# Practical: RNA-seq

Perry Moerland (p.d.moerland@amsterdamumc.nl)

March 12, 2021

## Contents



## 1 Introduction

In the context of a high-throughput RNA sequencing experiment, the final analysis steps generally consist of *read counting* and *differential expression analysis*. Read counting means counting the number of reads per gene (or exon). The result of this counting will typically be organized as a matrix where:

- each row represents a gene (or exon);
- each column represents a sequencing run (usually a given sample);

- and each value is the raw number of reads from the sequencing run that were *assigned* to the gene (or exon).

This count table is the main ingredient for finding genes differentially expressed between the conditions investigated in the experiment.

In this computer lab, you will learn how to generate a count table from aligned reads stored in BAM files and to perform a differential expression analysis. For this purpose you will use several R/Bioconductor packages. You will work with publicly available data from the article by Felix Haglund et al. [1] (`http://www.ncbi.nlm.nih.gov/pubmed/23024189`.) The purpose of the paired-end RNA-seq experiment described in this paper was to investigate the role of the estrogen receptor in parathyroid tumors. The investigators derived primary cultures of parathyroid adenoma cells from 4 patients. These primary cultures were treated with diarylpropionitrile (DPN), an estrogen receptor $\beta$ agonist, with 4-hydroxytamoxifen (OHT), a selective estrogen receptor modulator, or not treated (control). RNA was extracted at 24 hours and 48 hours from cultures under treatment (DPN, OHT) and control. The blocked design of the experiment allows for statistical analysis of the treatment effects while controlling for patient-to-patient variation. One sample (patient 4, 24 hours, control) was omitted by the authors due to low quality.

**First you will have to install quite some R/Bioconductor packages. Preferably you have already installed these in the CDW or on your own laptop. See the mail that I sent earlier this week for instructions.**

# 2 Creating a count table

The two main objects needed for generating a count table are:

1. genomic ranges of the genes (or exons) based on a *gene model*;
2. aligned reads as stored in BAM files.

## 2.1 Choosing and loading a gene model

To generate a count table one needs access to the genomic ranges of the genes (or exons). This information can be extracted from what it is called a *gene model*. Gene models for various organisms are provided by many annotation providers on the internet (UCSC, Ensembl, NCBI, TAIR, FlyBase, WormBase, etc.) In *Bioconductor* a gene model is typically represented as a `TxDb` object. The GenomicFeatures package contains tools for obtaining a gene model from these providers and store it in a `TxDb` object (the container for gene models). For convenience, the most commonly used gene models are available as *Bioconductor* data packages (called TxDb packages). Each TxDb package contains a `TxDb` object ready to use.

According to the vignette located in the parathyroidSE package that contains the data of Haglund et al. [1], the reads in the BAM files were aligned to the GRCh37 human reference genome. One must therefore be careful to choose a gene model based on the same reference genome as the one that was used to align the reads. If you want to use the gene model for Human provided by Ensembl, the latest release based on GRCh37 (release 75) is available via `feb2014.archive.ensembl.org`. Normally you would type the following R code, but since ADICT probably blocks access to dangerous websites built by Russian hackers such as Ensembl you should skip this:

```
## Requires internet access and takes several minutes.
library(GenomicFeatures)
txdb <- makeTxDbFromBiomart(biomart="ENSEMBL_MART_ENSEMBL",
                            dataset="hsapiens_gene_ensembl",host="feb2014.archive.ensembl.org")
## Type ?makeTxDbFromBiomart to get more information on this function
ex_by_gene <- exonsBy(txdb, by="gene")  # GRangesList object
save(ex_by_gene,file="ex_by_gene.RData")
```

This first makes a `TxDb` object containing the Ensembl gene model for Human and then creates an object from it containing the exon ranges grouped by gene using the `exonsBy` function.

You can now load the resulting object:

```
library(GenomicFeatures)
rooturl <- "http://bioinformatics.amc.nl/wp-content/uploads/"
download.file(paste0(rooturl,
               "gs-sequence-analysis/RNASequencing/ex_by_gene.zip"),
              destfile="ex_by_gene.zip")
unzip("ex_by_gene.zip")
load("ex_by_gene.RData")
ex_by_gene

## GRangesList object of length 64102:
## $ENSG00000000003
## GRanges object with 17 ranges and 2 metadata columns:
##        seqnames            ranges strand |   exon_id
##           <Rle>         <IRanges>  <Rle> | <integer>
##    [1]        X 99883667-99884983      - |    667145
##    [2]        X 99885756-99885863      - |    667146
##    [3]        X 99887482-99887565      - |    667147
##    [4]        X 99887538-99887565      - |    667148
##    [5]        X 99888402-99888536      - |    667149
##    ...      ...                 ...    ... .       ...
##   [13]        X 99890555-99890743      - |    667157
##   [14]        X 99891188-99891686      - |    667158
##   [15]        X 99891605-99891803      - |    667159
##   [16]        X 99891790-99892101      - |    667160
##   [17]        X 99894942-99894988      - |    667161
##             exon_name
##           <character>
##    [1] ENSE00001459322
##    [2] ENSE00000868868
##    [3] ENSE00000401072
##    [4] ENSE00001849132
##    [5] ENSE00003554016
##    ...             ...
##   [13] ENSE00003662440
##   [14] ENSE00001886883
##   [15] ENSE00001855382
##   [16] ENSE00001863395
##   [17] ENSE00001828996
##   -------
##   seqinfo: 722 sequences (1 circular) from an unspecified genome
##
## ...
## <64101 more elements>
```

For the purpose of this practical, we'll use a subset of the Ensembl genes. This subset is stored in the parathyroidSE package and is based on the GRCh37 human reference genome.

**Question 1**: In this exercise, we have a look at the `exonsByGene` data set included in the parathyroidSE package.

(a) Load the `exonsByGene` data set from the parathyroidSE package. Remember that you load a package

using the `library` function. Typing `help(package=parathyroidSE)` will then give you information on how to load `exonsByGene`. What does the resulting object contain?

(b) How many genes are represented in this object?

**Question 2**: Compare the contents of the first five elements of the `GRangesList` object `ex_by_gene` generated above with those contained in `exonsByGene`. Can you find any differences? If so, how can these be explained?

