Practical

Statistical analysis of RNA-Seq data

(solutions)

Ignacio González

Plateforme Bioinformatique – INRA Toulouse
Plateforme Biostatistique – IMT Université Toulouse III

Toulouse, 20/21 novembre 2014

Contents

1 Introduction

In this practical, you will learn how to read count table – such as arising from a RNA-Seq experiment – analyze count tables for differentially expressed genes, visualize the results, and cluster samples and genes using transformed counts. This practical covers two widely-used tools for this task: *DESeq2* and *edgeR*, both available as packages of the *Bioconductor* project.

2 Input data and preparations

For the purposes of this practical, we will make use of *pasilla* data. This data set is from an experiment on *Drosophila melanogaster* cell cultures and investigated the effect of RNAi knock-down of the splicing factor *pasilla* (Brooks et al. 2011)[1]. The detailed transcript of how we produced the pasilla count table from second generation sequencing (Illumina) FASTQ files is provided in the vignette of the data package *pasilla*. The *pasilla* data for this practical is suplied in the "RNAseq_data" file. After dowloading this file, you should have the following directory structure in your computer:

```
your directory
|-- *
|-- RNAseq_data
    |-- count_table_files
        |-- count_table.tsv
        `-- pasilla_design.txt
    |
    `-- separate_files
        |-- pasilla_design.txt
        |-- treated1fb.txt
        |-- treated2fb.txt
        |-- treated3fb.txt
        |-- untreated1fb.txt
        |-- untreated2fb.txt
        |-- untreated3fb.txt
        `-- untreated4fb.txt
|-- *
```

Considerations:

- As input, the *DESeq2* and *edgeR* packages expects count data in the form of a rectangular table of integer values. The table cell in the g-th row and the j-th column of the table tells how many reads have been mapped to gene g in sample j.
- The count values must be raw, unnormalized read counts. This precludes the use of transformed, non-count inputs such as RPKM (reads per kilobase model) and FPKM (fragments per kilobase model), depth-adjusted read counts or various other preprocessed RNA-seq expression measures.
- Furthermore, it is important that each column stems from an independent biological replicate. For technical replicates (e. g. when the same library preparation was distributed over multiple lanes of the sequencer), sum up their counts to get a single column, corresponding to a unique biological replicate. This is needed in order to allow *DESeq2* and *edgeR* to estimate variability in the experiment correctly.

There are different ways to read in RNA-seq data into *R*, depending on the "raw" data format at hand. In practice, the RNA-seq data would either be read from a count table (matrix) or from separate files perhaps generated by the *HTSeq* python package.[2]

[1] Brooks, Angela N. et al. (2011). "Conservation of an RNA regulatory map between Drosophila and mammals". In: Genome Research. http://dx.doi.org/10.1101/gr.108662.110

[2] available from http://www-huber.embl.de/users/anders/HTSeq

3 Running the DESeq2 pipeline

The data object class used by the *DESeq2* package to store the read counts is *DESeqDataSet*. This facilitates preparation steps and also downstream exploration of results.

A *DESeqDataSet* object must have an associated "design formula". The design is specified at the beginning of the analysis, as this will inform many of the *DESeq2* functions how to treat the samples in the analysis. The formula should be a tilde (~) followed by the variables with plus signs between them. The simplest design formula for differential expression would be `~ condition`, where `condition` specifies which of two (or more groups) the samples belong to. Note that *DESeq2* uses the same kind of formula as in base *R*, e. g., for use by the `lm()` function.

In this *DESeq2* pipeline, we will demonstrate the construction of the *DESeqDataSet* from two starting points:

1. from a count table (i.e. matrix) and a table of sample information
2. from separate files created by, e. g., the *HTSeq* python package.

We first load the *DESeq2* package.

```
library(DESeq2)
```

3.1 Starting from count table

First you will be to specify a variable which points to the directory in which the RNA-seq data is located.

```
directory = "/path/to/your/directory/RNAseq_data/count_table_files"
```

Use `dir()` to discover the files in the specified directory

```
dir(directory)
```

```
[1] "count_table.tsv"    "pasilla_design.txt"
```

and set the working directory

```
setwd(directory)
```

Exercise 3.1 *Read the "count_table.tsv" and "pasilla_design.txt" files in to R using the function `read.table()` and create variables `rawCountTable` and `sampleInfo` from it. Check the arguments `header`, `sep` and `row.names`.*

Solution 3.1

```
rawCountTable = read.table("count_table.tsv", header = TRUE, row.names = 1)

sampleInfo = read.table("pasilla_design.txt", header = TRUE, row.names = 1)
```

Here, `header = TRUE` indicates that the first line contains column names and `row.names = 1` means that the first column should be used as row names.

Exercise 3.2 *Look at the first few rows of the `rawCountTable` using the `head()` function to see how it is formatted. How many genes are there in this table? To do this use the `nrow()` function.*

Solution 3.2

```
head(rawCountTable)
```

	untreated1	untreated2	untreated3	untreated4	treated1	treated2	treated3
FBgn0000003	0	0	0	0	0	0	1
FBgn0000008	92	161	76	70	140	88	70
FBgn0000014	5	1	0	0	4	0	0
FBgn0000015	0	2	1	2	1	0	0
FBgn0000017	4664	8714	3564	3150	6205	3072	3334
FBgn0000018	583	761	245	310	722	299	308

The `head()` function restricts the output to the first few lines. In this count table, each row represents a gene, each column a sample (sequenced RNA library), and the values give the raw numbers of sequencing reads that were mapped to the respective gene in each library. The number of genes (rows) present in `rawCountTable` is

```
nrow(rawCountTable)
```

```
[1] 14599
```

Exercise 3.3 *Look at the `sampleInfo` table. Are sample names in the same order as in `rawCountTable`? Order the `rawCountTable` according to the `sampleInfo` if it is necessary.*

Solution 3.3

```
sampleInfo
```

```
                 type number.of.lanes total.number.of.reads exon.counts
treated1   single-read               5              35158667    15679615
treated2    paired-end               2         12242535 (x2)    15620018
treated3    paired-end               2         12443664 (x2)    12733865
untreated1 single-read               2              17812866    14924838
untreated2 single-read               6              34284521    20764558
untreated3  paired-end               2         10542625 (x2)    10283129
untreated4  paired-end               2         12214974 (x2)    11653031
```

Verifying the order of samples.

```
cbind(rownames(sampleInfo), colnames(rawCountTable))
```

```
     [,1]         [,2]
[1,] "treated1"   "untreated1"
[2,] "treated2"   "untreated2"
[3,] "treated3"   "untreated3"
[4,] "untreated1" "untreated4"
[5,] "untreated2" "treated1"
[6,] "untreated3" "treated2"
[7,] "untreated4" "treated3"
```

Ordering the `rawCountTable` according to the `sampleInfo`. We create an index which will put them in the same order: the `sampleInfo` comes first because this is the sample order we want to achieve.

```
Idx = match(rownames(sampleInfo), colnames(rawCountTable))

rawCountTable = rawCountTable[, Idx]
```

We then check to see that we have lined them up correctly.

```
head(rawCountTable)
```

```
            treated1 treated2 treated3 untreated1 untreated2 untreated3 untreated4
FBgn0000003        0        0        1          0          0          0          0
FBgn0000008      140       88       70         92        161         76         70
FBgn0000014        4        0        0          5          1          0          0
FBgn0000015        1        0        0          0          2          1          2
FBgn0000017     6205     3072     3334       4664       8714       3564       3150
FBgn0000018      722      299      308        583        761        245        310
```

Exercise 3.4 *Create a `"condition"` additional column in the `sampleInfo` data table specifying to which of both groups (`"treated"`, `"control"`) the samples belong.*

Solution 3.4

```
condition = c("treated", "treated", "treated",
              "control", "control", "control", "control")
sampleInfo = data.frame(sampleInfo, condition)
```

```
sampleInfo
```

```
                 type number.of.lanes total.number.of.reads exon.counts condition
treated1   single-read               5              35158667    15679615   treated
treated2    paired-end               2         12242535 (x2)    15620018   treated
treated3    paired-end               2         12443664 (x2)    12733865   treated
untreated1 single-read               2              17812866    14924838   control
untreated2 single-read               6              34284521    20764558   control
untreated3  paired-end               2         10542625 (x2)    10283129   control
untreated4  paired-end               2         12214974 (x2)    11653031   control
```

You now have all the ingredients to prepare your *DESeqDataSet* data object, namely:

- `rawCountTable`: a table with the read counts
- `sampleInfo`: a table with metadata on the count table's columns

Exercise 3.5 *Use the function `DESeqDataSetFromMatrix()` to construct a DESeqDataSet data object and create a variable `ddsFull` from it. For this function you should provide the counts matrix, the column information as a data.frame and the design formula.*

Solution 3.5

To construct the *DESeqDataSet* data object from the matrix of counts and the metadata table, use:

```
ddsFull = DESeqDataSetFromMatrix(countData = rawCountTable, colData = sampleInfo,
                                 design = ~ condition)
ddsFull
```

```
class: DESeqDataSet
dim: 14599 7
exptData(0):
assays(1): counts
rownames(14599): FBgn0000003 FBgn0000008 ... FBgn0261574 FBgn0261575
rowData metadata column names(0):
colnames(7): treated1 treated2 ... untreated3 untreated4
colData names(5): type number.of.lanes total.number.of.reads exon.counts
  condition
```

3.2 Starting from separate files

First you will want to specify a variable which points to the directory in which the separate files are located.

```
directory = "/path/to/your/directory/RNAseq_data/separate_files/"
```

Use `dir()` to discover the files in the specified directory

```
dir(directory)
```

```
[1] "pasilla_design.txt" "treated1fb.txt"     "treated2fb.txt"     "treated3fb.txt"
[5] "untreated1fb.txt"   "untreated2fb.txt"   "untreated3fb.txt"   "untreated4fb.txt"
```

For to construct a *DESeqDataSet* data object from separate files, you must provide a *data.frame* specifying which files to read and column information. This *data.frame* shall contain three or more columns. Each row describes one sample. The first column is the sample name, the second column the file name of each count file, and the remaining columns are sample metadata.

