RNA-Seq - differential expression using DESeq2

D. Puthier (adapted From Hugo Varet, Julie Auberta and J. van Helden)

First version: 2016-12-10; Last update: 2016-12-12

Contents

The Snf2 dataset

The RNA-Seq dataset we will use in this practical has been produced by Gierliński *et al* ([@pmid26206307, @pmid27022035]). The dataset is composed of 48 WT yeast samples vs 48 Snf2 knock-out mutant cell line. The prepared RNA-Seq libraries (unstranded) were pooled and sequenced on seven lanes of a single flow-cell on an Illumina HiSeq 2000 resulting in a total of 1 billion 50-bp single-end reads across the 96 samples. RNA-Seq reads have been cleaned, mapped and counted to generated a count data matrix containing 7126 rows/genes. The primary objective of this study was to check whether the observed gene read counts distribution where consistent with theorical models (e.g. negative binomial). More information can be obtained in the original paper (pdf)

Loading the dataset

R enables to download data directly from the Web. The expression matrix and phenotypic information will be loaded into **R** using the **read.table** function. Both table will be converted into a data.frame object when loaded into R. The 'count.table' object will contains counts for each gene (row) and each sample (column).

```
# Load the files content in an R data.frame
path.counts <- "http://jvanheld.github.io/stats_avec_RStudio_EBA/practicals/yeast_2x48_replicates/data/
count.table <- read.table(file=path.counts, sep="\t", header=TRUE, row.names=1)
#View(count.table)
```

Phenotypic data

The dataset contains RNA-Seq count data for WT and KO. All phenotypic informations are enclosed in a dedicated file. Note that the produced data.frame encodes the 'strains' columns as a factor. A factor is a vector with levels (categories).

```
## Download phenotypic information
path.expDesign <- "http://jvanheld.github.io/stats_avec_RStudio_EBA/practicals/yeast_2x48_replicates/da
expDesign <- read.table(file=path.expDesign, sep="\t", header=TRUE)
#View(expDesign)

## The number of sample in each class
table(expDesign$strain)
```

```
Snf  WT
 48  48
```

```
## Define colors for subsequent plots
col.pheno <- c("green","orange")[as.numeric(expDesign$strain)]
unique(col.pheno)
```

```
[1] "orange" "green"
```

```
head(col.pheno)
```

```
[1] "orange" "orange" "orange" "orange" "orange" "orange"
```

```
tail(col.pheno)
```

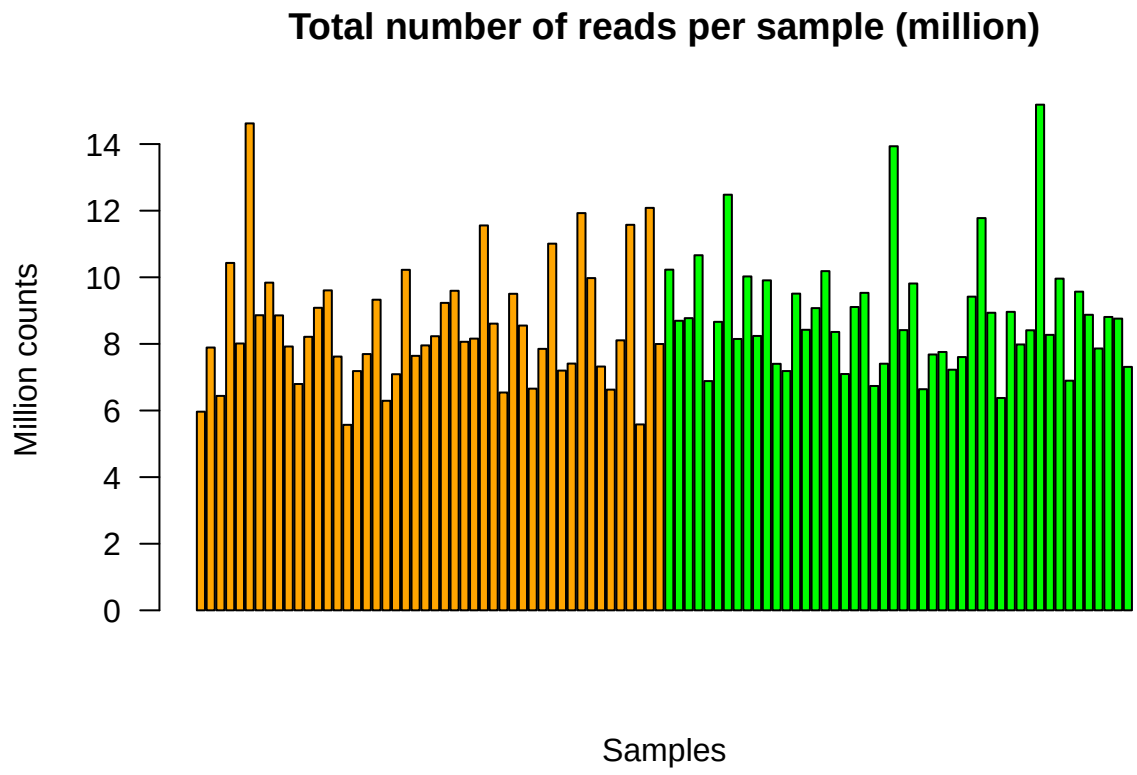
```
[1] "green" "green" "green" "green" "green" "green"
```