## 2.2 Aligned reads from BAM files

The parathyroidSE package contains RNA-seq data from the publication of Haglund et al. [1]. The package also includes BAM files for 3 of the 27 sequencing runs. The reads in these files are *paired-end reads* that were aligned using the TopHat aligner. To keep the package to a reasonable size, only a subset of all the aligned reads from the experiment have been placed in these files. More information on how these BAM files were obtained can be found in the vignette located in the parathyroidSE package (type `browseVignettes("parathyroidSE")` in R to get a link to the vignette in your browser).

To get the paths to the BAM files into R, do:

```
bamdir <- system.file("extdata", package="parathyroidSE")
bamfiles <- list.files(bamdir, pattern="bam$", full.names=TRUE)
bamfiles

## [1] "C:/Users/pdmoerland/R/R-3.6.1/library/parathyroidSE/extdata/SRR479052.bam"
## [2] "C:/Users/pdmoerland/R/R-3.6.1/library/parathyroidSE/extdata/SRR479053.bam"
## [3] "C:/Users/pdmoerland/R/R-3.6.1/library/parathyroidSE/extdata/SRR479054.bam"

## Then create a BamFileList that contains a pointer to all three BAM files
library(Rsamtools)
bamfile_list <- BamFileList(bamfiles, index=character())
## We need to use index=character() here because there are no
## BAM index files (.bam.bai extension) associated with our BAM files.
```

Of course, the paths to the BAM files depend on the particular folder in which the parathyroidSE package is installed on your computer and might be different in your case.

## 2.3 Counting the reads

Let us now see how aligned reads stored in BAM files can be counted with the `summarizeOverlaps` function from the GenomicAlignments package. With this function, the criteria used for assigning reads to genes are controlled via 2 arguments: the `mode` and `inter.feature` arguments. In addition to the help page for `summarizeOverlaps`, the "Counting reads with summarizeOverlaps" vignette (located in the GenomicAlignments package and accessible via `browseVignettes("GenomicAlignments")`) is recommended reading if you are planning to use this function.

**Question 3**:

(a) Use `summarizeOverlaps` on `exonsByGene` and `bamfile_list` to count the reads. Check the help page for the details. Note that because the RNA-seq protocol was not strand specific, you need to specify `ignore.strand=TRUE`. Also because the reads are *paired-end*, you need to specify `singleEnd=FALSE`.

(b) What is the default option used by `summarizeOverlaps` to resolve reads that overlap multiple features? What does this mean?

(c) The call to `summarizeOverlaps` returned a `RangedSummarizedExperiment` object containing the matrix of counts together with information about the genes and samples. The information in a `RangedSummarizedExperiment` object can be accessed with so-called accessor functions. For example, to get the read counts, we use the `assay` function. Extract the count table by applying the function `assay` to the `RangedSummarizedExperiment` object. What are its dimensions, i.e. how many rows and columns does the count table have?

**Question 4**: When `summarizeOverlaps` calls the reading function internally on each BAM file, by default it does so without specifying additional options for counting and filtering. If you want, for example, to discard PCR or optical duplicates and secondary alignments you have to define the `param` argument of `summarizeOverlaps`. You can define a suitable value for `param` using the functions `ScanBamParam` and `scanBamFlag`.

(a) Try to count the reads again but discard PCR or optical duplicates as well as secondary alignments. Hint: take a careful look at the possible arguments of `scanBamFlag` (`?scanBamFlag` is your friend).

(b) What is a secondary alignment?

### 2.4 Precomputed read counts in the parathyroidSE package

The count tables you generated above were based on a small portion of the data only. The parathyroidSE package also contains the `parathyroidGenesSE` data set which contains the counts of reads for all genes and all sequencing runs.

**Question 5**: In this exercise, we load the `parathyroidGenesSE` data set from the parathyroidSE package and perform some basic manipulations on it.

(a) Load the `parathyroidGenesSE` data set from the parathyroidSE package.

(b) Like above, extract the count table by applying the function `assay` to the `RangedSummarizedExperiment` object. What are the dimensions of the count table? Display the top left corner of the count table (e.g. first 6 rows and columns). Does it have row names or column names? What are the row names?

(c) In this matrix of read counts, each row represents an Ensembl gene, each column a sequencing run, and the values are the raw numbers of reads in each sequencing run that were assigned to the respective gene. How many reads were assigned to each of the samples? How many genes have non-zero counts?

(d) Use `colData` on `parathyroidGenesSE`. What do you get? How many rows does it have? Use the function `table` to summarize the number of runs for each treatment (Control, DPN, and OHT).

## 3 Differential expression analysis: DESeq2

### 3.1 Collapsing technical replicates

There are a number of samples which were sequenced in multiple runs. To see this, we extract the 'sample' column via the function `colData`:

```
colData(parathyroidGenesSE)$sample
```

```
##  [1] SRS308865 SRS308866 SRS308867 SRS308868 SRS308869
##  [6] SRS308870 SRS308871 SRS308872 SRS308873 SRS308873
## [11] SRS308874 SRS308875 SRS308875 SRS308876 SRS308877
## [16] SRS308878 SRS308879 SRS308880 SRS308881 SRS308882
```

```
## [21] SRS308883 SRS308884 SRS308885 SRS308885 SRS308886
## [26] SRS308887 SRS308887
## 23 Levels: SRS308865 SRS308866 SRS308867 ... SRS308887
```

For example, sample SRS308873 was sequenced twice. It is recommended to first add together technical replicates (i.e., libraries derived from the same samples), such that we have one column per sample. Otherwise technical replicates would be considered biological replicates, which would lead to underestimating biological variability and incorrectly reduced p-values. This can be easily done using the function `collapseReplicates` from the DESeq2:

```
library(DESeq2)
parathyroidGenesSE_new <-
  collapseReplicates(parathyroidGenesSE,groupby=colData(parathyroidGenesSE)$sample)
# Only keep the column data columns that we actually need for our analysis below
colData(parathyroidGenesSE_new) <-
  colData(parathyroidGenesSE_new)[ , c( "patient", "treatment", "time" ) ]

parathyroidGenesSE_new

## class: RangedSummarizedExperiment
## dim: 63193 23
## metadata(1): MIAME
## assays(1): counts
## rownames(63193): ENSG00000000003 ENSG00000000005 ...
##   LRG_98 LRG_99
## rowData names(0):
## colnames(23): SRS308865 SRS308866 ... SRS308886
##   SRS308887
## colData names(3): patient treatment time
```

## 3.2 Running the DESeq2 pipeline

The package DESeq2 provides methods to test for differential expression by use of negative binomial generalized linear models (see `browseVignettes("DESeq2")` for a nice and detailed vignette of the package). This section demonstrates a typical workflow for such an analysis.