Exercise 3.6 *List all files in your directory using `list.files()` and select those files which contain the count values. Create a variable `sampleFiles` containing the name of the selected files.*

Solution 3.6

```
list.files(directory)
```

```
[1] "pasilla_design.txt" "treated1fb.txt"     "treated2fb.txt"     "treated3fb.txt"
[5] "untreated1fb.txt"   "untreated2fb.txt"   "untreated3fb.txt"   "untreated4fb.txt"
```

```
sampleFiles = list.files(directory)[2:8]
sampleFiles
```

```
[1] "treated1fb.txt"   "treated2fb.txt"   "treated3fb.txt"   "untreated1fb.txt"
[5] "untreated2fb.txt" "untreated3fb.txt" "untreated4fb.txt"
```

Exercise 3.7 *Create a data frame `fileInfo` containing three columns, with the sample names in the first column, the file names in the second column, and the condition in the third column specifying to which of both groups (`"treated"`, `"control"`) the samples belong. Heads these columns by `sampleName`, `sampleFiles` and `condition` respectively.*

Solution 3.7

```
sampleName = unlist(strsplit(sampleFiles, "fb.txt", fixed = TRUE))

condition = c("treated", "treated", "treated",
              "control", "control", "control", "control")

fileInfo =  data.frame(sampleName, sampleFiles, condition)
fileInfo
```

```
  sampleName      sampleFiles condition
1   treated1   treated1fb.txt   treated
2   treated2   treated2fb.txt   treated
3   treated3   treated3fb.txt   treated
4 untreated1 untreated1fb.txt   control
5 untreated2 untreated2fb.txt   control
6 untreated3 untreated3fb.txt   control
7 untreated4 untreated4fb.txt   control
```

Exercise 3.8 *Use the function `DESeqDataSetFromHTSeqCount()` to construct the DESeqDataSet data object. For this function you should provide a data.frame specifying which files to read and column information, the directory relative to which the filenames are specified and the design formula.*

Solution 3.8

To construct the *DESeqDataSet* data object from separate files, use:

```
ddsHTSeq = DESeqDataSetFromHTSeqCount(sampleTable = fileInfo, directory = directory,
                                      design= ~ condition)
ddsHTSeq
```

```
class: DESeqDataSet
dim: 70463 7
exptData(0):
assays(1): counts
rownames(70463): FBgn0000003:001 FBgn0000008:001 ... FBgn0261575:001
  FBgn0261575:002
rowData metadata column names(0):
colnames(7): treated1 treated2 ... untreated3 untreated4
colData names(1): condition
```

3.3 Preparing the data object for the analysis of interest

Continue with the `ddsFull` data object constructed from the count table method above (see Section 3.1).

To analyse these samples, you will have to account for the fact that you have both single-end and paired-end method. To keep things simple at the start, first realize a simple analysis by using only the paired-end samples.

Exercise 3.9 *Select the subset paired-end samples from the `ddsFull` data object. Use the `colData()` function to get the colunm data (the metadata table), subset the `ddsFull` colunms accordingly and create a variable `dds` from it.*

Solution 3.9

```
pairedSamples = (colData(ddsFull)$type == "paired-end")

dds = ddsFull[, pairedSamples]
```

Now, you have data as follows:

```
dds
```

```
class: DESeqDataSet
dim: 14599 4
exptData(0):
assays(1): counts
rownames(14599): FBgn0000003 FBgn0000008 ... FBgn0261574 FBgn0261575
rowData metadata column names(0):
colnames(4): treated2 treated3 untreated3 untreated4
colData names(5): type number.of.lanes total.number.of.reads exon.counts
  condition
```

```
colData(dds)
```

```
DataFrame with 4 rows and 5 columns
                type number.of.lanes total.number.of.reads exon.counts condition
            <factor>       <integer>              <factor>   <integer>  <factor>
treated2   paired-end               2         12242535 (x2)    15620018   treated
treated3   paired-end               2         12443664 (x2)    12733865   treated
untreated3 paired-end               2         10542625 (x2)    10283129   control
untreated4 paired-end               2         12214974 (x2)    11653031   control
```

3.4 Data exploration and quality assesment

For data exploration and visualisation, use pseudocounts data, i. e., transformed versions of the count data of the form $y = \log_2(K + 1)$ where K represents the count values.

Exercise 3.10 *Use the `counts()` function to extract the count values from the `dds` data object and create a variable `pseudoCount` with the transformed count data.*

Solution 3.10

```
pseudoCount = log2(counts(dds) + 1)

head(pseudoCount)
```

```
           treated2 treated3 untreated3 untreated4
FBgn0000003    0.000    1.000      0.000      0.000
FBgn0000008    6.476    6.150      6.267      6.150
FBgn0000014    0.000    0.000      0.000      0.000
FBgn0000015    0.000    0.000      1.000      1.585
FBgn0000017   11.585   11.703     11.800     11.622
FBgn0000018    8.229    8.271      7.943      8.281
```

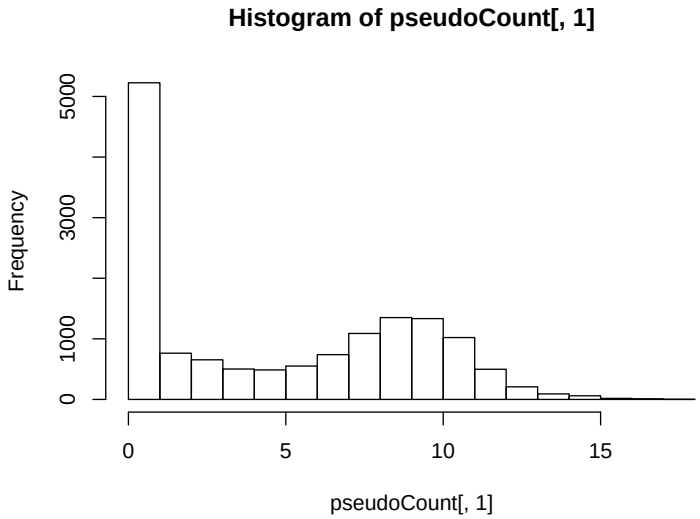
Inspect sample distributions

Exercise 3.11 *Use the `hist()` function to plot histograms from `pseudoCount` data for each sample.*

Solution 3.11

Histogram from the `treated2` sample.

```
hist(pseudoCount[, 1])
```

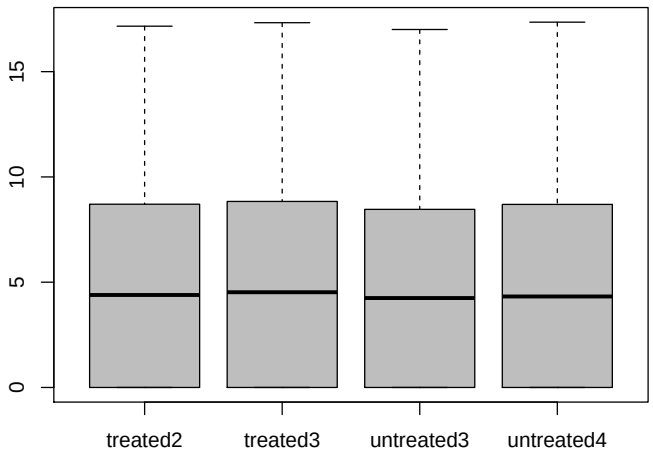


Exercise 3.12 *Use the `boxplot()` function to display parallel boxplots from `pseudoCount` data.*

Solution 3.12

Using `col = "gray"` in `boxplot()` to colour the bodies of the box plots.

```
boxplot(pseudoCount, col = "gray")
```



Inspect sample relations

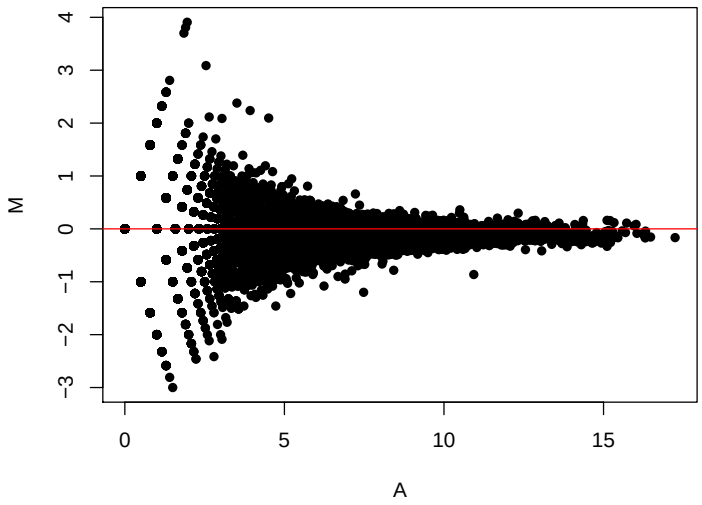
Exercise 3.13 *Create MA-plot from `pseudoCount` data for `"treated"` and `"control"` samples. Follow this steps:*

- *obtain the A-values, i.e., the $\log_2$-average level counts for each gene across the two samples,*
- *obtain the M-values, i.e., the $\log_2$-difference of level counts for each gene between two samples,*
- *create a scatterplot with the A-values in the x axis and the M-values in the y axis.*

Solution 3.13

MA-plot between treated samples. Use the `abline()` function to add one horizontal red line (at zero) to the current plot.