- Draw a barplot showing the number of reads in each sample. Use the **colSums** function or the **apply** function.
- What can you say from this diagram ?

View solution| Hide solution

Solution

```
barplot(colSums(count.table)/1000000,
        main="Total number of reads per sample (million)",
        col=col.pheno,
        names.arg = "",
        las=2,
        xlab="Samples",
        ylab="Million counts")
```



Descriptive statistics

Basic statistics

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

```
## Dimensions
nb_samples <- ncol(count.table)
print(nb_samples)
```

```
[1] 96
```

```
nb_genes <- nrow(count.table)
print(nb_genes)
```

```
[1] 7126
```

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

```
[1] 7126   96
```

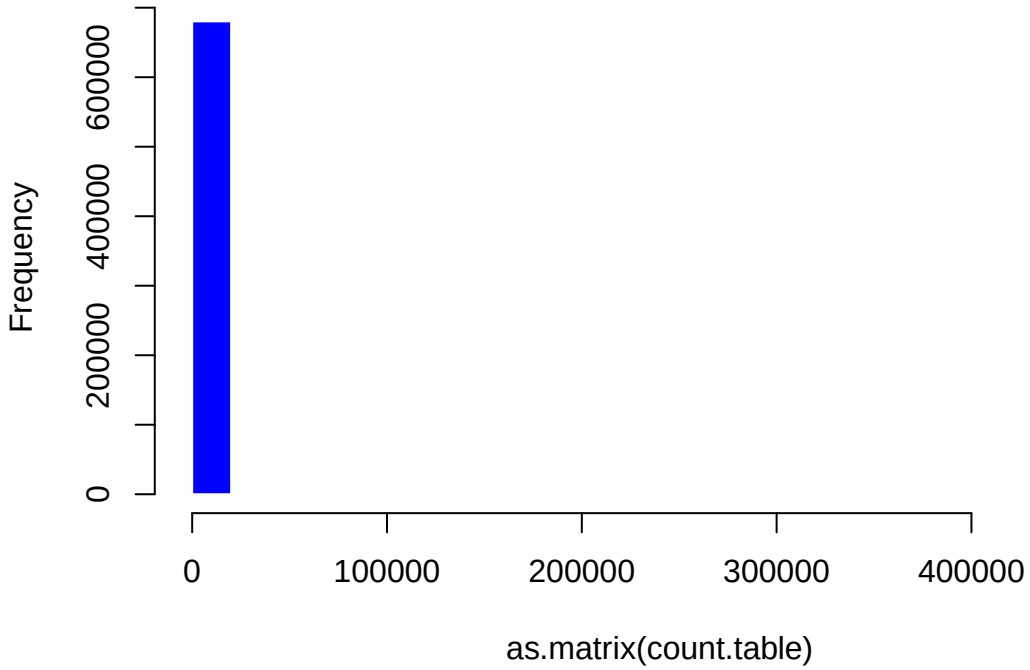
```
## Min, Max, median (...).
## Here on the first 4 samples
head(summary(count.table[,1:4]))
```

```
      WT1              WT2              WT3              WT4        
 Min.   :     0   Min.   :     0   Min.   :     0   Min.   :     0  
 1st Qu.:    49   1st Qu.:    88   1st Qu.:    74   1st Qu.:   104  
 Median :   224   Median :   371   Median :   297   Median :   430  
 Mean   :   837   Mean   :  1107   Mean   :   903   Mean   :  1464  
 3rd Qu.:   561   3rd Qu.:   853   3rd Qu.:   670   3rd Qu.:  1019  
 Max.   :188825   Max.   :196804   Max.   :172119   Max.   :328674  
```

