RNA-seq data analysis workshop

Instructor: **Mengjie Chen**
Course Assistant: **Kate Farris**

Welcome

This tutorial is going to walk you through basic RNA-seq data analysis using R. We will touch upon topics including quality control, differential expression analysis and downstream functional analysis.

Introduction

Transcriptomics is the study of the complete set of transcripts within a cell, which aims to document all species of transcripts, specifically mRNAs (along with ncRNAs), and quantify the gene expression levels during a particular biological process, in a development stage or under a set of unique pathological conditions like cancer. RNA-seq is a sequencing technique which uses next-generation sequencing (NGS) to reveal the presence and quantity of RNA in a biological sample at a given moment, analyzing the continuously changing cellular transcriptome. With maturation of NGS technologies, RNA-seq has become the main assay for transcriptomics studies.

The basic steps of RNA-seq data analysis are: assessing read quality, pre-processing (trimming), alignment to reference genome or assembly, quantification (of expression levels), and downstream functional analysis. In this workshop, we will skip some pre-processing steps and assume reads haven been aligned to the reference genome.

Step 1: Counting reads in genes

We will examine 8 samples from the `airway` package, which are from the paper by Himes et al: "RNA-seq Transcriptome Profiling Identifies CRISPLD2 as a Glucocorticoid Responsive Gene that Modulates Cytokine Function in Airway Smooth Muscle Cells".

To install these packages you can use the code (or if you are compiling the document, remove the `eval=FALSE` from the chunk.)

```
install.packages("devtools")
install.packages("pheatmap")
BiocManager::install(c("airway","Rsamtools","Rsubread","DESeq2","vsn",
                       "org.Hs.eg.db","GenomicFeatures", "clusterProfiler"))
```

This workshop will focus on a summarized version of an RNA-seq experiment: a count matrix, which has genes along the rows and samples along the columns. The values in the matrix are the number of reads which could be uniquely aligned to the exons of a given gene for a given sample. We will demonstrate how to build a count matrix for a subset of reads from an experiment, and then use a pre-made count matrix, to avoid having students download the multi-gigabyte BAM files containing the aligned reads. A new pipeline for building count matrices, which skips the alignment step, is to use fast pseudoaligners such as `Sailfish`, `Salmon` and `kallisto`, followed by the `tximport` package. See the package vignette for more details. Here, we will continue with the read counting pipeline.

First, make variables for the different BAM files and GTF file. Use the `sample.table` to contruct the BAM file vector, so that the count matrix will be in the same order as the `sample.table`.

```
library(airway)
dir <- system.file("extdata", package="airway", mustWork=TRUE)
csv.file <- file.path(dir, "sample_table.csv")
```

```
sample.table <- read.csv(csv.file, row.names=1)
bam.files <- file.path(dir, paste0(sample.table$Run, "_subset.bam"))
gtf.file <- file.path(dir, "Homo_sapiens.GRCh37.75_subset.gtf")
```

Next we create an `Rsamtools` variable which wraps our BAM files, and create a transcript database from the GTF file. We can ignore the warning about `matchCircularity`. Finally, we make a `GRangesList` which contains the exons for each gene.

```
library(Rsamtools)
bam.list <- BamFileList(bam.files)
library(GenomicFeatures)
txdb <- makeTxDbFromGFF(gtf.file, format="gtf")
exons.by.gene <- exonsBy(txdb, by="gene")
```

The following code chunk creates a `SummarizedExperiment` containing the counts for the reads in each BAM file (columns) for each gene in `exons.by.gene` (the rows). We add the `sample.table` as column data. Remember, we know the order is correct, because the `bam.list` was constructed from a column of `sample.table`.

```
library(GenomicAlignments)
se <- summarizeOverlaps(exons.by.gene, bam.list,
                        mode="Union",
                        singleEnd=FALSE,
                        ignore.strand=TRUE,
                        fragments=TRUE)
colData(se) <- DataFrame(sample.table)
```

Exercise: Can you check what is in the object `exons.by.gene`? Can you check what is in the object `se`?

Step 2: Visualizing sample-sample distances

We now load the full `SummarizedExperiment` object, counting reads over all the genes.

```
library(airway)
data(airway)
airway
colData(airway)
rowRanges(airway)
```

The counts matrix is stored in assay of a `SummarizedExperiment`.

```
head(assay(airway))
```

Note that, on the un-transformed scale, the high count genes have high variance. That is, in the following scatter plot, the points start out in a tight cone and then fan out toward the top right. This is a general property of counts generated from sampling processes, that the variance typically increases with the expected value. We will explore different scaling and transformations options below.

