RNA-seq analysis in R

Differential Expression of RNA-seq data

Contents

Load the data

In the previous session we read the results from Salmon into R and created a `txi` object, which we then saved into an "rds" file. We can now load the txi from that file to start the differential expression analysis. We will also need the sample meta data sheet

First load the packages we need.

```
library(DESeq2)
library(tidyverse)
```

Now load the data from the earlier session.

```
txi <- readRDS("RObjects/txi.rds")
sampleinfo <- read_tsv("data/samplesheet_corrected.tsv", col_types="cccc")
```

It is important to be sure that the order of the samples in rows in the sample meta data table matches the order of the columns in the data matrix - `DESeq2` will **not** check this. If the order does not match you will not be running the analyses that you think you are.

```
all(colnames(txi$counts)==sampleinfo$SampleName)
```

```
## [1] TRUE
```

The model formula and design matrices

Now that we are happy that the quality of the data looks good, we can proceed to testing for differentially expressed genes. There are a number of packages to analyse RNA-Seq data. Most people use DESeq2 (Love, Huber, and Anders 2014) or edgeR (Robinson, McCarthy, and Smyth 2010; McCarthy, Chen, and Smyth 2012). There is also the option to use the limma package and transform the counts using its `voom` function .They are all equally valid approaches (Ritchie et al. 2015). There is an informative and honest blog post here by Mike Love, one of the authors of DESeq2, about deciding which to use.

We will use **DESeq2** for the rest of this practical.

Create a DESeqDataSet object with the raw data

Creating the design model formula

First we need to create a design model formula for our analysis. `DESeq2` will use this to generate the model matrix, as we have seen in the linear models lecture.

We have two variables in our experiment: "Status" and "Time Point."

We will fit two models under two assumptions: no interaction and interaction of these two factors, however, to demonstrate the how `DESeq2` is used we will start with a simple model which considers Status but ignores Time Point.

First, create a variable containing the model using standard R 'formula' syntax.

```
simple.model <- as.formula(~ Status)
```

What does this look like as a model matrix?

```
model.matrix(simple.model, data = sampleinfo)
```

```
##    (Intercept) StatusUninfected
## 1            1                0
## 2            1                0
## 3            1                0
## 4            1                0
## 5            1                1
## 6            1                0
## 7            1                1
## 8            1                1
## 9            1                1
## 10           1                1
## 11           1                0
## 12           1                1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$Status
## [1] "contr.treatment"
```

The intercept has been set automatically to the group in the factor that is alphabetically first: `Infected`.

It would be nice if `Uninfected` were the base line/intercept. To get R to use `Uninfected` as the intercept we need to use a `factor`. Let's set factor levels on Status to use `Uninfected` as the intercept.

```
sampleinfo <- mutate(sampleinfo, Status = fct_relevel(Status, "Uninfected"))
model.matrix(simple.model, data = sampleinfo)
```

```
##    (Intercept) StatusInfected
## 1            1               1
## 2            1               1
## 3            1               1
## 4            1               1
## 5            1               0
## 6            1               1
## 7            1               0
## 8            1               0
## 9            1               0
## 10           1               0
## 11           1               1
## 12           1               0
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$Status
## [1] "contr.treatment"
```

Build a DESeq2DataSet

We don't actually need to pass `DESeq2` the model matrix, instead we pass it the design formula and the `sampleinfo` it will build the matrix itself.

```
# create the DESeqDataSet object
ddsObj.raw <- DESeqDataSetFromTximport(txi = txi,
                                       colData = sampleinfo,
                                       design = simple.model)
```

```
## using counts and average transcript lengths from tximport
```

When we summarised the counts to gene level, `tximport` also calculated an average transcript length for each gene for each sample. For a given gene the average transcript length may vary between samples if different samples are using alternative transcripts. `DESeq2` will incorporate this into its "normalisation."

