# Tutorial: Differential Gene Expression Analysis

*by*
*Roseric Azondekon, PhD*
*University of Wisconsin Milwaukee*

April 5, 2019

## Background

In a previous tutorial, we showed you how to download and process RNA-seq FASTQ files for read alignment on a reference sequence, and for read quantification. In this tutorial, we will show you how to conduct Differential Gene Expression (DGE) analysis using the `DESeq2`, `edgeR`, and `limma-voom` package.

We set our working directory to the `tuto` folder created in our first tutorial.

```
In [ ]: setwd('./tuto')
```

Now, let's install all the required packages for this tutorial.

```
In [ ]: # Indicate package repositories to R...
        repositories <- c("https://cloud.r-project.org",
                          "https://bioconductor.org/packages/3.7/bioc",
                          "https://bioconductor.org/packages/3.7/data/annotation",
                          "https://bioconductor.org/packages/3.7/data/experiment",
                          "https://www.stats.ox.ac.uk/pub/RWin",
                          "http://www.omegahat.net/R",
                          "https://R-Forge.R-project.org",
                          "https://www.rforge.net",
                          "https://cloud.r-project.org",
                          "http://www.bioconductor.org",
                          "http://www.stats.ox.ac.uk/pub/RWin")

        # Package list to download
        packages <- c("vsn", "UpSetR", "gplots", "NMF", "org.Hs.eg.db",
                      "pheatmap", "tximport", "readr","edgeR", "biomaRt",
                      "VennDiagram", "plyr", "dplyr","DESeq2", "AnnotationDbi",
                      "Biobase", "ensembldb", "ggpubr", "ggplot2", "limma", "magrittr")

        # Install and load missing packages
        new.packages <- packages[!(packages %in% installed.packages()[,"Package"])]
```

```
         if(length(new.packages)){
             install.packages(new.packages, repos = repositories)
         }

         lapply(packages, require, character.only = TRUE)
```

Let's load the `samples.txt` file which contains our samples information.

```
In [ ]: samples <- read.table("samples.txt")
        head(samples)
```

We can see that our samples names are contained in the 5th column of the `samples` table. We will use that information as our sample names.

The 9 RNA-seq samples are respectively from 3 new-born, 3 middle-aged, and 3 long-lived individuals. We represent that information in the `sampleTable` variable as follows:

```
In [ ]: conditions <- factor(rep(c("new_born", "middle_aged", "long_lived"), each = 3))
        sampleTable <- data.frame(condition = conditions)
        rownames(sampleTable) <- samples$V5
        sampleTable
```

Now that we have downloaded and loaded all the required packages and samples information, we can import the read counts into R.

## 1. Prepare data for DGE analysis from `STAR` feature counts data

### 1.1. Importing read counts data to `R` from `STAR` output

Remember that the read counts data from STAR were saved in the `featureCounts_results.txt` file inside the `read_counts` folder.

```
In [ ]: # get the table of read counts
        readcounts <- read.table("read_counts/featureCounts_results.txt", header = TRUE)

        # table overview
        head(readcounts)
```

```
In [ ]: # the gene IDs should be stored as row.names
        row.names(readcounts) <- readcounts$Geneid

        # exclude all columns that do not contain read counts
        readcounts <- readcounts[, -c(1:6)]

        head(readcounts)
```

```
In [ ]: # give meaningful sample names
        names(readcounts) <- samples$V5
        head(readcounts)
```

### 1.2. Preparing a DESeqDataSet for use with DESeq2

```
In [ ]: dds_star <- DESeqDataSetFromMatrix(readcounts, sampleTable, ~condition)
```

```
In [ ]: # remove genes without any counts
        dds_star <- dds_star[ rowSums(counts(dds_star)) > 0, ]
```

The `dds_star` object is now ready for the `DESeq()` function. For more, check the see DESeq2 vignette.

```
In [ ]: # Inspect DESeq dataset
        colData(dds_star) %>% head
```

```
In [ ]: # investigate different library sizes
        colSums(counts(dds_star))
```

Our read counts data table is now ready for the downstream DGE analysis. Next let's show how to import transcript-level estimates into R from the quantification files generated by `salmon`.

