RNA-Seq: differential expression using DESeq

Denis Puthier

12 November 2014

Contents

The Pasilla dataset

This dataset is available from the Pasilla Bioconductor library and is derived from the work from Brooks et al. (Conservation of an RNA regulatory map between Drosophila and mammals. Genome Research, 2010).

Alternative splicing is generally controlled by proteins that bind directly to regulatory sequence elements and either activate or repress splicing of adjacent splice sites in a target pre-mRNA. Here, the authors have combined RNAi and mRNA-seq to identify exons that are regulated by Pasilla (PS), the Drosophila melanogaster ortholog of the mammalian RNA-binding proteins NOVA1 and NOVA2.

Loading the dataset

To get the dataset, you need to install the pasilla library from Bioconductor then load this library.

```
## Install the library if needed then load it
if(!require("pasilla")){
  source("http://bioconductor.org/biocLite.R")
  biocLite("pasilla")
}
```

```
## Loading required package: pasilla
```

```
library("pasilla")
```

To get the path to the tabulated file containing count table use the command below. Here, the **system.file()** function is simply used to get the path to the directory containing the count table. The "pasilla_gene_counts.tsv" file contains counts for each gene (row) in each sample (column).

```
datafile <-  system.file( "extdata/pasilla_gene_counts.tsv", package="pasilla" )
```

Load the file using the read.table function.

```
## Read the data table
count.table<-  read.table( datafile, header=TRUE, row.names=1, quote="",
                           comment.char="" )
head(count.table)
```

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

Some genes were not detected in any of the sample. Delete them from the **count.table** data.frame.

View solution| Hide solution

Solution

```
## Some genes were not detected at all in these samples. We will discard them.
count.table <- count.table[rowSums(count.table) > 0,]
```

Phenotypic data

The dataset contains RNA-Seq count data for RNAi treated or S2-DRSC untreated cells (late embryonic stage). Some results were obtained through single-end sequencing whereas others were obtained using paired-end sequencing. We will store these informations in two vectors (cond.type and lib.type).

```
cond.type <-  c( "untreated", "untreated", "untreated","untreated", "treated", "treated", "treated" )
lib.type   <-  c( "single-end", "single-end", "paired-end", "paired-end",
                  "single-end", "paired-end", "paired-end" )
```

Next, we will extract a subset of the data containing only paired-end samples.

```r
## Select only Paired-end datasets
isPaired <-  lib.type == "paired-end"
show(isPaired)
```

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

```r
count.table <-  count.table[ , isPaired ]  ## Select only the paired samples
head(count.table)
```

```
##             untreated3 untreated4 treated2 treated3
## FBgn0000003          0          0        0        1
## FBgn0000008         76         70       88       70
## FBgn0000014          0          0        0        0
## FBgn0000015          1          2        0        0
## FBgn0000017       3564       3150     3072     3334
## FBgn0000018        245        310      299      308
```

```r
cond.type <-  cond.type[isPaired]
show(cond.type)
```

```
## [1] "untreated" "untreated" "treated"   "treated"
```

Descriptive statistics

Before going further in the analysis, we will compute some descriptive statistics on the dataset.

- What are the dimensions of the **count.table object**.
- Use the **summary** fonction with **count.table** as argument. What kind of information are displayed ?
- Draw the distribution of the **count.table** data. Is that really informative ? Why ?
- Draw the distribution of the **count.table** data in logarithme base 2 (add a pseudo-count to avoid logarithmic transformation of zero values).
- Draw the boxplot of each column of **count.table** using the **boxplot()** function.
- Use the **plotDensity()** from the **affy** library (BioC) to plot density estimates of each column of the **count.table** data.frame.
- Use the **pairs()** to produce a matrix of scatterplot from the **count.table** object.

View solution Hide solution

```r
## Dimensions
ncol(count.table)
```

Solution

```
## [1] 4
```

```r
nrow(count.table)
```

```
## [1] 12359
```

```r
dim(count.table)
```

```
## [1] 12359     4
```

```r
## Min, Max, median...
summary(count.table)
```

```
##    untreated3        untreated4          treated2           treated3
##  Min.   :     0.0   Min.   :     0.0   Min.   :     0.0   Min.   :     0
##  1st Qu.:     2.0   1st Qu.:     2.0   1st Qu.:     2.0   1st Qu.:     2
##  Median :    67.0   Median :    80.0   Median :    82.0   Median :    91
##  Mean   :   676.3   Mean   :   796.3   Mean   :   774.5   Mean   :   837
##  3rd Qu.:   466.0   3rd Qu.:   546.0   3rd Qu.:   552.0   3rd Qu.:   599
##  Max.   :131242.0   Max.   :167116.0   Max.   :146390.0   Max.   :164148
```

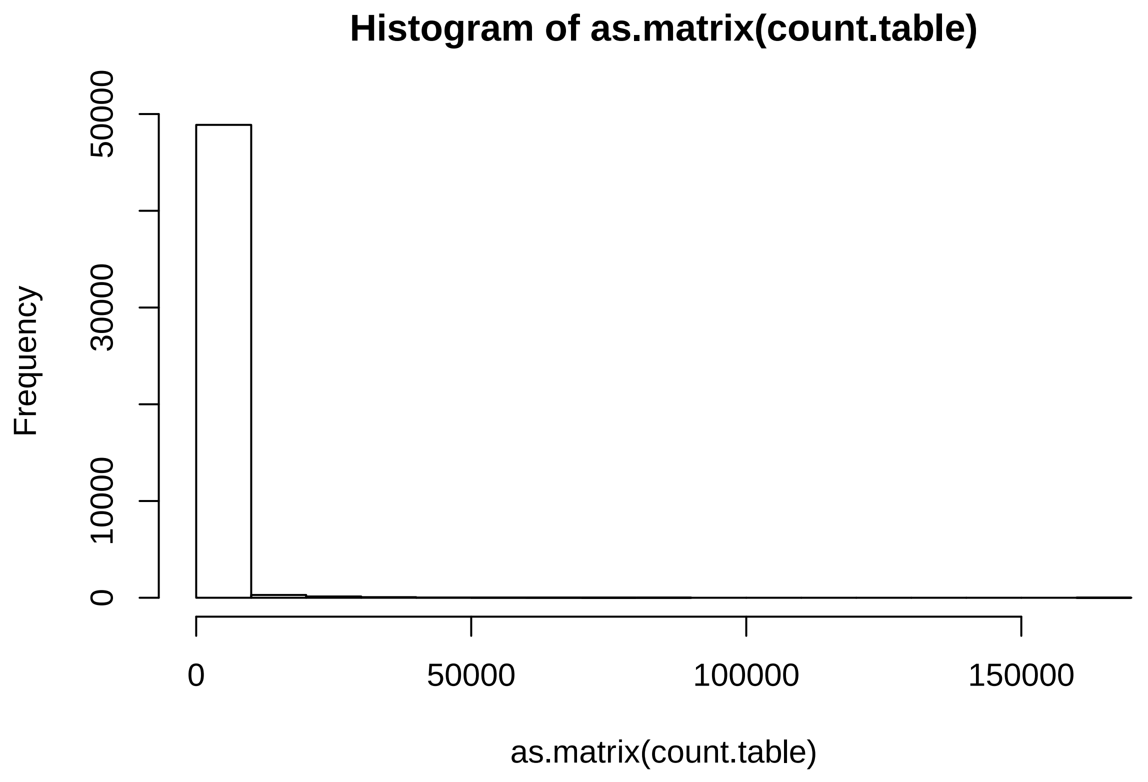
```r
## The summary only displays a few milestone values
## (mean, median, quartiles).
## In order to get a better intuition of the data,
## we can draw an histogram of all values.
col <- c("green","orange")[as.numeric(cond.type)] ## Define colors for subsequent plots
```

```
## Warning: NAs introduced by coercion
```