#### On the importance of raw counts

As input, the DESeq2 package expects count data as obtained, e. g., from a RNA-Seq experiment, in the form of a matrix of integer values. The value in the $i$-th row and the $j$-th column of the matrix tells how many reads have been mapped to gene $i$ in sample $j$.

The count values must be *raw* counts of sequencing reads. This is important for DESeq2's statistical model to hold, as only the actual counts allow assessing the measurement precision correctly. Hence, please do *not* supply other quantities, such as (rounded) normalized counts, or counts of covered base pairs – this will only lead to nonsensical results.

#### Preparing the data to be analyzed

In the previous section we prepared a `RangedSummarizedExperiment` object `parathyroidGenesSE_new` that can readily be used in a DESeq2 analysis.

First we load the `parathyroidGenesSE_new` object generated in the previous section:

```
rooturl <- "http://bioinformatics.amc.nl/wp-content/uploads/"
download.file(paste0(rooturl,"gs-sequence-analysis//RNASequencing/parathyroidGenesSE_new.zip"),
                     destfile="parathyroidGenesSE_new.zip")
unzip("parathyroidGenesSE_new.zip")
load("parathyroidGenesSE_new.RData")
```

Then we create a `DESeqDataSet` object. This requires specifying a design formula, which tells which factors in the column metadata table specify the experimental design and how these factors should be used in the analysis. We specify `~ patient + treatment`, which means that we want to test for the effect of treatment (the last factor), controlling for the effect of patient (the first factor). Here, it is sufficient to know that using this design formula, we will be performing something that is similar to a *paired* test exploiting that cells from the same patient received the same treatments (Control, DPN, OHT). You can use R's formula notation to express any experimental design that can be described within an ANOVA-like framework. Specifying a design formula can be quite intricate for complex experimental designs. Detailed examples for commonly used designs can be found in the user's guide of the edgeR package (type `library(edgeR);edgeRUsersGuide()` to open the PDF).

```
library(DESeq2)
ddsFull <- DESeqDataSet(se = parathyroidGenesSE_new, design = ~ patient + treatment)
ddsFull
```

```
## class: DESeqDataSet
## dim: 63193 23
## metadata(2): MIAME version
## assays(1): counts
## rownames(63193): ENSG00000000003 ENSG00000000005 ...
##   LRG_98 LRG_99
## rowData names(0):
## colnames(23): SRS308865 SRS308866 ... SRS308886
##   SRS308887
## colData names(3): patient treatment time
```

**Question 6**: Here we will analyze a subset of the samples, namely those taken after 48 hours, with either control or DPN treatment, taking into account the multifactor design. Select only the relevant columns from the full dataset `ddsFull` and assign it to variable `dds`. Since we selected a subset of the data, it is necessary to "refactor" the factors, since several levels have been dropped. Here, for example the treatment factor still contains the level "OHT", but no sample for this level:

```
dds$treatment
```

```
## [1] Control DPN     Control DPN     Control DPN     Control
## [8] DPN
## Levels: Control DPN OHT
```

```
dds$treatment <- factor(dds$treatment)
dds$treatment
```

```
## [1] Control DPN     Control DPN     Control DPN     Control
## [8] DPN
## Levels: Control DPN
```

```
## Do we have the right samples?
colData(dds)

## DataFrame with 8 rows and 3 columns
##            patient treatment     time
##           <factor>  <factor> <factor>
## SRS308866        1   Control      48h
## SRS308868        1       DPN      48h
## SRS308872        2   Control      48h
## SRS308874        2       DPN      48h
## SRS308878        3   Control      48h
## SRS308880        3       DPN      48h
## SRS308883        4   Control      48h
## SRS308885        4       DPN      48h
```

A call to the function `DESeq` would throw an error if we had omitted this step.

#### DESeq2 analysis

The DESeq2 analysis can now be run with a single call to the function `DESeq`:

```
dds <- DESeq(dds)

## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

This added all kinds of bells and whistles to the `dds` object, which you will investigate in more detail in the following sections.

### 3.3 Inspecting the results table

The results for the last variable in the design formula, in our case the `treatment` variable, can be extracted using the `results` function:

```
res <- results(dds)
res

## log2 fold change (MLE): treatment DPN vs Control
## Wald test p-value: treatment DPN vs Control
## DataFrame with 63193 rows and 6 columns
##                         baseMean      log2FoldChange
##                        <numeric>           <numeric>
## ENSG00000000003  623.422704159628  -0.017931994587485
## ENSG00000000005 0.680023404967342  -0.749857078573604
## ENSG00000000419     299.746242365 -0.0126005895662198
```

```
## ENSG00000000457  183.502856893626 -0.0907333019834504
## ENSG00000000460  200.464220731487   0.372176830780138
## ...                           ...                 ...
## LRG_94                          0                  NA
## LRG_96                          0                  NA
## LRG_97                          0                  NA
## LRG_98                          0                  NA
## LRG_99                          0                  NA
##                             lfcSE               stat
##                         <numeric>          <numeric>
## ENSG00000000003 0.0916336472442749  -0.19569224980953
## ENSG00000000005   2.59558008676392 -0.288897685106107
## ENSG00000000419  0.118923623345575 -0.105955311583506
## ENSG00000000457  0.150737013146947 -0.601931138803968
## ENSG00000000460  0.148315172480573   2.50936451446927
## ...                           ...                 ...
## LRG_94                         NA                  NA
## LRG_96                         NA                  NA
## LRG_97                         NA                  NA
## LRG_98                         NA                  NA
## LRG_99                         NA                  NA
##                            pvalue              padj
##                         <numeric>         <numeric>
## ENSG00000000003    0.8448510508845 0.996948189219851
## ENSG00000000005  0.772659675252425                NA
## ENSG00000000419   0.91561780871385 0.996948189219851
## ENSG00000000457  0.547219975705456 0.996948189219851
## ENSG00000000460 0.0120948603279287 0.276105175792322
## ...                           ...                 ...
## LRG_94                         NA                  NA
## LRG_96                         NA                  NA
## LRG_97                         NA                  NA
## LRG_98                         NA                  NA
## LRG_99                         NA                  NA
```

As `res` is a `DataFrame` object, it carries metadata with information on the meaning of the columns:

```
mcols(res)

