Lab Journal - Gene Expression Analysis

Noise exposures causing hearing loss generate proteotoxic stress and activate the proteostasis network

Student: Vincent Talen

Student number: 389015

Class: BFV2

Study: Bio-Informatica

Institute: Institute for Life Science & Technology

Teacher: Marcel Kempenaar

Date: 2022-04-15

Contents

1 R Setup

To correctly create this document and to be able to run all the R code a few settings have to be set and libraries imported. All the code chunks need to be visible in the end pdf so the echo is set to true and to render the pdf's faster whilst developing the cache is also set to true. Multiple packages are used in this project, all of them are placed in a vector which then can be used to load all the packages on a single line with lapply. For faster rendering times when using DESeq2 a higher core amount it specified.

```
# Set code chunks visibility in pdf output to true
knitr::opts_chunk$set(echo = TRUE)
# Use cache
knitr::opts_chunk$set(cache = TRUE)

# Create vector with all packages that will be used
packages <- c("affy", "pander", "scales", "BiocParallel",
              "edgeR", "pheatmap", "PoiClaClu", "ggplot2")
# Load each package in the vector with lapply
invisible(lapply(packages, library, character.only = TRUE))
# Drop the packages variable from memory since it will not be used again
remove(packages)
# Set cores to be used for when using Bioc DESeq2
register(MulticoreParam(6))
```

2 Exploratory Data Analysis

2.1 Preparing the data

To start this project out the data set will be loaded into the memory so the statistics can be performed on it. The first few lines will be shown to show what the contents of the data frame are like and dim() is used to show the length of the data set.

```
# Load in the data set into a data frame
myData <- read.table("GSE160639_RawReadCount.txt", header = T, sep = "\t")
# Change the int row names to the gene_id column and delete the gene_id column afterwards
row.names(myData) <- myData$gene_id
myData <- myData[-1]
# Show the first few lines of the data frame
head(myData)
```

```
##                    A1_70dB A2_70dB A3_70dB A4_70dB B1_94dB B2_94dB B3_94dB
## ENSMUSG00000064351  807021  844885  746507  600313  455027  512010  325579
## ENSMUSG00000069919  366445  395997  269558  284141  302984  427023  410969
## ENSMUSG00000052305  243346  341576  226918  233215  249790  301111  480944
## ENSMUSG00000015090  352912  323278  255015  234177  318835  364719  366617
## ENSMUSG00000064370  381672  361431  312361  262815  222349  253774  159885
## ENSMUSG00000001506  312965  305261  233933  193533  256679  279963  202360
##                    B4_94dB C1_105dB C2_105dB C4_105dB C5_105dB
## ENSMUSG00000064351  416803   745783   521567   444325   761621
## ENSMUSG00000069919  408831   390024   343923   474316   401705
## ENSMUSG00000052305  634849   320317   371479   321232   328205
## ENSMUSG00000015090  381912   307553   251633   333419   258235
## ENSMUSG00000064370  191874   360531   267048   265633   310993
## ENSMUSG00000001506  216174   340531   230902   196198   222483
```

```
# Print the amount of genes and columns
cat("Amount of genes:", dim(myData)[1], "\tColumns in dataframe", dim(myData)[2])
```

```
## Amount of genes: 35496   Columns in dataframe 12
```

By using the str() function the structure of the dataframe/dataset can be seen. Every column consists only of int values, so only the count data. (Resulting table not shown in PDF)

```
str(myData)
```

For future use the `groups` variable is already created since it will be needed multiple times. It is used to define what group a sample belongs to.

```
groups <- factor(rep(1:3, each=4), labels = c("70dB", "94dB", "105dB"))
```

2.2 Data Summary

To get to know the data set more the data will be visualized in multiple ways. Seeing how everything is grouped and divided gives us a deeper understanding of what the quality of the data set is and what types of statistics will have to be performed.

First a simple summary performing 6-number-statistic on the data columns will be done, this gives a summary overview of each replication for all groups.

```
# Disable printing 'table continues' lines between split sections of the table
panderOptions("table.continues", "")
# Pretty print the summary of the data frame
pander::pander(summary(myData), style = "rmarkdown")
```

A1_70dB	A2_70dB	A3_70dB	A4_70dB
Min. : 0.0	Min. : 0	Min. : 0.0	Min. : 0.0
1st Qu.: 1.0	1st Qu.: 0	1st Qu.: 0.0	1st Qu.: 0.0
Median : 21.0	Median : 20	Median : 16.0	Median : 16.0
Mean : 1161.2	Mean : 1192	Mean : 973.6	Mean : 877.3
3rd Qu.: 909.2	3rd Qu.: 915	3rd Qu.: 749.2	3rd Qu.: 690.0
Max. :807021.0	Max. :844885	Max. :746507.0	Max. :600313.0

B1_94dB	B2_94dB	B3_94dB	B4_94dB
Min. : 0.0	Min. : 0	Min. : 0	Min. : 0
1st Qu.: 0.0	1st Qu.: 0	1st Qu.: 0	1st Qu.: 0
Median : 16.0	Median : 17	Median : 15	Median : 18
Mean : 941.7	Mean : 1058	Mean : 910	Mean : 1091
3rd Qu.: 730.0	3rd Qu.: 817	3rd Qu.: 692	3rd Qu.: 813
Max. :455027.0	Max. :512010	Max. :480944	Max. :634849

