Exercise 2. Statistical Analysis of RNA-Seq Data

The DESeq2 developers provide a clearly written vignette how to use the software (https://www.bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html). Many of the examples used in this exercise are from this manual. When it comes time to work with your own data, we highly recommend that you first read this vignette.

Part 1. Basic Analysis: Identify differentially expressed (DE) genes

1. Start R.

There are several different versions of R installed on BioHPC computers. The most recent installed version is located in the directory `/programs/R-4.0.0`. If you create and navigate (`cd`) to the directory `/workdir/$USER/exercise2` before you start R, this will be your working directory within R. (*Note that* `$USER` *is a system environment variable that holds your BioHPC userid*). We'll also start a screen session in case you lose your connection.

```
mkdir /workdir/$USER/exercise2
cd /workdir/$USER/exercise2
screen
/programs/R-4.0.0/bin/R
```

2. Load the matrix and sample files into R, and examine their contents.

In the exercise from the first week of this workshop, you created a read count matrix file named `gene_count.txt`. This file contains read counts for 6 samples (wt1, wt2, wt3, mu1, mu2, mu3) . If you did not keep this file, you can use the `gene_count.txt` file located inside `/shared_data/RNAseq/exercise2/`. Also, there is tab-delimited text file `samples.txt` to describe the 6 samples, which looks like this:

Sample	Genotype
wt1	wt
wt2	wt
wt3	wt
mu1	mu
mu2	mu
mu3	mu

Two new R objects will be created: `cts` (from the `gene_count.txt` file) and `coldata` (from the `samples.txt` file). As the `gene_count.txt` file does not have the sample names, we will run the R command `colnames(cts) <- rownames(coldata)` to take the row names of `coldata` and use them as the column names for the `cts` matrix.

```
# Note that lines like this one (or # text snippets) that start with "#" are
comments
#   - you don't have to copy/paste them, but if you do they will be ignored
anyway

matrixFile <- "/shared_data/RNAseq/exercise2/gene_count.txt"
sampleFile <- "/shared_data/RNAseq/exercise2/samples.txt"
cts <- as.matrix(read.csv(matrixFile, sep="\t", row.names=1, header=FALSE))
coldata <- read.csv(sampleFile, sep="\t", row.names=1, header=TRUE)
head(coldata)
head(cts)  # the columns have generic names (V1,V2,...)

# assign the row names of coldata to the column names of cts
colnames(cts) <- rownames(coldata)
head(cts)  # looks good now
```

3. Load the DESeq2 library. Create a DESeq2 object named dds from the gene read count and sample information.

```
library(DESeq2)
dds <- DESeqDataSetFromMatrix(countData = cts, colData = coldata, design = ~
Genotype)
dds  # get some info on what's inside dds

# show the corresponding Design Matrix that DESeq2 will use
model.matrix(~ Genotype, coldata)

##     (Intercept) Genotypewt
## wt1           1          1
## wt2           1          1
## wt3           1          1
## mu1           1          0
## mu2           1          0
## mu3           1          0
```

4. Create a PCA plot from the DESeq2 object, using the default (500) number of most variable genes

(*Skip this step if you did it already in Exercise 1 last week*.)

```
library(ggplot2)
vsd <- vst(dds, blind=FALSE)
pcaData <- plotPCA(vsd, intgroup=c("Genotype"), returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=Genotype)) +
geom_point(size=3) +
xlim(-2.5, 2.5) +
ylim(-1, 1) +
xlab(paste0("PC1: ",percentVar[1],"% variance")) +
ylab(paste0("PC2: ",percentVar[2],"% variance")) +
geom_text(aes(label=name),vjust=2)
ggsave("myplot.png")

# The message:
## Saving 7 x 7 in image
```

```
# indicates that the plot was sucessfully created
```

The `vst(dds, blind=FALSE)` part performs a variance stabilizing transformation of the normalized counts, to prevent a handful genes with the highest expression levels and most variance from dominating the PCA plot.