## DataFrame with 6 rows and 2 columns
##                        type
##                 <character>
## baseMean       intermediate
## log2FoldChange      results
## lfcSE               results
## stat                results
## pvalue              results
## padj                results
##                                                  description
##                                                  <character>
## baseMean              mean of normalized counts for all samples
## log2FoldChange log2 fold change (MLE): treatment DPN vs Control
## lfcSE                  standard error: treatment DPN vs Control
## stat                   Wald statistic: treatment DPN vs Control
```

```
## pvalue              Wald test p-value: treatment DPN vs Control
## padj                                     BH adjusted p-values
```

The first column, `baseMean`, is a just the average of the normalized count values, taken over all samples. The remaining four columns refer to a specific *contrast*, namely the comparison of the levels *DPN* versus *Control* of the factor variable *treatment*. See the help page for `results` (by typing `?results`) for information on how to obtain other contrasts.

The column `log2FoldChange` is the effect size estimate. It tells us how much the gene's expression has changed due to treatment with DPN in comparison to control. This value is reported on a logarithmic scale to base 2: for example, a $\log_2$ fold change of 1.5 means that the gene's expression is increased by a factor of $2^{1.5} \approx 2.82$.

Of course, this estimate has an uncertainty associated with it, which is available in the column `lfcSE`, the standard error estimate for the log2 fold change estimate. We can also express the uncertainty of a particular effect size estimate as the result of a statistical test. The purpose of a test for differential expression is to test whether the data provides sufficient evidence to conclude that this value is really different from zero (and that the sign is correct). `DESeq2` performs for each gene a *hypothesis test* to see whether evidence is sufficient to decide against the *null hypothesis* that there is no effect of the treatment on the gene and that the observed difference between treatment and control was merely caused by experimental variability (i. e., the type of variability that you can just as well expect between different samples in the same treatment group). As usual in statistics, the result of this test is reported as a *p-value*, and it is found in the column `pvalue`. (Remember that a p-value indicates the probability that a fold change as strong as the observed one, or even stronger, would be seen under the situation described by the null hypothesis.)

Finally, we note that a subset of the p-values in `res` are `NA` ("not available"). This is `DESeq`'s way of reporting that all counts for this gene were zero, and hence no test was applied.

**Question 7**: The function `DESeq` takes the different library sizes into account in its statistical model. The estimated size factors estimated for this purpose have been included in the column metadata of the `dds` object.

```
colData(dds)$sizeFactor
```

```
## SRS308866 SRS308868 SRS308872 SRS308874 SRS308878 SRS308880
## 1.0765936 0.9932643 0.6706964 0.8010918 0.8652826 1.7399298
## SRS308883 SRS308885
## 0.8295786 1.4777134
```

(a) The function `DESeq` uses a robust method to estimate the size factors. Calculate the factors that you would obtain if you would normalize each sequencing run with respect to the average number of counts.

(b) Compare the values for both methods. Do you expect that the choice of normalization method will have a large effect on the down-stream analysis in this case?

(c) Check to see if the `baseMean` is indeed the mean of raw counts or the mean of normalized counts (hint: use the `counts` function)?

## 3.4 Multiple testing

Novices in high-throughput biology often assume that thresholding p-values at 0.05, as is often done in other settings, would be appropriate – but it is not. We briefly explain why:

There are 906 genes with a p-value below 0.05 among the 30434 genes, for which the test succeeded in reporting a p-value:

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

## [1] 906

table( is.na(res$pvalue) )

##
## FALSE  TRUE
## 30434 32759
```

Now, assume for a moment that the null hypothesis is true for all genes, i.e., no gene is affected by the treatment with DPN. Then, by the definition of *p-value*, we expect up to 5% of the genes to have a p-value below 0.05. This amounts to 1522 genes. If we just considered the list of genes with a p-value below 0.05 as differentially expressed, this list should therefore be expected to contain up to 1522/906 = 168% (let's say 100%) false positives!

DESeq2 uses the so-called Benjamini-Hochberg (BH) adjustment; in brief, this method calculates for each gene an *adjusted p-value* which answers the following question: if one called significant all genes with a p-value less than or equal to this gene's p-value threshold, what would be the fraction of false positives (the *false discovery rate*, FDR) among them (in the sense of the calculation outlined above)? These values, called the BH-adjusted p-values, are given in the column `padj` of the `results` object.

Hence, if we consider a fraction of 10% false positives acceptable, we can consider all genes with an *adjusted* p-value below 10%=0.1 as significant. How many such genes are there?

```
sum( res$padj < 0.1, na.rm=TRUE )

## [1] 236
```

We subset the results table to these genes and then sort it by the log2-fold-change estimate to get the significant genes with the strongest down-regulation

```
resSig <- res[ which(res$padj < 0.1 ), ]
head( resSig[ order( resSig$log2FoldChange ), ] )

## log2 fold change (MLE): treatment DPN vs Control
## Wald test p-value: treatment DPN vs Control
## DataFrame with 6 rows and 6 columns
##                         baseMean     log2FoldChange
##                        <numeric>          <numeric>
## ENSG00000163631 268.836661246947  -0.97188763049082
## ENSG00000145244 173.331658179128 -0.816530762063189
## ENSG00000169239 1547.61140356092  -0.76866722484542
## ENSG00000041982 1493.26304568347  -0.70389211528266
## ENSG00000119946 183.490954769971 -0.699677506586409
## ENSG00000155111 587.892288902484 -0.675326192945983
##                             lfcSE              stat
##                         <numeric>         <numeric>
## ENSG00000163631  0.151829596999077 -6.40117374807183
## ENSG00000145244  0.237157384161872 -3.44299109618221
## ENSG00000169239 0.0911072796563921 -8.43694628732656
## ENSG00000041982 0.0857431291127707 -8.20931219289758
## ENSG00000119946  0.166147415129077 -4.21118502531528
```

```
## ENSG00000155111 0.0987865765718096 -6.83621415360089
##                              pvalue                padj
##                           <numeric>           <numeric>
## ENSG00000163631 1.54186988030729e-10  1.1669582886418e-07
## ENSG00000145244 0.000575318254978076   0.0347273393296275
## ENSG00000169239  3.2573606261897e-17 1.06830570670268e-13
## ENSG00000041982 2.22459126174234e-16 5.47193835607071e-13
## ENSG00000119946 2.54034568821995e-05  0.00320441810594822
## ENSG00000155111 8.13133630488576e-12 8.88935754486344e-09
```