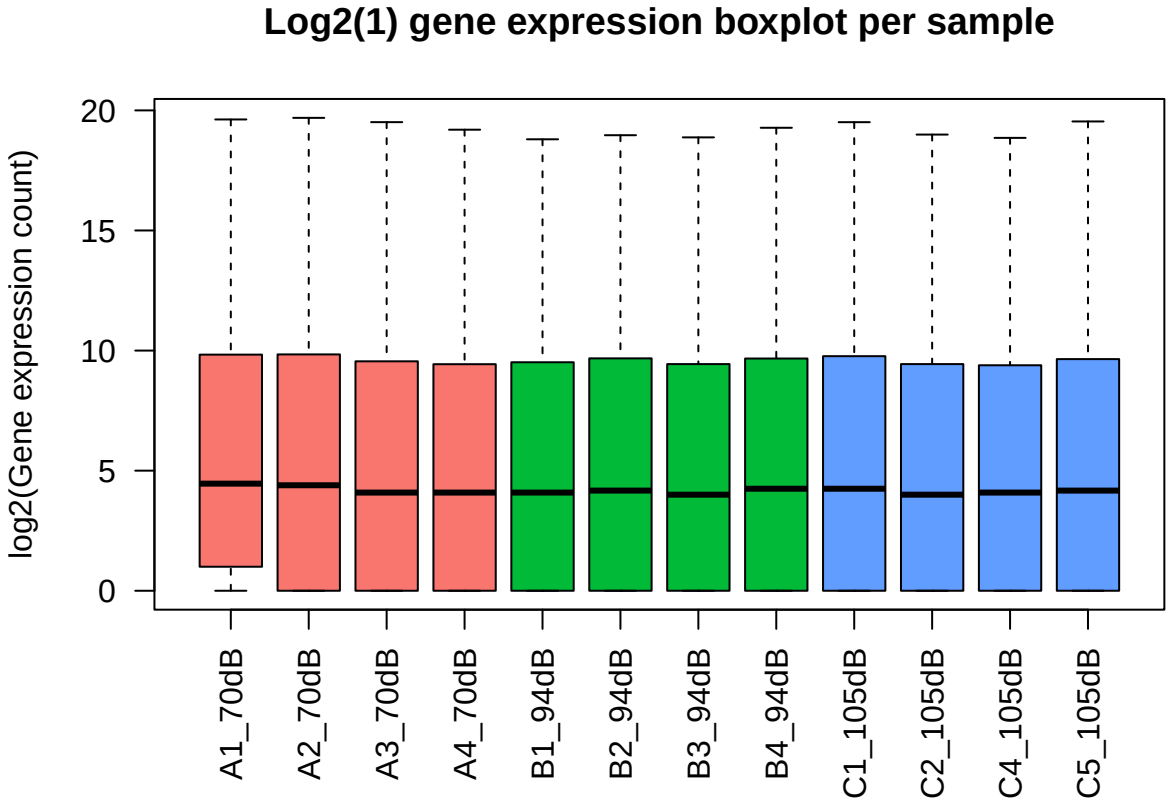
C1_105dB	C2_105dB	C4_105dB	C5_105dB
Min. : 0	Min. : 0.0	Min. : 0.0	Min. : 0
1st Qu.: 0	1st Qu.: 0.0	1st Qu.: 0.0	1st Qu.: 0
Median : 18	Median : 15.0	Median : 16.0	Median : 17
Mean : 1134	Mean : 913.6	Mean : 903.1	Mean : 1045
3rd Qu.: 870	3rd Qu.: 692.0	3rd Qu.: 669.0	3rd Qu.: 799
Max. :745783	Max. :521567.0	Max. :474316.0	Max. :761621

At first glance of the summary it can be seen that the expression at the lowest (70) dB genes seem to be expressed the most with the expression declining whilst the SPL increases. It also looks like there is a noticeable difference between each replication.

2.3 Boxplot

Now a boxplot will be made for each data column, visualizing this will help quickly spot irregularities after which further detailed statistics need to be performed to do any quality control. To increase the visibility a log2 transformation will be done to show changes more informative.

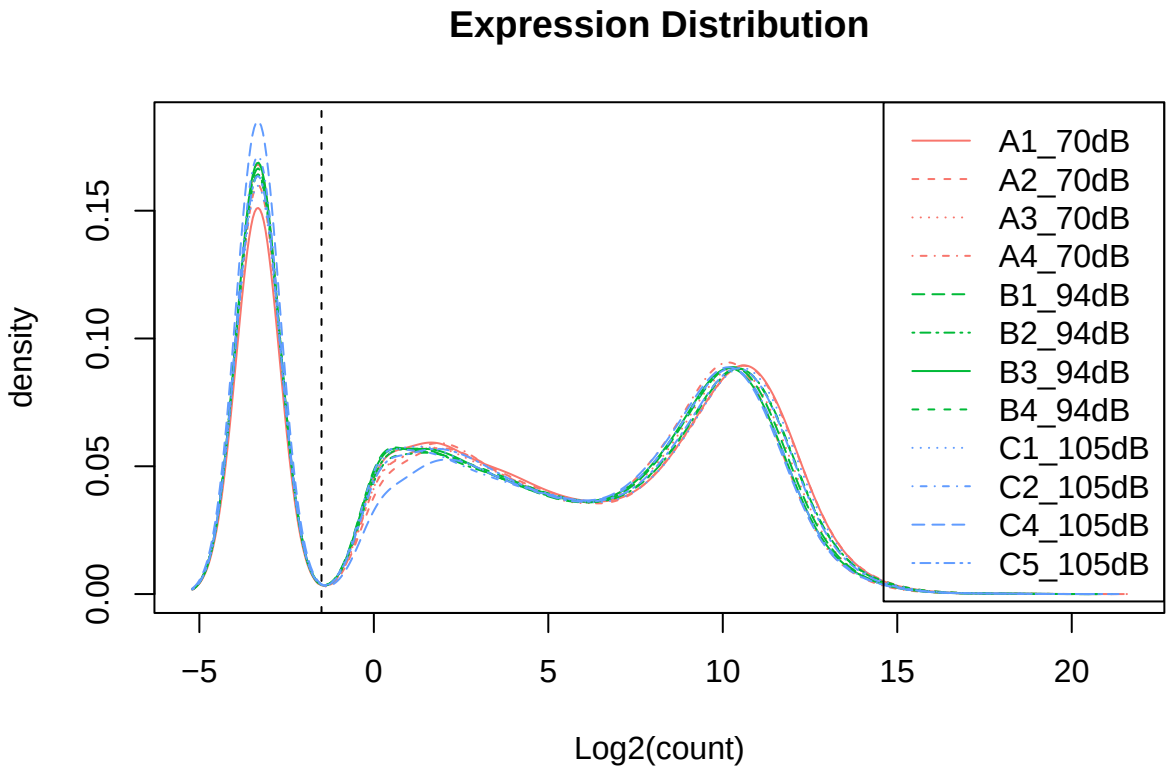
```
# Create a color pallet variable to use in graphs
myColors = rep(hue_pal()(3), each = 4)
# Plot the boxplot with log2(1) scale.
boxplot(log2(myData + 1), col = myColors,
        outline = FALSE,las = 2, ylab = "log2(Gene expression count)",
        main="Log2(1) gene expression boxplot per sample")
```



2.4 Density plot

Another useful way of visualizing the data set is using a density plot, it shows a distribution of the log2-transformed count data for all samples which makes spotting problems easier.

```
# Plot a density plot with log2(0.1) scale
# Use a different color for each group and line type for each replication
plotDensity(log2(myData + 0.1), col=myColors,
            lty=c(1:ncol(myData)), xlab="Log2(count)",
            main="Expression Distribution")

# Add a legend and a vertical line
legend("topright", names(myData), lty=c(1:ncol(myData)), col=myColors)
abline(v=-1.5, lwd=1, col="black", lty=2)
```