5. Create another PCA plot, this time using the 100 most variable genes (`ntop=100`), instead of the default of 500.

```
library(ggplot2) # if you skipped step 4
vsd <- vst(dds, blind=FALSE) # if you skipped step 4

pcaData100 <- plotPCA(vsd, intgroup=c("Genotype"), ntop=100, returnData=TRUE)
percentVar <- round(100 * attr(pcaData100, "percentVar"))
ggplot(pcaData100, aes(PC1, PC2, color=Genotype)) +
geom_point(size=3) +
xlim(-2.5, 2.5) +
ylim(-1, 1) +
xlab(paste0("PC1: ",percentVar[1],"% variance")) +
ylab(paste0("PC2: ",percentVar[2],"% variance")) +
geom_text(aes(label=name),vjust=2)
ggsave("PCA_100.png")
```

Use FileZilla to download the `PCA_100.png` file to your laptop, and double click the file to view the plot. How does it compare to the plot with the top 500 genes, `myplot.png`?

6. Get a list of differentially expressed genes.

```
dds<-DESeq(dds)  # the DESeq() function performs the analysis on the dds data
set

# save the normalized counts to a tab-delimited text file
write.table(counts(dds,normalized=TRUE),file='Part1_normCounts.txt',sep="\t",row
.names=TRUE,col.names=TRUE)

# get the DE results
resultsNames(dds)  # lists the coefficients
res <- results(dds, name="Genotype_wt_vs_mu")
summary(res) # print the summary on screen, you should see the following:

## out of 6435 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 1347, 21%
## LFC < 0 (down)     : 1020, 16%
## outliers [1]       : 0, 0%
## low counts [2]     : 248, 3.9%
## (mean count < 0)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results

# another way to see the results (header and first 6 genes)
head(res)

## log2 fold change (MLE): Genotype wt vs mu  # *** What if you want mu vs wt???
## Wald test p-value: Genotype wt vs mu
```

```
## DataFrame with 6 rows and 6 columns
##            baseMean log2FoldChange     lfcSE      stat     pvalue      padj
##           <numeric>      <numeric> <numeric> <numeric>  <numeric> <numeric>
## YDL248W   11.327012       0.446508  0.450896  0.990269 0.32204274 0.4930657
## YDL247W-A  0.000000             NA        NA        NA         NA        NA
## YDL247W    0.325112      -1.232607  2.682385 -0.459519 0.64586146        NA
## YDL246C    0.253939       2.167765  3.242605  0.668526 0.50379810        NA
## YDL245C    1.257693       4.388935  3.432627  1.278594 0.20104021 0.3530768
## YDL244W    4.854093      -3.115036  1.124672 -2.769730 0.00561028 0.0199832
```