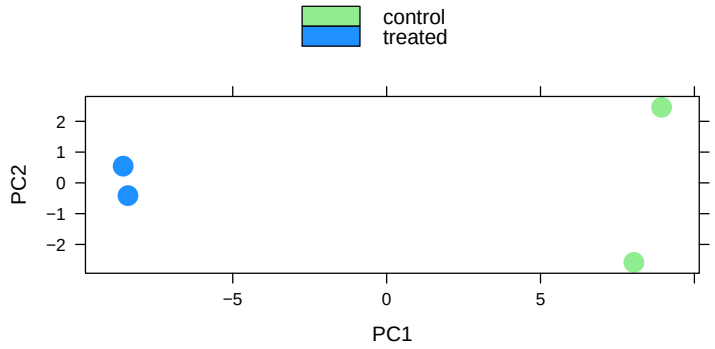
```
x = pseudoCount[, 1] # treated2 sample
y = pseudoCount[, 2] # treated3 sample

## A and M values
A = (x + y)/2
M = x - y

plot(A, M, pch = 16)
abline(h = 0, col = "red")
```



Exercise 3.14 *Invoke a variance stabilizing transformation (`varianceStabilizingTransformation()`) and create a variable `vsd` from it. Inspect a principal component analysis (PCA) plot from the transformated data using the `plotPCA()` function.*

Solution 3.14

```
vsd = varianceStabilizingTransformation(dds)

plotPCA(vsd, intgroup = "condition")
```



Exercise 3.15 *Explore the similarities between sample looking a clustering image map (CIM) or heatmap of sample-to-sample distance matrix. To avoid that the distance measure is dominated by a few highly variable genes, and have a roughly equal contribution from all genes, use it on the* `vsd`*-transformed data:*

i. extract the transformed data from the `vsd` *data object using the function* `assay()`*;*

ii. use the function `dist()` *to calculate the Euclidean distance between samples from the transformed data. First, use the function* `t()` *to transpose this data matrix, you need this because* `dist()` *calculates distances between data rows and your samples constitute the columns. Coerce the result of the* `dist()` *function to matrix using* `as.matrix()`*;*

iii. load the mixOmics package and use the utility function, `cim()`*, to produce a CIM.*

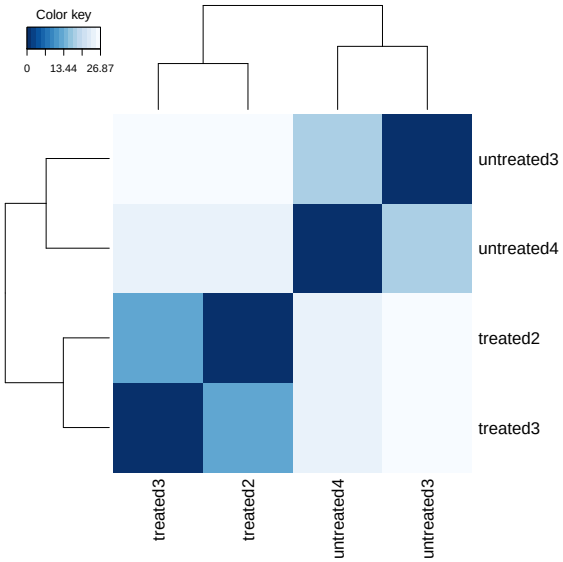
Solution 3.15

```
sampleDists = as.matrix(dist(t(assay(vsd))))

sampleDists
```

```
           treated2 treated3 untreated3 untreated4
treated2       0.00    12.77      26.40      24.53
treated3      12.77     0.00      26.87      24.03
untreated3    26.40    26.87       0.00      17.40
untreated4    24.53    24.03      17.40       0.00
```

Use `col = seqColor` for sequential colour schemes in the `cim()` function.

```
## For sequential colour schemes
library(RColorBrewer)
seqColor = colorRampPalette(rev(brewer.pal(9, "Blues")))(16)

## Load mixOmics
library(mixOmics)

cim(sampleDists, col = seqColor, symkey = FALSE)
```



3.5 Differential expression analysis

The standard differential expression analysis steps are wrapped into a single function, `DESeq()`. This function performs a default analysis through the steps:

1. estimation of size factors
2. estimation of dispersion
3. negative binomial fitting and Wald statistics

Exercise 3.16 *With the data object `dds` prepared, run the DESeq2 analysis calling to the function `DESeq()`.*

Solution 3.16

```
dds = DESeq(dds)
```

3.6 Inspecting the results

Results tables are generated using the function `results()`, which extracts results from a `DESeq()` analysis giving base means across samples, $\log_2$ fold changes, standard errors, test statistics, p-values and adjusted p-values. If the argument `independentFiltering = TRUE` (the default) independent filtering is applied automatically.

Exercise 3.17 *Extract the results from `DESeq()` output using the `results()` function and create a variable `res` from it. Visualize and inspect this variable.*

Solution 3.17

```
res = results(dds)

res
```

```
log2 fold change (MAP): condition treated vs control
Wald test p-value: condition treated vs control
DataFrame with 14599 rows and 6 columns
             baseMean log2FoldChange     lfcSE      stat    pvalue      padj
            <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
FBgn0000003    0.2243        0.09911    0.1901    0.5213   0.60218        NA
FBgn0000008   76.2956       -0.06366    0.2918   -0.2182   0.82728    0.9551
FBgn0000014    0.0000             NA        NA        NA        NA        NA
FBgn0000015    0.7811       -0.36440    0.3330   -1.0941   0.27389        NA
FBgn0000017 3298.6822       -0.25932    0.1282   -2.0223   0.04315    0.2357
...               ...            ...       ...       ...       ...       ...
FBgn0261571     0.000             NA        NA        NA        NA        NA
FBgn0261572     5.273       -0.66152    0.5217   -1.2680    0.2048        NA
FBgn0261573  1728.420        0.05192    0.1157    0.4486    0.6537    0.9017
FBgn0261574  3129.037       -0.04206    0.1388   -0.3030    0.7619    0.9385
FBgn0261575     2.659        0.20201    0.4858    0.4158    0.6775        NA
```

Note that a subset of the p-values in `res` are NA ("not available"). The p-values and adjusted p-values can be set to NA here for three reasons: 1) all samples had a count of zero (in which case the `baseMean` column will have a zero and tests cannot be performed); 2) a count outlier was detected (in which case the p-value and adjusted p-value will be set to NA to avoid a potential false positive call of differential expression); or 3) the gene was filtered by automatic independent filtering (in which case only the adjusted p-value will be set to NA).

Exercise 3.18 *Obtain information on the meaning of the columns using the `mcols()` function.*

Solution 3.18

```
mcols(res)
```

```
DataFrame with 6 rows and 2 columns
          type                                         description
   <character>                                         <character>
1 intermediate                           the base mean over all rows
2      results log2 fold change (MAP): condition treated vs control
3      results         standard error: condition treated vs control
4      results         Wald statistic: condition treated vs control
5      results      Wald test p-value: condition treated vs control
6      results                                 BH adjusted p-values
```

The `padj` column in the table `res` contains the adjusted p-values for multiple testing with the Benjamini-Hochberg procedure (i.e. FDR). This is the information that we will use to decide whether the expression of a given gene differs significantly across conditions (e.g. we can arbitrarily decide that genes with an FDR < 0.01 are differentially expressed).