Filter out the unexpressed genes

Just as we did in session 7, we should filter out genes that uninformative.

```
keep <- rowSums(counts(ddsObj.raw)) > 5
ddsObj.filt <- ddsObj.raw[keep,]
```

Differential expression analysis with DESeq2

The `DESeq2` work flow

The main `DESeq2` work flow is carried out in 3 steps:

estimateSizeFactors

First, Calculate the "median ratio" normalisation size factors for each sample and adjust for average transcript length on a per gene per sample basis.

```
ddsObj <- estimateSizeFactors(ddsObj.filt)
```

```
## using 'avgTxLength' from assays(dds), correcting for library size
```

Let's have a look at what that did DESeq2 has calculated a normalizsation factor for each gene for each sample.

```
normalizationFactors(ddsObj.filt)
```

```
## NULL
```

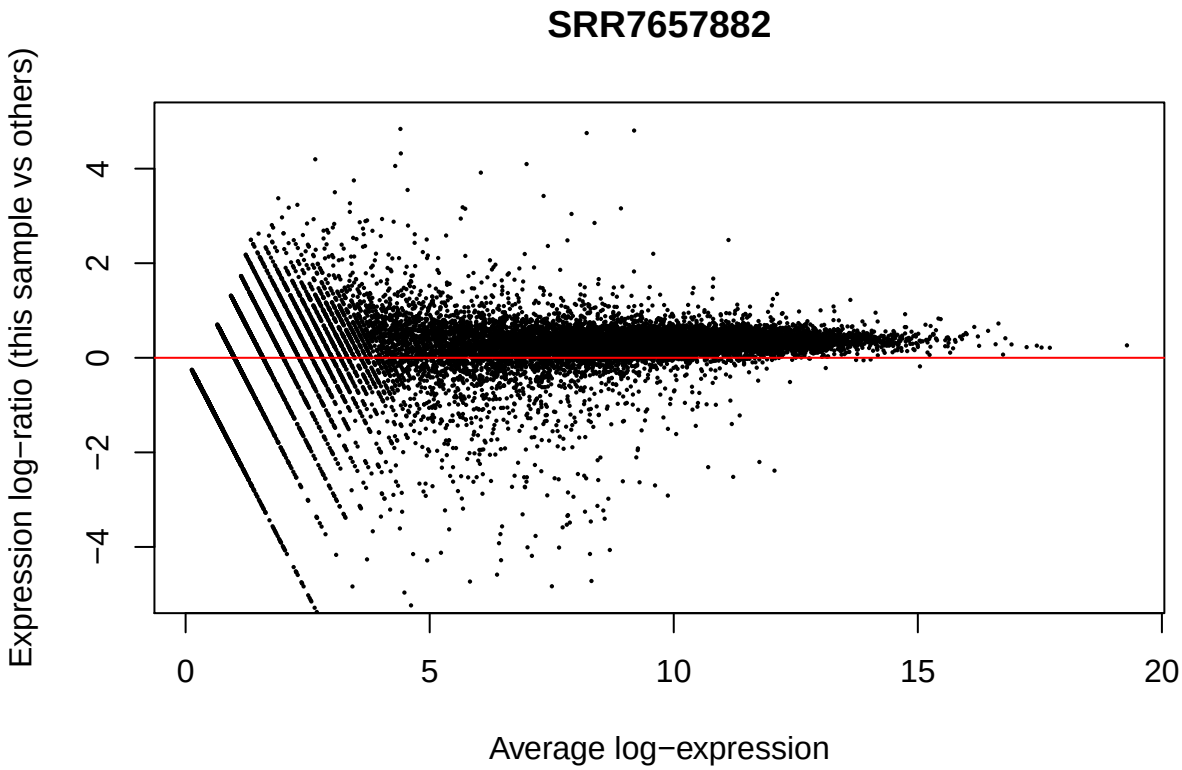
```
normalizationFactors(ddsObj)
```

```
##                     SRR7657878 SRR7657881 SRR7657880 SRR7657874 SRR7657882
## ENSMUSG00000000001  0.9650391 0.96894491 0.94927826  0.9018189 1.20992201
## ENSMUSG00000000028  1.1565743 1.00859533 0.94221210  0.8798304 1.22726399
## ENSMUSG00000000037  0.9372735 1.11991330 0.52624731  1.0355843 0.77648936
##                     SRR7657872 SRR7657877 SRR7657876 SRR7657879 SRR7657883
## ENSMUSG00000000001  0.9615574  1.1078271 1.01950314 0.95990882  0.9028895
## ENSMUSG00000000028  1.0086842  1.2535886 0.95400723 0.77315414  0.7550835
## ENSMUSG00000000037  0.8718128  2.2493164 0.77345594 0.67601042  0.5549084
##                     SRR7657873 SRR7657875
## ENSMUSG00000000001  1.0472683 1.04743598
## ENSMUSG00000000028  1.0609078 1.12777706
## ENSMUSG00000000037  1.8562184 2.13169578
##  [ reached getOption("max.print") -- omitted 20088 rows ]
```