and with the strongest upregulation

```
tail( resSig[ order( resSig$log2FoldChange ), ] )
```

```
## log2 fold change (MLE): treatment DPN vs Control
## Wald test p-value: treatment DPN vs Control
## DataFrame with 6 rows and 6 columns
##                         baseMean    log2FoldChange
##                        <numeric>         <numeric>
## ENSG00000158457 301.551036962356 0.622084934994109
## ENSG00000159307 258.895063901787 0.633776974537171
## ENSG00000156414 136.907007840686   0.7832401157074
## ENSG00000103257 168.152231856987 0.823774509133976
## ENSG00000101255 284.997129109888 0.879295115645584
## ENSG00000092621 594.182981210876 0.918414129671324
##                            lfcSE             stat
##                        <numeric>        <numeric>
## ENSG00000158457 0.161437806114784  3.8534030532588
## ENSG00000159307  0.14753572306804 4.29575265811987
## ENSG00000156414 0.181358524007566 4.31873891780639
## ENSG00000103257 0.171008811943237 4.81714655387122
## ENSG00000101255 0.159616385918632 5.50880231114757
## ENSG00000092621  0.12216463853399 7.51783937391829
##                              pvalue                padj
##                           <numeric>           <numeric>
## ENSG00000158457 0.000116487398849416   0.0105916926874801
## ENSG00000159307 1.74101617734789e-05  0.00231484569850349
## ENSG00000156414 1.56923284505185e-05  0.00211502492636509
## ENSG00000103257 1.45625731150969e-06 0.000333211992742879
## ENSG00000101255  3.6128335874206e-08  1.2695239166654e-05
## ENSG00000092621 5.56888600703128e-14 7.82746706045439e-11
```

**Question 8**: What is the proportion of down- and up-regulation among the genes with adjusted p-value less than 0.1?

Of course it is often useful to visualize the counts of reads for a single gene across the treatments. Here we use the function `plotCounts` and specify the gene which had the smallest p-value in the results table created above (Fig. 1):

```
plotCounts(dds, gene=which.min(res$pvalue), intgroup="treatment")
```

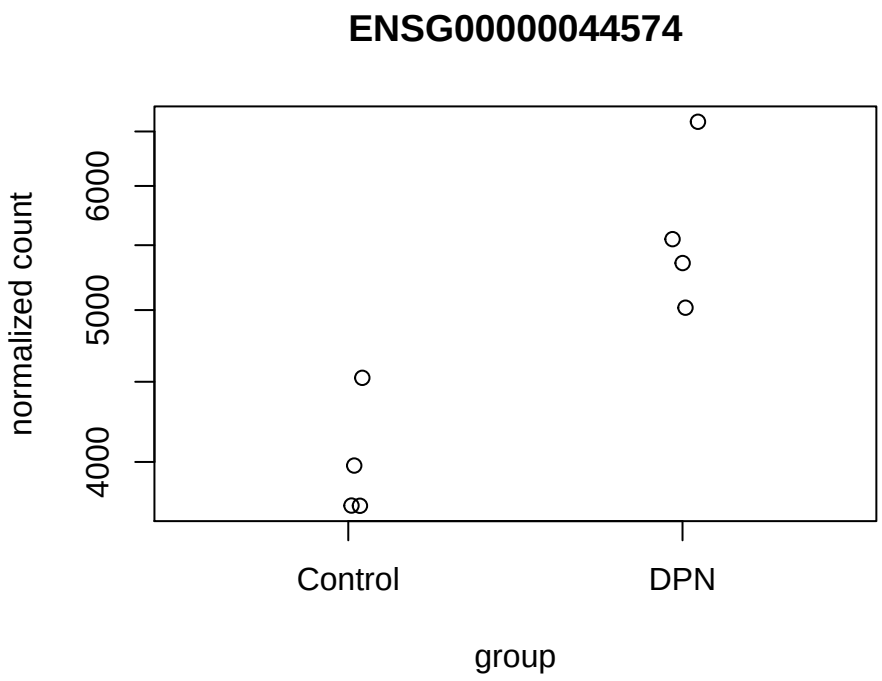


Figure 1: Normalized counts for the gene with the smallest p-value in the comparison of DPN versus control treatment.

### 3.5 Diagnostic plots

A so-called MA-plot provides a useful overview for an experiment with a two-group comparison:

```
plotMA(dds, ylim = c(-0.75, 0.75) )
```

The plot (Fig. 2) represents each gene with a dot. The $x$ axis is the average expression over all samples, the $y$ axis the $\log_2$ fold change between treatment and control. Genes with an adjusted p-value below a threshold (here 0.1, the default) are shown in red.

This plot demonstrates that only genes with an average normalized count above 100 contain sufficient information to yield a significant call, and only above about 1000 counts can smaller fold-changes become significant.

It is actually more useful to visualize the MA-plot for the shrunken log fold changes (LFC). When count values are too low to allow an accurate estimate of the LFCs, the value is "shrunken" towards zero to avoid that these values, which otherwise would frequently be unrealistically large, dominate the top-ranked log fold changes. To shrink the LFCs, we pass the `dds` object to the function `lfcShrink` indicating the comparison of interest:

```
resLFC <- lfcShrink(dds, coef="treatment_DPN_vs_Control")

## using 'normal' for LFC shrinkage, the Normal prior from Love et al (2014).
##
## Note that type='apeglm' and type='ashr' have shown to have less bias than type='normal'.
```