Distributions

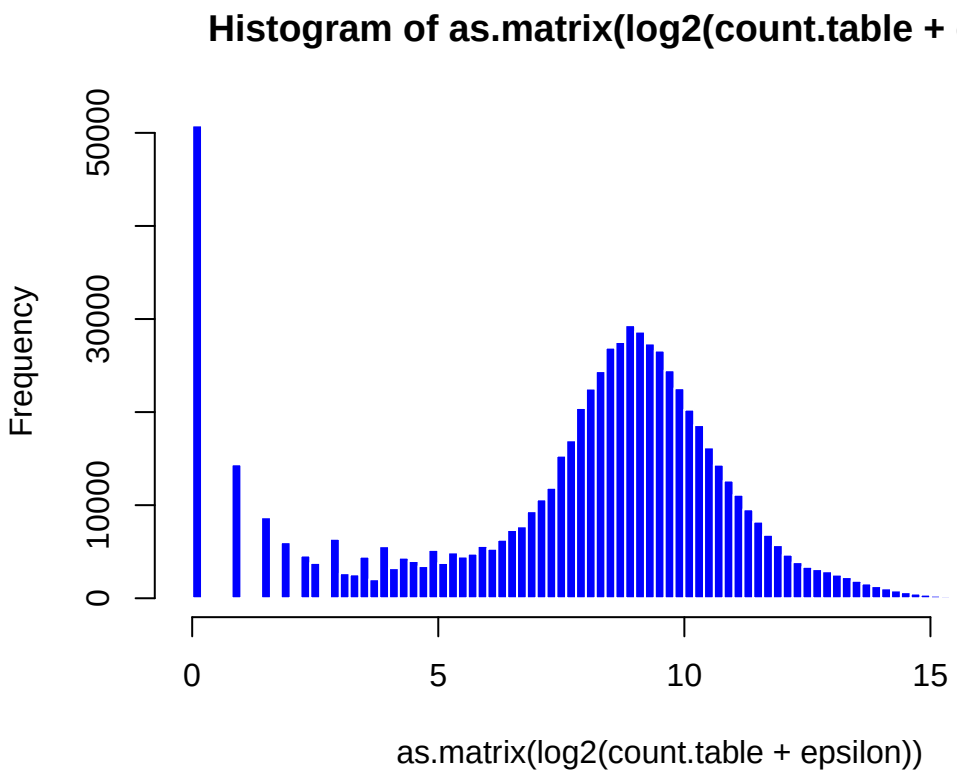
The summary only displays a few milestone values (mean, median, quartiles). In order to get a better intuition of the data, we can draw an histogram of all values.

```
## Data distribution
hist(as.matrix(count.table), col="blue", border="white")
```