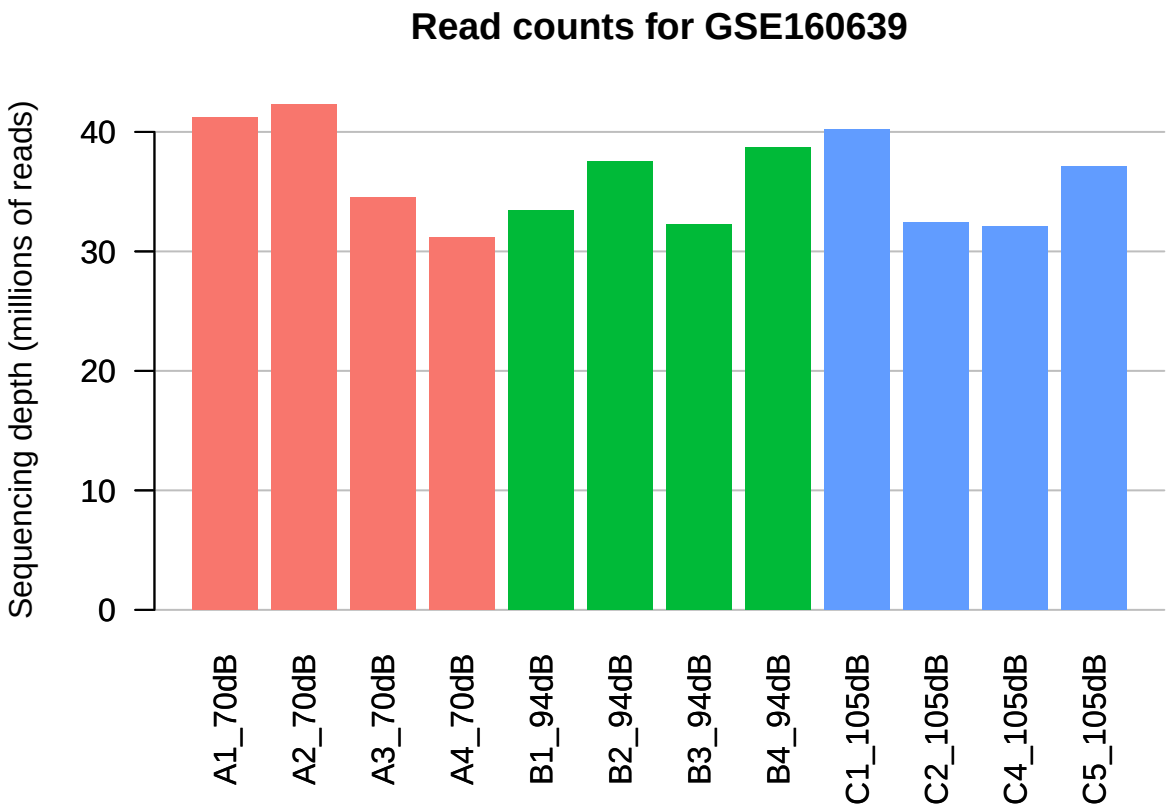
Looking at the density plot of the expression distribution it looks like all replications have a relatively similar amount of reads sequenced since all the peaks lie together.

2.5 Normalizing the data

Checking read depth with barplot

Sometimes there seems to be a difference in expression between groups even if there actually isn't, if there is a big difference in count values within the samples of a group that could be due to the read depth of the samples. To see the read depth of all samples fast, the total number of mapped reads per sample can be shown in a bar plot.

```
# Initialize plot
barplot(colSums(myData) / 1e6, las = 2, border = NA)

# Add horizontal axis
axis(2, tck = 1, lty = 1, col = "gray", labels = FALSE)

# Create the plot
par(new = TRUE)
barplot(colSums(myData) / 1e6, main = "Read counts for GSE160639",
        las = 2, col = myColors, border = NA,
        ylab = "Sequencing depth (millions of reads)")
```



Looking at the bar plot that was created it can be seen that all the samples have a good, similar, sequencing depth. Though all the read depths are high and similar the data will still need to be normalized, because there might still be unwanted variation disrupting our analysis.

Performing the data normalization with a variance stabilizing transformation

To normalize the data the *vsc* function of the DESeq2 library will be used. First a *DESeqDataSet*-object will need to be created since that's what is needed to use the function. There are multiple techniques to normalize data but for now only normalized data from a variance stabilizing transformation will be used.

```
# Create DESeqDataSet object
ddsMat <- DESeqDataSetFromMatrix(countData = myData, design = ~ 1,
                                 colData = data.frame(samples = names(myData)))
# Perform normalization
rld.dds <- vst(ddsMat)
# 'Extract' normalized values
rld <- assay(rld.dds)

# Pretty print the head of the normalized data
panderOptions("table.continues", "")
pander::pander(head(rld), split.tables = 70)
```

	A1_70dB	A2_70dB	A3_70dB	A4_70dB
ENSMUSG00000064351	19.36	19.44	19.54	19.37
ENSMUSG00000069919	18.22	18.35	18.07	18.29
ENSMUSG00000052305	17.63	18.14	17.83	18
ENSMUSG00000015090	18.17	18.06	17.99	18.01
ENSMUSG00000064370	18.28	18.22	18.29	18.18
ENSMUSG00000001506	18	17.98	17.87	17.73

	B1_94dB	B2_94dB	B3_94dB	B4_94dB
ENSMUSG00000064351	18.87	18.9	18.49	18.61
ENSMUSG00000069919	18.28	18.63	18.83	18.59
ENSMUSG00000052305	18.01	18.13	19.05	19.22
ENSMUSG00000015090	18.36	18.41	18.66	18.49
ENSMUSG00000064370	17.84	17.88	17.47	17.49
ENSMUSG00000001506	18.04	18.02	17.81	17.67