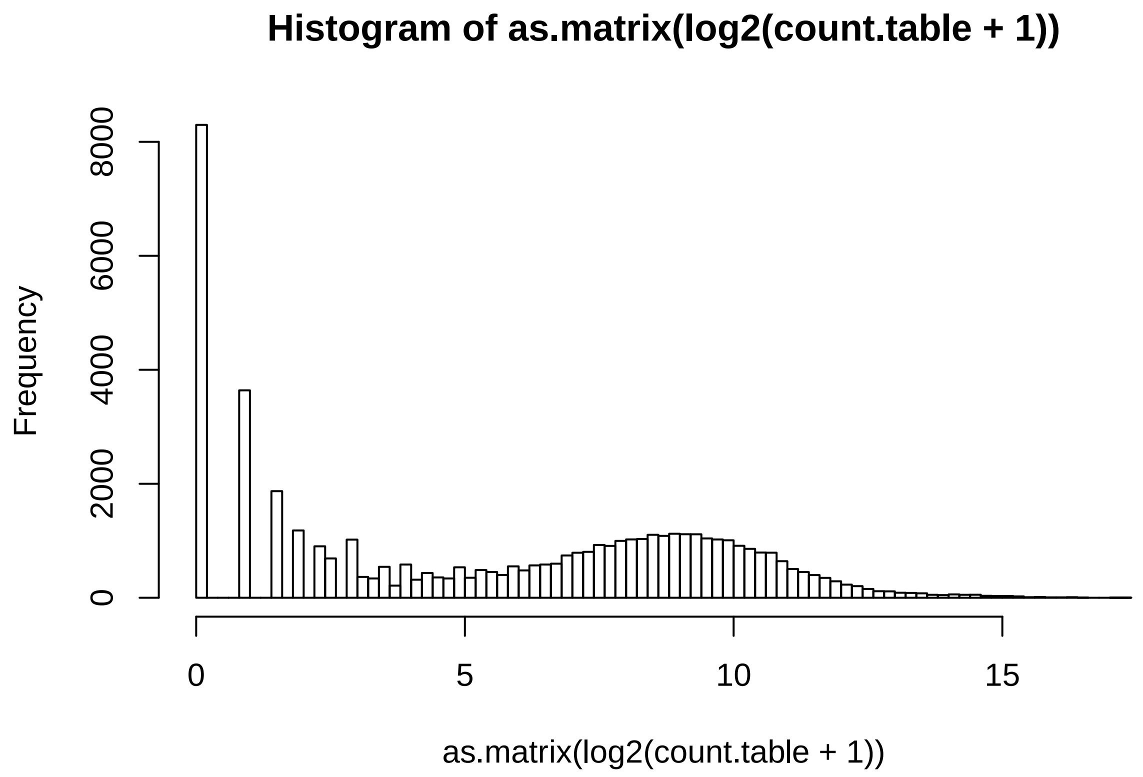
```r
show(col)
```

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

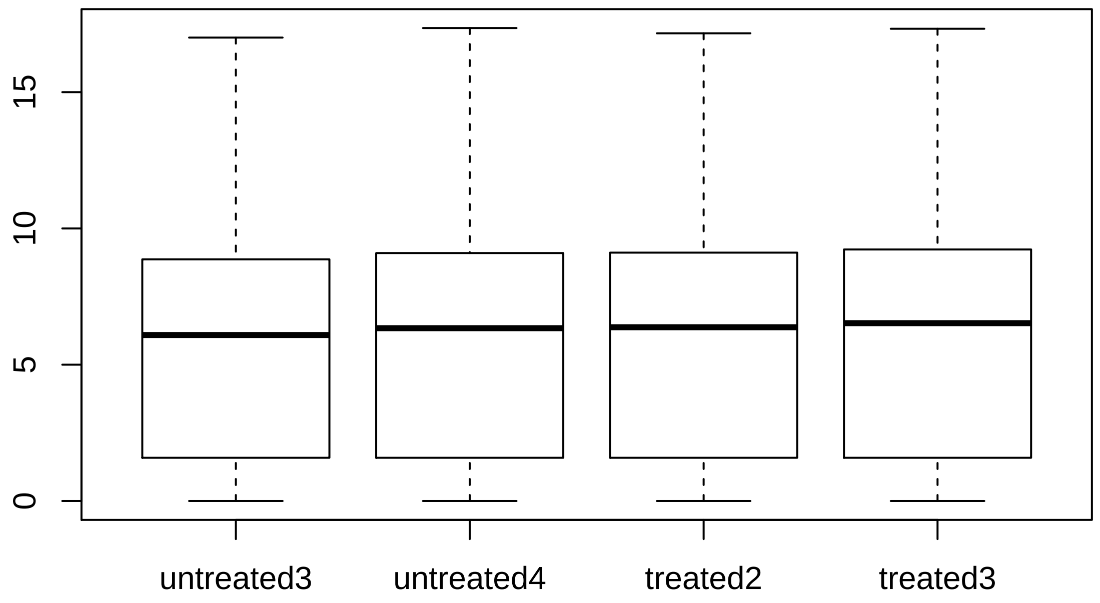
```r
hist(as.matrix(count.table))
```



```
## The histogram is not very informative so far, apparently due
## to the presence of a few very high count values, that impose
## a very large scale onthe X axis. We can use a logarithmic transformation
## to improve the readability. Note that we will add pseudo count to avoid
## problems with "zero" counts observed for some genes in some samples.

## Data distribution in log scale.
## Note that a pseudo count is added before logarithmic transformation
hist(as.matrix(log2(count.table + 1)),  breaks=100)
```