Exercise 3.19 *Consider all genes with an adjusted p-value below 1% = 0.01 (`alpha = 0.01`) as significant. How many such genes are there?*

Solution 3.19

```
alpha = 0.01
sum(res$padj < alpha, na.rm = TRUE)
```

```
[1] 513
```

Exercise 3.20 *Select the significant genes (`alpha = 0.01`) and subset the `res` table to these genes. Sort it by the $\log_2$-fold-change estimate to get the significant genes with the strongest down-regulation.*

Solution 3.20

```
sigDownReg = res[res$padj < alpha & !is.na(res$padj), ]
sigDownReg = sigDownReg[order(sigDownReg$log2FoldChange), ]

head(sigDownReg)
```

```
log2 fold change (MAP): condition treated vs control
Wald test p-value: condition treated vs control
DataFrame with 6 rows and 6 columns
            baseMean log2FoldChange     lfcSE      stat     pvalue      padj
           <numeric>      <numeric> <numeric> <numeric>  <numeric> <numeric>
FBgn0039155    463.44         -4.228    0.1983   -21.322 7.146e-101 1.346e-97
FBgn0039827    188.59         -3.855    0.2653   -14.530  7.853e-48 7.394e-45
FBgn0085359     36.47         -3.569    0.4546    -7.850  4.145e-15 4.956e-13
FBgn0034434     82.89         -3.376    0.3347   -10.087  6.284e-24 1.821e-21
FBgn0024288     42.56         -3.300    0.4162    -7.929  2.201e-15 2.674e-13
FBgn0034736    162.04         -3.268    0.2590   -12.617  1.697e-36 7.991e-34
```

Exercise 3.21 *Repet the Exercise 3.20 for the strongest up-regulated genes.*

Solution 3.21

```
sigUpReg = res[res$padj < alpha & !is.na(res$padj), ]
sigUpReg = sigUpReg[order(sigUpReg$log2FoldChange, decreasing = TRUE), ]

head(sigUpReg)
```

```
log2 fold change (MAP): condition treated vs control
Wald test p-value: condition treated vs control
DataFrame with 6 rows and 6 columns
             baseMean log2FoldChange     lfcSE      stat     pvalue       padj
            <numeric>      <numeric> <numeric> <numeric>  <numeric>  <numeric>
FBgn0025111   1340.23          2.887    0.1248    23.129 2.359e-118 5.922e-115
FBgn0035189    197.47          2.786    0.2437    11.431  2.941e-30  1.231e-27
FBgn0033764     53.95          2.749    0.3719     7.392  1.443e-13  1.489e-11
FBgn0037290     50.45          2.488    0.3682     6.759  1.388e-11  1.149e-09
FBgn0000071    302.37          2.468    0.1879    13.136  2.054e-39  1.105e-36
FBgn0051092    128.83          2.250    0.2535     8.878  6.820e-19  1.223e-16
```

Exercise 3.22 *Create persistent storage of results. Save the result tables as a csv (comma-separated values) file using the `write.csv()` function (alternative formats are possible).*

Solution 3.22

```
write.csv(sigDownReg, file = "sigDownReg.csv")

write.csv(sigUpReg, file = "sigUpReg.csv")
```

3.7 Diagnostic plot for multiple testing

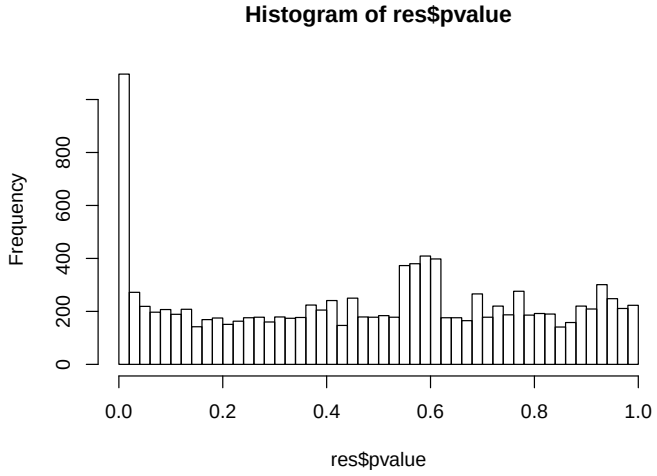
For diagnostic of multiple testing results it is instructive to look at the histogram of p-values.

Exercise 3.23 *Use the `hist()` function to plot a histogram from (unadjusted) p-values in the `res` data object.*

Solution 3.23

Use `breaks = 50` in `hist()` to generate this plot.

```
hist(res$pvalue, breaks = 50)
```



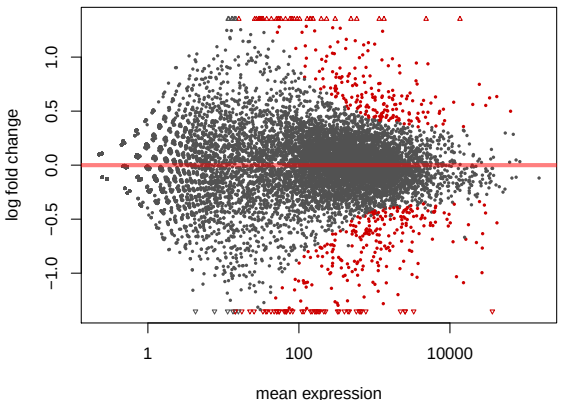
3.8 Interpreting the DE analysis results

MA-plot

Exercise 3.24 *Create a MA-plot using the `plotMA()` function showing the genes selected as differentially expressed with a 1% false discovery rate.*

Solution 3.24

```
plotMA(res, alpha = 0.01)
```

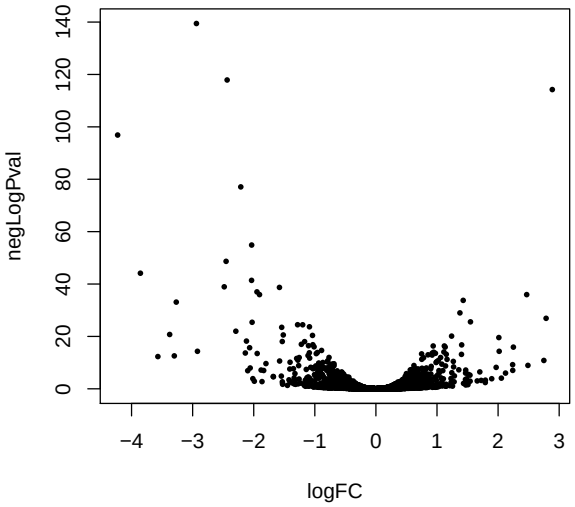


Volcano plot

Exercise 3.25 *Create a volcano-plot from the `res` data object. First construct a table containing the $\log_2$ fold change and the negative $\log_{10}$-transformed p-values, remove rows with NA adjusted p-values, then generate the volcano plot using the standard `plot()` command.* Hint: *the negative $\log_{10}$-transformed value of x is `-log10(x)`.*

Solution 3.25

```
resVol = data.frame(logFC = res$log2FoldChange, negLogPval = -log10(res$padj))
resVol = resVol[!is.na(res$padj), ]

plot(resVol, pch = 16, cex = 0.6)
```



Gene clustering

The first step to make a CIM (or heatmap) from RNA-seq analyzed data is to transform the normalisez counts of reads to (approximately) homoskedastic data.

Exercise 3.26 *Transform the normalized counts using the `varianceStabilizingTransformation()` function and create a variable `vsnd` from it. Set the `blind` argument in appropriate form.*

Solution 3.26

```
vsnd = varianceStabilizingTransformation(dds, blind = FALSE)
```

Since the clustering is only relevant for genes that actually are differentially expressed, carry it out only for a gene subset of most highly differential expression.

Exercise 3.27 *Extract the transformated data from the `vsnd` using the `assay()` function and select those genes that have adjusted p-values below 0.01 and absolute $\log_2$-fold-change above 2 from it, then use the function `cim()` to produce a CIM from this data.*

Solution 3.27

```
deg = (abs(res$log2FoldChange) > 2 & res$padj < 0.01 & !is.na(res$padj))
vsnd = vsnd[deg, ]
```

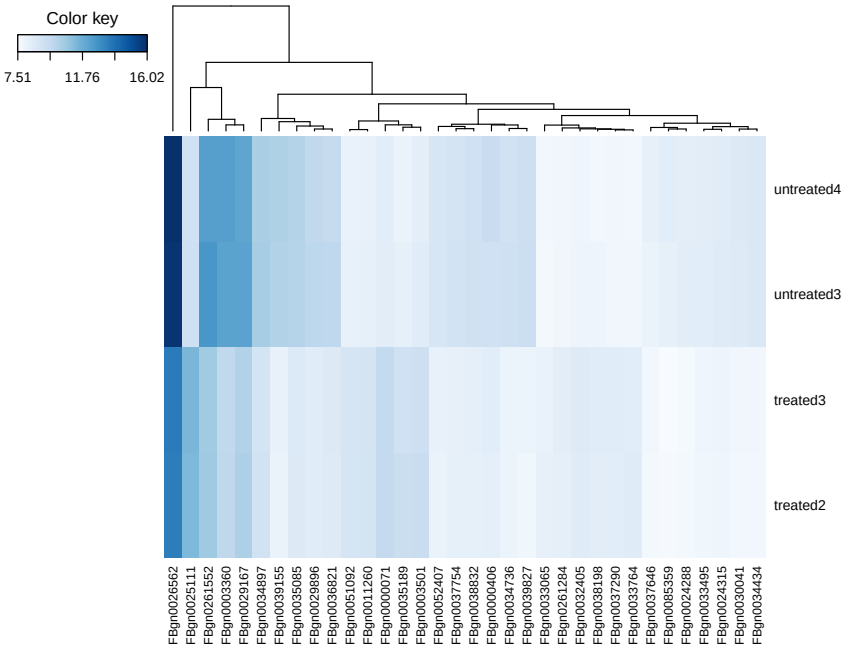
Transformed values for the first ten selected genes.

```
head(assay(vsnd), 10)
```

	treated2	treated3	untreated3	untreated4
FBgn0000071	9.651	9.710	8.406	8.441
FBgn0000406	8.270	8.424	9.218	9.449
FBgn0003360	9.817	9.749	12.174	12.309
FBgn0003501	9.458	9.328	8.474	8.373
FBgn0011260	9.027	9.005	8.233	8.176
FBgn0024288	7.651	7.616	8.399	8.368
FBgn0024315	7.906	7.928	8.544	8.498
FBgn0025111	11.376	11.419	9.257	9.213
FBgn0026562	13.462	13.529	15.817	16.017
FBgn0029167	10.309	10.162	12.209	12.088