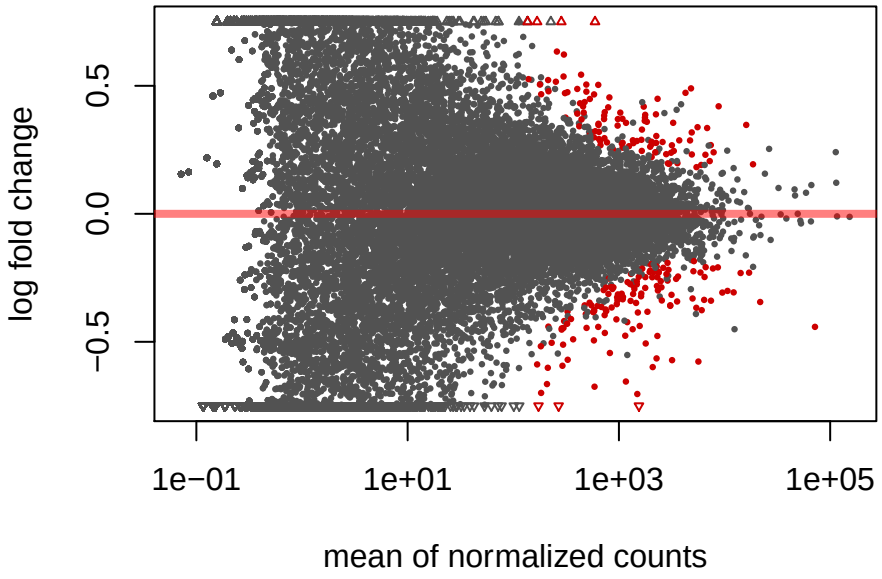


Figure 2: The MA-plot shows the $\log_2$ fold changes from the treatment over the mean of normalized counts, i.e. the average of counts normalized by size factor. Points are colored red if the adjusted p-value is less than 0.1. Points which fall out of the window are plotted as open triangles pointing either up or down.

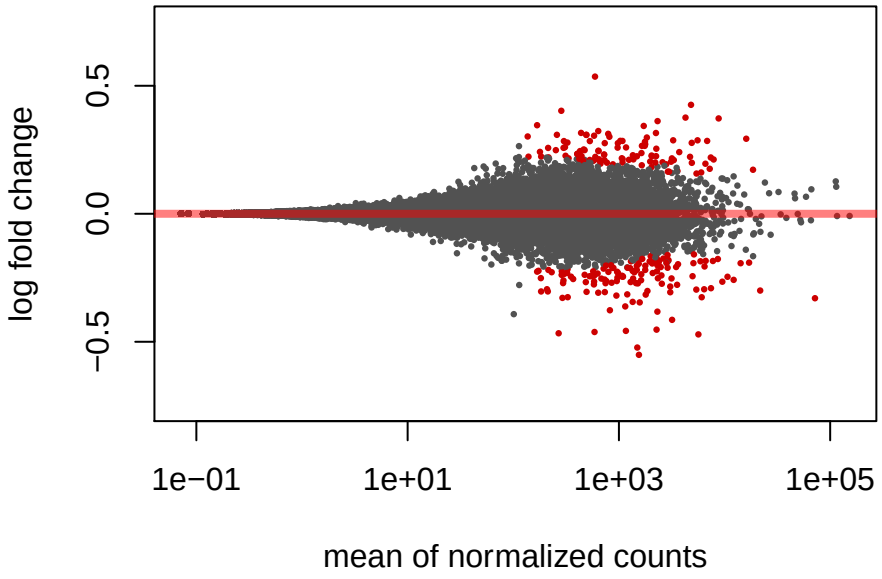


Figure 3: The MA-plot shows the shrunken $\log_2$ fold changes from the treatment over the mean of normalized counts, i.e. the average of counts normalized by size factor. Points are colored red if the adjusted p-value is less than 0.1. Points which fall out of the window are plotted as open triangles pointing either up or down. Shrinkage incorporates a prior on $\log_2$ fold changes, resulting in moderated estimates from genes with low counts and highly variable counts, as can be seen by the narrowing of the vertical spread of points on the left side of the plot.

```
## See ?lfcShrink for more details on shrinkage type, and the DESeq2 vignette.
## Reference:  https://doi.org/10.1093/bioinformatics/bty895
```

The MA-plot for the LFCs (Fig. 3) is then created as follows:

```
plotMA(resLFC, ylim = c(-0.75, 0.75) )
```

Whether a gene is called significant depends not only on its LFC but also on its within-group variability, which DESeq2 quantifies as the *dispersion*. For strongly expressed genes, the dispersion can be understood as a squared coefficient of variation: a dispersion value of 0.01 means that the gene's expression tends to differ by typically $\sqrt{0.01} = 10\%$ between samples of the same treatment group. For weakly expressed genes, the Poisson noise is an additional source of noise, which is added to the dispersion.

The function `plotDispEsts` visualizes DESeq2's dispersion estimates (Fig. 4):

```
plotDispEsts( dds )
```

The black dots are the dispersion estimates for each gene as obtained by considering the information from each gene separately. Unless one has many samples, these values fluctuate strongly around their true values. Therefore, we fit the red trend line, which shows the dispersions' dependence on the mean, and

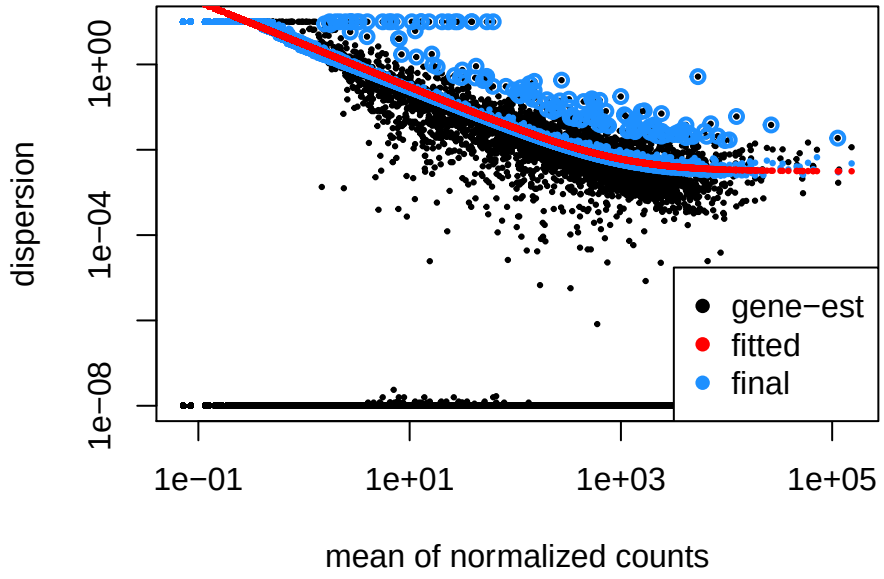


Figure 4: Plot of dispersion estimates. See text for details.

then shrink each gene's estimate towards the red line to obtain the final estimates (blue circles) that are then used in the hypothesis test.

Another useful diagnostic plot is the histogram of the p-values (Fig. 5).

```
hist( res$pvalue, breaks=100 )
```

**Question 9**: Revisit the discussion about p-values and multiple testing in the previous section. Which part of the histogram is caused by genes that are called significant? And which part is caused by those that are truly significant? Why are there "spikes" at intermediate values?