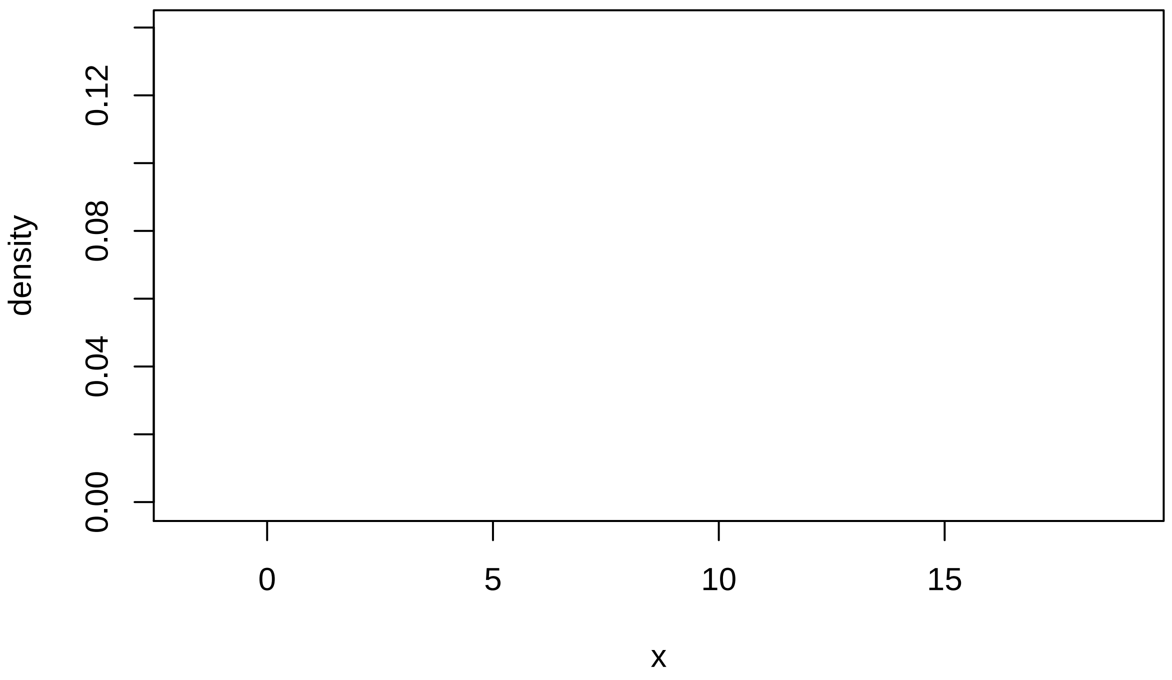
```
## Boxplot
boxplot(log2(count.table + 1), col=col)
```



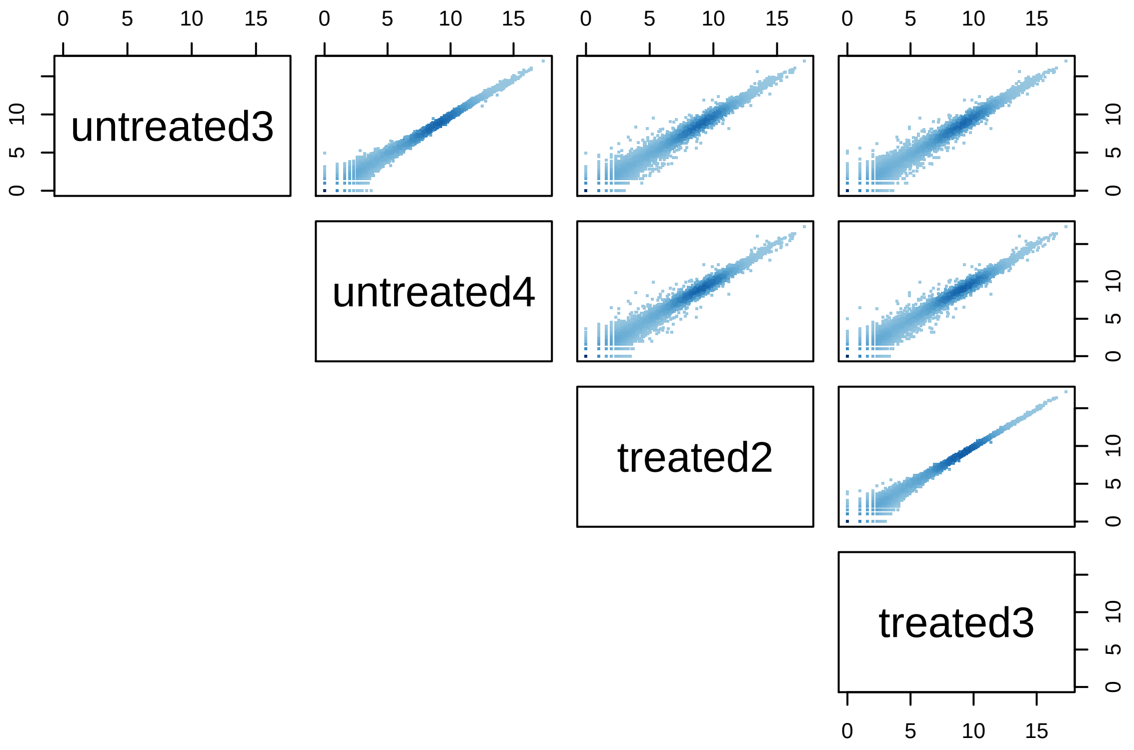
```
## Density
## We will require one function from the affy package
if(!require("affy")){
  source("http://bioconductor.org/biocLite.R")
  biocLite("affy")

}
```

```
## Loading required package: affy
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##     xtabs
##
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, duplicated, eval, evalq, Filter, Find, get,
##     intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rep.int, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unlist
##
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
library(affy)
plotDensity(log2(count.table + 1), lty=1, col=col, lwd=2)
```



```
## Let's have a look at the scatter plots using the pairs() function
plotFun <- function(x,y){ dns <- densCols(x,y); points(x,y, col=dns, pch=".") }
pairs(log2(count.table + 1), panel=plotFun, lower.panel = NULL)
```



The command *pairs()* draws a scatter plot for each pair of columns of the input dataset. The plot shows a fairly good reproducibility between samples of the same type (treated or untreated, respectively): all points are aligned along te diagonal, with a relatively wider dispersion at the bottom, corresponding to small number fluctuations.

In contrast, on all the plots comparing an untreated and a treated sample, we can see some points (genes) discarding from the diagonal, on the side of either treated (up-regulated) or untreated (down-regulated) sample.

Creating a CountDataSet dataset

Now that we have all the required material, we will create a **CountDataSet** object (named **cds**) that will be used by DESeq to perform differential expression call. The CountDataSet has some important useful accessor methods (**counts**, **conditions**, **estimateSizeFactors**, **sizeFactors**, **estimateDispersions** and **nbinomTest**) that will be used later in this tutorial.

- Install and load the DESeq library (it should be installed from BioC).
- Create an object of class **CountDataSet** using the **newCountDataSet()** function and get some help about this object.

