Table of Contents

title: “DEG_July2019_Kopp” author: “Soheila” date: ‘2019-07-28’ output: word_document: default html_document: default

{r, echo=TRUE}

knitr::opts_chunk$set(error = TRUE)

```
knitr::opts_chunk$set(echo = TRUE, eval = FALSE)

Read_count_all_removedp53_July28_2018<-
read.csv("Data/Read_count_all_removedp53_Oct22_2018.csv", header =
TRUE, row.names = 1)
head(Read_count_all_removedp53_July28_2018)

##     Gene  Du2  Du3  Du7 Du10 Du11 Du12 Du17 Du18_19 Du20 Du22 Du23
Du24
## 1   Xkr4    7    6    6    1    0    4    0       6    8    1    4
5
## 2    Rp1    0    0    0    0    0    1    0       2    0    1    0
0
## 3  Sox17 2462 2402 3573 2393   75 3124 1037   16911  296 1124  811
760
## 4 Mrpl15 1797 1680 2070 1528 1219 1687  869    3050 2451 1511 1768
1783
## 5 Lypla1 2526 2475 2630 1943  887 1922  622    4767 2234 1862 1495
1728
## 6  Tcea1 3829 3609 2099 2900 1226 2465 1015    2957 2194 1432 1607
1779
##   Du25  AC3  AC5  AC7  AC8 AC10 AC12 AC13 AC14 AC16 AC17 AC19 AC23
## 1    9    5    0    2    2    6    2    0    0    2    3    2    1
## 2    0    0    0    1    1    0    0    0    0    1    0    0    0
## 3  662 3833 3094 6873 2318  589 6373 6072 5273 6859 4167 5033  292
## 4 1587 1515 1521 1001 1316 1387 1083 1455 1772  995 1182 1061 1351
## 5 2033 2583 2313  967 2224 2905 1857 1762 1852 1642 1525 1569 1059
## 6 1893 2836 2983 1090 2934 2837  914 1395 1425 1069 1105  902 1171

all_reads_July2019<- Read_count_all_removedp53_July28_2018
View(all_reads_July2019)
```

Count matrix input Alternatively, the function DESeqDataSetFromMatrix can be used if you already have a matrix of read counts prepared from another source. Another method for quickly producing count matrices from alignment files is the

featureCounts function (Liao, Smyth, and Shi 2013) in the Rsubread package. To use DESeqDataSetFromMatrix, the user should provide the counts matrix, the information about the samples (the columns of the count matrix) as a DataFrame or data.frame, and the design formula.

To demonstate the use of DESeqDataSetFromMatrix, we will read in count data. We read in a count matrix, which we will name cts, and the sample information table, which we will name coldata. Further below we describe how to extract these objects from, e.g. featureCounts output.

```
sort(all_reads_July2019$Gene)
row.names(all_reads_July2019) <- all_reads_July2019$Gene
head(all_reads_July2019)
cts_July_2019 <- all_reads_July2019[,-1]
head(cts_July_2019)
sampleinfo_Oct23<-
read.csv("Data/phenotype_removed_Com_all_samples_Oct23_2018.csv",
row.names = 1)
```

Note that these are not in the same order with respect to samples!

It is absolutely critical that the columns of the count matrix and the rows of the column data (information about samples) are in the same order. DESeq2 will not make guesses as to which column of the count matrix belongs to which row of the column data, these must be provided to DESeq2 already in consistent order.

```
library(DESeq2)
sampleinfo_Oct23
colnames(cts_July_2019)
rownames(sampleinfo_Oct23)
colData<- sampleinfo_Oct23
all(rownames(colData) %in% colnames(cts_July_2019))
```

If you have used the featureCounts function (Liao, Smyth, and Shi 2013) in the Rsubread package, the matrix of read counts can be directly provided from the "counts" element in the list output. The count matrix and column data can typically be read into R from flat files using base R functions such as read.csv or read.delim. For htseq-count files, see the dedicated input function below.