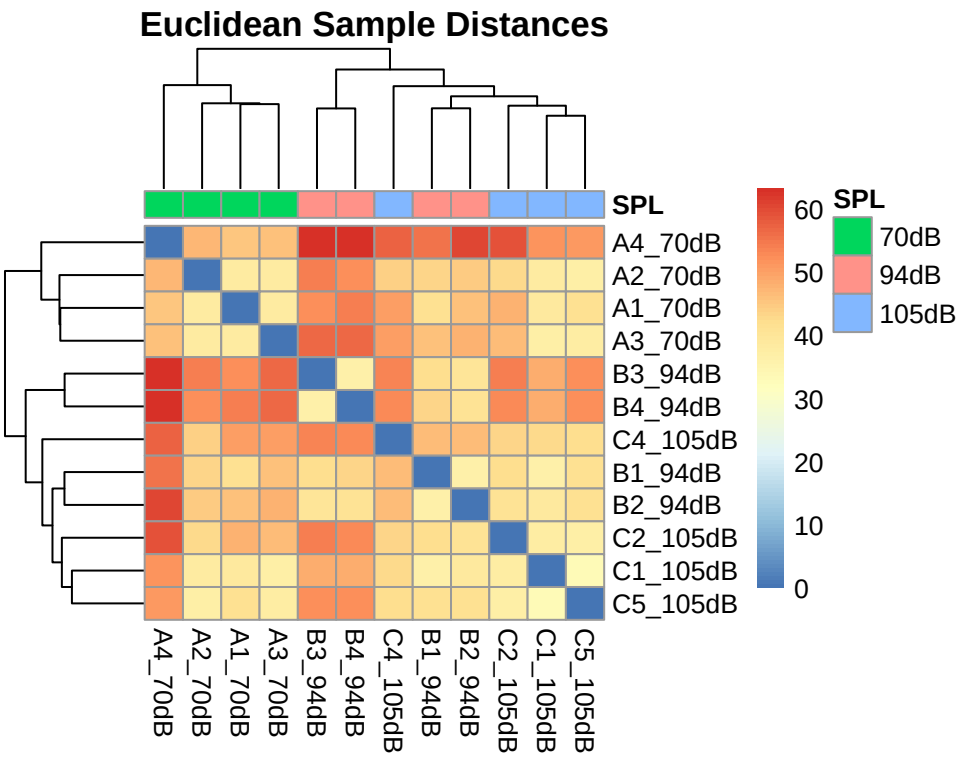
	C1_105dB	C2_105dB	C4_105dB	C5_105dB
ENSMUSG00000064351	19.34	19.15	18.96	19.49
ENSMUSG00000069919	18.41	18.55	19.05	18.56
ENSMUSG00000052305	18.12	18.66	18.49	18.27
ENSMUSG00000015090	18.07	18.1	18.55	17.93
ENSMUSG00000064370	18.3	18.18	18.22	18.2
ENSMUSG00000001506	18.21	17.97	17.78	17.71

Normalized data has now been obtained with values that are comparable, unlike the raw read counts. What looked like differences in expression in the count data could now with the normalized data look no different at all between the groups.

2.6 Creating a heatmap to see sample distances

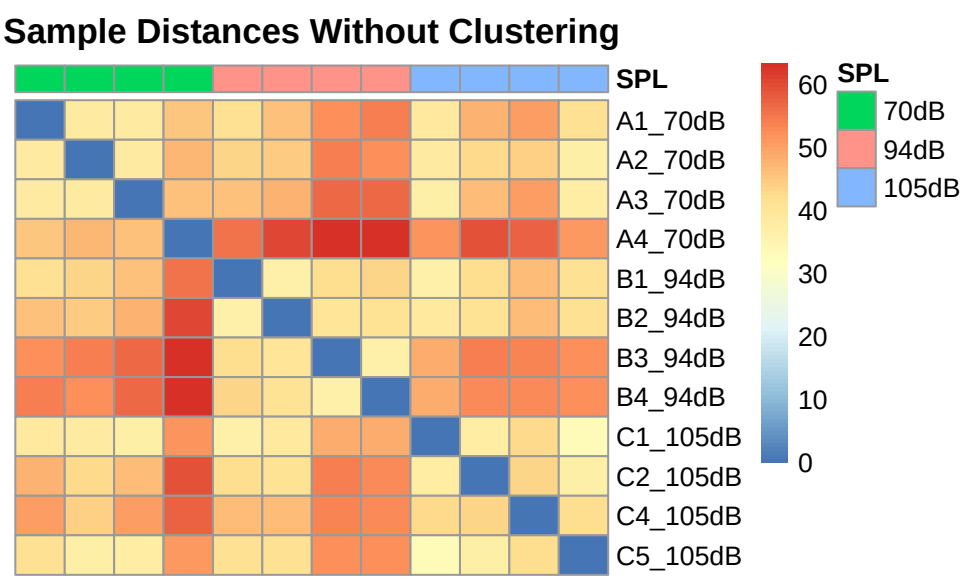
To see what the differences in expression between groups are, the data will be visualized with a heatmap. This heatmap will show the Euclidean distances between all samples. To make a heatmap distance calculations need to be performed, for each combination of samples a distance metric is calculated that will be used to check for variation within the sample groups. The normalized data will be transposed and only after that can be used to calculate the sample distances.

```
# Calculate basic distance metric (using euclidean distance)
sampledists <- dist(t(rld))

# Convert the 'dist' object into a matrix for creating a heatmap
sampleDistMatrix <- as.matrix(sampledists)

# The annotation is an extra layer that will be plotted above the heatmap columns
annotation <- data.frame(SPL = groups)

# Set the rownames of the annotation dataframe to the sample names
rownames(annotation) <- names(myData)

# Create heatmap
pheatmap(sampleDistMatrix, main = "Euclidean Sample Distances",
         annotation_col = annotation, show_colnames = TRUE,
         clustering_distance_rows = sampledists,
         clustering_distance_cols = sampledists)
```



The resulting heatmap is not as beautiful as desired and thus does not give that much insight in the data. This is probably mainly due to the fact that all the samples are from the exact same type of tissue and the only differentiating factor is the Sound Pressure Level the mice were exposed to. With the idea that it might be more beneficial to create a heatmap that doesn't change the clustering a second one was created.

```
# Create second heatmap without changed clustering
pheatmap(sampleDistMatrix, main = "Sample Distances Without Clustering",
         annotation_col = annotation, show_colnames = FALSE,
         cluster_rows = FALSE, cluster_cols = FALSE)
```



In this second heatmap it can be better seen that there are trends between and within sample groups, almost creating borders between groups. Because there are a few samples with noticeable differentiating values from the other samples in their own group the first heatmap became less readable since there is only 1 factor that differs. One could also say that by looking at this heatmap sample A4 looks to be an outlier and might cause noise in statistics later on.