We can use `plotMA` from `limma` to look at the of these normalisation factors on data in an MA plot. Let's look at **SRR7657882**, the fifth column, which has the largest normalisation factors.

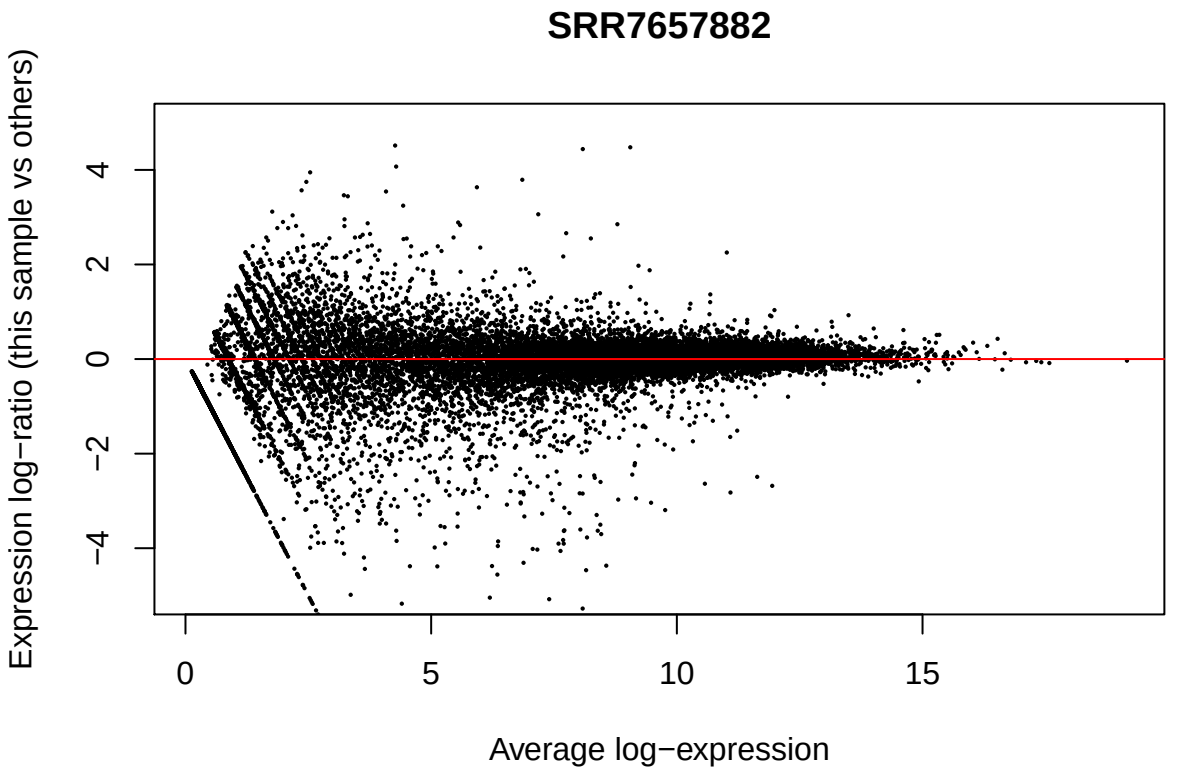
```
logcounts <- log2(counts(ddsObj, normalized=FALSE)  + 1)

limma::plotMA(logcounts, array = 5, ylim =c(-5, 5))
abline(h=0, col="red")
```




```
logNormalizedCounts <- log2(counts(ddsObj, normalized=TRUE)  + 1)

limma::plotMA(logNormalizedCounts, array=5, ylim =c(-5, 5))
abline(h=0, col="red")
```