7. Switch the order of the levels so wildtype is the "control" or "reference" level.

Note the the comparison is wild type (`wt`) vs mutant (`mu`), because DESeq2 (R, actually) decides which level to use as the reference (control) level based their alphabetical order. We can force the mutant vs wildtype comparison (*i.e.*, make wildtype the control) by specifying the contrast, as follows:

```
# see the current ordering of the levels for the factor Genotype
dds$Genotype
## [1] wt wt wt mu mu mu
## Levels: mu wt

# swap the levels, then confirm that it worked (wt will now be the reference
level)
dds$Genotype <- factor(dds$Genotype, levels = c("wt","mu"))
dds$Genotype
## [1] wt wt wt mu mu mu
## Levels: wt mu

# re-run the analyis with the re-ordered levels
dds<-DESeq(dds)
resultsNames(dds)
## [1] "Intercept"         "Genotype_mu_vs_wt"  # *** Ah, that's better now!!!

res <- results(dds, name="Genotype_mu_vs_wt")
summary(res) # same as before but up and down are flipped

head(res)
# now the signs of the log2FoldChange are flipped:
## log2 fold change (MLE): Genotype mu vs wt  # *** Ah, that's better now!!!
## Wald test p-value: Genotype mu vs wt
## DataFrame with 6 rows and 6 columns
##            baseMean log2FoldChange     lfcSE      stat     pvalue      padj
##           <numeric>      <numeric> <numeric> <numeric>  <numeric> <numeric>
## YDL248W   11.327012      -0.446508  0.450896 -0.990269 0.32204274 0.4930657
## YDL247W-A  0.000000             NA        NA        NA         NA        NA
## YDL247W    0.325112       1.232607  2.682385  0.459519 0.64586146        NA
## YDL246C    0.253939      -2.167765  3.242605 -0.668526 0.50379810        NA
## YDL245C    1.257693      -4.388935  3.432627 -1.278594 0.20104021 0.3530768
## YDL244W    4.854093       3.115036  1.124672  2.769730 0.00561028 0.0199832


# get information on what the columns contain:
mcols(res)$description
```

```
## [1] "mean of normalized counts for all samples"
## [2] "log2 fold change (MLE): Genotype mu vs wt"
## [3] "standard error: Genotype mu vs wt"
## [4] "Wald statistic: Genotype mu vs wt"
## [5] "Wald test p-value: Genotype mu vs wt"
## [6] "BH adjusted p-values"  # *** accounting for multiple testing (=FDR)


# Filter the results for FDR < 0.05 and
#  absolute value of Fold-Change > 2 (i.e., at least double or half the level of
expression)
res05 <- res[which(res$padj<0.05 & abs(res$log2FoldChange)>log2(2)), ]

# sort by padj and write to file
res05Ordered <- res05[order(res05$padj),]
write.csv(as.data.frame(res05Ordered), file="wt_vs_mu_fdr05_fc2_results.csv")
```

You can download the file "wt_vs_mu_fdr05_results.csv" with FileZilla and open it in Excel.

8. Shrinkage of log fold change of genes with very low expression.

Genes with very low expression usually have very noisy estimates of log2 fold changes. It is desirable to shrink the fold change of genes with low read counts, but not shrink the fold change of highly expressed genes too much. DEseq2 has implemented several different algorithms for shrinkage. The DESeq2 developers recommend to use `apeglm` method for shrinkage. Shrinkage is especially important if you plan to use LFC to rank genes for enrichment analysis (e.g., GSEA, to be covered next week).

Here you will compare the MA plot with or without shrinkage. The commands provided here write the plots to pdf files, and you can download and view them on your laptop.

```
# no shrinkage
res <- results(dds, name="Genotype_mu_vs_wt")
pdf("res_no_shrink.pdf")
plotMA(res, main = "No shrinkage", alpha=0.05, ylim=c(-4,4))
dev.off()

# LFC estimates shrunken by apeglm method
res_shrink <- lfcShrink(dds, coef="Genotype_mu_vs_wt", type="apeglm")
pdf("res_shrink.pdf")
plotMA(res_shrink, main = "Shrinkage by apeglm", alpha=0.05, ylim=c(-4,4))
dev.off()
```

Part 2. Interactions

1. The data files.

In this experiment, there are 12 samples, with two variables: strains (wt vs mut), and sample collection time (0 vs 120 minutes of osmotic stress). You will use two input files: `fission_gene_count.csv` (count matrix) and fission_sample.csv (sample annotation). The sample annotation file (`fission_sample.csv`) looks like this:

	strain	minute
GSM1368273	wt	0
GSM1368274	wt	0
GSM1368275	wt	0
GSM1368285	wt	120
GSM1368286	wt	120
GSM1368287	wt	120
GSM1368291	mut	0
GSM1368292	mut	0
GSM1368293	mut	0
GSM1368303	mut	120
GSM1368304	mut	120
GSM1368305	mut	120

Reference:
Leong, S. H, Dawson, K., Wirth, C., Li, Y., Connolly, Y., Smith, L. D, Wilkinson, R. C, Miller, J. C (2014). "A global non-coding RNA system modulates fission yeast protein levels in response to stress." *Nat Commun*, **5**, 3947. http://www.ncbi.nlm.nih.gov/pubmed/24853205.