Exercise: Can you plot the first two columns of `assay(airway)`? What do you observe? Can you plot samples from two different cell lines? Can you plot samples from two different treatment groups? What do you observe?

```
plot(assay(airway)[,1:2], cex=.1)
```

Step 3: Creating a DESeqDataSet object

We will use the DESeq2 package to normalize the sample for sequencing depth. The DESeqDataSet object is just an extension of the SummarizedExperiment object, with a few changes. The matrix in assay is now accessed with counts and the elements of this matrix are required to be non-negative integers (0,1,2,...).

We need to specify an experimental design here, for later use in differential analysis. The design starts with the tilde symbol ~, which means, model the counts (log2 scale) using the following formula. Following the tilde, the variables are columns of the colData, and the + indicates that for differential expression analysis we want to compare levels of dex while controlling for the cell differences.

```
library(DESeq2)
dds <- DESeqDataSet(airway, design= ~ cell + dex)
head(dds)
```

Step 4: Normalization for sequencing depth

The goal of normalization: to make RNA-seq samples comparable by taking into account differences in sequencing depths of RNA-seq libraries. Typical normalization methods include quantile normalization, median of ratios and trimmed mean of M values. Normalization method in DEseq2 is median of ratios, which counts for sequencing depth and RNA composition.

Size factors in DEseq2 are calculated by the median ratio of samples to a pseudo-sample (the geometric mean of all samples). In other words, for each sample, we take the exponent of the median of the log ratios in this histogram.

```
loggeomeans <- rowMeans(log(counts(dds)))
hist(log(counts(dds)[,1]) - loggeomeans,
     col="grey", main="", xlab="", breaks=40)
```

The size factor for the first sample:

```
exp(median((log(counts(dds)[,1]) - loggeomeans)[is.finite(loggeomeans)]))
```

```
sizeFactors(dds)[1]
```

In DEseq2, a size factor will be estimated for each sample.

```
dds <- estimateSizeFactors(dds)
sizeFactors(dds)
```

```
colSums(counts(dds))
```

```
plot(sizeFactors(dds), colSums(counts(dds)))
abline(lm(colSums(counts(dds)) ~ sizeFactors(dds) + 0))
```

Exercise: Is your calculated exponent of the median of the log ratios the same as output of function sizeFactors? Can you repeat the analysis for sample 5? Based on the above plot, what is the relationship between sizeFactors(dds) and column sum of count matrix? Are they equal? Why?

Make a matrix of log normalized counts (plus a pseudocount):

```
log.norm.counts <- log2(counts(dds, normalized=TRUE) + 1)
```

Another way to make this matrix, and keep the sample and gene information is to use the function normTransform. The same matrix as above is stored in assay(log.norm).

```
log.norm <- normTransform(dds)
```

Examine the log counts and the log normalized counts (plus a pseudocount).

```
rs <- rowSums(counts(dds))
par(mfrow=c(1,2))
boxplot(log2(counts(dds)[rs > 0,] + 1)) # not normalized
boxplot(log.norm.counts[rs > 0,]) # normalized
```

Exercise: How do you think of the normalization results? Are you satisfied? What happened after normalization? Make a scatterplot of log normalized counts against each other. Did you note the fanning out of the points in the lower left corner, for points less than 2^5=32? Would you concern about it? Can you repeat for Sample 3 vs. Sample 5?

Make a scatterplot of log normalized counts against each other.

```
plot(log.norm.counts[,1:2], cex=.1)
```

Step 5: Stabilizing count variance

We will use a sophisticated transformation to address the variance for low counts. It uses the variance model for count data to shrink together the log-transformed counts for genes with very low counts. For genes with medium and high counts, the `rlog` is very close to `log2`.

We use the argument `blind=FALSE` which means that the global dispersion trend should be estimated by considering the experimental design, but the design is not used for applying the transformation itself. See the `DESeq2` vignette for more details.

```
rld <- rlog(dds, blind=FALSE)
plot(assay(rld)[,1:2], cex=.1)
```

Another transformation for stabilizing variance in the DESeq2 package is `varianceStabilizingTransformation`. These two tranformations are similar, the rlog might perform a bit better when the size factors vary widely, and the `varianceStabilizingTransformation` is much faster when there are many samples.

```
vsd <- varianceStabilizingTransformation(dds, blind=FALSE)
plot(assay(vsd)[,1:2], cex=.1)
```

We can examine the standard deviation of rows over the mean for the log plus pseudocount and the rlog. Note that the genes with high variance for the log come from the genes with lowest mean. If these genes were included in a distance calculation, the high variance at the low count range might overwhelm the signal at the higher count range.

```
library(vsn)
meanSdPlot(log.norm.counts, ranks=FALSE)
```

Exercise: Can you compare the scatter plot of sample 1 and 2 before and after variance stablization transformation? Can you make a meanSdPlot for rlog and VST, respectively, and then compare those with untransformed data? Can you use the same ranges for Y axis? Does the transformation fix your previous problem on high variance of low counts?