## 2. Importing transcript-level estimates to `R` from `salmon` output

To import the transcript-level estimates, we need a gene annotation table for *Homo sapiens*. Such a table can be obtained from the `org.Hs.eg.db` R package.

```
In [ ]: # get gene annotation table for Homo sapiens
        tx2g <- select(org.Hs.eg.db,
                       keys = keys(org.Hs.eg.db),
                       columns = c("REFSEQ","SYMBOL","GENENAME"))
```

```
In [ ]: tx2g <- tx2g[, -1]
        colnames(tx2g) <- c("TXNAME", "GENEID", "GENENAME")
```

```
In [ ]: head(tx2g)
```

Let's find the paths to the quantification. Recall that in the previous tutorial, we saved all the quantification files in the `quants` folder.

```
In [ ]: # search for '.sf' file starting from the current working directory
        files <- file.path('.', list.files(pattern = "\\.sf$", recursive = TRUE))
        files
```

We now can import all the quantification files using the `tximport()` function provided by the `tximport` package.

```
In [ ]: txi <- tximport(files, type = "salmon", tx2gene = tx2g,ignoreTxVersion=T)
```

```
In [ ]: # give meaningful sample names
        colnames(txi[['counts']]) <- samples$V5
        colnames(txi[['length']]) <- samples$V5
        colnames(txi[['abundance']]) <- samples$V5
```

```
In [ ]: names(txi)
```

Let's look at the `txi` object data.

```
In [ ]: # Looking at the counts table
        head(txi$counts)
```

### 2.1. Preparing a DESeqDataSet for use with DESeq2

```
In [ ]: dds_salmon <- DESeqDataSetFromTximport(txi, sampleTable, ~condition)
```

```
In [ ]: # remove genes without any counts
        dds_salmon <- dds_salmon[ rowSums(counts(dds_salmon)) > 0, ]
```

The `dds_salmon` object is now ready for the `DESeq()` function. For more, check the see DESeq2 vignette.

```
In [ ]: # Inspect DESeq dataset
        colData(dds_salmon) %>% head
```

```
In [ ]: # investigate different library sizes
        colSums(counts(dds_salmon)) # should be the same as colSums(txi$counts)
```

```
In [ ]: # investigate different library sizes
        colSums(txi$counts) # should be the same as colSums(counts(dds))
```

From this point, we choose to demonstrate the DGE analysis with `dds_star`. To make it easy to switch between `dds_star` and `dds_salmon`, we provide the following statement which you may change as you wish:

```
In [ ]: dds <- dds_star # dds <- dds_salmon
```

## 3. Normalization and Transformation

## 3.1. Normalization for sequencing depth differences

```
In [ ]: # calculate the size factor and add it to the data set
        dds <- estimateSizeFactors(dds)
        sizeFactors(dds)
```

```
In [ ]: # counts() allows you to immediately retrieve the normalized read counts
        counts.sf_normalized <- counts(dds, normalized = TRUE)
```

```
In [ ]: # inspect counts.sf_normalized
        head(counts.sf_normalized)
```

### 3.2. Transformation of sequencing-depth-normalized read counts

```
In [ ]: # transform size-factor normalized read counts
        # to log2 scale using a pseudocount of 1
        log.norm.counts <- log2(counts.sf_normalized + 1)
```

Let's see how the log2 transformation compares to the normalized read counts

```
In [ ]: # first, boxplots of non-transformed read counts (one per sample)
        boxplot(counts.sf_normalized,
                notch = TRUE,
                main = "untransformed  read  counts",
                ylab = "read  counts")
```

```
In [ ]: # box plots of log2-transformed read counts
        boxplot(log.norm.counts,
                notch = TRUE,
                main = "log2 -transformed  read  counts",
                ylab = "log2(read  counts)")
```

### 3.3. Visually exploring normalized read counts

Let's get an impression of how similar read counts are between replicates

```
In [ ]: plot(log.norm.counts[, 1:2], cex = 0.1,
             main = "Normalized log2 (read counts)")
```

Checking for heteroskedasticity...

```
In [ ]: msd_plot <- meanSdPlot(log.norm.counts, ranks = FALSE, plot = FALSE)
        msd_plot$gg + ggtitle("sequencing depth normalized log2(read counts)") +
                      ylab("standard deviation")
```

A clear bump on the left-hand side in the figure will indicate that the variance is higher for smaller read counts compared to the variance for greater read counts.