DESeq2 doesn't actually normalise the counts, it uses raw counts and includes the normalisation factors in the modeling as an "offset." Please see the DESeq2 documentation if you'd like more details on exactly how they are incorporated into the algorithm. For practical purposes we can think of it as a normalisation.

estimateDispersions

Next we need to estimate the dispersion parameters for each gene.

```
ddsObj <- estimateDispersions(ddsObj)
```

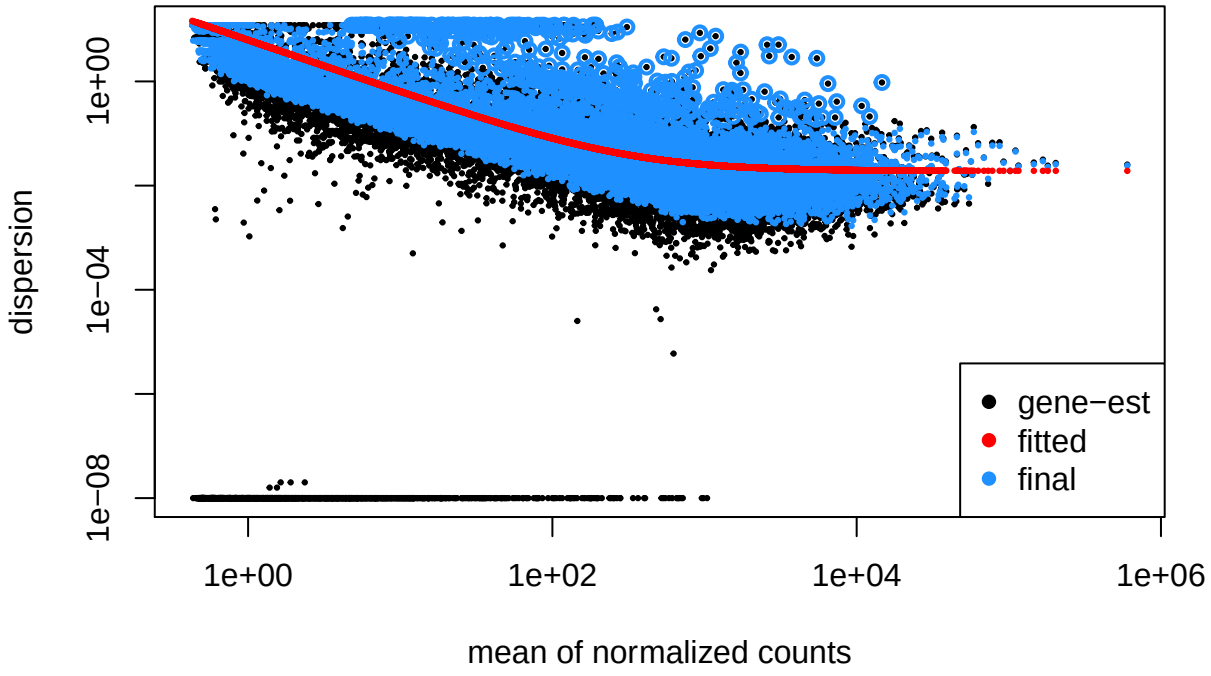
```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

We can plot all three sets of dispersion estimates. It is particularly important to do this if you change any of the default parameters for this step.

```
plotDispEsts(ddsObj)
```



nbinomWaldTest

Finally, apply Negative Binomial GLM fitting and calculate Wald statistics.