2. Load data files into R.

As the values of one of the variables, "minute", are numeric, the minute variable needs to be converted to a factor.

```
matrixFile <- "/shared_data/RNAseq/exercise2/fission_gene_count.csv"
sampleFile <- "/shared_data/RNAseq/exercise2/fission_sample.csv"
cts <- as.matrix(read.csv(matrixFile, row.names=1))
coldata <- read.csv(sampleFile, row.names=1)

# convert the numeric variable "minute" into a factor
coldata$minute<-as.factor(coldata$minute)
```

3. Test of interactions of the strain and minute variable.

```
library(DESeq2)  # not needed if this is already loaded
dds <- DESeqDataSetFromMatrix(countData = cts,
     colData = coldata,
     design = ~ strain + minute + strain:minute)

# show the corresponding Design Matrix that DESeq2 will use
model.matrix(~ strain + minute + strain:minute, coldata)

##             (Intercept) strainwt minute120 strainwt:minute120
## GSM1368273           1        1         0                  0
## GSM1368274           1        1         0                  0
```

```
## GSM1368275           1        1         0                  0
## GSM1368285           1        1         1                  1
## GSM1368286           1        1         1                  1
## GSM1368287           1        1         1                  1
## GSM1368291           1        0         0                  0
## GSM1368292           1        0         0                  0
## GSM1368293           1        0         0                  0
## GSM1368303           1        0         1                  0
## GSM1368304           1        0         1                  0
## GSM1368305           1        0         1                  0


# perform a likelihood ratio test, testing the significance of the
#  strain:minute interaction (by omitting that term from the reduced model)
ddsLRT <- DESeq(dds, test="LRT", reduced= ~ strain + minute)
resLRT <- results(ddsLRT)
summary(resLRT)
# no significant interactions were found:
## out of 6703 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 0, 0%
## LFC < 0 (down)     : 0, 0%
## outliers [1]       : 1, 0.015%
## low counts [2]     : 0, 0%
## (mean count < 0)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results

# see more info on the test performed and the results for the 6 most signif
genes:
options(width=150) # increase the width of the screen output
head(resLRT[order(resLRT$pvalue),])
## log2 fold change (MLE): strainwt.minute120
## LRT p-value: '~ strain + minute + strain:minute' vs '~ strain + minute'
## DataFrame with 6 rows and 6 columns
##                 baseMean log2FoldChange     lfcSE      stat      pvalue
padj
##                <numeric>      <numeric> <numeric> <numeric>   <numeric>
<numeric>
## SPAC11D3.01c  313.13721       1.793633  0.463539  14.76077 0.000122048
0.817967
## SPNCRNA.1621    9.30094      -3.095840  0.997907  10.26607 0.001354990
1.000000
## SPCC1281.04    20.18763       2.487489  0.829423   9.29592 0.002296643
1.000000
## SPAC5H10.09c  129.22395       0.664890  0.225595   8.70302 0.003176835
1.000000
## SPBC336.16      7.65615      -3.029877  1.103006   7.79431 0.005241094
1.000000
## SPRRNA.32    4780.82382       0.737386  0.269399   7.47301 0.006263084
1.000000


# save the normalized counts to a tab-delimited text file
fissionNormCounts <- counts(ddsLRT,normalized=TRUE)
write.table(fissionNormCounts,file='fission_normCounts.txt',sep="\t",row.names=T
RUE,col.names=TRUE)
```

Since no significant interactions were found (after correction for multiple testing), we can conclude that gene expression in the two yeast strains (wild type and mutant) responds to 120 minutes of osmotic stress in a similar fashion, as far as we can tell given the statistical power of this experiment. We were not able to detect any genes in the mutant that responded differently than in the wild type. An interaction would have been detected, for example, if there was a gene in that significantly increased its expression level in the mutant in response to stress, but did not change much in the wild type. If such genes truly exist, we'll need more biological replicates (more statistical power) to detect them. Indeed, Leong et al. (2014) concluded that "general osmotic stress responses of fission yeast were not impaired by *atf21* deletion."