### 3.4. Transformation of read counts including variance shrinkage

Let's reduce the amount of heteroskedasticity by using the dispersion-mean trend that can be observed for the entire data set as a reference.

```
In [ ]: # obtain regularized log-transformed values
        DESeq.rlog  <- rlog(dds, blind = TRUE)
        rlog.norm.counts  <- assay(DESeq.rlog)
        head(rlog.norm.counts)
```

```
In [ ]: # mean-sd plot for rlog-transformed data
        msd_plot <- meanSdPlot(rlog.norm.counts, ranks = FALSE, plot = FALSE)

        msd_plot$gg + ggtitle("rlog-transformed read counts") +
                      ylab("standard deviation")
```

Let's re-examine how similar the rlog-transformed read counts are between replicates.

```
In [ ]: plot(rlog.norm.counts[, 1:2], cex = 0.1,
             main = "Normalized log2(read counts)")
```

## 3.5. Exploring global read count patterns

An important step before diving into the identification of differentially expressed genes is to check whether expectations about basic global patterns are met. The similarity of expression patterns can be assessed with various methods: - Pairwise correlation - Hierarchical clustering, and - Principal Components Analysis (PCA)

Assessing the similarity of RNA-seq samples in a pair-wise fashion...

```
In [ ]: cor(counts.sf_normalized, method = "pearson")
```

Hierarchical clustering can be used to determine whether the different sample types can be separated in an unsupervised fashion (i.e., samples of different conditions are more dissimilar to each other than replicates within the same condition).

```
In [ ]: # cor() calculates the correlation between columns of a matrix
        distance.m_rlog <- as.dist(1 - cor(rlog.norm.counts, method = "pearson"))

        # plot() can directly interpret the output of hclust()
        plot(hclust(distance.m_rlog),
             labels = colnames(rlog.norm.counts),
             main = "rlog-transformed read counts\ndistance: Pearson correlation")
```

PCA is a complementary approach to determine whether samples display greater variability between experimental conditions than between replicates of the same treatment is principal components analysis.

The goal is to find groups of genes that have certain patterns of expression across different samples, so that the information from thousands of genes is captured and represented by a reduced number of groups.

In base R, the function prcomp() can be used to perform PCA:

```
In [ ]: pc <- prcomp(t(rlog.norm.counts))
        plot(pc$x[, 1], pc$x[, 2],
             col = colData(dds)[, 1],
             main = "PCA of seq.depth normalized\n and rlog-transformed read counts")
```

PCA can also be performed using the R package `DESeq2` which offers a convenience function based on `ggplot2` to do PCA directly on a `DESeqDataSet`:

```
In [ ]: # PCA
        P <- plotPCA(DESeq.rlog)
        # plot cosmetics
        P <- P + theme_bw() + ggtitle("rlog-transformed counts")
        print(P)
```

## 4. Differential Gene Expression Analysis (DGE)

Let's recall that the two basic tasks of all DGE tools are: 1. Estimate the *magnitude* of differential expression between two or more conditions based on read counts from replicated samples, i.e., calculate the fold change of read counts, taking into account the differences in sequencing depth and variability. 2. Estimate the *significance* of the difference and correct for multiple testing.

When it comes to DGE analysis, R offers various tools among which, the best performing are: - `edgeR` (recommended for experiments with fewer than 12 replicates) - `DESeq`/`DESeq2` (better control of false positives and more conservative than `edgeR`) - `limma-voom` (also more conservative than `edgeR`)

All three packages rely on a *negative binomial* model to fit the observed read counts to arrive at the estimate for the difference.