```
ddsObj <- nbinomWaldTest(ddsObj)
```

The DESeq command

In practice the 3 steps above can be performed in a single step using the `DESeq` wrapper function. Performing the three steps separately is useful if you wish to alter the default parameters of one or more steps, otherwise the `DESeq` function is fine.

```
ddsObj <- DESeq(ddsObj.filt)
```

```
## estimating size factors
## using 'avgTxLength' from assays(dds), correcting for library size
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

Generate a results table

We can generate a table of differential expression results from the DDS object using the `results` function of DESeq2.

```
results.simple <- results(ddsObj, alpha=0.05)
results.simple
```

```
## log2 fold change (MLE): Status Infected vs Uninfected
## Wald test p-value: Status Infected vs Uninfected
## DataFrame with 20091 rows and 6 columns
##                      baseMean log2FoldChange     lfcSE       stat      pvalue
##                     <numeric>      <numeric> <numeric>  <numeric>   <numeric>
## ENSMUSG00000000001 1102.56094    -0.00802952  0.102877  -0.078050    0.937788
## ENSMUSG00000000028   58.60055     0.30498077  0.254312   1.199239    0.230435
## ENSMUSG00000000037   49.23586    -0.05272685  0.416862  -0.126485    0.899348
## ENSMUSG00000000049    7.98789     0.38165132  0.644869   0.591827    0.553966
## ENSMUSG00000000056 1981.00402    -0.16921845  0.128542  -1.316449    0.188024
##                          padj
##                     <numeric>
## ENSMUSG00000000001    0.975584
## ENSMUSG00000000028    0.480598
## ENSMUSG00000000037    0.961314
## ENSMUSG00000000049    0.772123
## ENSMUSG00000000056    0.426492
##  [ reached getOption("max.print") -- omitted 6 rows ]
```

How many genes have an adjusted p-value of less than 0.05?

```
sum(results.simple$padj < 0.05)
```

```
## [1] NA
```

Independent filtering

You will notice that some of the adjusted p-values (`padj`) are NA.

```
sum(is.na(results.simple$padj))
```

```
## [1] 1584
```

Remember in Session 7 we said that there is no need to pre-filter the genes as DESeq2 will do this through a process it calls 'independent filtering.' The genes with `NA` are the ones `DESeq2` has filtered out.

From `DESeq2` manual: "The results function of the `DESeq2` package performs independent filtering by default using the mean of normalized counts as a filter statistic. A threshold on the filter statistic is found which optimizes the number of adjusted p values lower than a [specified] significance level."

The default significance level for independent filtering is `0.1`, however, you should set this to the FDR cut off you are planning to use. We will use `0.05` - this was the purpose of the `alpha` argument in the previous command.

How many genes have an adjusted p-value of less than 0.05?

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

```
## [1] 2884
```

Exercise 1

So far we have fitted a simple model considering just "Status," but in reality we want to model the effects of both "Status" and "Time Point."

Let's start with the model with only main effects - an additive model with no interaction. The main assumption here is that the effects of Status and the effects of Time Point are indepedent.

Recapitulate the above steps to generate a new DESeq2 object with the additive model. Then we will extract the results table as above.

Load the raw data, remembering to set the factor on the Status so that "Uninfected" will be set as the intercept:

```
txi <- readRDS("RObjects/txi.rds")
sampleinfo <- read_tsv("data/samplesheet_corrected.tsv", col_types="cccc") %>%
                  mutate(Status = fct_relevel(Status, "Uninfected"))
```

```
additive.model <- as.formula(~ TimePoint + Status)
```

Create the model:

```
ddsObj.raw <- DESeqDataSetFromTximport(txi = txi,
                                       colData = sampleinfo,
                                       design = additive.model)
```

Then build the DESeq from the raw data, the sample meta data and the model:

```
keep <- rowSums(counts(ddsObj.raw)) > 5
ddsObj.filt <- ddsObj.raw[keep,]
```

Filter the data set: You are now ready to run the differential gene expression analysis Run the DESeq2 analysis