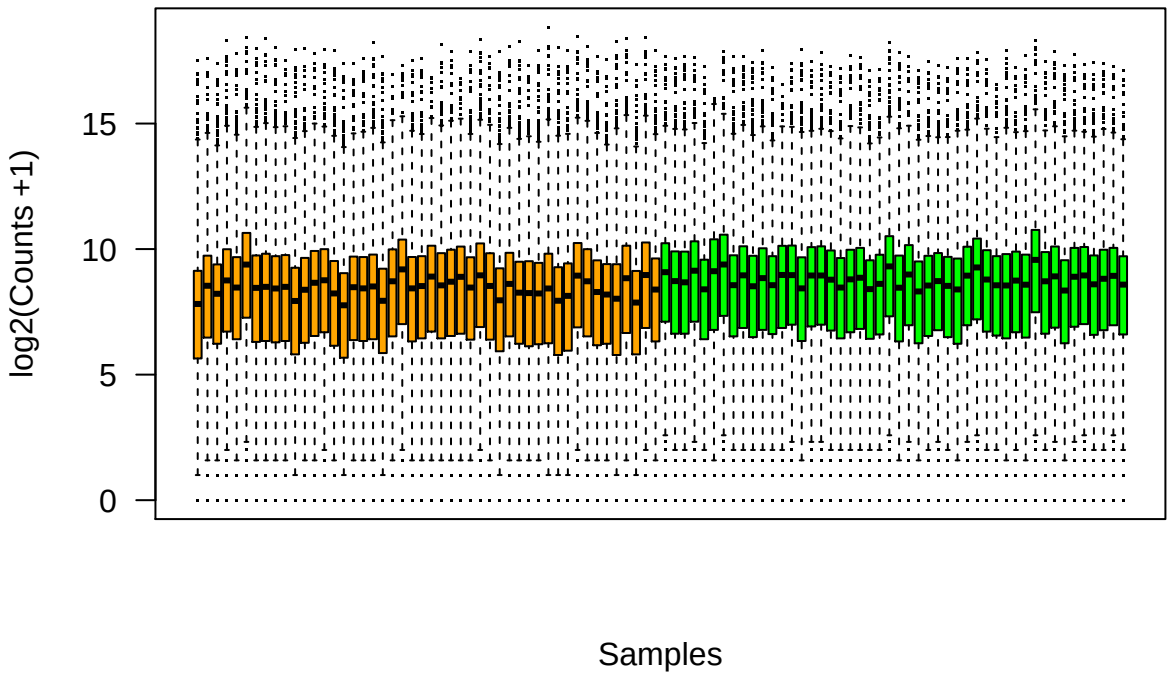


The histogram is not very informative so far, apparently due to the presence of a few very high count values, that impose a very large scale on the X axis. We can use a logarithmic transformation to improve the readability. Note that we will add pseudo count to avoid problems with "zero" counts observed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

```
## Data distribution in log scale.

epsilon <- 1 # pseudo-count to avoid problems with log(0)

## Logarithmic transformation
hist(as.matrix(log2(count.table + epsilon)), breaks=100, col="blue", border="white")
```

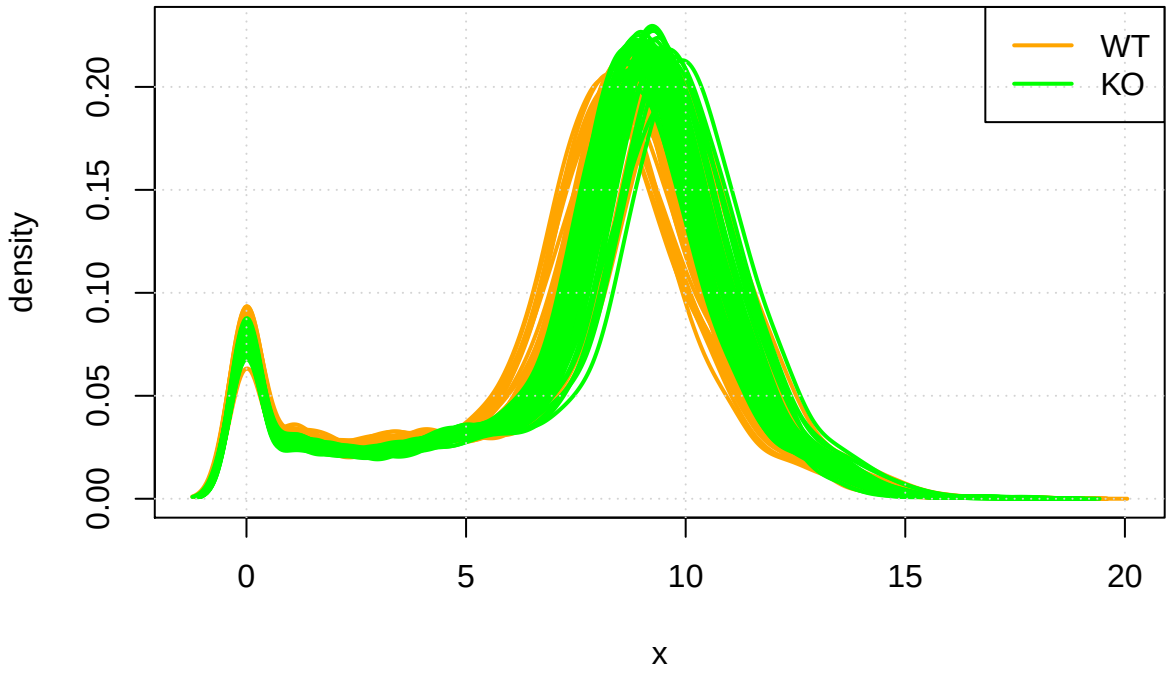



```
## Boxplots
boxplot(log2(count.table + epsilon), col=col.pheno, pch=".",
        las=1, xaxt='n', xlab="Samples", ylab="log2(Counts +1)")
```