### 4.1. Running DGE analysis with DESeq2:

```
In [ ]: # DESeq uses the levels of the condition
        # to determine the order of the comparison
        str(colData(dds)$condition)
```

```
In [ ]: # set 'new_born' as the first-level factor
        colData(dds)$condition <- relevel(colData(dds)$condition, "new_born")
```

Now, we can run the DGE analysis using the `DESeq()` functiuon provided by the `DESeq2` R package:

```
In [ ]: # Running DGE analysis using the DESeq() function
        dds2 <- DESeq(dds)
```

The `results()` function lets you extract the base means across samples, moderated log2 fold changes, standard errors, test statistics etc. for every gene.

```
In [ ]: DGE.results  <- results(dds2, independentFiltering = TRUE, alpha = 0.05)
        summary(DGE.results)
```

```
In [ ]: # the DESeqResult object can basically be handled like a data.frame
        head(DGE.results)
```

```
In [ ]: # number of differentially expressed genes
        table(DGE.results$padj < 0.05)
```

```
In [ ]: # list all differentially expressed genes
        rownames(subset(DGE.results, padj < 0.05))
```

The `DESeq()` function is a wrapper around the functions `estimateSizeFactors()`, `stimateDispersions()`, and `nbinomWaldTest()`, the DGE analysis can alternatively be performed as follows:

```
In [ ]: # sequencing depth normalization between the samples
        dds3 <- estimateSizeFactors(dds)
```

```
# gene-wise dispersion estimates across all samples
dds3 <- estimateDispersions(dds3)

# this fits a negative binomial GLM and
# applies Wald statistics to each gene
dds3 <- nbinomWaldTest(dds3)
```

#### 4.1.1. Exploratory plots following DGE analysis with DESeq2

A simple and fast way of inspecting how frequently certain values are present in a data set is to plot a histogram of p-values:

```
In [ ]: hist(DGE.results$pvalue,
             col = "grey",
             border = "white",
             xlab = "",
             ylab = "",
             main = "frequencies  of p-values")
```

MA plots provide a general view of the relationship between the expression change between condition

```
In [ ]: plotMA(DGE.results,
               alpha = 0.05,
               main = "new-born vs. middle-aged vs long-lived conditions",
               ylim = c(-4,4))
```

Another way to provide a general view of the relationship between the expression change between condition is to use a volcano plot:

```
In [ ]: # Volcano plot for a threshold of adjusted pval=0.05 and logFC=7
        with(DGE.results,
             plot(log2FoldChange, -log10(padj), pch = 20,
                  main = "Volcano plot", xlim = c(-10,10)))

        with(subset(DGE.results, padj < 0.05),
             points(log2FoldChange,-log10(padj), pch = 20, col = "blue"))

        with(subset(DGE.results, padj < 0.05 & abs(log2FoldChange) > 7),
             points(log2FoldChange, -log10(padj), pch = 20, col = "red"))
```

Heatmaps are a popular means to visualize the expression values across the individual samples.

```
In [ ]: # aheatmap needs a matrix of values, e.g., a matrix of DE genes
        # with the transformed read counts for each replicate
        # sort the results according to the adjusted p-value
        # DGE.results.sorted <- DGE.results[order(DGE.results$padj), ]
        # sort the results according to the log2FoldChange
        DGE.results.sorted <- DGE.results[order(DGE.results$log2FoldChange), ]

        # identify genes with the desired adjusted p-value cut -off
        # DGEgenes <- rownames(subset(DGE.results.sorted , padj < 0.05))

        # identify genes with the desired cut-off
        DGEgenes <- rownames(subset(DGE.results.sorted, abs(log2FoldChange) > 7))
        length(DGEgenes)
```

```
In [ ]: # extract the normalized read counts for DE genes into a matrix
        hm.mat_DGEgenes <- log.norm.counts[DGEgenes, ]
```

```
In [ ]: # scale the read counts per gene to emphasize
        # the sample type-specific differences
        aheatmap(hm.mat_DGEgenes,
                 Rowv = TRUE,
                 Colv = TRUE,
                 distfun = "euclidean",
                 hclustfun = "average",
                 scale = "row")
        # values are transformed into distances from the center
        # of the row-specific average:
        # (actual value - mean of the group)/standard deviation
```