With the count matrix, cts, and the sample information, coldata, we can construct a DESeqDataSet:

```
dds <- DESeqDataSetFromMatrix ( countData = cts_July_2019,
                                colData,
                                design = ~ B + P + Sex + Condition )

dds
```

Pre-filtering While it is not necessary to pre-filter low count genes before running the DESeq2 functions, there are two reasons which make pre-filtering useful: by

removing rows in which there are very few reads, we reduce the memory size of the dds data object, and we increase the speed of the transformation and testing functions within DESeq2. Here we perform a minimal pre-filtering to keep only rows that have at least 10 reads total. Note that more strict filtering to increase power is automatically applied via independent filtering on the mean of normalized counts within the results function.

```
library(dplyr)
library("BiocParallel")
register(MulticoreParam(4))
colData ( dds) %>% head
assay ( dds) %>% head
rowRanges ( dds) %>% head
dds <- dds[ rowSums ( counts ( dds)) > 1, ]
```

Note on factor levels By default, R will choose a reference level for factors based on alphabetical order. Then, if you never tell the DESeq2 functions which level you want to compare against (e.g. which level represents the control group), the comparisons will be based on the alphabetical order of the levels. There are two solutions: you can either explicitly tell results which comparison to make using the contrast argument (this will be shown later), or you can explicitly set the factors levels. You should only change the factor levels of variables in the design before running the DESeq2 analysis, not during or afterward. Setting the factor levels can be done in two ways, either using factor:

```
dds$Condition<- factor(dds$Condition, levels = c("A", "D"))
dds$Condition<- relevel(dds$Condition, ref = "A")
```

Using parallelization The above steps should take less than 30 seconds for most analyses. For experiments with complex designs and many samples (e.g. dozens of coefficients, ~100s of samples), one can take advantage of parallelized computation. Parallelizing DESeq, results, and lfcShrink can be easily accomplished by loading the BiocParallel package, and then setting the following arguments: parallel=TRUE and BPPARAM=MulticoreParam(4), for example, splitting the job over 4 cores. Note that results for coefficients or contrasts listed in resultsNames(dds) is fast and will not need parallelization. As an alternative to BPPARAM, one can register cores at the beginning of an analysis, and then just specify parallel=TRUE to the functions when called.

```
library("BiocParallel")
register(MulticoreParam(4))
```

Data transformations and visualization Count data transformations However for other downstream analyses – e.g. for visualization or clustering – it might be useful to work with transformed versions of the count data. One makes use of the concept of variance stabilizing transformations (VST) (Tibshirani 1988; Huber et al. 2003; Anders and Huber 2010), and the other is the regularized logarithm or rlog, which incorporates a prior on the sample differences (Love, Huber, and Anders 2014).

Both transformations produce transformed data on the log2 scale which has been normalized with respect to library size or other normalization factors.

```
#vsd <- vst(dds, blind=FALSE)
rld <- rlog(dds, blind=FALSE)
head(assay(rld), 3)
```

Effects of transformations on the variance The figure below plots the standard deviation of the transformed data, across samples, against the mean, using the shifted logarithm transformation, the regularized log transformation and the variance stabilizing transformation. The shifted logarithm has elevated standard deviation in the lower count range, and the regularized log to a lesser extent, while for the variance stabilized data the standard deviation is roughly constant along the whole dynamic range.

Note that the vertical axis in such plots is the square root of the variance over all samples, so including the variance due to the experimental conditions. While a flat curve of the square root of variance over the mean may seem like the goal of such transformations, this may be unreasonable in the case of datasets with many true differences due to the experimental conditions.

```
# this gives log2(n + 1)
ntd <- normTransform(dds)
library("vsn")
meanSdPlot(assay(ntd))

meanSdPlot(assay(vsd))

meanSdPlot(assay(rld))
```

Data quality assessment by sample clustering and visualization Data quality assessment and quality control (i.e. the removal of insufficiently good data) are essential steps of any data analysis. These steps should typically be performed very early in the analysis of a new data set, preceding or in parallel to the differential expression testing.