View solution Hide solution

```
## Install the library if needed then load it
if(!require("DESeq")){
  source("http://bioconductor.org/biocLite.R")
  biocLite("DESeq")
}
```

```
## Loading required package: DESeq
## Loading required package: locfit
## locfit 1.5-9.1    2013-03-22
## Loading required package: lattice
##     Welcome to 'DESeq'. For improved performance, usability and
##     functionality, please consider migrating to 'DESeq2'.
```

```
library("DESeq")

## Use the newCountDataSet function to create a CountDataSet object
cds <-  newCountDataSet( count.table, cond.type )
show(cds)
```

```
## CountDataSet (storageMode: environment)
## assayData: 12359 features, 4 samples
##   element names: counts
## protocolData: none
## phenoData
##   sampleNames: untreated3 untreated4 treated2 treated3
##   varLabels: sizeFactor condition
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```

```r
## What kind of object is it ?
is(cds)
```

```
## [1] "CountDataSet"     "eSet"             "VersionedBiobase"
## [4] "Versioned"
```

```r
isS4(cds)
```

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

```r
## What does it contain ?
# The list of slot names
# Not really meaningful here...
slotNames(cds)
```

```
##  [1] "fitInfo"                 "dispTable"
##  [3] "multivariateConditions" "assayData"
##  [5] "phenoData"               "featureData"
##  [7] "experimentData"          "annotation"
##  [9] "protocolData"            ".__classVersion__"
```

```r
## Get some help about the "CountDataSet" class.
## NOT RUN
#?"DESeqDataSet-class"
```

Normalization

The normalization procedure (RLE) is implemented through the **estimateSizeFactors** function.

How is it computed ?

From DESeq help files: Given a matrix or data frame of count data, this function estimates the size factors as follows: Each column is divided by the **geometric means** of the rows. The **median** (or, if requested, another location estimator) **of these ratios** (skipping the genes with a geometric mean of zero) is used as the size factor for this column.

- Create a new object **cds.norm** that will contain normalized data. The method associated with normalization for the CountDataSet class is **estimateSizeFactors()**.
- Implement a function that will compute the size factors for each sample.
- Now, check that the results obtained with your function are the same as those produced by DESeq (you can get the size factors from a CountDataSet object using the **sizeFactors()** function.
- Check the distribution before and after normalization using the **boxplot()** and **plotDensity()** functions. Do you see some differences. Does the count look more balanced across samples ?

Solution| Hide solution

```
### Let's implement such a function
### cds is a countDataset

estimSf <- function (cds){
    # Get the count matrix
    cts <- counts(cds)

    # Compute the geometric mean
    geomMean <- function(x) prod(x)^(1/length(x))

    # Compute the geometric mean over the line
    gm.mean  <-  apply(cts, 1, geomMean)

    # Zero values are set to NA (avoid subsequentcdsdivision by 0)
    gm.mean[gm.mean == 0] <- NA

    # Divide each line by its corresponding geometric mean
    # sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)
    # MARGIN: 1 or 2 (line or columns)
    # STATS: a vector of length nrow(x) or ncol(x), depending on MARGIN
    # FUN: the function to be applied
    cts <- sweep(cts, 1, gm.mean, FUN="/")

    # Compute the median over the columns
    med <- apply(cts, 2, median, na.rm=TRUE)

    # Return the scaling factor
    return(med)

}


## Normalizing with using the method for an object of class"CountDataSet"
cds.norm <-  estimateSizeFactors(cds)
sizeFactors(cds.norm)
```

Solution

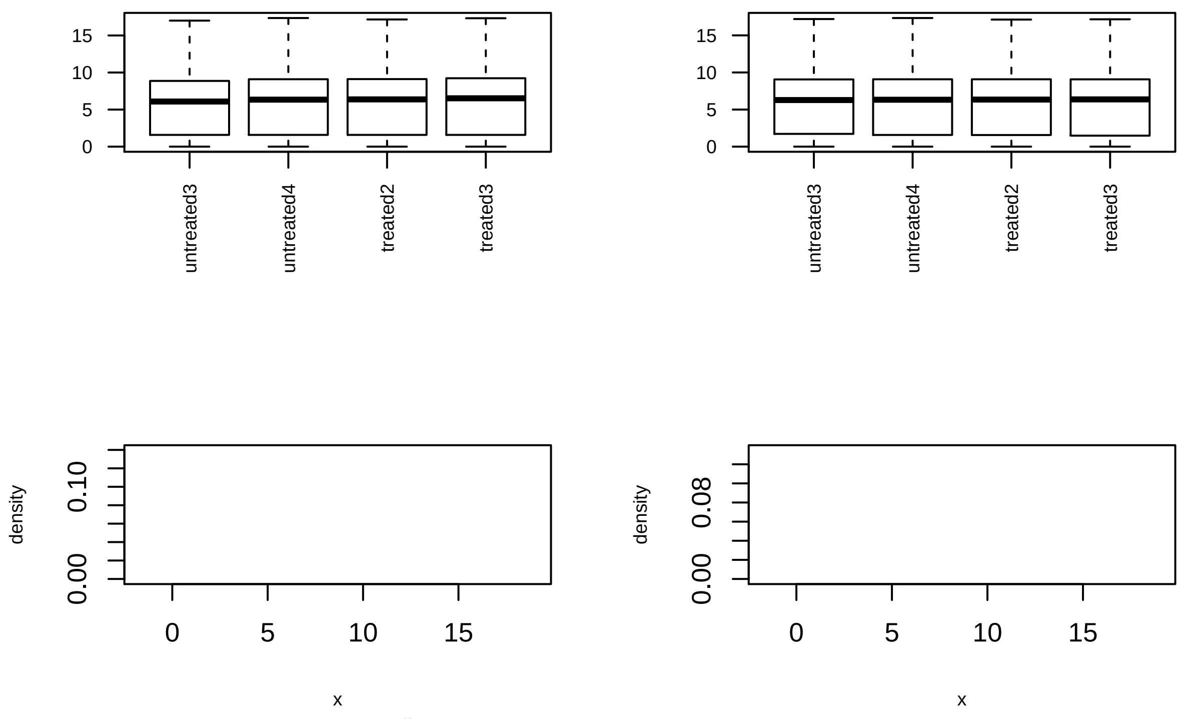
```
## untreated3 untreated4   treated2   treated3
##  0.8730966  1.0106112  1.0224517  1.1145888
```

```
## Now get the scaling factor with our homemade function.cds.norm
estimSf(cds)
```

```
## untreated3 untreated4   treated2   treated3
##  0.8730966  1.0106112  1.0224517  1.1145888
```