### 4.2. Running DGE analysis with edgeR

```
In [ ]: # We need to specify the sample types, similarly to what we did for DESeq2.
        sample_info.edger <- sampleTable$condition
        sample_info.edger <- relevel(sample_info.edger , ref = "new_born")

        # DGEList() is the function that converts the count matrix into an edgeR object.
        # readcounts <- txi$counts # uncomment this line when data from salmon
        edgeR.DGElist <- DGEList(counts = readcounts, group = sample_info.edger)
        keep <- rowSums( cpm(edgeR.DGElist) >= 1) >= 5
        edgeR.DGElist <- edgeR.DGElist[keep, ]
        edgeR.DGElist <- calcNormFactors(edgeR.DGElist, method = "TMM")
```

```
In [ ]: # check  the  result
        edgeR.DGElist[1:5,]
```

The package `edgeR` recommends removing genes with almost no coverage. In order to determine a sensible cutoff, we plot a histogram of counts per million calculated by `edgeR`'s `cpm()` function.

```
In [ ]: # get an impression of the coverage across samples
        hist(log2(rowSums(cpm(edgeR.DGElist))))
```

```
In [ ]: # specify the design setup - the design matrix looks a bit intimitating,
        # but if you just focus on the formula [~sample_info.edger]
        # you can see that it's exactly what we used for DESeq2, too
        design  <- model.matrix(~sample_info.edger)

        # estimate the dispersion for all read counts across all samples
        edgeR.DGElist  <- estimateDisp(edgeR.DGElist, design)

        # fit the negative binomial model
        edger_fit  <- glmFit(edgeR.DGElist, design)

        # perform the testing for every gene using the neg. binomial model
        edger_lrt  <- glmLRT(edger_fit)
```

```
In [ ]: # extract results from edger_lrt$table
        DGE.results_edgeR <- topTags(edger_lrt, n = Inf, # to  retrieve  all  genes
                                     sort.by = "PValue",
                                     adjust.method = "BH")
```

```
In [ ]: DGE.results_edgeR[1:10,]
```

```
In [ ]: DGE.res_edgeR.sort <- DGE.results_edgeR$table[order(DGE.results_edgeR$table$FDR), ]

        # identify genes with the desired cut-off
        DGEgenes_edgeR <- rownames(subset(DGE.res_edgeR.sort, FDR <= 0.05))
        length(DGEgenes_edgeR)
```

Fit a quasi-likelihood negative binomial generalized log-linear model to count data:

```
In [ ]: fit2 <- glmQLFit(edgeR.DGElist, design)

        # Conduct genewise statistical tests for a given coefficient or contrast.
        qlf2 <- glmQLFTest(fit2,coef=2)
        sm<-topTags(qlf2, n = Inf, # to  retrieve  all  genes
                    sort.by = "PValue",
                    adjust.method = "BH")

        # explore results table
        sm[1:10,]
```

```
In [ ]: DGEgenes_edgeR.QL <- rownames(subset(sm$table, abs(logFC) > 7))
        length(DGEgenes_edgeR.QL)
```

```
In [ ]: hist(sm$table$PValue, col = "grey", border = "white",
             xlab = "", ylab = "", main = "frequencies  of p-values")
```

Another way to provide a general view of the relationship between the expression change between condition is to use a volcano plot:

```
In [ ]: # Volcano plot for a threshold of PValue=0.05 and logFC=7
        with(sm$table, plot(logFC, -log10(PValue), pch = 20,
                            main = "Volcano plot", xlim = c(-10,10)))

        with(subset(sm$table, PValue < 0.05),
             points(logFC,-log10(PValue), pch = 20, col = "blue"))

        with(subset(sm$table, PValue < 0.05 & abs(logFC) > 7),
             points(logFC, -log10(PValue), pch = 20, col = "red"))
```