Another way to get an intuition of the value distributions is to use the *plotDensity()* function, which draws one density curve for each sample.

```
## Density
## We will require one function from the affy package
if(!require("affy")){
  source("http://bioconductor.org/biocLite.R")
  biocLite("affy")
}
library(affy)
plotDensity(log2(count.table + 1), lty=1, col=col.pheno, lwd=2)
grid()
legend("topright", legend=c("WT", "KO"), col=unique(col.pheno), lwd=2)
```

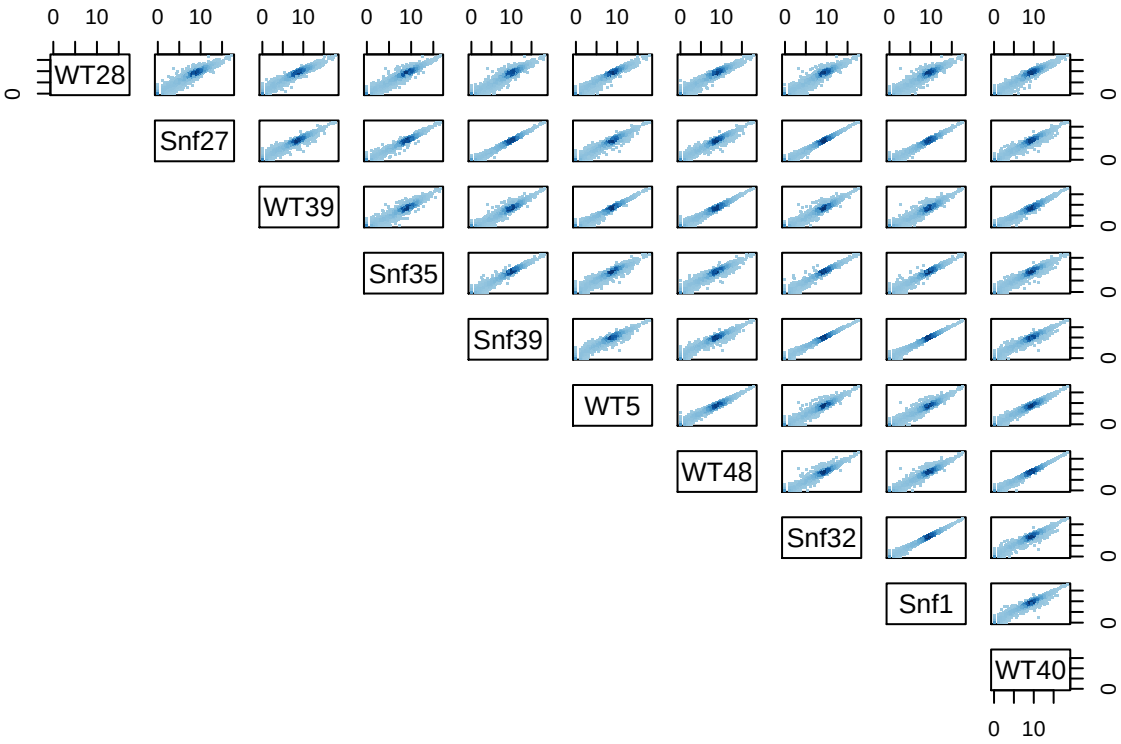


Beware: the R function *plotDensity()* does not display the actual distribution of your values, but a polynomial fit. The representation thus generally looks smoother than the actual data. It is important to realize that, in some particular cases, the fit can lead to extrapolated values which can be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