2.7 Using Multi-Dimensional Scaling to spot clustering

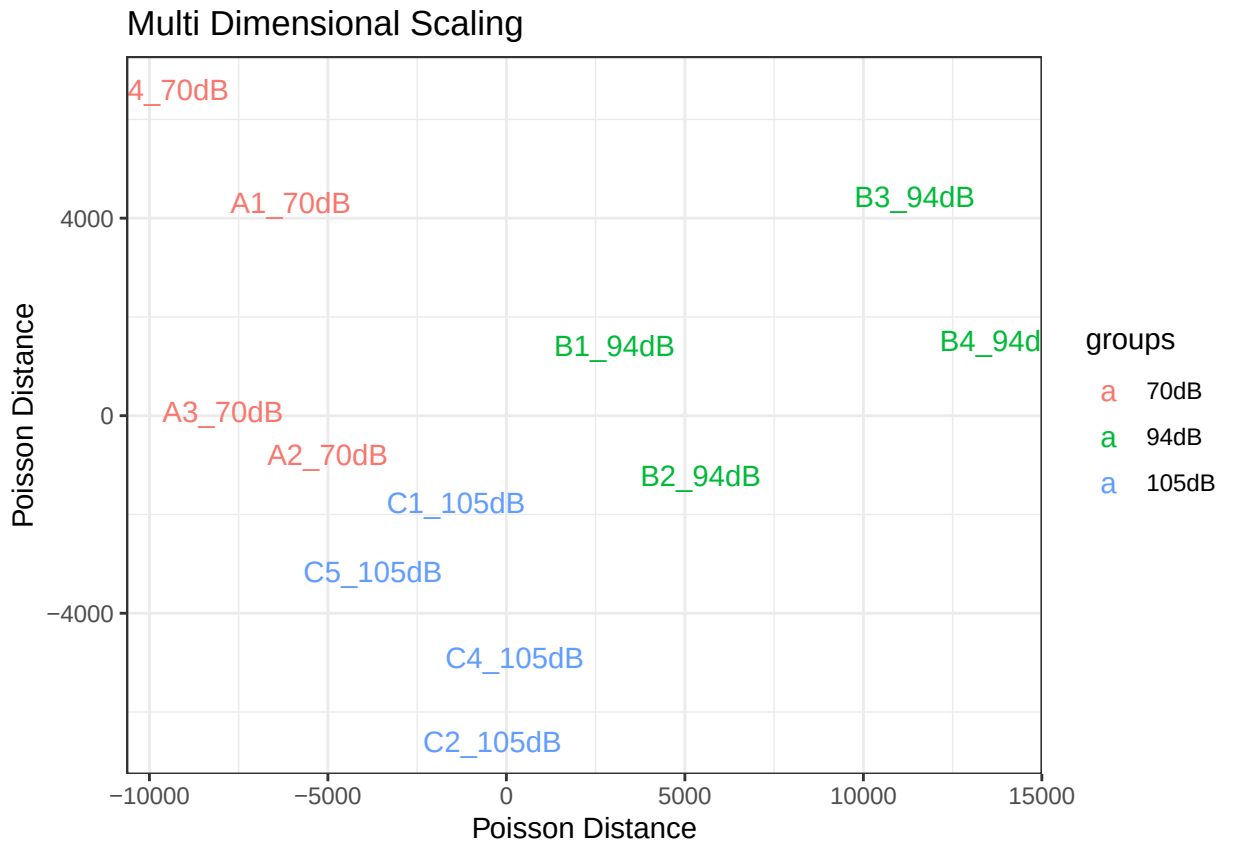
Another way to see how distant samples are from each other is by performing Multi-Dimensional Scaling (MDS), this creates a 2D plot where it is the hope that distinct clusters are formed. What's different with MDS is that a slightly different distance metric is used; Poisson instead of the previously used Euclidean with the dist() function. To get the *Poisson Distance* data the 'PoiClaClu' library is used, it uses the raw count data (keep in mind the 'ddsMat' DataSet still contains raw count data, only the 'rld' matrix is normalized).

```
# Note: uses the raw-count data, PoissonDistance performs normalization
# set by the 'type' parameter (uses DESeq)
dds <- assay(ddsMat)
poisd <- PoissonDistance(t(dds), type = "deseq")
# Extract the matrix with distances
samplePoisDistMatrix <- as.matrix(poisd$dd)

# Calculate the MDS and get the X- and Y-coordinates
mdsPoisData <- data.frame( cmdscale(samplePoisDistMatrix) )
# And set some better readable names for the columns
names(mdsPoisData) <- c('x_coord', 'y_coord')
mdsPoisData
```

```
##       x_coord     y_coord
## 1  -6039.3225  4308.88762
## 2  -5004.5860  -782.13899
## 3  -7939.7214    90.81125
## 4  -9463.8490  6607.43794
## 5   3032.6580  1431.82351
## 6   5442.7042 -1212.37390
## 7  11440.5382  4445.34593
## 8  13835.0550  1522.68025
## 9  -1409.1580 -1757.52593
## 10  -389.4577 -6594.13393
## 11   233.9949 -4905.50780
## 12 -3738.8558 -3155.30595
```

Now the MDS coordinates used for plotting the distances using Poisson Distance are calculated they'll be plotted with the ggplot library.

```
# Separate the annotation factor (as the variable name is used as label)
coldata <- names(myData)

# Create the plot using ggplot
ggplot(mdsPoisData, aes(x_coord, y_coord, color = groups, label = coldata)) + theme_bw() +
       geom_text(size = 4) + ggtitle('Multi Dimensional Scaling') +
       labs(x = "Poisson Distance", y = "Poisson Distance")
```



The three different groups seem to cluster pretty nicely in the generated plot, there is a noticeable separation between the different SPL groups' coordinates.

Based on all of the exploratory data analysis that was done it looks like all the samples are usable so that there does not seem to be a need to do any further data cleaning. This marks the end of this chapter and now the 'real' analysis will start.

3 Discovering Differentially Expressed Genes (DEGs)

The goal of this project and what will be focused on now is to try to discover differentially expressed genes, these are genes that show a difference in expression between sample groups. Simply said it's done by calculating the ratios for all genes between samples to determine the **fold-change** (FC) denoting the factor of change in expression between groups. Then, filter out only those genes that actually show a difference. While this might look like it immediately works, what's almost more important is that the DEGs are statistically significant.

In the chosen research the interactive gene expression analysis kit *iGEAK* was used, which is based on the R/shiny platform, so most methods can be replicated in the environment this project is being done (RStudio).

3.1 Pre-processing

Sample removal

After doing the exploratory data analysis there does not seem to be a direct need to remove samples from the data set. It will however be checked if removing the A4 sample groups causes a reduction noise and lower p-values thus if it results in higher overall statistical significance.

Filtering (partially) inactive genes

An important step in pre-processing is to filter out (partially) inactive genes from the data set, this is even required for using the `edgeR` library. This step is done manually, since removing genes can be very subjective to the experiment, it might be expected (thus important) when comparing different tissues or knockout experiments. Genes with a (very) low read count (< 5) can give a very high (artificial) FC value, when comparing two samples where one has a value of 2 and the other 11, this reads as an up-regulated gene by a factor of 5.5 while it might actually just be noise.

Since the chosen research also filtered very low signals, the same formula will be used. The researchers filtered the genes by requiring a minimum read count of 8 in at least 4 samples. The way it is done below here is that an index filter is created saying which genes have passed the criteria by marking them as TRUE or FALSE. For each gene/row it gets the sum of how many samples have 8 or more counts, if a gene has 4 or more samples with at least 8 counts then it returns it as TRUE. The index is then applied to the original data frame resulting in a new data frame with only the genes that match the criteria.

```
# Create filter index
idx <- rowSums( myData >= 8 ) >= 4
# Use the index to create the new data frame with only the genes that passed
myData.filtered <- myData[idx,]
```

To be sure the filtering was successful, a manual check was done. It can be seen that the index has TRUE and FALSE values for each gene/row, when applying the filter and checking the data frame for which genes are left with which values it all seems to have worked great. Before filtering there were **35496** genes and after filtering there are **20315** genes left. This means that more than a third, **15181** genes to be exact, were filtered out.

