Worksheet 6.2/7.1: Differential gene expression analysis with DESeq2

Author: Jacob Stanley (jacob.stanley@colorado.edu)

Resources:

- DESeq2 documentation
- DESeq2 tutorial

*The purpose of DESeq2 is to identify which genomic loci demonstrate a statistically significant difference in expression level between two or more conditions (referred to as "gene differential expression analysis"). It does so by modeling the variance in expression level across the full range of baseline expression levels present in the data, and determines if the differential expression level for each loci is significantly greater than this variance. DESeq2 takes as an input the **unnormalized** count values for each (non-overlapping) loci in each sample. We recommend that you use featureCounts() to compute count values. DESeq2 performs best when provided multiple replicates per experimental condition (preferably 5+ replicates), in order to get an accurate estimation of within condition variance. DESeq2 is only to be used for non-overlapping, unique genomic loci. If one's aim is to compute differential expression of transcripts, DESeq2 is not appropriate.*

Note: All commands are executed within the R environment. We will be executing them manually, from the R command line, but they can also be compiled into a single script to be executed together.

Prepare experiment metadata table and count matrix from RData file:

There are many ways of preparing your metadata table. Below we should you two. The important thing is that the table contains (at minimum) the data file names, sample IDs, and experimental condition which you want DESeq2 to evaluate. Be sure you are associated the correct metadata with each file.

1. Load in matrix of count data (generated last time)
```
> load("/scratch/Users/USERNAME/RNA-seq/featureCounts_workspace.RData")
```
2. Define your sample table containing treatment conditions (option 1)
   a. First let's define a variable that contains our conditions, corresponding to each sample
   ```
   > condition <- c("ctrl", "ctrl", "ifn", "ifn")
   ```
   b. Now let's combine our input files, sample IDs, and experimental conditions into a single dataframe to produce our experiment table.
   ```
   > exptable <- data.frame(input, sample_ID, condition)
   ```
3. Define your sample table containing treatment conditions (option 2): Instead of defining the metadata table internally in R, one can instead create a CSV file with that information (with a header that specifies the field names) and load it into R all at once.
   a. Create your CSV metadata file (`bam_table.csv`):
   ```
   filename,samp_ID,condition
   /path/to/my/file/BAM1_rep1.bam,BSA1,ctrl
   /path/to/my/file/BAM1_rep2.bam,BSA2,ctrl
   /path/to/my/file/BAM2_rep1.bam,IFN1,ifn
   /path/to/my/file/BAM2_rep2.bam,IFN2,ifn
   ```
   b. In your R environment, load in the metadata text file:
   ```
   > exptable <- read.csv(paste0(path_to_file, bam_table.csv))
   ```
   (You may need to convert the various columns to "factors" so DESeq2 will identify them)
   ```
   > exptable$condition <- factor(exptable$condition)
   ```

RNA-seq differential expression analysis with DESEq2:

1. Load DESeq2 library into your R session. This contains all the necessary functions for the rest of the analysis:
```
> library("DESeq2")
```

2. Generate your DESeq2 data structure (*DESeqDataSet*). This contains all the count data, sample IDs, and experimental conditions in a format appropriate for running *DESeq()*:
```
> dds <- DESeqDataSetFromMatrix(
+ countData = count_matrix$counts,
+ colData = exptable,
+ design =~ condition
+ )
```

3. Filter out lowly expressed features (optional):
```
> dds <- dds[rowSums(counts(dds)) > 1, ]
```

4. Run deseq() method (estimates dispersion, fits GLM, calculates LFC):
```
> DEdds <- DESeq(dds)
```
This will then specify the steps it is taking to calculated the final log-fold changes.
```
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
-- note: fitType='parametric', but the dispersion trend was not well captured by the
   function: y = a/x + b, and a local regression fit was automatically substituted.
   specify fitType='local' or 'mean' to avoid this message next time.
final dispersion estimates
fitting model and testing
>
```
Here we see a warning that the assumed model for the dispersion relationship did not fit well. Since DESeq2 makes a number of assumptions about the underlying distribution of your count data, checking the dispersion relationship is important to evaluating the reliability and interpretability of your results.

5. Define your alpha value and the experimental conditions you want to compare:
```
> alpha_value = 0.05
> treatment = "nutlin"
> control = "dmso"
```
Define your contrast:
```
> contrast <- c("condition", treatment, control)
```

6. Extract significant results:
```
> res <- results(DEdds, alpha=alpha_value, contrast=contrast)
```

7. Perform log-fold change shrinkage (for visualization):
```
> res_shrink <- lfcShrink(DEdds, contrast=contrast, res=res)
```

Plotting and saving results:

1. Generate plot of the dispersion estimates:
    a. Define your filename:
    ```
    > Dispname <- paste0("Disp_est_", treatment, "_", control)
    ```

   b. Open a pdf in which to save your dispersion estimates plot:

```
> pdf(paste0(outdir, Dispname, ".pdf"))
```

   c. Generate plot, using the DESeq2 *plotDispEsts()* function:

```
> plotDispEsts(DEdds)
```

   Close graphics file:

```
> dev.off()
```

2. Generate MA plot:
   a. Define your filename, plot title, and the LFC limits of your plot

```
> MAname <- paste0("MA_plot_", treatment, "_", control)
> title <- paste0(treatment, " vs ", control)
> limits <- c(-4, 4)
```

   b. Open a pdf in which to save your MA plot:

```
> pdf(paste0(outdir, MAname, ".pdf"))
```

   c. Generate plot, using the DESeq2 *plotMA()* function:

```
> ma <- plotMA(res_shrink, main=title, alpha=alpha_value, ylim=limits)
```

   Close graphics file:

```
> dev.off()
```

3. Save results:
   a. Define filename:

```
> res_file <- paste0("res_", treatment, "_", "control")
```

   b. Sort results by the adjusted p-values:

```
> res_shrink <- res_shrink[order(res_shrink$padj), ]
```

   c. Select only those results that are significant, based on your specified value for alpha:

```
> res_shrink_sig <- subset(res_shrink, padj < alpha_value)
```

   d. Save significant results to .csv file:

```
> write.csv(res_shrink_sig, file=paste0(outdir, res_file, ".csv"))
```