Use `col = seqColor` for sequential colour schemes in the `cim()` function.

```
## For sequential colour schemes
library(RColorBrewer)
seqColor = colorRampPalette(brewer.pal(9, "Blues"))(255)

cim(t(assay(vsnd)), dendrogram = "column", col = seqColor, symkey = FALSE)
```



4 Running the edgeR pipeline

edgeR stores data in a simple list-based data object called a *DGEList* (Digital Gene Expression data - class). This type of object can be manipulated like any list in *R*. If the table of counts is already available as a matrix or a data frame, `countData` say, then a *DGEList* object can be made by

```
dge = DGEList(counts = countData, group = group)
```

where `group` is a factor identifying the group membership of each sample.

In this *edgeR* pipeline, we will demonstrate the construction of the *DGEList* from two starting points:

1. from a count table (i.e. matrix) and a table of sample information
2. from separate files created by, e. g., the *HTSeq* python package.

We first load the *edgeR* package.

```
library(edgeR)
```

4.1 Starting from count table

Continue with the `rawCountTable` and the `sampleInfo` data objects constructed in the Section 3.1 (see Exercises 3.1 – 3.4).

Exercise 4.1 *Use the function `DGEList()` to construct a DGEList data object and create a variable `dgeFull` from it. For this function you should provide the counts matrix and the vector or factor giving the experimental group/condition for each sample.*

Solution 4.1

To construct the *DGEList* data object from the matrix of counts and the group membership of each sample, use:

```
dgeFull = DGEList(counts = rawCountTable, group = sampleInfo$condition)
dgeFull
```

```
An object of class "DGEList"
$counts
            treated1 treated2 treated3 untreated1 untreated2 untreated3 untreated4
FBgn0000003        0        0        1          0          0          0          0
FBgn0000008      140       88       70         92        161         76         70
FBgn0000014        4        0        0          5          1          0          0
FBgn0000015        1        0        0          0          2          1          2
FBgn0000017     6205     3072     3334       4664       8714       3564       3150
14594 more rows ...

$samples
             group lib.size norm.factors
treated1   treated 18670279            1
treated2   treated  9571826            1
treated3   treated 10343856            1
untreated1 control 13972512            1
untreated2 control 21911438            1
untreated3 control  8358426            1
untreated4 control  9841335            1
```

Exercise 4.2 *Include the `sampleInfo` information in the `dgeFull` data object.*

Solution 4.2

```
dgeFull$sampleInfo = sampleInfo
dgeFull
```

```
An object of class "DGEList"
$counts
            treated1 treated2 treated3 untreated1 untreated2 untreated3 untreated4
FBgn0000003        0        0        1          0          0          0          0
FBgn0000008      140       88       70         92        161         76         70
FBgn0000014        4        0        0          5          1          0          0
FBgn0000015        1        0        0          0          2          1          2
FBgn0000017     6205     3072     3334       4664       8714       3564       3150
14594 more rows ...

$samples
             group lib.size norm.factors
treated1   treated 18670279            1
treated2   treated  9571826            1
treated3   treated 10343856            1
untreated1 control 13972512            1
untreated2 control 21911438            1
untreated3 control  8358426            1
untreated4 control  9841335            1

$sampleInfo
                  type number.of.lanes total.number.of.reads exon.counts condition
treated1   single-read               5              35158667    15679615   treated
treated2    paired-end               2          12242535 (x2)    15620018   treated
treated3    paired-end               2          12443664 (x2)    12733865   treated
untreated1 single-read               2              17812866    14924838   control
untreated2 single-read               6              34284521    20764558   control
untreated3  paired-end               2          10542625 (x2)    10283129   control
untreated4  paired-end               2          12214974 (x2)    11653031   control
```

4.2 Starting from separate files

First you will want to specify a variable which points to the directory in which the separate files are located.

```
directory = "/path/to/your/directory/RNAseq_data/separate_files/"
```

Use `dir()` to discover the files in the specified directory

```
dir(directory)
```

```
[1] "pasilla_design.txt" "treated1fb.txt"     "treated2fb.txt"     "treated3fb.txt"
[5] "untreated1fb.txt"   "untreated2fb.txt"   "untreated3fb.txt"   "untreated4fb.txt"
```

and set the working directory

```
setwd(directory)
```

For to construct a *DGEList* data object from separate files, you must provide a *data.frame* specifying which files to read and column information. This *data.frame* shall contain three or more columns. Each row describes one sample. A column called `files` with the sample name, other column called `group` containing the group to which each sample belongs and the remaining columns with sample information.

Exercise 4.3 *Read the "pasilla_design.txt" file in to R using the function `read.table()` and create the variable `fileInfo` from it. Check the arguments `header` and `sep`.*

Solution 4.3

```
fileInfo = read.table("pasilla_design.txt", header = TRUE)
fileInfo
```

```
             files        type number.of.lanes total.number.of.reads exon.counts
1   treated1fb.txt single-read               5              35158667    15679615
2   treated2fb.txt  paired-end               2          12242535 (x2)    15620018
3   treated3fb.txt  paired-end               2          12443664 (x2)    12733865
4 untreated1fb.txt single-read               2              17812866    14924838
5 untreated2fb.txt single-read               6              34284521    20764558
6 untreated3fb.txt  paired-end               2          10542625 (x2)    10283129
7 untreated4fb.txt  paired-end               2          12214974 (x2)    11653031
```

Exercise 4.4 *Create an additional column in the `fileInfo` data table, called `group`, specifying to which of both groups (`"treated"`, `"control"`) the samples belong.*

Solution 4.4

```
group = c("treated", "treated", "treated", "control", "control", "control", "control")

fileInfo =  data.frame(fileInfo, group)
fileInfo
```

```
             files        type number.of.lanes total.number.of.reads exon.counts   group
1   treated1fb.txt single-read               5              35158667    15679615 treated
2   treated2fb.txt  paired-end               2          12242535 (x2)    15620018 treated
3   treated3fb.txt  paired-end               2          12443664 (x2)    12733865 treated
4 untreated1fb.txt single-read               2              17812866    14924838 control
5 untreated2fb.txt single-read               6              34284521    20764558 control
6 untreated3fb.txt  paired-end               2          10542625 (x2)    10283129 control
7 untreated4fb.txt  paired-end               2          12214974 (x2)    11653031 control
```

Exercise 4.5 *Use the function `readDGE()` to construct a readDGE data object. For this function you should provide a data.frame, which, under the headings `files` and `group`, are the filename and the group information.*

Solution 4.5

To construct a *readDGE* data object from separate files, use:

```
dgeHTSeq = readDGE(fileInfo)
dgeHTSeq
```

```
An object of class "DGEList"
$samples
             files        type number.of.lanes total.number.of.reads exon.counts   group
1   treated1fb.txt single-read               5              35158667    15679615 treated
2   treated2fb.txt  paired-end               2          12242535 (x2)    15620018 treated
3   treated3fb.txt  paired-end               2          12443664 (x2)    12733865 treated
4 untreated1fb.txt single-read               2              17812866    14924838 control
5 untreated2fb.txt single-read               6              34284521    20764558 control
6 untreated3fb.txt  paired-end               2          10542625 (x2)    10283129 control
7 untreated4fb.txt  paired-end               2          12214974 (x2)    11653031 control
  lib.size norm.factors
1 88834542            1
2 22381538            1
3 22573930            1
4 37094861            1
5 66650218            1
6 19318565            1
7 20196315            1
```

```
$counts
                1 2 3 4 5 6 7
FBgn0000008:001 0 0 0 0 0 0 0
FBgn0000008:002 0 0 0 0 0 1 0
FBgn0000008:003 0 1 0 1 1 1 0
FBgn0000008:004 1 0 1 0 1 0 1
FBgn0000008:005 4 1 1 2 2 0 1
70461 more rows ...
```

4.3 Preparing the data object for the analysis of interest

Continue with the `dgeFull` data object constructed from the count table method above (see Section 4.1).

To analyse these samples, you will have to account for the fact that you have both single-end and paired-end method. To keep things simple at the start, first realize a simple analysis by using only the paired-end samples.