FPM Normalization

Before using other libraries another manual step will be done, the normalization of the data will be re-done with a different technique. Performing the simple method of calculating the *fragments per million mapped fragments* (FPM) value and then log2 transforming it. This does not include the gene-length, which is not known for the data set, so it can't be used anyways. FPM is calculated by dividing the count values by the total number of reads of that sample divided by 1 million. For now all genes, not only the filtered genes are used, to later see the difference of what the results look like without filtering and with.

```
# Perform a naive FPM normalization
# Note: log transformation includes a pseudo count of 1
myData.fpm <- log2( (myData / (colSums(myData) / 1e6)) + 1 )
head(myData.fpm)
```

```
##                     A1_70dB  A2_70dB  A3_70dB  A4_70dB  B1_94dB  B2_94dB
## ENSMUSG00000064351 14.25710 14.32325 14.14466 13.83024 13.43051 13.60071
## ENSMUSG00000069919 13.08021 13.19209 12.63727 12.71327 12.80589 13.30091
## ENSMUSG00000052305 12.78191 13.27105 12.68108 12.72057 12.81961 13.08915
## ENSMUSG00000015090 13.46830 13.34178 12.99962 12.87666 13.32182 13.51577
## ENSMUSG00000064370 13.47913 13.40052 13.19004 12.94090 12.69971 12.89040
## ENSMUSG00000001506 13.02546 12.98951 12.60561 12.33214 12.73946 12.86471
##                     B3_94dB  B4_94dB C1_105dB C2_105dB C4_105dB C5_105dB
## ENSMUSG00000064351 12.94761 13.30393 14.14326 13.62739 13.39617 14.17357
## ENSMUSG00000069919 13.24563 13.23810 13.17017 12.98871 13.45243 13.21274
## ENSMUSG00000052305 13.76467 14.16518 13.17835 13.39211 13.18246 13.21344
## ENSMUSG00000015090 13.52326 13.58222 13.26985 12.98036 13.38634 13.01772
## ENSMUSG00000064370 12.22400 12.48708 13.39692 12.96395 12.95628 13.18371
## ENSMUSG00000001506 12.39648 12.49173 13.14723 12.58680 12.35187 12.53322
```

3.2 The *Fold Change* value

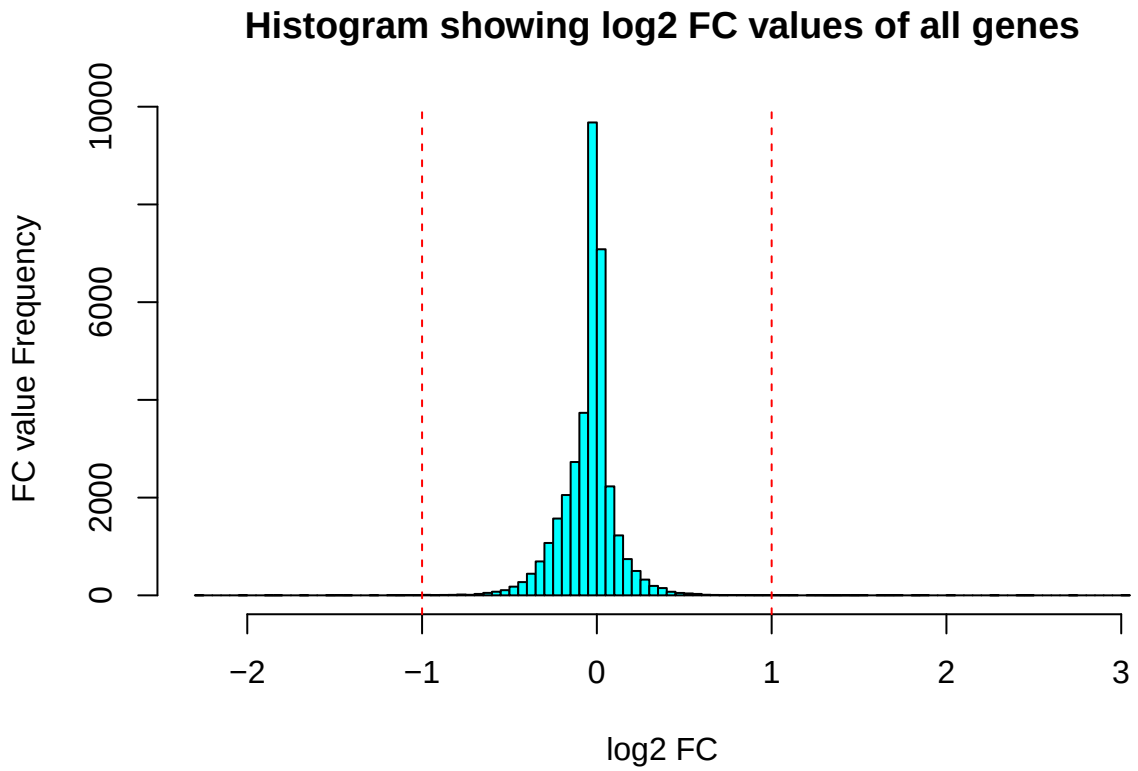
Fold Change (FC) is a measure that describes the degree of quantity change between final and original values, it is usually given as the calculated `log2` of the case/control ratio. Ratio values > 1 indicate increased expression in the experiment in relation to the control and values between 0 and 1 indicate lower expression but can have a huge range of numbers. Using log2 values to display the ratios makes it easier to distinguish between down-regulated expression `log2(ratio) < 0` and up-regulated expression `log2(ratio) > 0`.

For example, gene A has an average expression of 30 mapped reads in the control group and 88 reads in the experiment group, the *ratio* case/control (`88/30`) is `2.93`. If the counts were reversed, the ratio would have been `30/88`, which is `0.34`. The previously calculated value of `2.93` means a *3-fold up regulation* while the `0.34` value means *3-fold down regulation* but as you can see the range of numbers is very different. For easier comparing the log2 values are used, the log2 transformed value of the ratio is calculated with `log2(ratio)`. In this example the log2 values would be `log2(2.93) = 1.55` and `log2(0.34) = -1.55`, these log2 fold changes compare much better and easier.