```
round(estimSf(cds),6) == round(sizeFactors(cds.norm), 6)
```

```
## untreated3 untreated4   treated2   treated3
##       TRUE       TRUE       TRUE       TRUE
## Checking the normalization
par(mfrow=c(2,2),cex.lab=0.7)
boxplot(log2(counts(cds.norm)+1),  col=col, cex.axis=0.7, las=2)
boxplot(log2(counts(cds.norm, normalized=TRUE)+1),  col=col,, cex.axis=0.7, las=2)
plotDensity(log2(counts(cds.norm)+1),  col=col, cex.lab=0.7)
plotDensity(log2(counts(cds.norm, normalized=TRUE)+1),  col=col, cex.lab=0.7)
```



Beware: the R function *plotDensity()* does not display the actual distribution of your values, but a polynomial fit. The representation thus generally looks smoother than the actual data. It is important to realize that, in some particular cases, the fit can lead to extrapolated values which can be misleading.

Differential expression

In this section we will search for genes whose expression is affected by the si-RNA treatment.

Some intuition about the problematic

Let say that we would produce a lot of RNA-Seq experiments from the same samples (technical replicates). For each gene g the measured read counts would be expected to vary rather slighlty around the expected mean and would be probably well-modeled using a poisson distribution. However, when working with biological replicates more variations are intrinsically expected. Indeed, due to sample purity, cell-synchronization issues or reponses to environment (e.g. heat-shock) the measured expression values for each genes are expected to fluctuate more importantly. The poisson distribution has only one parameter λ and the mean and variance of the distribution are both equal to λ. Thus in most cases, the poisson distribution is not expected to fit very well with the count distribution since some over-dispersion (greater variability) due to biological noise is expected. As a consequence, when working with RNA-Seq data, many of the current approaches for differential expression call rely on the negative binomial distribution (note that this hold true also for other -Seq approaches, e.g. ChIP-Seq with replicates).

What is the negative binomial?

The negative binomial distribution is a discrete distribution that can be used to model over-dispersed data (in this case this overdispersion is relative to the poisson model). There are two ways to parametrize the negative binomial distribution. The negative binomial distribution is a discrete distribution that can be used to model over-dispersed data (in this case this overdispersion is relative to the poisson model). There are two ways to parametrize the negative binomial distribution.

The probability of x failures before n success First, given a Bernouilli trial with a probability p of success, the **negative binomial** distribution describes the probability of observing x failures before a target number of successes n is reached. In this case the parameters of the distribution will thus be p, n (in **dnbinom()** function of R, n and p are denoted by arguments **size** and **prob** respectively).

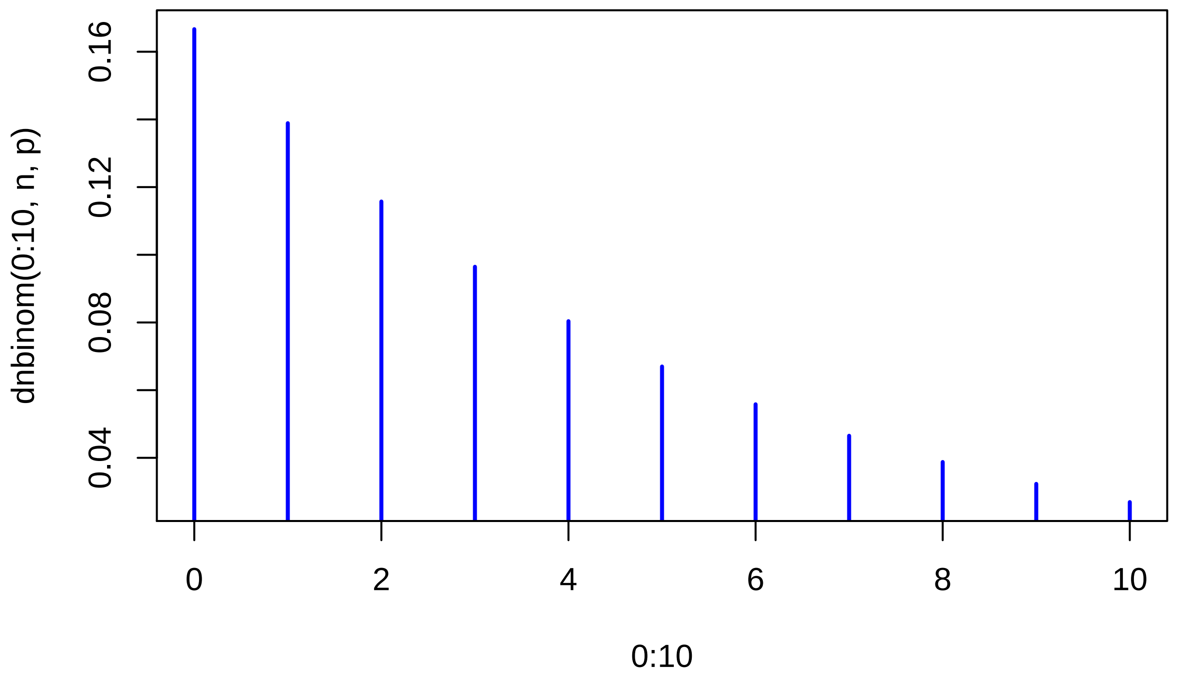
$$P_{NegBin}(x; n, p) = \binom{x+n-1}{x} \cdot p^n \cdot (1-p)^x = C^x_{x+n-1} \cdot p^n \cdot (1-p)^x$$

In this formula, p^n denotes the probability to observe n successes, $(1-p)^x$ the probability of x failures, and the binomial coefficient C^x_{x+n-1} indicates the number of possible ways to dispose x failures among the $x+n-1$ trials that precede the last one (the problem statement imposes for the last trial to be a success).

The negative binomial distribution has expected value $n\frac{q}{p}$ and variance $n\frac{q}{p^2}$. Some examples of using this distribution in R are provided below.

Particular case: when $n = 1$ the negative binomial corresponds to the the **geometric distribution**, which models the probability distribution to observe the first success after x failures: $P_{NegBin}(x; 1, p) = P_{geom}(x; p) = p \cdot (1-p)^x$.

```r
par(mfrow=c(1,1))
## Some intuition about the negative binomial parametrized using n and p.
## The simple case, one success (see geometric distribution)
# Let's have a look at the density
p <- 1/6 # the probability of success
n <- 1   # target for number of successful trials

# The density function
plot(0:10, dnbinom(0:10, n, p), type="h", col="blue", lwd=2)
```



```
# the probability of zero failure before one success.
# i.e the probability of success
dnbinom(0, n , p)
```

```
## [1] 0.1666667
```

```
## i.e the probability of at most 5 failure before one success.
sum(dnbinom(0:5, n , p)) # == pnbinom(5, 1, p)
```

```
## [1] 0.665102
```

```
## The probability of at most 10 failures before one sucess
sum(dnbinom(0:10, n , p)) # == pnbinom(10, 1, p)
```

```
## [1] 0.865412
```

```
## The probability to have more than 10 failures before one sucess
1-sum(dnbinom(0:10, n , p)) # == 1 - pnbinom(10, 1, p)
```