However, **one gene (SPAC11D3.01c) has a quite low p-value** that, sadly, did not survive correction for multiple testing -- it certainly seems worthy of further study! Let's check it out:

```
data<-plotCounts(ddsLRT,
  gene="SPAC11D3.01c",
  intgroup=c("strain","minute"),
  returnData=TRUE)
ggplot(data, mapping = aes(x=strain, y=count, color=minute,group=minute)) +
geom_point() +
scale_y_log10()
ggsave("SPAC11D3.01c.png")
```

Examining this plot (after downloading it with FileZilla) it seems that the expression level of the gene SPAC11D3.01c responds much more dramatically to osmotic stress in the wild type than in the mutant. This is an interaction! (*But it still doesn't survive correction for multiple testing, as it has a 82% chance of being a false positive, according the the padj*.)

Part 3. Batch effects

1. The data files.

In this experiment, there are 7 samples from two conditions (untreated and treated), and the data are from two different batches (single- and paired-end sequencing). We will correct for the batch effect in this analysis.

Reference for this data set: Brooks AN, Yang L, Duff MO, Hansen KD, Park JW, Dudoit S, Brenner SE, Graveley BR (2011) Conservation of an RNA regulatory map between Drosophila and mammals Genome Res. 21(2):193-202, Epub 2010 Oct 4, PMID: 20921232.

Here's what the sample annotation file (`pasilla_sample_annotation.csv`) looks like:

file	condition	type
treated1fb	treated	single-read
treated2fb	treated	paired-end
treated3fb	treated	paired-end
untreated1fb	untreated	single-read
untreated2fb	untreated	single-read
untreated3fb	untreated	paired-end
untreated4fb	untreated	paired-end

Run the following commands to load the data set into R. There are two files: the read count file `pasilla_gene_counts.tsv` and the sample annotation file `pasilla_sample_annotation.csv`.

```
matrixFile <- "/programs/R-
3.5.0s/library/pasilla/extdata/pasilla_gene_counts.tsv"
sampleFile <- "/programs/R-
3.5.0s/library/pasilla/extdata/pasilla_sample_annotation.csv"
cts <- as.matrix(read.csv(matrixFile,sep="\t",row.names="gene_id"))
coldata <- read.csv(sampleFile, row.names=1)
coldata <- coldata[,c("condition","type")]
```

We need to remove the two extra characters ("fb") in sample names of the sample annotation file (and convert the "-" in the type to "_" to remove some warning messages). We also need to fix the order of samples in the read count matrix, to be consistent with sample annotation file (*it's crucial that the order matches!*).

```
rownames(coldata) <- sub("fb", "", rownames(coldata))
coldata$type <- sub("-", "_", coldata$type)
coldata$type <- as.factor(coldata$type)

# fix the order of the samples in the read count matrix
cts <- cts[, rownames(coldata)]
```

2. DE test accounting for the batch effect

We include the batch effect variable `type` in the model by using the design formula `type + condition`. Then we retrieve the results for the factor `condition`, with batch effect `type` corrected.

```
library(DESeq2) # if it is not already loaded

dds <- DESeqDataSetFromMatrix(countData = cts,
  colData = coldata,
  design = ~ type + condition)
dds <- DESeq(dds)

# show the corresponding Design Matrix that DESeq2 will use
model.matrix(~ type + condition, coldata)
```

```
##            (Intercept) typesingle_read conditionuntreated
## treated1             1               1                   0
## treated2             1               0                   0
## treated3             1               0                   0
## untreated1           1               1                   1
## untreated2           1               1                   1
## untreated3           1               0                   1
## untreated4           1               0                   1

# save the normalized counts to a tab-delimited text file
pasillaNormCounts <- counts(dds,normalized=TRUE)
write.table(pasillaNormCounts,file='pasilla_normCounts.txt',sep="\t",row.names=T
RUE,col.names=TRUE)

resultsNames(dds)
## [1] "Intercept"                      "type_single_read_vs_paired_end"
"condition_untreated_vs_treated"

# force the comparison of treated vs untreated using "contrast"
res <- results(dds, contrast=c("condition","treated","untreated"))
summary(res)
head(res[order(res$pvalue),])
```