Let's generate the heatmap of the differentially expressed determined by the quasi-likelihood negative binomial generalized log-linear model.

```
In [ ]: # extract the normalized read counts for DE genes into a matrix
        hm.mat_DGEgenes.edgeR <- log.norm.counts[DGEgenes_edgeR.QL, ]

        # plot the normalized read counts of DE genes sorted by the adjusted p-value
        #aheatmap(hm.mat_DGEgenes.edgeR, Rowv = NA, Colv = NA)
```

```
In [ ]: # scale the read counts per gene to emphasize
        # the sample type-specific differences
        aheatmap(hm.mat_DGEgenes.edgeR,
                 Rowv = TRUE,
                 Colv = TRUE,
                 distfun = "euclidean",
                 hclustfun = "average",
                 scale = "row")
        # values are transformed into distances from the center
        # of the row-specific average:
        # (actual value - mean of the group)/standard deviation
```

## 4.3. Running DGE analysis with `limma-voom`

```
In [ ]: # limma also needs a design matrix, just like edgeR
        design <- model.matrix(~sample_info.edger)

        # transform the count  data to log2-counts -per -million and estimate
        # the mean-variance relationship, which is used to compute weights
        # for each count -- this is supposed to make the read counts
        # amenable to be used with linear models
        design <- model.matrix(~sample_info.edger)
        rownames(design) <- colnames(edgeR.DGElist)
        voomTransformed <- voom(edgeR.DGElist, design, plot=F)
```

```
In [ ]: # fit a linear model for each gene
        voomed.fitted <- lmFit(voomTransformed, design = design)

        # compute moderated t-statistics, moderated F-statistics,
        # and log-odds of differential expression
        voomed.fitted <- eBayes(voomed.fitted)
```

```
In [ ]: # extract gene list with logFC and statistical measures
        colnames(design) # check how the coefficient is named
```

```
In [ ]: DGE.results_limma <- topTable(voomed.fitted,
                                      coef = "sample_info.edgermiddle_aged",
                                      number = Inf,
                                      adjust.method = "BH",
                                      sort.by = "logFC")
```

```
In [ ]: head(DGE.results_limma[DGE.results_limma$logFC>3,])
```

```
In [ ]: DGE.results_lima.sorted <- DGE.results_limma[order(DGE.results_limma$adj.P.Val), ]
```

```
In [ ]: # identify genes with the desired cut-off
        DGEgenes_lima <- rownames(subset(DGE.results_lima.sorted , abs(logFC) > 7))
        length(DGEgenes_lima)
```

```
In [ ]: # extract the normalized read counts for DE genes into a matrix
        hm.mat_DGEgenes.lima <- log.norm.counts[DGEgenes_lima, ]

        # plot the normalized read counts of DE genes sorted by the adjusted p-value
        #aheatmap(hm.mat_DGEgenes.edgeR , Rowv = NA, Colv = NA)
```

```
In [ ]: # scale the read counts per gene to emphasize
        # the sample type-specific differences
        aheatmap(hm.mat_DGEgenes.lima,
                 Rowv = TRUE,
                 Colv = TRUE,
                 distfun = "euclidean",
                 hclustfun = "average",
                 scale = "row")
        # values are transformed into distances from the center
        # of the row -specific  average:
        # (actual value - mean of the group)/standard deviation
```

## 4.4. Venn Diagram and Upset plot

```
In [ ]: # make a Venn diagram
        DE_list <- list(edger = rownames(subset(DGE.results_edgeR$table, abs(logFC) > 7))
                        ,edger_QL = rownames(subset(sm$table, abs(logFC) > 7))
                        ,deseq2 = rownames(subset(DGE.results, abs(log2FoldChange) > 7))
                        ,limma = rownames(subset(DGE.results_limma, abs(logFC) > 7))
                       )

        gplots::venn(DE_list)
```

```
In [ ]: # more sophisticated venn alternative, especially if you
        # are comparing  more than 3 lists
        DE_gns <- UpSetR::fromList(DE_list)
        UpSetR::upset(DE_gns, order.by = "freq")
```

```
In [ ]: # list the Differentially Expressed genes
        # by categories displayed in the Venn Diagram
        out <- gplots::venn(DE_list, show.plot = F)
        out
```

The Venn Diagram and the Upset plot conclude our tutorial on DGE analysis. You may want to check the documentation of each of the packages used in this tutorial.