```
## [1] 0.134588
```

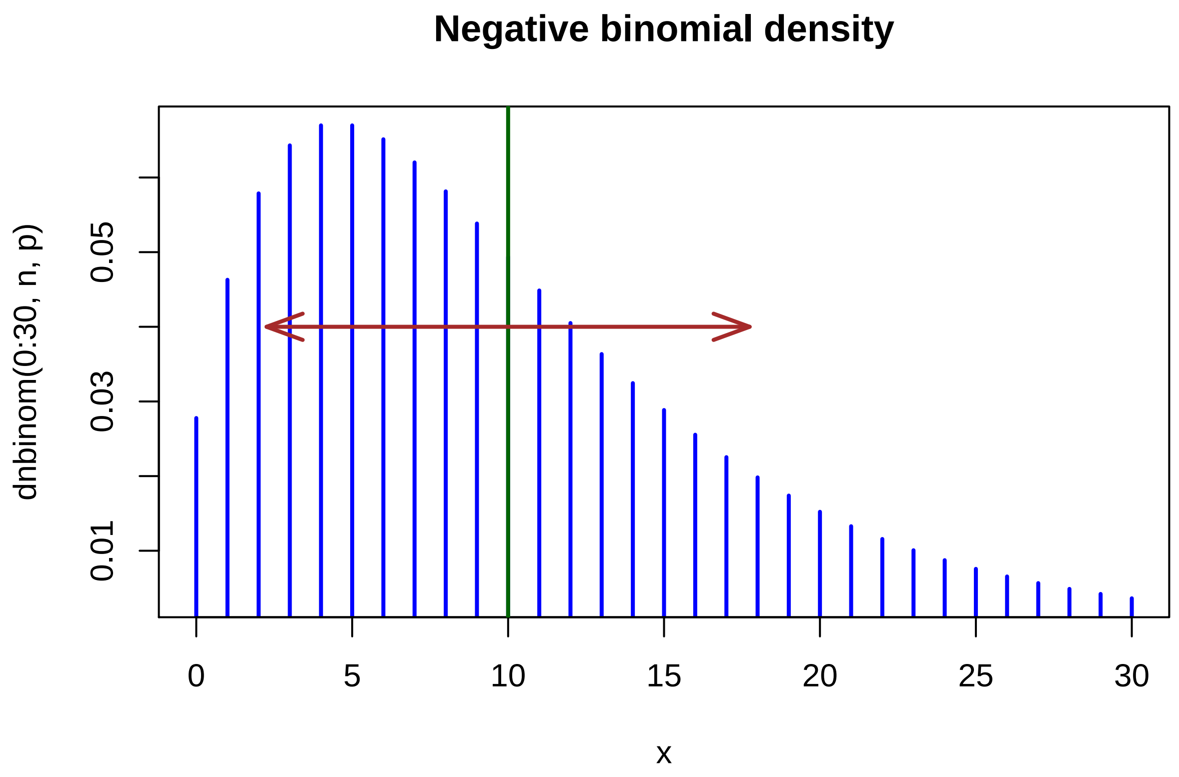
```
## With two successes
## The probability of x failure before two success (e.g. two six)
n <- 2
plot(0:30, dnbinom(0:30, n, p), type="h", col="blue", lwd=2,
     main="Negative binomial density", xlab="x")

# Expected value
q <- 1-p
(ev <- n*q/p)
```

```
## [1] 10
```

```
abline(v=ev, col="darkgreen", lwd=2)

# Variance
(v <- n*q/p^2)
```

```
## [1] 60
```

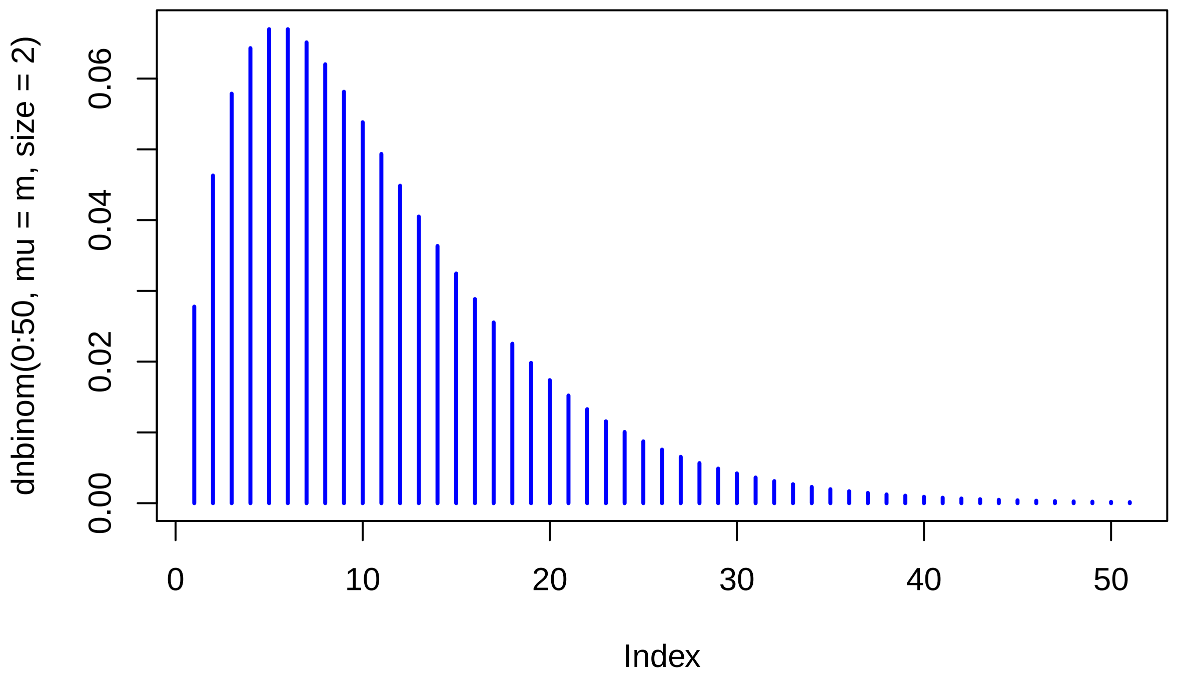
```
arrows(x0=ev-sqrt(v), y0 = 0.04, x1=ev+sqrt(v), y1=0.04, col="brown",lwd=2, code=3, , length=0.2, angle=
```



Using mean and dispersion The second way of parametrizing the distribution is using the mean value m and the dispersion parameter r (in **dnbinom()** function of R, m and r are denoted by arguments **mu** and **size** respectively). The variance of the distribution can then be computed as $m + m^2/r$.

This second parameterization is used in DESeq as it will use it in an attempt to model the variance for each gene given a particular mean.