3. Plot PCA before and after removing batch effect.

Although the batch effect was accounted for in the above DE analysis, it will still be present in the variance stabilized counts and visible in the PCA (and can be diagnosed from that) unless you explicitly remove it with with `limma::removeBatchEffect`.

a. PCA plot before removing batch effect

```
library(ggplot2)
vsd <- vst(dds, blind=FALSE)
pcaData <- plotPCA(vsd, intgroup=c("condition", "type"), returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=condition, shape = type)) +
geom_point(size=3) +
xlim(-12, 12) +
ylim(-10, 10) +
xlab(paste0("PC1: ",percentVar[1],"% variance")) +
ylab(paste0("PC2: ",percentVar[2],"% variance")) +
geom_text(aes(label=name),vjust=2)
ggsave("myPCAWithBatchEffect.png")
```

b. PCA plot after removing the batch effect with `limma::removeBatchEffect`. This is a good idea if you plan to use the variance stabilized counts (`vsd`) in downstream analysis such as WGCNA (covered next week).

```
assay(vsd) <- limma::removeBatchEffect(assay(vsd), vsd$type)

pcaData <- plotPCA(vsd, intgroup=c("condition", "type"), returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=condition, shape = type)) +
geom_point(size=3) +
xlim(-12, 12) +
ylim(-10, 10) +
```

```
xlab(paste0("PC1: ",percentVar[1],"% variance")) +
ylab(paste0("PC2: ",percentVar[2],"% variance")) +
geom_text(aes(label=name),vjust=2)
ggsave("myPCABatchEffectRemoved.png")

# save the batch effect-corrected, variance stabilized counts to a tab-delimited
text file
write.table(assay(vsd),file='pasilla_batchCorrectedVSD.txt',sep="\t",row.names=T
RUE,col.names=TRUE)
```

4. Heatmap of the distances between the samples

Examining the distances between samples is an additional method, along with PCA, for identifying outlier samples.

```
library("pheatmap")
library("RColorBrewer")
sampleDists <- dist(t(assay(vsd)))
sampleDistMatrix <- as.matrix(sampleDists)
rownames(sampleDistMatrix) <- paste(vsd$condition, vsd$type, sep="-")
colnames(sampleDistMatrix) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pdf("heatmapSampleDists.pdf", height = 6, width = 9)
pheatmap(sampleDistMatrix,
         clustering_distance_rows=sampleDists,
         clustering_distance_cols=sampleDists,
         col=colors)
dev.off()
```

5. Heatmap of batch-corrected variance stabilized counts for the 20 most siginificant genes

This provides a visual diplay of the differential expression of the most significant genes. Plotting the deviation from the mean variance stabilized count (after batch correction) makes the contrasts more clear.

```
# get the gene names for the top 20
top20 <- res[order(res$pvalue),]
top20 <- rownames(top20[1:20,])

# get a data frame containing the plot annotation data
df <- as.data.frame(colData(dds)[,c("condition","type")])

mat <- assay(vsd)[top20,]  # get the vsd for the top 20 genes
mat <- mat - rowMeans(mat)  # plot the deviation from the mean var stab count
pdf("heatmapVSDCounts.pdf", height = 8, width = 10)
pheatmap(mat,annotation_col=df)
dev.off()
```

Exit R

When you are done, exit R as follows:

```
q()
## Save workspace image? [y/n/c]:   <--- Type "n" for no.
```