Hints for changing the Y ranges.

```
  library("ggplot2")
  msd <- meanSdPlot(assay(vsd), ranks=FALSE)
  msd$gg + ggtitle("") + scale_y_continuous(limits = c(0, 1))
```

Next we will introduce two useful analyses for visualization of sample to sample differences/distance. The principal components (PCA) plot is a useful diagnostic for examining relationships between samples. In PCA, the high dimensional data are projected into a lower dimensional space (usually 2D), where the largest variability is retained.

```
plotPCA(log.norm, intgroup="dex")
```

In addition, we can plot a dendrogram based on hierarchical clustering on Euclidean distance matrix.

```
plot(hclust(dist(t(log.norm.counts))), labels=colData(dds)$dex)
```

Exercise: Can you make a PCA plot for rlog? Can you make a PCA plot for VST? How do you interpret PC1 and PC2 before and after variance Can you make a dendrogram for rlog and VST? Based on HC plot, can you comments on the effect of variance stablizing transformation?

Step 6: Differential gene expression

1) Modeling counts using a negative binomial distribution

We will now perform differential gene expression on the counts, to try to find genes in which the differences in expected counts across samples due to the condition of interest rises above the biological and technical variance we observe.

We will use an overdispersed Poisson distribution – called the negative binomial – to model the raw counts in the count matrix. The model will include the size factors into account to adjust for sequencing depth. The formula will look like:

$$K_{ij} \propto \text{NB}(s_{ij} q_{ij}, \alpha_i)$$

where K_{ij} is a single raw count in our count table, s_{ij} is a size factor or more generally a normalization factor, q_{ij} is proportional to gene expression (what we want to model with our design variables), and α_i is a dispersion parameter.

For the negative binomial, the variance parameter is called disperison, and it links the mean value with the expected variance. The reason we see more dispersion than in a Poisson is mostly due to changes in the proportions of genes across biological replicates – which we would expect due to natural differences in gene expression.

```
par(mfrow=c(3,1))
n <- 10000
brks <- 0:400
hist(rpois(n,lambda=100),
     main="Poisson / NB, disp=0",xlab="",breaks=brks,col="black")
hist(rnbinom(n,mu=100,size=1/.01),
     main="NB, disp = 0.01",xlab="",breaks=brks,col="black")
hist(rnbinom(n,mu=100,size=1/.1),
     main="NB, disp = 0.1",xlab="",breaks=brks,col="black")
```

The square root of the dispersion is the coefficient of variation – SD/mean – after subtracting the variance we expect due to Poisson sampling.

```
disp <- 0.5
mu <- 100
v <- mu + disp * mu^2
sqrt(v)/mu
sqrt(v - mu)/mu
sqrt(disp)
```

A number of methods for assessing differential gene expression from RNA-seq counts use the negative binomial distribution to make probabilistic statements about the differences seen in an experiment. A few such methods

are edgeR, DESeq2, and DSS. Other methods, such as limma+voom find other ways to explicitly model the mean of log counts and the observed variance of log counts.

DESeq2 performs a similar step to limma in using the variance of all the genes to improve the variance estimate for each individual gene. In addition, DESeq2 shrinks the unreliable fold changes from genes with low counts, which will be seen in the resulting MA-plot.

2) Experimental design and running DESeq2

Remember, we had created the DESeqDataSet object earlier using the following line of code (or alternatively using DESeqDataSetFromMatrix).

```
dds <- DESeqDataSet(airway, design= ~ cell + dex)
```

First, we setup the design of the experiment, so that differences will be considered across time and protocol variables. We can read and if necessary reset the design using the following code.

```
design(dds)
design(dds) <- ~ cell + dex
```

The last variable in the design is used by default for building results tables (although arguments to results can be used to customize the results table), and we make sure the "control" or "untreated" level is the first level, such that log fold changes will be treated over control, and not control over treated.

```
dds$dex <- relevel(dds$dex, "untrt")
levels(dds$dex)
```

The following line runs the DESeq2 model. After this step, we can build a results table, which by default will compare the levels in the last variable in the design, so the dex treatment in our case:

```
dds <- DESeq(dds)
res <- results(dds)
```

3) Examining results tables

```
head(res)
table(res$padj < 0.1)
```

A summary of the results can be generated:

```
summary(res)
```

For testing at a different threshold, we provide the alpha to results, so that the mean filtering is optimal for our new FDR threshold.

```
res2 <- results(dds, alpha=0.05)
table(res2$padj < 0.05)
```

4) Visualizing results

The MA-plot provides a global view of the differential genes, with the log2 fold change on the y-axis over the mean of normalized counts:

```
plotMA(res, ylim=c(-4,4))
```

