the first argument about the $m/z$ values in the second argument.

```r
pl_cal <- mzMatch(pl_all,cal_df$m.z)
```

Optional arguments `binMargin` (mass error to be allowed) and `use_ppm` (mass error measured in ppm or Da) can also be specified, but otherwise default to:

- `binMargin` = 0.3, and
- `use_ppm` = FALSE.

The function `mzMatch` returns the subset of the input peaklist with an added column, `PeakGroup`, specifying the theoretical mass that peak is matched too. Not sure how this would react to overlapping mass windows but it was not intended for that.

## 4.2 Tolerance Clustering

When the masses of interest are not known, peakgroups can be formed via a one-dimensional clustering of the $m/z$ values of the peaks. Tolerance Clustering is one of the simplest ways of doing this, and boils down to finding the equivalence classes of the relation defined on two peaks as 'being within some tolerance `tol` of each other'.

```r
pl_tol <- groupPeaks(pl_all)
```

The optional tolerance argument `tol` defaults to a value of 0.1Da, and an additional optional argument `minGroupSize` can be specified to label any equivalence classes with less than that many peaks in them zero. By default all peaks will be labeled. The function `groupPeaks` returns the the input peaklist with an added column, `PeakGroup`, specifying a peakgroup label.

## 4.3 DBSCAN

A more sophisticated clustering approach is to use a density based clustering such as `DBSCAN`, or more precisely its deterministic version `DBSCAN*`, as described in [1].

```r
pl_dbs <- dbscan_lw(pl_all,pp=FALSE)
```

The function `dbscan_lw` works similarly to the function `groupPeaks`, in that it takes a peaklist and returns the same peaklist with an added column, `PeakGroup`, containing a peakgroup label. The function `dbscan_lw` also takes optional arguments `eps` (similar to the `tol` of the tolerance clustering, specifying the width of the rectangular kernal used), `mnpts` (the minimum number of points within a eps-neighbourhood considered significant – adjusting mnpts can fix the problem in large datasets of different masses being combined), cvar (specifying the variable in the input peaklist to be clustered) and pp (print progress to console logical). These optional arguments default to:

- eps = 0.05,
- mnpts = 100,
- cvar ="m.z", and
- pp =TRUE

## 5 Simple Plots

I also have some functions that make plots specially for peaklist data. This part is the most hacked up, as I have repeatadly modified these functions to perform various different tasks, while trying to maintain backwards compatability and its all ended in a giant mess. These are such a mess I am not even going to bother trying to explain them, and instead just reccomend you write your own plotting functions, as then you can be sure your plotting the thing you want to plot. Your also welcome to look at the code under spatialPlot and acquisitionPlot and canabilise code to your hearts content. I provide an example of one use of the function spatialPlot below when I produce a DIPPS map using it, although it can do alot more than this – I plan on coming back and re-writing the plotting functions at some point so they make more sense.

## 6 DIPPS

Now say you have produced some peakgroups one way or another, and now you have two regions you want to compare using DIPPS. For example, here I have annotation of the center of cancer tumours stored in an xml ‘ROI’ file. So I’ll read the annotations into R using the XML package and merge them onto the LXY variable as a ‘ROI’ column with value ‘None’ for spectra not in any annotated region.

```
library(XML)
fname <- 'A1_annotation.xml'
doc   = xmlInternalTreeParse(fname)
rois  = xpathSApply(doc,
                   "/ClinProtSpectraImport/Class",
                   xmlGetAttr,"Name")
nSpec = xpathSApply(doc,
                    "/ClinProtSpectraImport/Class",
                    xmlSize)
spec  = xpathSApply(doc,
                    "/ClinProtSpectraImport/Class/Element",
                    xmlGetAttr,"Path")
temp  = data.frame(Peaklist = Peaklist_ID(spec),
                   ROI = rep(rois,nSpec),
                   stringsAsFactors = FALSE)
LXY <- merge(LXY,temp,
             all.x = TRUE)
LXY[is.na(LXY$ROI),'ROI'] = "None"
```

We can take a quick look at these annotations by plotting them. Figure 1 demonstrates this, as well as providing a simple (less confusing?) example of a straightforward way to make spatial plots without using my gargantuan `spatialPlot` function (although you could equally make this plot using `spatialPlot` if you really wanted to.

```
p = (ggplot(LXY,aes(x=X,y=Y,
                    fill=ROI,
                    alpha= 1-(ROI=='None')),
            colour=NA)
     + geom_tile()
     + coord_fixed()
     + guides(alpha = FALSE)
     + scale_x_reverse(breaks=seq(75,175, 50))
     + scale_y_continuous(breaks=seq(50, 200, 50))
     + ylab("")
     + xlab("")
)
print(p)
```