Exercise 4.6 *Select the subset paired-end samples from the `dgeFull` data object and create a variable `dge` from it.*

Solution 4.6

```
pairedSamples = (dgeFull$sampleInfo$type == "paired-end")

dge = dgeFull[, pairedSamples]
dge$sampleInfo = dgeFull$sampleInfo[pairedSamples, ]
```

Now, you have data as follows:

```
dge
```

```
An object of class "DGEList"
$counts
            treated2 treated3 untreated3 untreated4
FBgn0000003        0        1          0          0
FBgn0000008       88       70         76         70
FBgn0000014        0        0          0          0
FBgn0000015        0        0          1          2
FBgn0000017     3072     3334       3564       3150
14594 more rows ...

$samples
             group lib.size norm.factors
treated2   treated  9571826            1
treated3   treated 10343856            1
untreated3 control  8358426            1
untreated4 control  9841335            1

$sampleInfo
                 type number.of.lanes total.number.of.reads exon.counts condition
treated2   paired-end               2         12242535 (x2)    15620018   treated
treated3   paired-end               2         12443664 (x2)    12733865   treated
untreated3 paired-end               2         10542625 (x2)    10283129   control
untreated4 paired-end               2         12214974 (x2)    11653031   control
```

4.4 Data exploration and quality assesment

For data exploration and visualisation, use pseudocounts data, i. e., transformed versions of the count data of the form $y = \log_2(K + 1)$ where K represents the count values.

Exercise 4.7 *Extract the count values from the `dge` data object and create a variable `pseudoCount` with the transformed values.*

Solution 4.7

```
pseudoCount = log2(dge$counts + 1)

head(pseudoCount)
```

```
            treated2 treated3 untreated3 untreated4
FBgn0000003    0.000    1.000      0.000      0.000
FBgn0000008    6.476    6.150      6.267      6.150
FBgn0000014    0.000    0.000      0.000      0.000
FBgn0000015    0.000    0.000      1.000      1.585
FBgn0000017   11.585   11.703     11.800     11.622
FBgn0000018    8.229    8.271      7.943      8.281
```

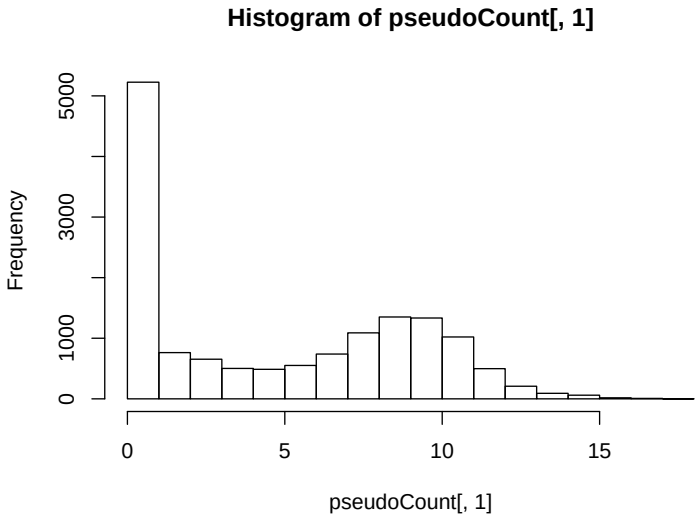
Inspect sample distributions

Exercise 4.8 *Use the `hist()` function to plot histograms from `pseudoCount` data for each sample.*

Solution 4.8

Histogram from the `treated2` sample.

```
hist(pseudoCount[, 1])
```

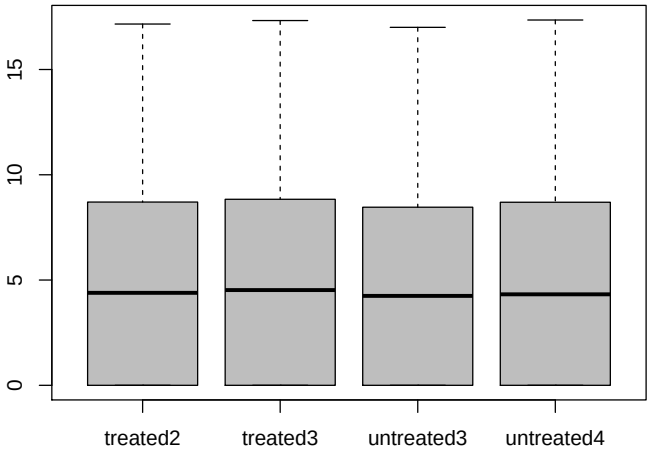


Exercise 4.9 *Use the `boxplot()` function to display parallel boxplots from `pseudoCount` data.*

Solution 4.9

Using `col = "gray"` in boxplot() to colour the bodies of the box plots.

```
boxplot(pseudoCount, col = "gray")
```



Inspect sample relations

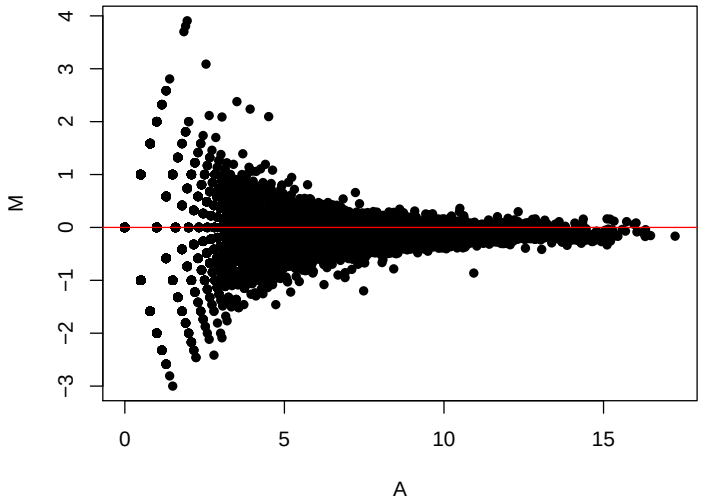
Exercise 4.10 *Create MA-plot from* `pseudoCount` *data for* `"treated"` *and* `"control"` *samples. Follow this steps:*

- *obtain the A-values, i.e., the* $\log_2$*-average level counts for each gene across the two samples,*
- *obtain the M-values, i.e., the* $\log_2$*-difference of level counts for each gene between two samples,*
- *create a scatterplot with the A-values in the* x *axis and the M-values in the* y *axis.*

Solution 4.10

MA-plot between treated samples. Use the `abline()` function to add one horizontal red line (at zero) to the current plot.

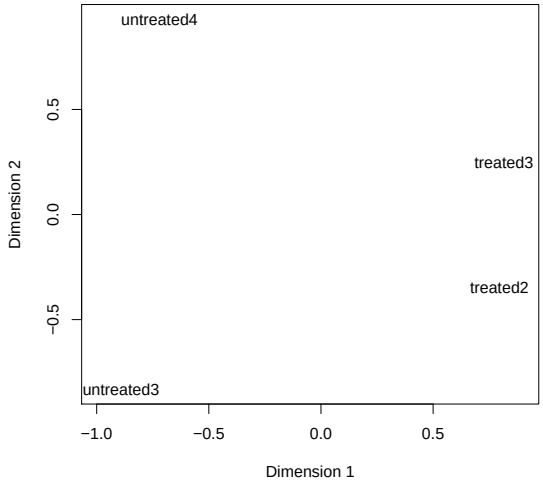
```
x = pseudoCount[, 1] # treated2 sample
y = pseudoCount[, 2] # treated3 sample

## A and M values
A = (x + y)/2
M = x - y

plot(A, M, pch = 16)
abline(h = 0, col = "red")
```



Exercise 4.11 *Inspect a multidimensional scaling plot from the `pseudoCount` data using the `plotMDS()` function.*

Solution 4.11

```
plotMDS(pseudoCount)
```



Exercise 4.12 *Explore the similarities between sample looking a clustering image map (CIM) or heatmap of sample-to-sample distance matrix. To avoid that the distance measure is dominated by a few highly variable genes, and have a roughly equal contribution from all genes, use it on the `pseudoCount` data:*

i. use the function `dist()` to calculate the Euclidean distance between samples from the transformed data. First, use the function `t()` to transpose this data matrix, you need this because `dist()` calculates distances between data rows and your samples constitute the columns. Coerce the result of the `dist()` function to matrix using `as.matrix()`;

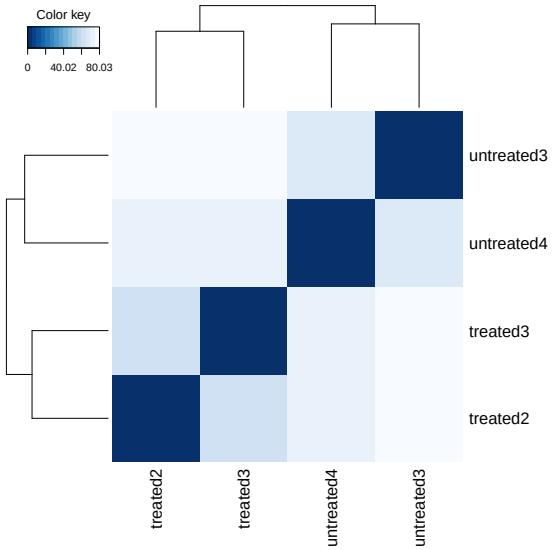
ii. load the mixOmics package and use the utility function, `cim()`, to produce a CIM.