Scatter plots

Let's have a look at the scatter plots using the *pairs()* function. We will only represent 10 randomly selected samples.

```
plotFun <- function(x,y){ dns <- densCols(x,y); points(x,y, col=dns, pch=".") }
set.seed(123) # forces the random number generator to produce fixed results.
pairs(log2(count.table[,sample(ncol(count.table),10)] + 1), panel=plotFun, lower.panel = NULL)
```



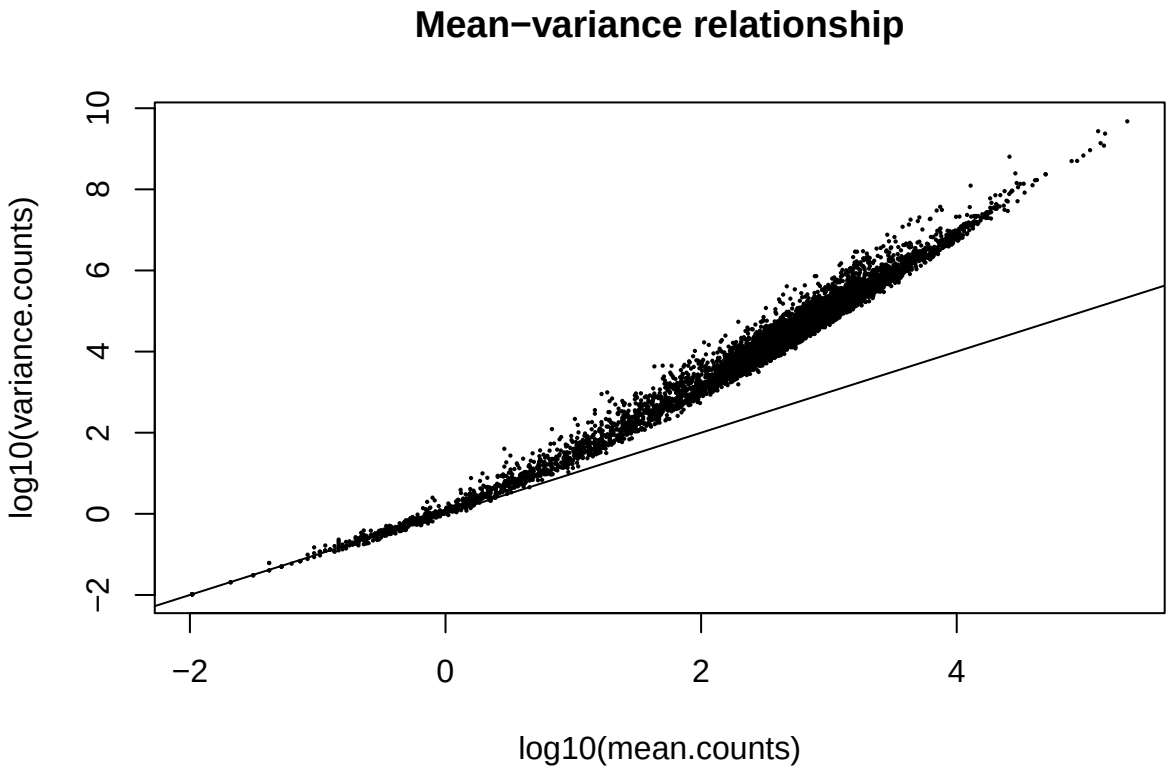
The command *pairs()* draws a scatter plot for each pair of columns of the input dataset. The plot shows a fairly good reproducibility between samples of the same type (WT or KO, respectively): all points are aligned along te diagonal, with a relatively wider dispersion at the bottom, corresponding to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

In contrast, on all the plots comparing a WT and a KO sample, we can see some points (genes) discarding from the diagonal.