```
## The density function with m=10 and r=2
m <- 10
r <- 2
plot(dnbinom(0:50, mu=m, size=2), type="h", col="blue", lwd=2)
```



```
## Note that the same results can be obtained using both parametrizations
n <- r <- 10
p <- 1/6
q <- 1-p
ev <- n*q/p

all(dnbinom(0:100, mu=ev, size=n) == dnbinom(0:100, size=n, prob=p))
```

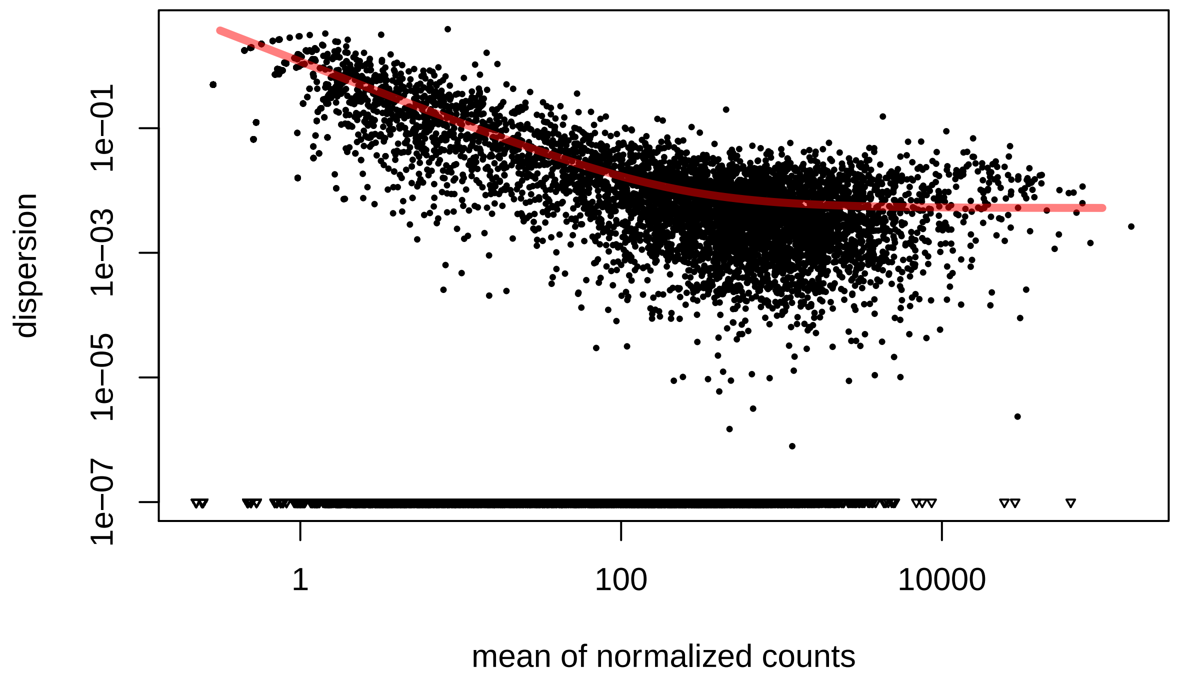
```
## [1] TRUE
```

Modelling read counts through a negative binomial

To perform diffential expression call DESeq will assume that, for each gene, the read counts are generated by a negative binomial distribution. One problem here will be to estimate, for each gene, the two parameters of the negative binomial distribution: mean and dispersion.

- The mean will be estimated from the observed normalized counts in both conditions.
- The first step will be to compute a gene-wise dispersion. When the number of available samples is insufficient to obtain a reliable estimator of the variance for each gene, DESeq will apply a shrinkage strategy, which assumes that counts produced by genes with similar expression level (counts) have similar variance (note that this is a strong assumption). DESeq will regress the gene-wise dispersion onto the means of the normalized counts to obtain an estimate of the dispersion that will be subsequently used to build the binomial model for each gene.

```
## Performing estimation of dispersion parameter
 cds.norm <- estimateDispersions(cds.norm)

## A diagnostic plot which
## shows the mean of normalized counts (x axis)
## and dispersion estimate for each genes
plotDispEsts(cds.norm)
```



Performing differential expression call

Using the nbinomTest() function

Now that a negative binomial model has been fitted for each gene, the **nbinomTest** can be used to test for differential expression. The output is a data.frame which contains nominal p-values, as well as FDR values (correction for multiple tests computed with the Benjamini–Hochberg procedure).

```
## Perform differential expression call
res <- nbinomTest(cds.norm, "untreated", "treated")

## What is the object returned by nbinomTest()
is(res) # a data.frame
```

```
## [1] "data.frame"       "list"             "oldClass"
## [4] "data.frameOrNULL" "vector"
```

```
head(res)
```

```
##            id     baseMean   baseMeanA    baseMeanB foldChange
## 1 FBgn0000003    0.2242980    0.000000    0.4485959        Inf
## 2 FBgn0000008   76.2956431   78.155755   74.4355310  0.9523999
## 3 FBgn0000014    0.0000000    0.000000    0.0000000        NaN
## 4 FBgn0000015    0.7810873    1.562175    0.0000000  0.0000000
## 5 FBgn0000017 3298.6821506 3599.474078 2997.8902236  0.8328690
## 6 FBgn0000018  289.0312286  293.677741  284.3847165  0.9683564
##   log2FoldChange      pval      padj
## 1            Inf 1.0000000 1.0000000
## 2    -0.07036067 0.8354725 1.0000000
## 3            NaN        NA        NA
## 4           -Inf 0.4160556 1.0000000
## 5    -0.26383857 0.2414208 0.8811746
## 6    -0.04638999 0.7572819 1.0000000
```

```
## The column names of the data.frame
## Note the column padj
## contains FDR values (computed Benjamini-Hochberg procedure)
colnames(res)
```

```
## [1] "id"              "baseMean"       "baseMeanA"      "baseMeanB"
## [5] "foldChange"      "log2FoldChange" "pval"           "padj"
```

```
## Order the table by decreasing p-valuer
res <- res[order(res$padj),]
head(res)
```