Histogram showing FC

To visualize the FC ratio of *all* genes between the 105db (case) and the ambient (control) group a histogram is made. For each group the average of the log2(FPM) is calculated per gene and added as a new column to the previously created FPM normalized data frame. The FC values are then calculated for the 105dB group against the 70dB control group after which they are plotted in a histogram. Two ablines will also be shown in the histogram at -1 and 1, these indicate where the border is of which FC values could be considered DEGs.

```r
# Create new column for each group with the average of genes' log2(FPM) value's
myData.fpm$avg70dB <- rowMeans(myData.fpm[1:4])
myData.fpm$avg94dB <- rowMeans(myData.fpm[5:8])
myData.fpm$avg105dB <- rowMeans(myData.fpm[8:12])

# Calculate FC for 105dB against 70dB control group
myData.fpm$FC105dB_70dB <- myData.fpm$avg105dB - myData.fpm$avg70dB

# Plot histogram showing the FC
hist(myData.fpm$FC105dB_70dB, breaks=100, col = "cyan",
     main = "Histogram showing log2 FC values of all genes",
     xlab = "log2 FC", ylab = "FC value Frequency")

# Add ablines indicating DEG territory (outside = DEG)
abline(v=c(-1, 1), lty=2, col="red")
```



Looking at the histogram almost all genes lie within -1 and 1 (the red ablines), outside the ablines the frequency is so low that they practically can't be seen. However this does not mean there are not any high enough FC values outside the lines!

An FC value by itself does not mean much, because if the variation *within* a group is very high, it could just be biological noise or a sequencing error. It needs to be determined if the observed FC is statistically significant. To determine significance, **t-test**s are performed using all the replicates of a group. These t-tests will however not be done manually but by using the Bioconductor package `edgeR`, it performs the t-tests and significance is then indicated by the resulting p-value. If a genes' p-value is below the alpha of 0.05 it is considered a statistically-significant differentially expressed gene (SDEG).

3.3 Using Bioconductor Packages for DEG Analysis

To make the DEG analysis easier and more automatic rather than performing manual steps like done above a package will be used, there are a range of packages to choose from but since the chosen research used Bioconductor's `edgeR` package, this will be the one being used. In the chosen research the data was normalized using `edgeR`'s TMM (Trimmed Mean of M-values) normalization method and the resulting P-values were adjusted using the *Benjamini and Hochberg's* approach for controlling the False Discovery Rate (FDR). Genes with an adjusted P value < 0.05 and fold change > 1.5 found by edgeR were assigned as differentially expressed genes.

Creating a `Design Matrix`

To use any of these packages it needs to be properly specified how the samples are grouped. With more than two samples it is important to know what differences will want to be looked for. The time is not relevant for this particular experiment since all samples are taken at the same time after exposure, the main thing this project is looking for is the difference in expression caused by different sound pressure levels. Since the first group, with the 70dB SPL, is our control group nothing has to be done about the levels.

```
# Create a design matrix using the previously created `groups` variable
(myDesign <- model.matrix( ~ groups))
```

```
##    (Intercept) groups94dB groups105dB
## 1            1          0           0
## 2            1          0           0
## 3            1          0           0
## 4            1          0           0
## 5            1          1           0
## 6            1          1           0
## 7            1          1           0
## 8            1          1           0
## 9            1          0           1
## 10           1          0           1
## 11           1          0           1
## 12           1          0           1
## attr(,"assign")
## [1] 0 1 1
## attr(,"contrasts")
## attr(,"contrasts")$groups
## [1] "contr.treatment"
```

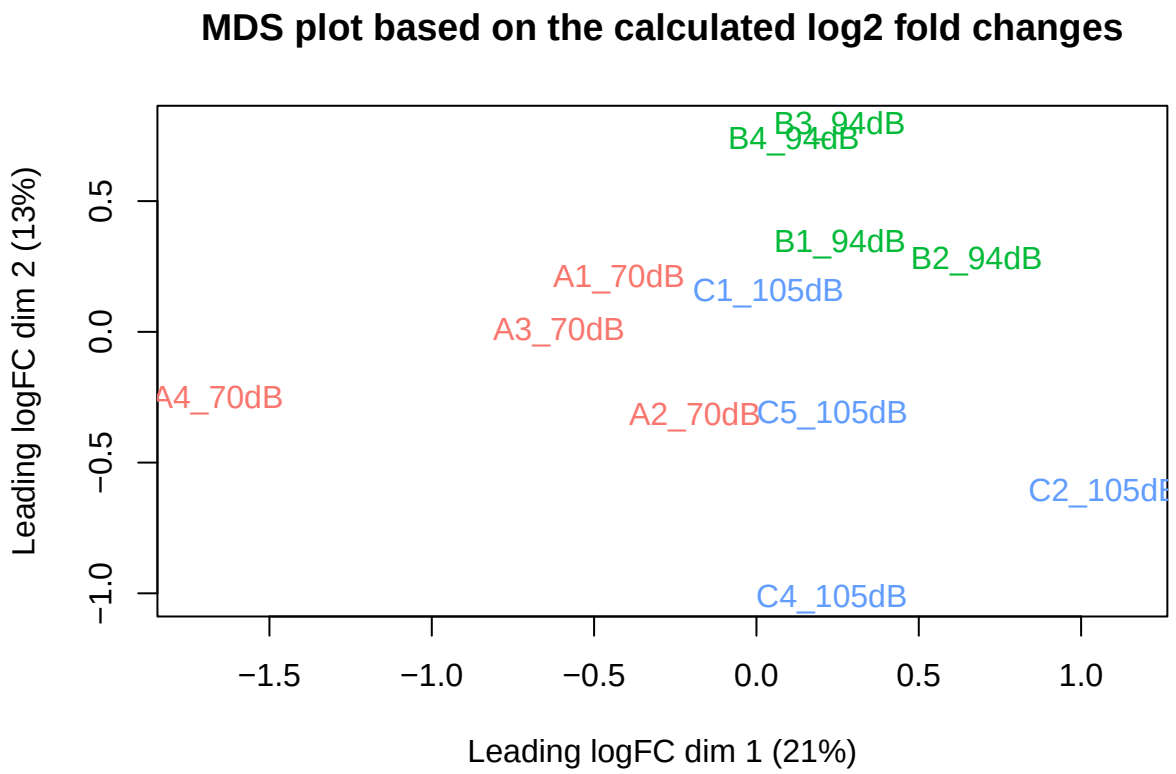
edgeR

For `edgeR` the data needs to be in a `DGEList` object, it contains a `counts` matrix containing the count integer data and a data.frame `samples` containing information about the samples (like the group the sample belongs to). Because in #3.1 filtering was already done, the built-in filtering of edgeR does not need to be used. It was used and checked but using it did not even remove any other genes from the data. Earlier in the chapter a manual normalization was performed but now the built-in edgeR normalization will be used based on TTM instead, this is the same method as used in the chosen research.

```
# Create DGEList object with filtered data and group factor
myDgeList <- DGEList(counts=myData.filtered,group=groups)
```