Count variance is related to mean

As you can see from the following plot the relationship between variance and mean is not strictly linear. This can be shown by the poor fit that is obtained using a linear regression.

```
## Computing mean and variance
mean.counts <- rowMeans(count.table)
variance.counts <- apply(count.table, 1, var)

## Mean and variance relationship
plot(x=log10(mean.counts), y=log10(variance.counts), pch=16, cex=0.3, main="Mean-variance relationship"
abline(a=0, b=1)
```



Eliminating undetected genes

All genes from genome the *S. cerevisiae* where quantified. However only a fraction of them where expressed and some of them where to weakly expressed to be detected in any of the sample. As a result the count table may contain rows with only zero values (null counts).

- What is the percentage of gene having null counts per sample. Draw a barplot.
- Some genes were not detected in any of the sample. Count their number, and delete them from the **count.table** data.frame.

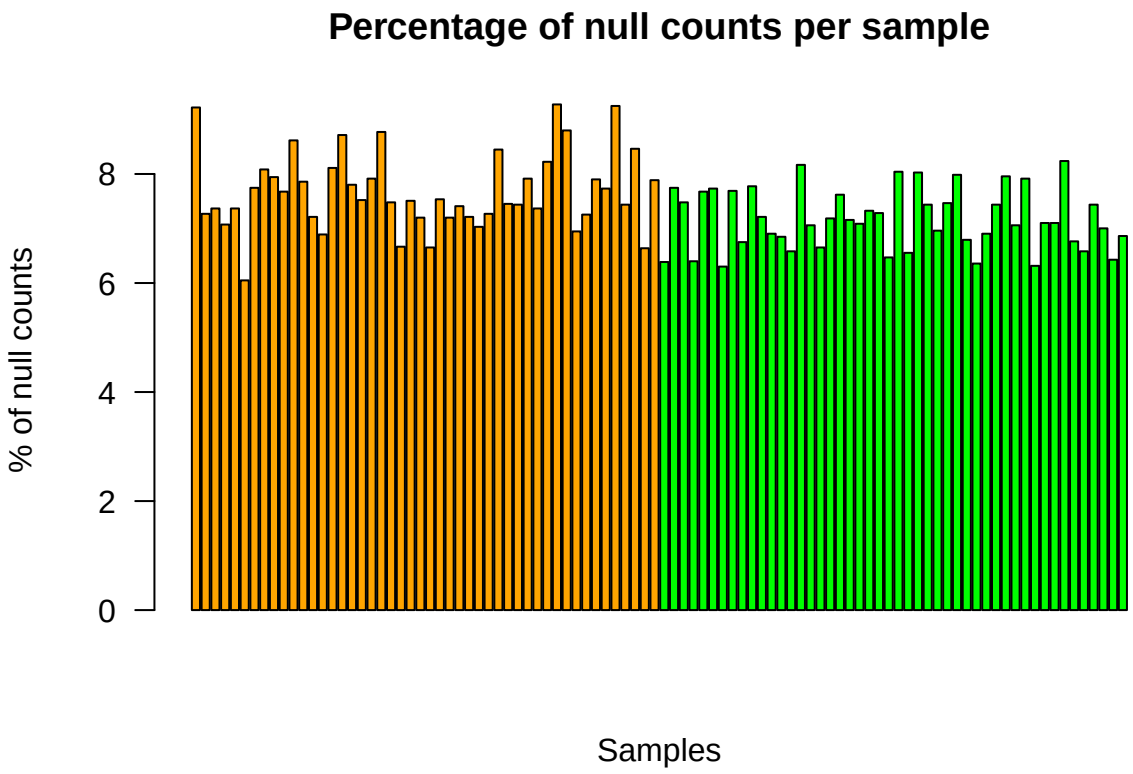
View solution| Hide solution

Solution

```
prop.null <- apply(count.table, 2, function(x) 100*mean(x==0))
print(head(prop.null))
```

```
 WT1  WT2  WT3  WT4  WT5  WT6
9.22 7.27 7.37 7.07 7.37 6.05
```

```
barplot(prop.null, main="Percentage of null counts per sample", col=col.pheno, xaxt='n', xlab='Samples'
```




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

Selecting randomly samples

One of the question that will drive the analysis will be to define the impact of the number of sample on the analysis.