Now I create a simplified peaklist variable `pl_uni` which intially has only two variables, `Acquisition` (indexing the originating spectrum) and `PeakGroup` (indexing the peakgroup). I also make sure the rows of `pl_uni` are unique – this is important, as having multiple peaks from the same peakgroup in the same spectrum will otherwise affect your results, although in the senario that this occurs more than a couple of times I would suggest perhaps revisiting whatever decisions you made at your peakgrouping step. I add a third variable, `Group` to the peaklist `pl_uni`, identifying each peak as originating from either an annotated region (2), or not (1). Now that we have cleaned up `pl_uni` and ensured it has the three neccessary columns, and that they are correctly formatted we can plug this right into the `DIPPS` function.

```
pl_uni = unique(pl_tol[,c("Acquisition","PeakGroup")])
temp = match(pl_uni$Acquisition,LXY$Acquisition)
pl_uni$Group = 1+(LXY[temp,]$ROI != "None")

dipsum = DIPPS(pl_uni)
```

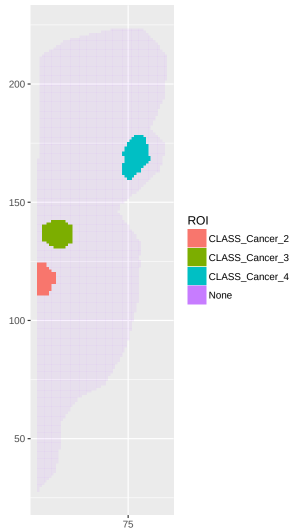

Figure 1: Annotation Regions

```
## Error in `[<-.data.frame`(`*tmp*`, prop$Group == 1, "p", value =
c(0.000298596595998806, :  missing values are not allowed in subscripted
assignments of data frames

nStar = dippsHeur(pl_uni,dipsum)

## Error in match(colnames(u.m), dsum_in$PeakGroup):  object 'dipsum'
not found
```

Note that in this case I used pl_tol to make pl_uni which I used in the DIPPS analyses – this was the peaklist with the PeakGroup column created by tolerance clustering. I could easily have used pl_dbs or even p_cal (if I was only interested in the calibrants) instead of pl_tol. Another option would be to bin the peaks in a data-independant manner using the R base function cut – a potentially useful option when interested in prediction, because of its independance on the data.

Also note how I generate nStar, which is the number of variables suggested to be optimal by the heuristic. One way to visualise these top nStar 'DIPPS features' is in a 'DIPPS map', so I might as well demonstrate how to do that using the hugely dodgey spatialPlot function:

```
dipsum = dipsum[rev(order(dipsum$d)),]

## Error in eval(expr, envir, enclos):  object 'dipsum' not found

peakgroups = dipsum[1:nStar,"PeakGroup"]
```

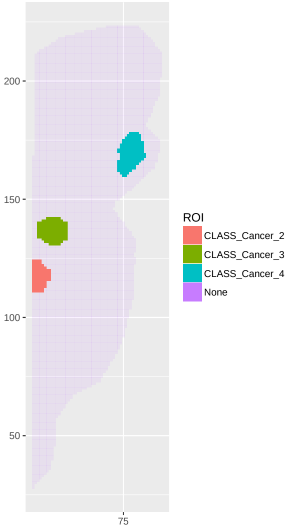

Figure 2: DIPPS Map

```
## Error in eval(expr, envir, enclos):  object 'dipsum' not found

plp = pl_uni[!is.na(match(pl_uni$PeakGroup,peakgroups)),]

## Error in match(pl_uni$PeakGroup, peakgroups):  object 'peakgroups'
not found

p = spatialPlot(plp,
                fExists,
                plot_var = "count",
                save_plot = FALSE,
                minX_in = min(LXY$X),
                minY_in = min(LXY$X)
)

## Error in data.table(x):  object 'plp' not found

p = (p + scale_x_reverse(breaks=seq(75,175, 50))
     + scale_y_continuous(breaks=seq(50, 200, 50))
     + ylab("")
     + xlab("")
)
# DIPPS Map
print(p)
```