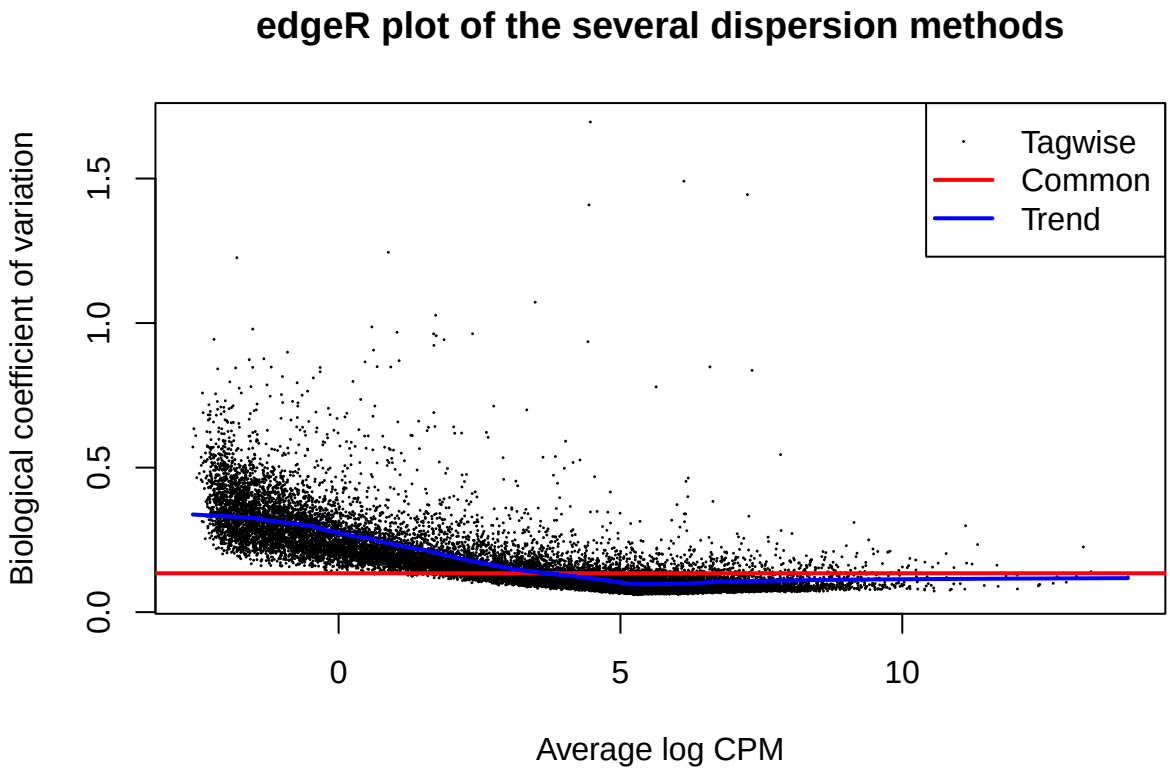
To further prepare the data the `calcNormFactors` function normalizes the library sizes by finding a set of scaling factors for the library sizes that minimizes the log-fold changes between the samples for most genes. A MDS plot will be made to see what the log2 FC distances are between samples.

```
# Perform edgeR's normalization
myDgeList <- calcNormFactors(myDgeList)

# Create MDS plot
plotMDS(myDgeList, col=myColors,
        main="MDS plot based on the calculated log2 fold changes")
```



Then the `Cox-Reid profile-adjustedlikelihood (CR)` method that is based on the idea of approximate conditional likelihood is used. When given the DGEList object and the design matrix, generalized linear models are fitted. This allows valid estimation of the dispersion, since all systematic sources of variation are accounted for. To estimate common dispersion, trended dispersions and tagwise dispersions in one run the `estimateDisp()` function is called.

```
# Estimate all dispersions
myDgeList <- estimateDisp(myDgeList, myDesign)

# Create plot showing the dispersions
plotBCV(myDgeList, main="edgeR plot of the several dispersion methods")
```



To finally get DEGs a few more steps need to be done with the edgeR package. To begin the `glmQLFit()` function is used which fits the negative binomial ***Generalized linear models (GLM)*** for each tag and produces an object of the `DGEGLM` class. This object can then be passed to the `glmQLFTest()` function that carries out the ***quasi-likelihood (QL)*** F-test.

```
# Fit negative binomial GLMs
fit <- glmQLFit(myDgeList, myDesign)

# Perform QL F-test of 94dB against 70dB
qlf.94dBvs70dB <- glmQLFTest(fit, coef=2)
# Perform QL F-test of 105dB against 70dB
qlf.105dBvs70dB <- glmQLFTest(fit, coef=3)
# Perform QL F-test of 105dB against 94dB
qlf.105dBvs94dB <- glmQLFTest(fit, contrast=c(0,-1,1))
```

```
# Find genes that are different between any of the groups
qlf <- glmQLFTest(fit, coef=2:3)

# Show 105dB vs 70dB top DEGs
topTags(qlf.105dBvs70dB)
```

```
## Coefficient:  groups105dB
##                       logFC   logCPM        F       PValue          FDR
## ENSMUSG00000020108 2.663762 7.325319 318.0243 1.388976e-11 2.821704e-07
## ENSMUSG00000021025 1.166068 6.701394 235.0666 1.224944e-10 6.121148e-07
## ENSMUSG00000048546 1.128406 6.461517 234.9156 1.230583e-10 6.121148e-07
## ENSMUSG00000002910 1.668081 6.025176 231.7275 1.356748e-10 6.121148e-07
## ENSMUSG00000024190 1.825942 7.175793 222.7171 1.800266e-10 6.121148e-07
## ENSMUSG00000024042 1.537038 6.482945 222.5853 1.807871e-10 6.121148e-07
## ENSMUSG00000090698 2.177447 5.026683 212.4147 2.521430e-10 7.317549e-07
## ENSMUSG00000020893 1.898047 7.189611 192.1677 5.124513e-10 1.301306e-06
## ENSMUSG00000021250 2.691451 5.596028 188.0131 5.978380e-10 1.349453e-06
## ENSMUSG00000028195 1.718826 7.705419 184.1496 6.919440e-10 1.405684e-06
```

```
# Create plot showing DEGs
deGenes <- decideTestsDGE(qlf.105dBvs70dB, p=0.05)
deGenes <- rownames(qlf.105dBvs70dB)[as.logical(deGenes)]
plotSmear(qlf.105dBvs70dB, de.tags=deGenes,
          main="edgeR results; genes marked in red are DEGs")
abline(h=c(-1, 1), col=2)
```