## 3.6 Exploratory analysis

Many common statistical methods for exploratory analysis of multidimensional data, especially methods for clustering and ordination (e. g., principal-component analysis), work best for (at least approximately) homoskedastic data; this means that the variance of an observable (i.e., here, the expression strength of a gene) does not depend on the mean. In RNA-Seq data, however, variance grows with the mean. For example, if one performs PCA directly on a matrix of normalized read counts, the result typically depends only on the few most strongly expressed genes because they show the largest absolute differences between samples. A simple and often used strategy to avoid this is to take the logarithm of the normalized count values; however, now the genes with low counts tend to dominate the results because, due to the strong Poisson noise inherent to small count values, they show the strongest relative differences between samples.

As a solution, DESeq2 offers the *regularized-logarithm transformation*, or *rlog* for short. For genes with high counts, the rlog transformation differs not much from an ordinary $\log_2$ transformation. For genes with lower counts, however, the values are shrunken towards the genes' averages across all samples. Using an

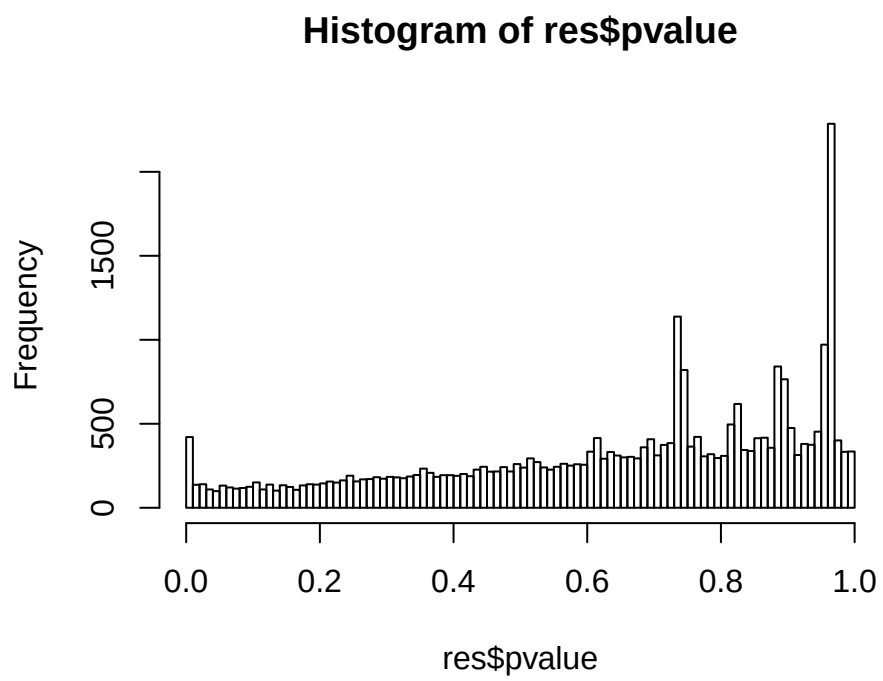


Figure 5: Histogram of the p-values returned by the test for differential expression.

empirical Bayesian prior in the form of a *ridge penality*, this is done such that the rlog-transformed data are approximately homoskedastic.

The function `rlogTransform` returns a `DESeqTransform` object which contains the rlog-transformed values in its `assay` slot:

```
rld <- rlogTransformation(dds)
head( assay(rld) )
```

```
##                  SRS308866  SRS308868  SRS308872  SRS308874
## ENSG00000000003  9.7281273  9.6975828  9.1310119  9.1894125
## ENSG00000000005 -0.6610708 -0.5302481 -0.6338938 -0.7231434
## ENSG00000000419  8.0983673  8.1085864  8.2400300  8.2911769
## ENSG00000000457  7.4439013  7.3018366  7.8173333  7.7094949
## ENSG00000000460  7.5733354  7.6718827  7.9893714  8.1747855
## ENSG00000000938  3.2943821  3.1750959  4.0677929  3.6405037
##                 SRS308878  SRS308880  SRS308883  SRS308885
## ENSG00000000003  8.952581  8.8604213  9.0904959  9.1108783
## ENSG00000000005 -0.724811 -0.7306887 -0.7243747 -0.7295391
## ENSG00000000419  8.302976  8.3084428  8.2650841  8.1707219
## ENSG00000000457  7.220910  7.3417436  7.6014490  7.4952816
## ENSG00000000460  7.133437  7.4527080  7.0154215  7.3733657
## ENSG00000000938  3.096320  3.4243846  3.3973629  3.5014756
```

A popular way to visualize sample-to-sample distances is a principal-components analysis (PCA). In this ordination method, the data points (i.e., here, the samples) are projected onto the 2D plane such that they spread out optimally (Fig. 6).

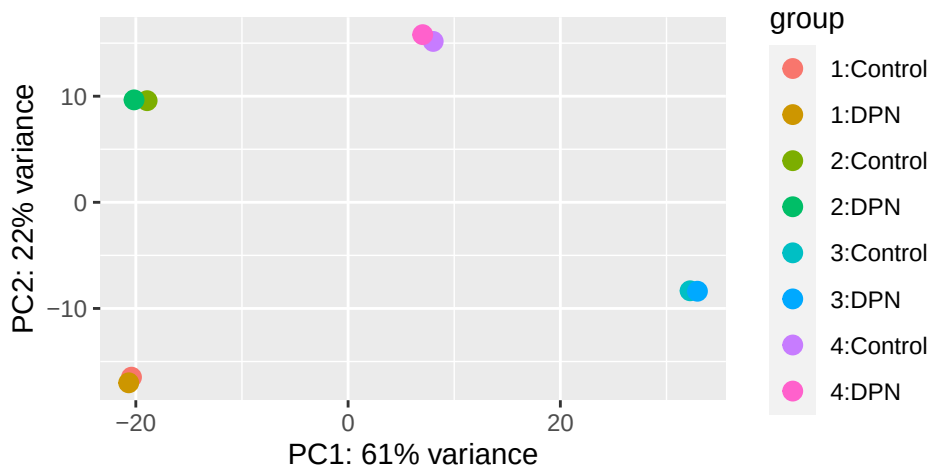


Figure 6: The PCA plot shows that the difference between patients is much greater than the difference between treatments

```
print( plotPCA( rld, intgroup = c( "patient", "treatment") ) )
```

Here, we have used the function `plotPCA` which comes with DESeq2. The two terms specified as `intgroup` are column names from our sample data; they tell the function to use them to choose colours.