1. Run the size factor estimation, dispersion estimation and modelling steps using the `DESeq` command as above.
2. Extract the default contrast using the `results` command into a new object called `results.additive`
   a) What contrast are these results for? If you have constructed the model correctly, then it should be the same as previous `results.simple`
   b) How many genes have an adjusted p-value of less than 0.05

The default contrast of `results`

The `results` function has returned the results for the contrast "Infected vs Uninfected." Let's have a look at the model matrix to understand why `DESeq2` has given us this particular contrast.

```
model.matrix(additive.model, data = sampleinfo)
```

```
##    (Intercept) TimePointd33 StatusInfected
## 1            1            0              1
## 2            1            0              1
## 3            1            0              1
## 4            1            1              1
## 5            1            1              0
## 6            1            1              1
## 7            1            0              0
## 8            1            0              0
## 9            1            0              0
## 10           1            1              0
## 11           1            1              1
## 12           1            1              0
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$TimePoint
## [1] "contr.treatment"
##
## attr(,"contrasts")$Status
## [1] "contr.treatment"
```

By default, `results` has returned the contrast encoded by the final column in the model matrix. `DESeq2` has the command `resultsNames` that allows us to view the contrasts that are available directly from the DESeq2 object.

```
resultsNames(ddsObj)
```

```
## [1] "Intercept"                     "TimePoint_d33_vs_d11"
## [3] "Status_Infected_vs_Uninfected"
```

Let's just rename `results.additive` so that we know which contrast results it contains.

```
results.InfectedvUninfected <- results.additive
rm(results.additive)
```

Let's get the top 100 genes by adjusted p-value

```
topGenesIvU <- as.data.frame(results.InfectedvUninfected) %>%
    rownames_to_column("GeneID") %>%
    top_n(100, wt=-padj)
topGenesIvU
```

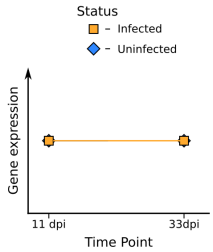
Exercise 2

If we want a different contrast we can just pass the `results` function the **name** of the contrast, as given by `resultsNames(ddsObj)`. Look at the help page for the `results` command to see how to do this.

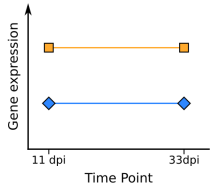
1. Retrieve the results for the contrast of d33 versus d11.
2. How many differentially expressed genes are there at FDR < 0.05?

Should we be using the interaction model?

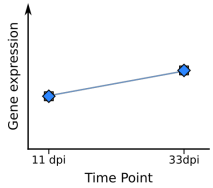
So far we have modeled gene expression as a function of Status and Time Point with an additive model. Realistically, we expect the two factors interact such that differences in gene expression between infected and uninfected mice are not the same at different time point:



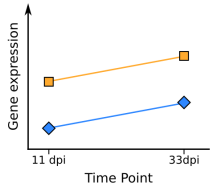
No difference at all.



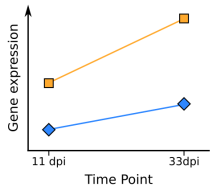
Simple model: gene expression changes with time point, but not status:
~ TimePoint



Simple model: gene expression changes with status, but not time point:
~ Status



Additive model: gene expression changes with status and time point, but the difference in gene expression difference between infected and uninfected is the same irregardless of the time point:
~ TimePoint + Status



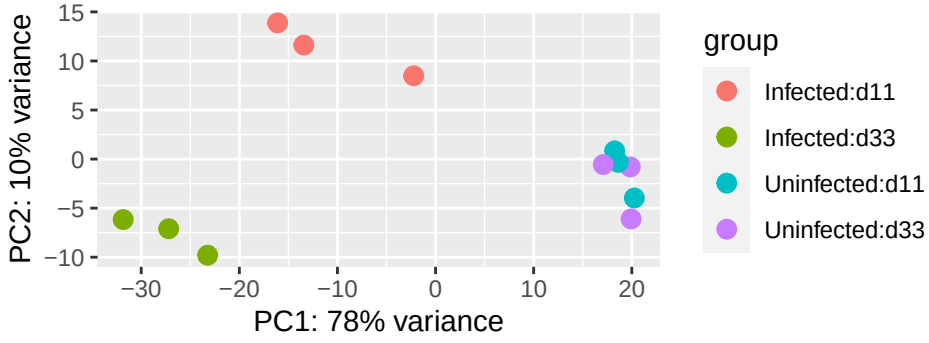
Interaction model: gene expression changes with status and time point, but the difference in gene expression between infected and uninfected changes depending on the time point:
~ TimePoint + Status + TimePoint:Status