We can also test against a different null hypothesis. For example, to test for genes which have fold change more than doubling or less than halving:

```
res.thr <- results(dds, lfcThreshold=1)
plotMA(res.thr, ylim=c(-4,4))
```

A sorted results table:

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

Examine the counts for the top gene, sorting by p-value:

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

A more sophisticated plot of counts:

```
library(ggplot2)
data <- plotCounts(dds, gene=which.min(res$padj), intgroup=c("dex","cell"), returnData=TRUE)
ggplot(data, aes(x=dex, y=count, col=cell)) +
  geom_point(position=position_jitter(width=.1,height=0)) +
  scale_y_log10()
```

Connecting by lines shows the differences which are actually being tested by results given that our design includes cell + dex

```
ggplot(data, aes(x=dex, y=count, col=cell, group=cell)) +
  geom_point() + geom_line() + scale_y_log10()
```

A heatmap of the top genes:

```
library(pheatmap)
topgenes <- head(rownames(resSort),20)
mat <- assay(rld)[topgenes,]
mat <- mat - rowMeans(mat)
df <- as.data.frame(colData(dds)[,c("dex","cell")])
pheatmap(mat, annotation_col=df)
```

Exercise: Can you plot histogram for p-values? With what you have learned from previous lectures, is the histrogram with a desired shape? Can you make a heatmap based on top 50 genes? What do you observe?

5) Getting alternate annotations

We can then check the annotation of these highly significant genes:

```
library(org.Hs.eg.db)
keytypes(org.Hs.eg.db)
anno <- select(org.Hs.eg.db, keys=topgenes,
               columns=c("SYMBOL","GENENAME"),
               keytype="ENSEMBL")
anno[match(topgenes, anno$ENSEMBL),]
```

Exercise: Based on the annotation, what is biological functions of top genes? Can you run a GO enrichment analysis using top 100 genes? You can use the following package or this website (http://geneontology.org/page/go-enrichment-analysis).

```
library(clusterProfiler)
top100genes <- head(rownames(resSort), 100)
annoList <- anno[match(top100genes, anno$ENSEMBL),]
ego <- enrichGO(gene          = annoList$ENSEMBL,
                universe      = rownames(resSort),
```

```
                OrgDb         = org.Hs.eg.db,
                keyType       = 'ENSEMBL',
                ont           = "CC",
                pAdjustMethod = "BH",
                pvalueCutoff  = 0.01,
                qvalueCutoff  = 0.05,
                readable      = TRUE)
head(ego)
```

6) Looking up different results tables

The `contrast` argument allows users to specify what results table should be built. See the help and examples in `?results` for more details:

```
results(dds, contrast=c("cell","N061011","N080611"))
```

Exercise: Would you check the biological functions of top significantly differential expressed genes between cell line N061011 and N080611? Would you perform GO enrichment analysis on top genes as well? Any significant findings? Do you want to make any conclusion? Why or why not?

Hints:

```
res_contrast <- results(dds, contrast=c("cell","N061011","N61311"))
res_contrastSort <- res_contrast[order(res_contrast$padj),]
top100genes <- head(rownames(res_contrastSort), 100)
annoList <- anno[match(top100genes, anno$ENSEMBL),]
ego <- enrichGO(gene          = annoList$ENSEMBL,
                OrgDb         = org.Hs.eg.db,
                universe      = rownames(res_contrastSort),
                keyType       = 'ENSEMBL',
                ont           = "CC",
                pAdjustMethod = "BH",
                pvalueCutoff  = 0.01,
                qvalueCutoff  = 0.05,
                readable      = TRUE)
head(ego)
```

Follow-up activities

`DESeq2` can be used to analyze time course experiments, for example to find those genes that react in a condition-specific manner over time, compared to a set of baseline samples. Here we demonstrate a basic time course analysis with the fission data package, which contains gene counts for an RNA-seq time course of fission yeast (Leong et al. 2014). The yeast were exposed to oxidative stress, and half of the samples contained a deletion of the gene `atf21`. We use a design formula that models the strain difference at time 0, the difference over time, and any strain-specific differences over time (the interaction term strain:minute).

```
library("fission")
data("fission")
ddsTC <- DESeqDataSet(fission, ~ strain + minute + strain:minute)
```

The following chunk of code performs a likelihood ratio test, where we remove the strain-specific differences over time. Genes with small p values from this test are those which at one or more time points after time 0 showed a strain-specific effect.

```
ddsTC <- DESeq(ddsTC, test="LRT", reduced = ~ strain + minute)
```

Exercise: Can you visualize results from the above analysis? Can you perform functional analysis? Can you propose other tests on this dataset?

Acknowledgement

This workshop contains online materials written by Rafael Irizarry and Michael Love.

Session information

Here is the session information.

```
devtools::session_info()
```