One could also produce individual intensity plots of particular peakgroups,

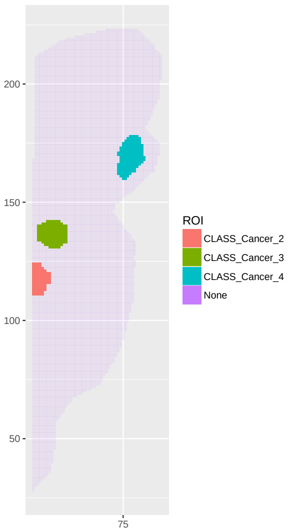

Figure 3: Intensity Map for peakgroup most highly ranked by DIPPS

for example for the peakgroup with highest DIPPS statistic value:

```
plp = subset(pl_tol,PeakGroup==dipsum[1,"PeakGroup"])

## Error in eval(expr, envir, enclos):  object ’dipsum’ not found

p = spatialPlot(plp,
                fExists,
                save_plot = FALSE,
                minX_in = min(LXY$X),
                minY_in = min(LXY$X)
)

## Error in data.table(x):  object ’plp’ not found

p = (p + scale_x_reverse(breaks=seq(75,175, 50))
     + scale_y_continuous(breaks=seq(50, 200, 50))
     + ylab("")
     + xlab("")
)
# Intensity Map
print(p)
```

and you could calculate ‘Abundance Weighted Means’ (weighting $m/z$ values by intensity, or in the following example signal-to-noise ratio) or various other statistics for these peakgroups as well if you wanted. For example:

```
pg_df = ddply(pl_tol,
              "PeakGroup",
              summarise,
              AWM = weighted.mean(m.z,SN),
              Range = max(m.z) - min(m.z),
              nDupPeaks = length(Acquisition) -
                length(unique(Acquisition))
)
pg_df = merge(pg_df,dipsum)

## Error in as.data.frame(y):  object 'dipsum' not found

pg_df = pg_df[rev(order(pg_df$d)),]

## Error in order(pg_df$d):  argument 1 is not a vector

head(pg_df)

##   PeakGroup       AWM Range nDupPeaks
## 1         1  999.5056 0.096         0
## 2         2 1000.5035 0.200         0
## 3         3 1001.3020 0.000         0
## 4         4 1001.5008 0.166         0
## 5         5 1002.2590 0.000         0
## 6         6 1002.4702 0.091         0

pg_df[57:61,]

##    PeakGroup      AWM Range nDupPeaks
## 57        57 1030.220 0.000         0
## 58        58 1030.495 0.089         0
## 59        59 1030.815 0.304         0
## 60        60 1031.132 0.000         0
## 61        61 1031.501 0.255         0
```

In addition to calculating the AWM here I have also calculated the $m/z$ range over which each peakgroup spans, and the number of duplicated peaks (a number above zero indicated there are individual spectra with more than one peak in the indicated peakgroup). Notice that the most highly ranked variables by DIPPS seem fine, but that there are some (AWM = 1155 and AWM = 1776) for which it seems the peakgrouping has failed pretty badly, producing peakgroups that span 4.16Da and 2.789Da respectively. Note that you can look at such things without having done the DIPPS step, and it is worth doing so as a quality-control/sanity-check step. You could for example try using the dbscan peakgroups instead – I'll leave that as an exercise.

# References

[1] Ricardo JGB Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.

[2] Johan OR Gustafsson, James S Eddes, Stephan Meding, Tomas Koudelka, Martin K Oehler, Shaun R McColl, and Peter Hoffmann. Internal calibrants allow high accuracy peptide matching between MALDI imaging MS and LC-MS/MS. *Journal of Proteomics*, 75(16):5093–5105, 2012.

# Session Info and pdflatex Version

```
sessionInfo()

## R version 3.1.0 (2014-04-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [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] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] plyr_1.8.3       ggplot2_2.1.0    reshape2_1.4.1   data.table_1.9.6
## [5] stringr_1.0.0    knitr_1.12.3
##
## loaded via a namespace (and not attached):
##  [1] chron_2.3-47     colorspace_1.2-6 digest_0.6.9     evaluate_0.9
##  [5] formatR_1.3      gtable_0.2.0     highr_0.5.1      magrittr_1.5
##  [9] munsell_0.4.3    Rcpp_0.12.4      scales_0.4.0     stringi_1.0-1
## [13] tools_3.1.0
```

```
system('pdflatex --version',intern=TRUE)

## [1] "MiKTeX-pdfTeX 2.9.5840 (1.40.16) (MiKTeX 2.9 64-bit)"
## [2] "Copyright (C) 1982 D. E. Knuth, (C) 1996-2014 Han The Thanh"
## [3] "TeX is a trademark of the American Mathematical Society."
## [4] "compiled with zlib version 1.2.8; using 1.2.8"
## [5] "compiled with libpng version 1.6.19; using 1.6.19"
## [6] "compiled with poppler version 0.32.0"
## [7] "compiled with jpeg version 8.4"
```