Heatmap of the sample-to-sample distances Another use of the transformed data is sample clustering. Here, we apply the dist function to the transpose of the transformed count matrix to get sample-to-sample distances.

```
sampleDists <- dist(t(assay(rld)))
sampleDists
plot ( hclust ( sampleDists ),
       labels = colnames ( sampleDists),
       main = " rld transformed read counts \ ndistance : Pearson
correlation ")
#obtain regularized log - transformed values

library("RColorBrewer")
sampleDistMatrix <- as.matrix(sampleDists)
```

```
rownames(sampleDistMatrix) <- paste(vsd$condition, vsd$type, sep="-")
colnames(sampleDistMatrix) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
         clustering_distance_rows=sampleDists,
         clustering_distance_cols=sampleDists,
         col=colors)
```

Principal component plot of the samples Related to the distance matrix is the PCA plot, which shows the samples in the 2D plane spanned by their first two principal components. This type of plot is useful for visualizing the overall effect of experimental covariates and batch effects.

```
plotPCA(vsd, intgroup= c("Condition", "Sex"))
plotPCA(vsd, intgroup= c("Condition"))

library(ggplot2)
jpeg("PCA_July2019", width = 4, height = 4, units = 'in', res = 300)

pcaData <- plotPCA(vsd, intgroup=c("Condition", "Sex"),
returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=Condition, shape=Sex)) +
  geom_point(size=3) +
  xlab(paste0("PC1: ",percentVar[1],"% variance")) +
  ylab(paste0("PC2: ",percentVar[2],"% variance")) +
  coord_fixed()
dev.off()
```

Variations to the standard workflow Wald test individual steps The function DESeq runs the following functions in order:

```
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
dds <- nbinomWaldTest(dds)
```

Differential expression analysis Likelihood ratio test DESeq2 offers two kinds of hypothesis tests: the Wald test, where we use the estimated standard error of a log2 fold change to test if it is equal to zero, and the likelihood ratio test (LRT). The LRT examines two models for the counts, a full model with a certain number of terms and a reduced model, in which some of the terms of the full model are removed. The test determines if the increased likelihood of the data using the extra terms in the full model is more than expected if those extra terms are truly zero.

The LRT is therefore useful for testing multiple terms at once, for example testing 3 or more levels of a factor at once, or all interactions between two variables. The LRT for count data is conceptually similar to an analysis of variance (ANOVA) calculation in linear regression, except that in the case of the Negative Binomial GLM, we use an analysis of deviance (ANODEV), where the deviance captures the difference in likelihood between a full and a reduced model.

The likelihood ratio test can be performed by specifying test="LRT" when using the DESeq function, and providing a reduced design formula, e.g. one in which a number of terms from design(dds) are removed. The degrees of freedom for the test is obtained from the difference between the number of parameters in the two models. A simple likelihood ratio test, if the full design was ~condition would look like:

```
dds <- DESeq(dds, test="LRT",  reduced = ~ P + B + Sex, parallel=TRUE)
res <- results(dds)
res
```

Log fold change shrinkage for visualization and ranking Shrinkage of effect size (LFC estimates) is useful for visualization and ranking of genes. To shrink the LFC, we pass the dds object to the function lfcShrink. Below we specify to use the apeglm method for effect size shrinkage (Zhu, Ibrahim, and Love 2018), which improves on the previous estimator.

We provide the dds object and the name or number of the coefficient we want to shrink, where the number refers to the order of the coefficient as it appears in resultsNames(dds).

```
resultsNames(dds)
```

p-values and adjusted p-values We can order our results table by the smallest p value:

```
resOredered_July2019<- res[order(res$pvalue),]
```

We can summarize some basic tallies using the summary function.

```
summary(res)
```

How many adjusted p-values were less than 0.1?

```
sum(res$padj< 0.05, na.rm = TRUE)
```

For a particular gene, a log2 fold change of -1 for condition treated vs untreated means that the treatment induces a multiplicative change in observed gene expression level of $2^{-1}=0.5$ compared to the untreated condition. If the variable of interest is continuous-valued, then the reported log2 fold change is per unit of change of that variable.