pdf version

Let's plot a PCA from `vst` transformed data. Can you anticipate if the interaction term will be important?

```
vstcounts <- vst(ddsObj.raw, blind=TRUE)
```

```
## using 'avgTxLength' from assays(dds), correcting for library size
plotPCA(vstcounts,  intgroup = c("Status", "TimePoint"))
```



In this case we can, from both the PCA and our understanding of the biology, be fairly certain that the interaction model is the appropriate model to use. This is not always the case and so we need a way to compare two models.

Comparing two design models

Let's take a simple example to start with.

Suppose we thought that maybe `TimePoint` was irrelevant and really the only differences might be between `Infected` and `Uninfected` groups. We could fit the simpler model and this would give us more degrees of freedom and therefore more power, but how would we know if it was a better model of not?

We can compare two models by using the "likelihood ratio test" (LRT).

To do so we provide the LRT with a simpler model (one with less parameters) than the one currently being used.

Currently `ddsObj` is using the model `~TimePoint + Status`. Here we want to compare to a model without the `TimePoint` parameter: `~Status`, this was our `simple.model` from earlier.

```
ddsObj.LRT <- DESeq(ddsObj, test = "LRT", reduced = simple.model)
```

```
## using pre-existing normalization factors
## estimating dispersions
## found already estimated dispersions, replacing these
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

```
results.Additive_v_Simple <- results(ddsObj.LRT)
results.Additive_v_Simple
```

```
## log2 fold change (MLE): Status Infected vs Uninfected
## LRT p-value: '~ TimePoint + Status' vs '~ Status'
## DataFrame with 20091 rows and 6 columns
##                      baseMean log2FoldChange     lfcSE      stat    pvalue
##                     <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSMUSG00000000001 1102.56094     -0.0110965  0.106195 0.3226536 0.5700173
## ENSMUSG00000000028   58.60055      0.3007930  0.265626 0.0560662 0.8128250
## ENSMUSG00000000037   49.23586     -0.0481414  0.429685 0.0318005 0.8584663
## ENSMUSG00000000049    7.98789      0.4110498  0.656171 0.4373766 0.5083914
## ENSMUSG00000000056 1981.00402     -0.1907691  0.119694 2.7850130 0.0951499
##                         padj
##                    <numeric>
## ENSMUSG00000000001  0.939329
## ENSMUSG00000000028  0.978399
## ENSMUSG00000000037  0.982281
## ENSMUSG00000000049  0.926027
## ENSMUSG00000000056  0.695076
##  [ reached getOption("max.print") -- omitted 6 rows ]
```

The second line of the results output shows us the test we are doing:

```
LRT p-value: '~ TimePoint + Status' vs '~ Status'
```

The null hypothesis is that there is no significant difference between the two models, i.e. the simpler model is sufficient to explain the variation in gene expression between the samples. If the the adjusted p-value for a gene passes a significance threshold (e.g. padj < 0.05) then we should consider using the more complex model for this gene.

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