Solution 4.12

```
sampleDists = as.matrix(dist(t(pseudoCount)))

## Distance matrix between samples from the pseudoCount data
sampleDists
```

```
           treated2 treated3 untreated3 untreated4
treated2       0.00    60.43      75.96      70.65
treated3      60.43     0.00      80.03      71.72
untreated3    75.96    80.03       0.00      65.98
untreated4    70.65    71.72      65.98       0.00
```

```
## For sequential colour schemes
library(RColorBrewer)
seqColor = colorRampPalette(rev(brewer.pal(9, "Blues")))(16)

## loading the mixOmics package
library(mixOmics)

cim(sampleDists, col = seqColor, symkey = FALSE)
```



4.5 Differential expression analysis

Typically, a *edgeR* differential expression analysis is performed in three steps: count normalisation, dispersion estimation and differential expression test.

Exercise 4.13 *In edgeR, is recommended to remove genes with very low counts. Remove genes (rows) which have zero counts for all samples from the* `dge` *data object.*

Solution 4.13

```
dge = dge[rowSums(dge$counts) > 0, ]

head(dge$counts)
```

```
            treated2 treated3 untreated3 untreated4
FBgn0000003        0        1          0          0
FBgn0000008       88       70         76         70
FBgn0000015        0        0          1          2
FBgn0000017     3072     3334       3564       3150
FBgn0000018      299      308        245        310
FBgn0000024        7        5          3          3
```

Exercise 4.14 *Estimate normalization factors using the* `calcNormFactors()` *function.*

Solution 4.14

```
dge = calcNormFactors(dge)

dge$samples
```

```
             group lib.size norm.factors
treated2   treated  9571826       1.0081
treated3   treated 10343856       1.0179
untreated3 control  8358426       1.0041
untreated4 control  9841335       0.9706
```

Exercise 4.15 *Estimate common and tagwise dispersions using the functions* `estimateCommonDisp()` *and* `estimateTagwiseDisp()` *respectively.*

Solution 4.15

```
dge = estimateCommonDisp(dge)
dge = estimateTagwiseDisp(dge)
dge
```

```
An object of class "DGEList"
$counts
            treated2 treated3 untreated3 untreated4
FBgn0000003        0        1          0          0
FBgn0000008       88       70         76         70
FBgn0000015        0        0          1          2
FBgn0000017     3072     3334       3564       3150
FBgn0000018      299      308        245        310
11496 more rows ...

$samples
             group lib.size norm.factors
treated2   treated  9571826       1.0081
treated3   treated 10343856       1.0179
untreated3 control  8358426       1.0041
untreated4 control  9841335       0.9706

$sampleInfo
                 type number.of.lanes total.number.of.reads exon.counts condition
treated2   paired-end               2         12242535 (x2)    15620018   treated
treated3   paired-end               2         12443664 (x2)    12733865   treated
untreated3 paired-end               2         10542625 (x2)    10283129   control
untreated4 paired-end               2         12214974 (x2)    11653031   control

$common.dispersion
[1] 0.004761

$pseudo.counts
             treated2  treated3 untreated3 untreated4
FBgn0000003 0.000e+00 9.154e-01  2.776e-17  2.776e-17
FBgn0000008 8.671e+01 6.273e+01  8.564e+01  6.960e+01
FBgn0000015 2.776e-17 2.776e-17  1.167e+00  1.990e+00
FBgn0000017 3.025e+03 3.008e+03  4.032e+03  3.133e+03
FBgn0000018 2.944e+02 2.777e+02  2.777e+02  3.083e+02
11496 more rows ...

$pseudo.lib.size
[1] 9499789

$AveLogCPM
[1] -2.083  3.036 -1.793  8.440  4.940
11496 more elements ...

$prior.n
[1] 5

$tagwise.dispersion
[1] 7.439e-05 8.684e-03 7.439e-05 6.891e-03 3.601e-03
11496 more elements ...
```

Exercise 4.16 *Perform an exact test for the difference in expression between the two conditions "treated" and "control" using the exactTest() function and create a variable dgeTest from it.*

Solution 4.16

```
dgeTest = exactTest(dge)

dgeTest
```

```
An object of class "DGEExact"
$table
               logFC logCPM  PValue
FBgn0000003  2.25662 -2.083 1.00000
FBgn0000008 -0.05492  3.036 0.82398
FBgn0000015 -3.78099 -1.793 0.25001
FBgn0000017 -0.24803  8.440 0.04289
FBgn0000018 -0.03655  4.940 0.78928
11496 more rows ...

$comparison
[1] "control" "treated"

$genes
NULL
```

4.6 Independent filtering

By removing the weakly-expressed genes from the input to the FDR procedure, you can find more genes to be significant among those which are kept, and so improve the power of your test.

Exercise 4.17 *Load the HTSFilter package and perform independent filtering from results of the exact test. Use the HTSFilter() function on the dgeTest data object and create an object dgeTestFilt of the same class as dgeTest containing the data that pass the filter.*

Solution 4.17

```
library(HTSFilter)

dgeTestFilt = HTSFilter(dgeTest, DGEList = dge, plot = FALSE)$filteredData
```

4.7 Diagnostic plot for multiple testing

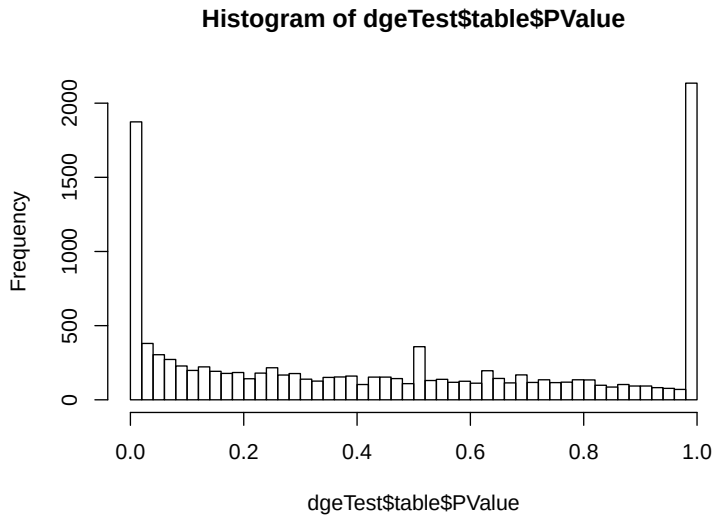
For diagnostic of multiple testing results it is instructive to look at the histogram of p-values.

Exercise 4.18 *Use the hist() function to plot a histogram from (unadjusted) p-values in the dgeTest data object.*

Solution 4.18

Use `breaks = 50` in `hist()` to generate this plot.

```
hist(dgeTest$table$PValue, breaks = 50)
```

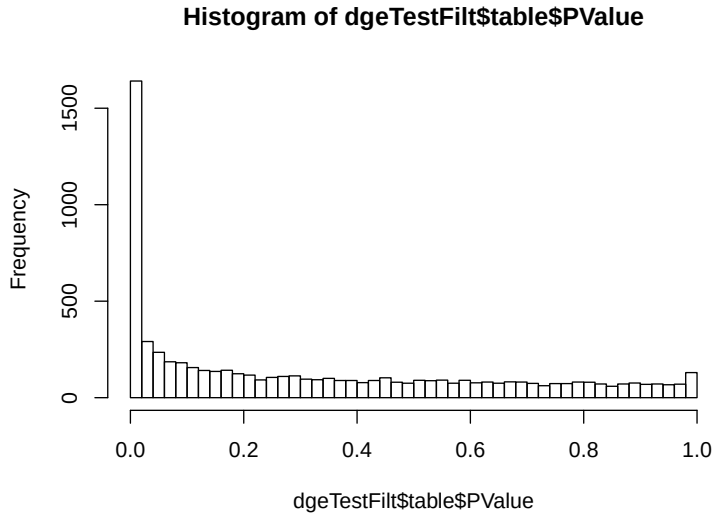


Exercise 4.19 *Plot a histogram from (unadjusted) p-values after independent filtering.*

Solution 4.19

Use `breaks = 50` in `hist()` to generate this plot.

```
hist(dgeTestFilt$table$PValue, breaks = 50)
```



4.8 Inspecting the results

Results tables are generated using the function `topTags()`, which extracts a table with $\log_2$ fold changes, p-values and adjusted p-values.

Exercise 4.20 *Use the `topTags()` function to extract a tabular summary of the differential expression statistics from test results before and after independent filtering (check the `n` argument). Create variables `resNoFilt` and `resFilt` from it. Visualize and inspect these variables. Are genes sorted? If yes, these are sorted by?*

Solution 4.20

```
resNoFilt = topTags(dgeTest, n = nrow(dgeTest))$table
```

```
resFilt = topTags(dgeTestFilt, n = nrow(dgeTestFilt))$table

head(resNoFilt)
```

```
             logFC logCPM     PValue        FDR
FBgn0039155 -4.378  5.588 4.293e-184 4.937e-180
FBgn0025111  2.943  7.159 2.758e-152 1.586e-148
FBgn0003360 -2.961  8.059 1.939e-151 7.432e-148
FBgn0039827 -4.129  4.281 5.594e-104 1.608e-100
FBgn0026562 -2.447 11.903 2.260e-102  5.198e-99
FBgn0035085 -2.499  5.542  1.241e-96  2.379e-93
```

```
head(resFilt)
```

```
             logFC logCPM     PValue        FDR
FBgn0039155 -4.378  5.588 4.293e-184 2.841e-180
FBgn0025111  2.943  7.159 2.758e-152 9.128e-149
FBgn0003360 -2.961  8.059 1.939e-151 4.277e-148
FBgn0039827 -4.129  4.281 5.594e-104 9.257e-101
FBgn0026562 -2.447 11.903 2.260e-102  2.992e-99
FBgn0035085 -2.499  5.542  1.241e-96  1.369e-93
```

By default, genes are sorted by p-value (`"PValue"`).