From the PCA plot, we see that the difference between patients is much larger than the difference between treatment and control samples of the same patient. This shows why it was important to account for this paired design ("paired", because each treated sample is paired with one control sample from the *same* patient). We did so by using the design formula `~ patient + treatment` when setting up the data object in the beginning. Had we used an un-paired analysis, by specifying only `~ treatment`, we would not have found many hits, because then, the patient-to-patient differences would have drowned out any treatment effects.

**Question 10**: How many genes differentially expressed with an adjusted p-value below 0.1 would you have found with a design that ignores the pairing of the samples?

Here, we have performed the exploratory data analysis towards the end of our analysis. In practice, however, this is a step suitable to give a first overview on the data. Hence, one will typically carry out this analysis as one of the first steps.

Now try to perform the same analysis as above using BioJupies (`https://maayanlab.cloud/biojupies/`). BioJupies is a web server that automatically generates RNA-seq data analysis notebooks. The data of Haglund et al. [1] can be found in BioJupies by searching for GSE37211. Compare and contrast the analyses that you performed above with the results that you obtain with BioJupies.

# 4 Further pointers

Worked out solutions and code for the exercises can be found at `https://bioinformatics.amc.nl/education/gs-bioinformatics-sequence-analysis/`

# 5 System requirements

## 5.1 Installation of required R/Bioconductor packages

For the L0/L01 desktops only. See the mail I sent earlier this week for installation instructions for the CDW or your own laptop:

- Open R version 3.4.4 or RStudio.
- Now you'll have to install a number of *Bioconductor* packages:
  1. type `.libPaths("C:/Scratch")` at the R prompt
  2. Type `source("http://bioconductor.org/biocLite.R")` at the R prompt
  3. Install the following packages via `biocLite(c("BiocParallel","DESeq2","edgeR", "GenomicAlignments","GenomicFeatures","parathyroidSE","Rsamtools"))`.
  4. If you are asked "Update all/some/none? [a/s/n]:", answer "n".

# 6 Session info

As last part of this document, we call the function `sessionInfo`, which reports the version numbers of R and all the packages used in this session. It is good practice to always keep such a record as it will help to trace down what has happened in case that an R script ceases to work because a package has been changed in a newer version.

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252
## [2] LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
##  [1] DESeq2_1.26.0               GenomicAlignments_1.22.1
##  [3] Rsamtools_2.2.3             Biostrings_2.54.0
##  [5] XVector_0.26.0              parathyroidSE_1.24.0
```

```
##  [7] SummarizedExperiment_1.16.1 DelayedArray_0.12.2
##  [9] BiocParallel_1.20.1         matrixStats_0.55.0
## [11] GenomicFeatures_1.38.2      AnnotationDbi_1.48.0
## [13] Biobase_2.46.0              GenomicRanges_1.38.0
## [15] GenomeInfoDb_1.22.0         IRanges_2.20.0
## [17] S4Vectors_0.24.0            BiocGenerics_0.32.0
## [19] knitr_1.25
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6           bit64_0.9-7
##  [3] RColorBrewer_1.1-2     progress_1.2.2
##  [5] httr_1.4.1             tools_3.6.1
##  [7] backports_1.1.5        R6_2.4.0
##  [9] rpart_4.1-15           Hmisc_4.3-1
## [11] DBI_1.1.0              colorspace_1.4-1
## [13] nnet_7.3-12            tidyselect_1.1.0
## [15] gridExtra_2.3          prettyunits_1.0.2
## [17] bit_1.1-14             curl_4.2
## [19] compiler_3.6.1         htmlTable_1.13.2
## [21] labeling_0.3           rtracklayer_1.46.0
## [23] scales_1.0.0           checkmate_1.9.4
## [25] genefilter_1.68.0      askpass_1.1
## [27] rappdirs_0.3.1         stringr_1.4.0
## [29] digest_0.6.22          foreign_0.8-71
## [31] base64enc_0.1-3        pkgconfig_2.0.3
## [33] htmltools_0.4.0        dbplyr_1.4.2
## [35] highr_0.8              htmlwidgets_1.5.3
## [37] rlang_0.4.10           rstudioapi_0.10
## [39] RSQLite_2.1.2          acepack_1.4.1
## [41] dplyr_0.8.3            RCurl_1.95-4.12
## [43] magrittr_1.5           GenomeInfoDbData_1.2.2
## [45] Formula_1.2-3          Matrix_1.2-17
## [47] Rcpp_1.0.2             munsell_0.5.0
## [49] lifecycle_1.0.0        stringi_1.4.3
## [51] zlibbioc_1.32.0        BiocFileCache_1.10.2
## [53] grid_3.6.1             blob_1.2.0
## [55] crayon_1.3.4           lattice_0.20-41
## [57] splines_3.6.1          annotate_1.64.0
## [59] hms_0.5.2              locfit_1.5-9.1
## [61] pillar_1.4.7           geneplotter_1.64.0
## [63] codetools_0.2-16       biomaRt_2.42.0
## [65] XML_3.99-0.3           glue_1.3.1
## [67] evaluate_0.14          latticeExtra_0.6-28
## [69] data.table_1.12.6      vctrs_0.3.6
## [71] gtable_0.3.0           openssl_1.4.1
## [73] purrr_0.3.3            assertthat_0.2.1
## [75] ggplot2_3.3.3          xfun_0.21
## [77] xtable_1.8-4           survival_3.1-8
## [79] tibble_3.0.6           snow_0.4-3
## [81] memoise_1.1.0          cluster_2.1.0
## [83] ellipsis_0.3.0
```

## Acknowledgements

I would like to thank the Bioconductor community (in particular, Hervé Pagès, Michael Love, Simon Anders, Wolfgang Huber, Sean Davis, Mark Robinson and Gordon Smyth) for providing the packages, vignettes and course material that formed the basis for these exercises.

## References

[1] Felix Haglund, Ran Ma, Mikael Huss, Luqman Sulaiman, Ming Lu, Inga-Lena Nilsson, Anders Höög, Christofer C. Juhlin, Johan Hartman, and Catharina Larsson. Evidence of a Functional Estrogen Receptor in Parathyroid Adenomas. *Journal of Clinical Endocrinology & Metabolism*, September 2012.