```
## [1] 66
```

We can see that for 66 genes the more complex model does fit the data better. Although we have a result for each gene, in practice we should choose one model or the other and apply it to all genes. Curiously then, this suggests that overall the simple model is more appropriate than the additive model. Let's look into the interaction model.

Exercise 3

When we looked at the PCA it did seem that an interaction model might be warranted. Let's test that.

1. Create a new DESeq2 object using a model with an interaction between TimePoint and Status. The model formula should be

   `~TimePoint + Status + TimePoint:Status`

   where `TimePoint:Status` is the parameter for the interaction beteween TimePoint and Status.

Note that `*` can be used as shortcut to add the interaction term, e.g. `~TimePoint * Status`, however, writing out in long form is clearer here. > Remember to filter to remove uninformative genes.

2. Run the statistical analysis using the `DESeq` command and create a new analysis object called `ddsObj.interaction`.
3. Use the LRT to compare this to the simpler additive model (`~TimePoint + Status`)
4. Extract a table of results using `results`. For how many genes is interaction model a better fit?

Extracting specific contrasts from an interactive model

If we are settled on using the interaction model, then we need to extract our contrasts with reference to this. That is, we can no longer ask the general question "What is the difference in gene expression between Infected and Uninfected?" but must rather ask two quesions:

- "What is the difference in gene expression between Infected and Uninfected at 11 days post infection?"
- "What is the difference in gene expression between Infected and Uninfected at 33 days post infection?"

If we view the `resultsNames` for the interaction model, we can see the intercept is Uninfected and 11 days post infection:

```
resultsNames(ddsObj.interaction)
```

```
## [1] "Intercept"                     "TimePoint_d33_vs_d11"
## [3] "Status_Infected_vs_Uninfected" "TimePointd33.StatusInfected"
```

The main effect `Status_Infected_vs_Uninfected` is therefore the difference between Infected and Uninfected **at 11 days post infection**.

```
results.interaction.11 <- results(ddsObj.interaction,
                                  name="Status_Infected_vs_Uninfected",
                                  alpha=0.05)
```

To get the results for Infected versus Uninfected at 33 days post infection, we would need to add the interaction term `TimePointd33.StatusInfected`.

In the help page for `results` it shows us how to do this with a `contrast` in example 3.

```
results.interaction.33 <- results(ddsObj.interaction,
          contrast = list(c("Status_Infected_vs_Uninfected", "TimePointd33.StatusInfected")),
                                  alpha=0.05)
```

Number of genes with padj < 0.05 for Test v Control at day 11:

```
sum(results.interaction.11$padj < 0.05, na.rm = TRUE)
```

```
## [1] 1072
```

Number of genes with padj < 0.05 for Test v Control at day 33:

```
sum(results.interaction.33$padj < 0.05, na.rm = TRUE)
```

```
## [1] 2782
```

We can see that there is a strong difference in the effects of infection on gene expression between days 11 and 33.

Exercise 4

Let's investigate the uninfected mice

1. Extract the results for d33 v d11 for uninfected mice How many genes have an adjusted p-value less than 0.05?
   Is this remarkable? Does this suggest another approach to analysing this data set?

Save the results

Finally save the corrected sample metadata, the DESeq2 dataset, and the two DESeq2 results objects.

```
write_tsv(sampleinfo, "results/samplesheet_corrected.tsv")
saveRDS(ddsObj.interaction, "results/DESeqDataSet.interaction.rds")
saveRDS(results.interaction.11, "results/DESeqResults.interaction_d11.rds")
saveRDS(results.interaction.33, "results/DESeqResults.interaction_d33.rds")
```

References

Love, Michael I., Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for RNA-Seq Data with DESeq2." *Genome Biology* 15: 550. https://doi.org/10.1186/s13059-014-0550-8.

McCarthy, Davis J, Yunshun Chen, and Gordon K Smyth. 2012. "Differential Expression Analysis of Multifactor RNA-Seq Experiments with Respect to Biological Variation." *Nucleic Acids Research* 40 (10): 4288–97. https://doi.org/10.1093/nar/gks042.

Ritchie, Matthew E, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. 2015. "limma Powers Differential Expression Analyses for RNA-Sequencing and Microarray Studies." *Nucleic Acids Research* 43 (7): e47. https://doi.org/10.1093/nar/gkv007.

Robinson, Mark D, Davis J McCarthy, and Gordon K Smyth. 2010. "edgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data." *Bioinformatics* 26 (1): 139–40. https://doi.org/10.1093/bioinformatics/btp616.