Note on p-values set to NA: some values in the results table can be set to NA for one of the following reasons:

If within a row, all samples have zero counts, the baseMean column will be zero, and the log2 fold change estimates, p value and adjusted p value will all be set to NA. If a row contains a sample with an extreme count outlier then the p value and adjusted p value will be set to NA. These outlier counts are detected by Cook's distance. Customization of this outlier filtering and description of functionality for replacement of outlier counts and refitting is described below If a row is filtered by

automatic independent filtering, for having a low mean normalized count, then only the adjusted p value will be set to NA. Description and customization of independent filtering is described below

By default the argument alpha is set to 0.1. If the adjusted p value cutoff will be a value other than 0.1, alpha should be set to that value:

```
resall <- results(dds, alpha = 0.05)
summary(resall)
write.csv(resall, "resSig_D_A_July2019_all.csv")
```

filter for significant genes, according to some chosen threshold for the false dicovery rate (FDR),

```
resOredered_July2019<- res[order(res$padj),]

resSig_D_A_05_July2019 = subset(resOredered_July2019, padj < 0.05)
resSig_D_A_05_July2019
write.csv(resSig_D_A_05_July2019, "resSig_D_A_05_July2019.csv")

#####heatmap for 0.05 DEG
DEgenes_D_AJUly2019_0.05 <- rownames(resOredered_July2019
[resOredered_July2019$padj<0.05 & !is.na(resOredered_July2019$padj),
])#[1:200]
DEgenes_D_AJUly2019_0.05
DEgenes_D_AJUly2019_0.05 <- assay(rld)[DEgenes_D_AJUly2019_0.05, ]
DEgenes_D_AJUly2019_0.05
DEgenes_D_AJUly2019_0.05 <- DEgenes_D_AJUly2019_0.05 -
rowMeans(DEgenes_D_AJUly2019_0.05)
DEgenes_D_AJUly2019_0.05
write.csv(DEgenes_D_AJUly2019_0.05, "DEgenes_D_AJUly2019_0.05.csv")


#pdf("heatmap_DEgenes_D_AJUly2019_0.05.pdf", width=5, height=25)
jpeg("heatmap_July2019_600_68", width = 12, height = 45, units = 'in',
res = 600)
df <- as.data.frame(colData(rld)[,c("Condition","Sex")])

pheatmap(DEgenes_D_AJUly2019_0.05, annotation_col=df,
         show_rownames=T,
         cluster_cols=T,
         cluster_rows=T,
         scale="row",
         clustering_distance_rows="euclidean",
         clustering_distance_cols="euclidean",
         clustering_method="complete",
         border_color=FALSE,
         cex=0.4)
dev.off()
```

Exploring and exporting results MA-plot In DESeq2, the function plotMA shows the log2 fold changes attributable to a given variable over the mean of normalized counts for all the samples in the DESeqDataSet. Points will be colored red if the adjusted p value is less than 0.1. Points which fall out of the window are plotted as open triangles pointing either up or down.

```
plotMA(res, ylim= c(-2,2))
```

Plot counts It can also be useful to examine the counts of reads for a single gene across the groups. A simple function for making this plot is plotCounts, which normalizes counts by sequencing depth and adds a pseudocount of 1/2 to allow for log scale plotting. The counts are grouped by the variables in intgroup, where more than one variable can be specified. Here we specify the gene which had the smallest p value from the results table created above. You can select the gene to plot by rowname or by numeric index.

```
plotCounts(dds, gene=which.min(res$padj), intgroup="Condition")
```

More information on results columns Information about which variables and tests were used can be found by calling the function mcols on the results object.

```
mcols(res)$description
```

For customized plotting, an argument returnData specifies that the function should only return a data.frame for plotting with ggplot.

```
d <- plotCounts(dds, gene=which.min(res$padj), intgroup="Condition",
                returnData=TRUE)
library("ggplot2")
ggplot(d, aes(x=Condition, y=count)) +
  geom_point(position=position_jitter(w=0.1,h=0)) +
  scale_y_log10(breaks=c(25,100,400))

par(mar=c(8,5,2,2))
boxplot(log10(assays(dds)[["cooks"]]), range=0, las=2)
```