Exercise 4.21 *Compare the number of genes found at an FDR of 0.05 from the differential analysis before and after independent filtering.*

Solution 4.21

```
sum(resNoFilt$FDR < 0.05)
```

```
[1] 1347
```

```
sum(resFilt$FDR < 0.05)
```

```
[1] 1363
```

Continue with the independent filtered data in the `resFilt` data object.

The `FDR` column in the table `resFilt` contains the adjusted p-values for multiple testing with the Benjamini-Hochberg procedure (i.e. FDR). This is the information that we will use to decide whether the expression of a given gene differs significantly across conditions (e.g. we can arbitrarily decide that genes with an FDR < 0.01 are differentially expressed).

Exercise 4.22 *Consider all genes with an adjusted p-value below 5%=0.05 (`alpha = 0.05`) and subset the results table to these genes. Sort it by the $\log_2$-fold-change estimate to get the significant genes with the strongest down-regulation.*

Solution 4.22

```
sigDownReg = resFilt[resFilt$FDR < alpha, ]
sigDownReg = sigDownReg[order(sigDownReg$logFC), ]

head(sigDownReg)
```

```
             logFC logCPM     PValue        FDR
FBgn0085359 -5.153  1.966  2.843e-26  3.764e-24
FBgn0039155 -4.378  5.588 4.293e-184 2.841e-180
FBgn0024288 -4.208  2.161  1.359e-33  2.645e-31
FBgn0039827 -4.129  4.281 5.594e-104 9.257e-101
FBgn0034434 -3.824  3.107  1.732e-52  7.644e-50
FBgn0034736 -3.482  4.060  5.794e-68  3.196e-65
```

Exercise 4.23 *Repet the Exercise 4.22 for the strongest up-regulated genes.*

Solution 4.23

```
sigUpReg = resFilt[resFilt$FDR < alpha, ]
sigUpReg = sigUpReg[order(sigUpReg$logFC, decreasing = TRUE), ]

head(sigUpReg)
```

```
            logFC logCPM     PValue        FDR
FBgn0033764 3.268  2.612  3.224e-29  4.743e-27
FBgn0035189 2.973  4.427  7.154e-48  2.492e-45
FBgn0025111 2.943  7.159 2.758e-152 9.128e-149
FBgn0037290 2.935  2.523  1.192e-25  1.461e-23
FBgn0038198 2.670  2.587  4.163e-19  3.167e-17
FBgn0000071 2.565  5.034  1.711e-78  1.416e-75
```

Exercise 4.24 *Create persistent storage of results. Save the result tables as a csv (comma-separated values) file using the `write.csv()` function (alternative formats are possible).*

Solution 4.24

```
write.csv(sigDownReg, file = "sigDownReg.csv")

write.csv(sigUpReg, file = "sigUpReg.csv")
```

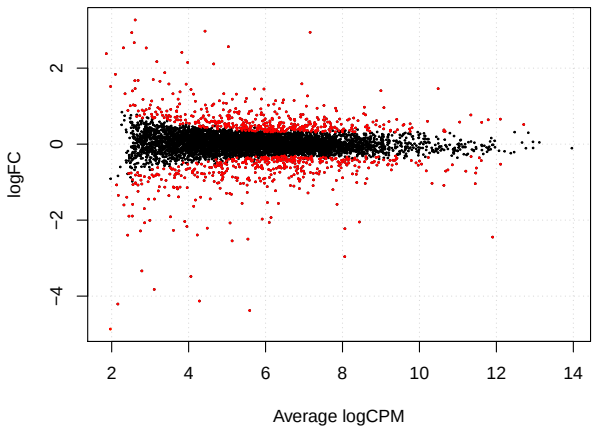
4.9 Interpreting the DE analysis results

MA-plot

Exercise 4.25 *Create a MA-plot using the `plotSmear()` showing the genes selected as differentially expressed with a 1% false discovery rate.*

Solution 4.25

```
de.genes = rownames(resFilt[resFilt$FDR < 0.01, ])

plotSmear(dgeTestFilt, de.tags = de.genes)
```



Volcano plot

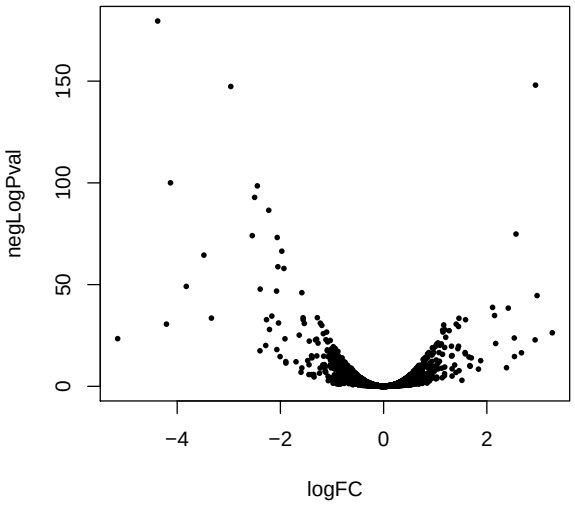
Exercise 4.26 *Create a volcano-plot from the `res` data object. First construct a table containing the $\log_2$ fold change and the negative $\log_{10}$-transformed p-values, then generate the volcano plot using the standard `plot()` command.* Hint: *the negative $\log_{10}$-transformed value of x is `-log10(x)`.*

Solution 4.26

```
tab = data.frame(logFC = resFilt$logFC, negLogPval = -log10(resFilt$FDR))

## First few rows of the table containing the log£_2£ fold change and the negative
## log10-transformed p-values
head(tab)
```

```
   logFC negLogPval
1 -4.378     179.55
2  2.943     148.04
3 -2.961     147.37
4 -4.129     100.03
5 -2.447      98.52
6 -2.499      92.86
```

```
plot(tab, pch = 16, cex = 0.6)
```



Gene clustering

To draw a CIM (or heatmap) of individual RNA-seq samples, *edgeR* suggest using moderated log-counts-per-million. This can be calculated by the `cpm()` function with positive values for `prior.count`, for example

```
y = cpm(dge, prior.count = 1, log = TRUE)
```

where `dge` is the normalized `DGEList` object. This produces a matrix of $\log_2$ counts-per-million (`logCPM`), with undefined values avoided and the poorly defined log-fold-changes for low counts shrunk towards zero. Larger values for `prior.count` produce more shrinkage.

Exercise 4.27 *Transform the normalized counts from `dge` data object using the `cpm()` function.*

Solution 4.27

```
cpm.mat = cpm(dge, prior.count = 1, log = TRUE)

head(cpm.mat)
```

```
            treated2 treated3 untreated3 untreated4
FBgn0000003  -3.2526  -2.3226     -3.253     -3.253
FBgn0000008   3.2056   2.7556      3.195      2.894
FBgn0000015  -3.2526  -3.2526     -2.158     -1.670
FBgn0000017   8.3151   8.3073      8.730      8.366
FBgn0000018   4.9585   4.8757      4.873      5.025
FBgn0000024  -0.2681  -0.7863     -1.113     -1.255
```

Since the clustering is only relevant for genes that actually are differentially expressed, carry it out only for a gene subset of most highly differential expression.

Exercise 4.28 *Select those genes that have adjusted p-values below 0.01 and absolute $\log_2$-fold-change above 1.5 from the trasformed data, and use the function `cim()` to produce a CIM.*

Solution 4.28

```
deg.idx = (abs(resFilt$logFC) > 1.5 & resFilt$FDR < 0.01)
deg = rownames(resFilt)[deg.idx]

cpm.mat = cpm.mat[deg, ]
```

Transformed values for the first ten selected genes.

```
head(cpm.mat)
```

```
            treated2 treated3 untreated3 untreated4
FBgn0039155   2.0155    2.335      6.466      6.608
FBgn0025111   7.9476    7.996      5.059      5.006
FBgn0003360   5.9800    5.878      8.802      8.970
FBgn0039827   0.6377    1.516      5.252      5.212
FBgn0026562  10.1798   10.247     12.544     12.769
FBgn0035085   3.8172    3.843      6.279      6.363
```

Use `col = seqColor` for sequential colour schemes in the `cim()` function.

```
## For sequential colour schemes
library(RColorBrewer)
seqColor = colorRampPalette(brewer.pal(9, "Blues"))(255)

cim(t(cpm.mat), dendrogram = "column", col = seqColor, symkey = FALSE)
```