```
##                 id  baseMean baseMeanA  baseMeanB foldChange log2FoldChange
## 8817 FBgn0039155  463.4369  884.9640   41.90977  0.0473576      -4.400260
## 2132 FBgn0025111 1340.2282  311.1697 2369.28680  7.6141316       2.928680
## 570  FBgn0003360 2544.2512 4513.9457  574.55683  0.1272848      -2.973868
## 2889 FBgn0029167 2551.3113 4210.9571  891.66551  0.2117489      -2.239574
## 9234 FBgn0039827  188.5927  357.3299   19.85557  0.0555665      -4.169641
## 6265 FBgn0035085  447.2485  761.1898  133.30718  0.1751300      -2.513502
##                 pval          padj
## 8817 1.641210e-124 1.887556e-120
## 2132 3.496915e-107 2.010901e-103
## 570   1.552884e-99  5.953239e-96
## 2889  4.346335e-78  1.249680e-74
## 9234  1.189136e-65  2.735251e-62
## 6265  3.145997e-56  6.030352e-53
```

Looking at the results with a MA plot

One popular diagram in dna chip analysis is the M versus A plot (MA plot) between two conditions a and b. In this representation :

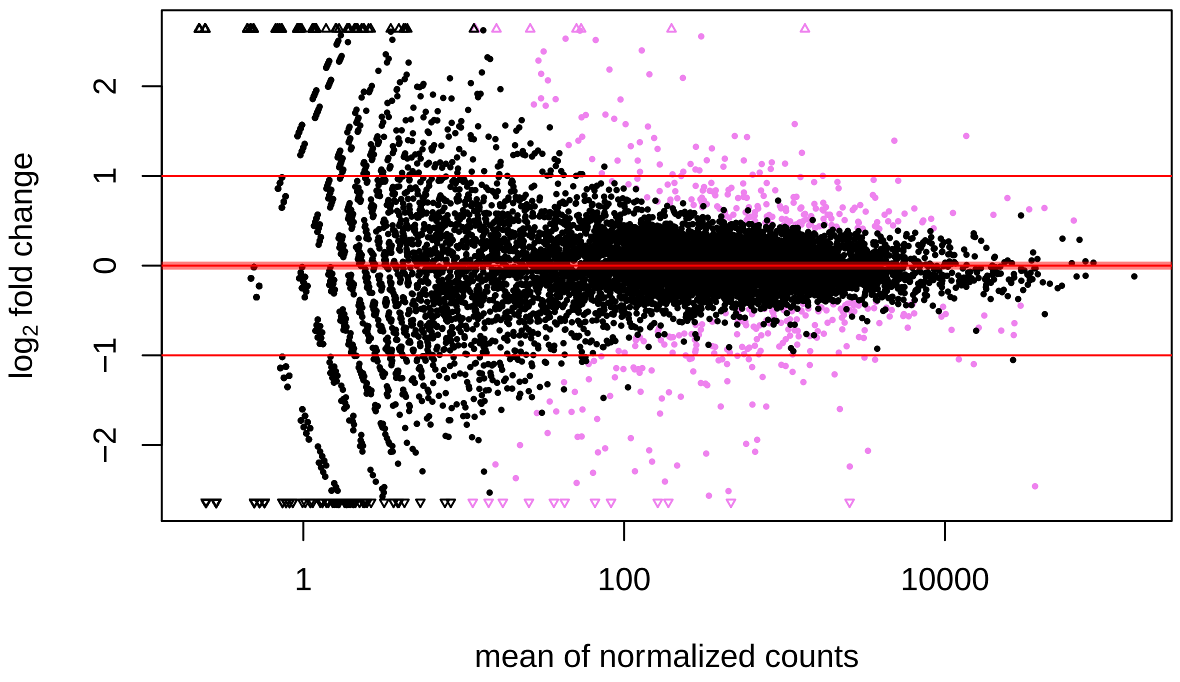
- M (Minus) is the log ratio of counts calculated for any gene.

$$M_g = log2(\bar{x}_{g,a}) - log2(\bar{x}_{g,b})$$

- A (add) is the average log counts which corresponds to an estimate of the gene expression level.

$$A_g = \frac{1}{2}(log2(\bar{x}_g, a) + log2(\bar{x}_g, b))$$

```
## Draw a MA plot.
## Genes with FDR values below 1% are shown
plotMA(res, col = ifelse(res$padj >=0.01, "black", "violet"))
abline(h=c(-1:1), col="red")
```



Hierarchical clustering

To ensure that the selected genes distinguish well between "treated"" and "untreated" condition we will perform a hierachical clustering using the **heatmap.2()** function from the gplots library.

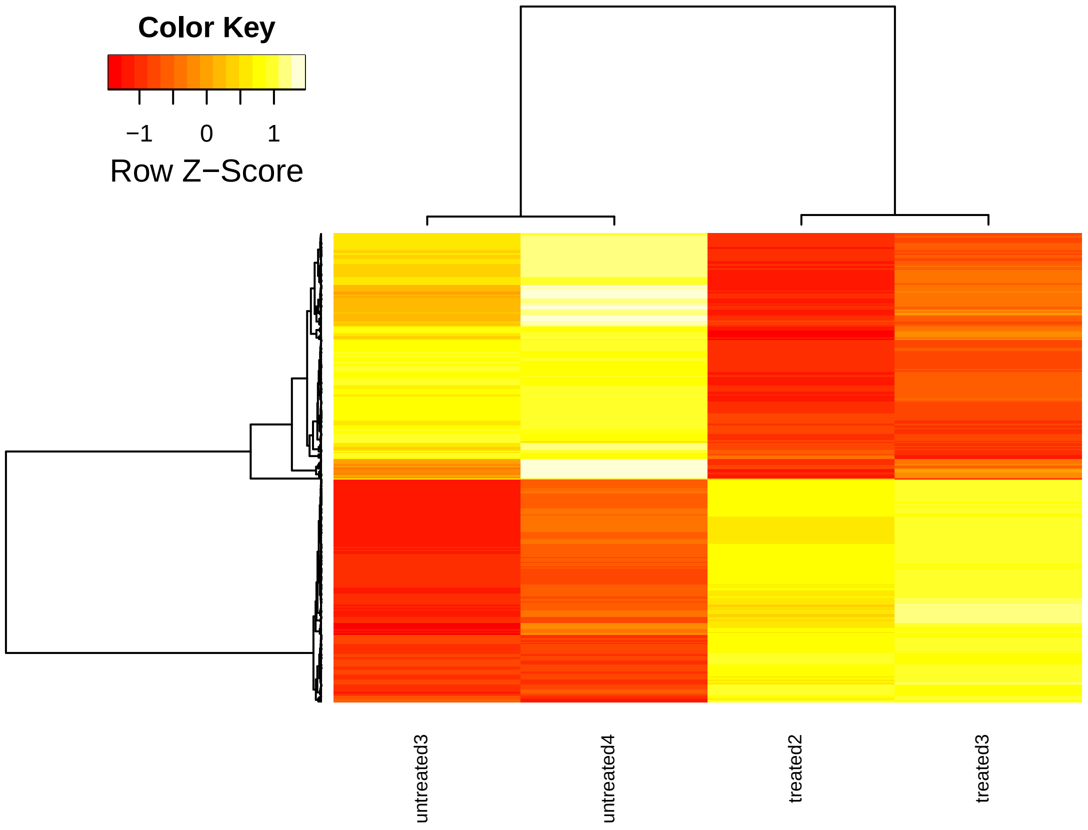
```
## We select gene names based on FDR (1%)
gene.kept <- res$id[res$padj <= 0.01 & !is.na(res$padj)]

## We retrieve the normalized counts for gene of interest
count.table.kept <- log2(count.table + 1)[gene.kept, ]
dim(count.table.kept)
```

```
## [1] 464   4
```

```
## Install the gplots library if needed then load it
if(!require("gplots")){
  install.packages("gplots")
}
```

```
## Loading required package: gplots
##
## Attaching package: 'gplots'
##
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library("gplots")

## Perform the hierarchical clustering with
## A distance based on Pearson-correlation coefficient
## and average linkage clustering as agglomeration criteria
heatmap.2(as.matrix(count.table.kept),
          scale="row",
          hclust=function(x) hclust(x,method="average"),
          distfun=function(x) as.dist((1-cor(t(x)))/2),
          trace="none",
          density="none",
          labRow="",
          cexCol=0.7)
```


Color Key
Row Z−Score
untreated3
untreated4
treated2
treated3