The original study contained 48 replicates per genotype, what happens if we select a smaller number?

Each attendee of this course select a given number (e.g. 3,4,5,10,15,20, 35, 40, 45...) and adapt the code below run the analysis with that number of replicates per genotype. We will at the end then compare the results (number of genes, significance, ...).

```
nb.replicates <- 10 ## Each attendee chooses a number (3,4,5,10,15 or 20)

samples.WT <- sample(1:48, size=nb.replicates, replace=FALSE)

## Random sampling of the Snf2 replicates (columns 49 to 96)
samples.Snf2 <- sample(49:96, size=nb.replicates, replace=FALSE)

selected.samples <- c(samples.WT, samples.Snf2)

# Don't forget to update colors
col.pheno.selected <- col.pheno[selected.samples]
```

Differential analysis with DESeq2

In this section we will search for genes whose expression is affected by the genetic invalidation. You will first need to install the **DESeq2** bioconductor library then load it.

```
## Install the library if needed then load it
if(!require("DESeq2")){
  source("http://bioconductor.org/biocLite.R")
  biocLite("DESeq2")
}
library("DESeq2")
```

Creating a DESeqDataSet dataset

We will then create a **DESeqDataSet** using the **DESeqDataSetFromMatrix()** function. Get some help about the **DESeqDataSet** and have a look at some important accessor methods: **counts**, **conditions**, **estimateSizeFactors**, **sizeFactors**, **estimateDispersions** and **nbinomTest**.

```
## Use the DESeqDataSetFromMatrix to create a DESeqDataSet object
dds0 <- DESeqDataSetFromMatrix(countData = count.table[,selected.samples ], colData = expDesign[selecte
print(dds0)
```

```
class: DESeqDataSet
dim: 6887 20
metadata(1): version
assays(1): counts
rownames(6887): 15s_rrna 21s_rrna ... ty(gua)m2 ty(gua)o
rowData names(0):
colnames(20): WT46 WT22 ... Snf12 Snf6
colData names(2): label strain
```

```
## What kind of object is it ?
is(dds0)
```

```
[1] "DESeqDataSet"                "RangedSummarizedExperiment"
[3] "SummarizedExperiment"        "Vector"
[5] "Annotated"
```

```
isS4(dds0)
```

```
[1] TRUE
```

```
## What does it contain ?
# The list of slot names
slotNames(dds0)
```

```
[1] "design"             "dispersionFunction" "rowRanges"
[4] "colData"            "assays"             "NAMES"
[7] "elementMetadata"    "metadata"
```

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

Normalization

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

How is the scaling factor computed ?

Given a matrix with p columns (samples) and n rows (genes) this function estimates the size factors as follows: Each column element is divided by the **geometric means** of the rows. For each sample, 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.

The scaling factor for sample j is thus obtained as:

$$sf_j = median(\frac{K_{g,j}}{(\prod_{j=1}^{p} K_{g,j})^{1/p}})$$

```
### 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)
}
```

Now, check that the results obtained with our function are the same as those produced by DESeq. The method associated with normalization for the "CountDataSet" class is **estimateSizeFactors()**.

```
## Normalizing using the method for an object of class"CountDataSet"
dds.norm <-  estimateSizeFactors(dds0)
sizeFactors(dds.norm)
```

```
 WT46  WT22  WT32  WT26   WT5  WT39  WT11   WT2  WT14  WT38 Snf43 Snf33
0.646 1.536 0.686 1.036 0.940 0.759 0.684 0.986 1.175 0.689 1.300 1.667
Snf30 Snf45 Snf29 Snf31 Snf23 Snf25 Snf12  Snf6
1.001 1.054 1.144 0.916 1.061 0.971 1.034 1.478
```

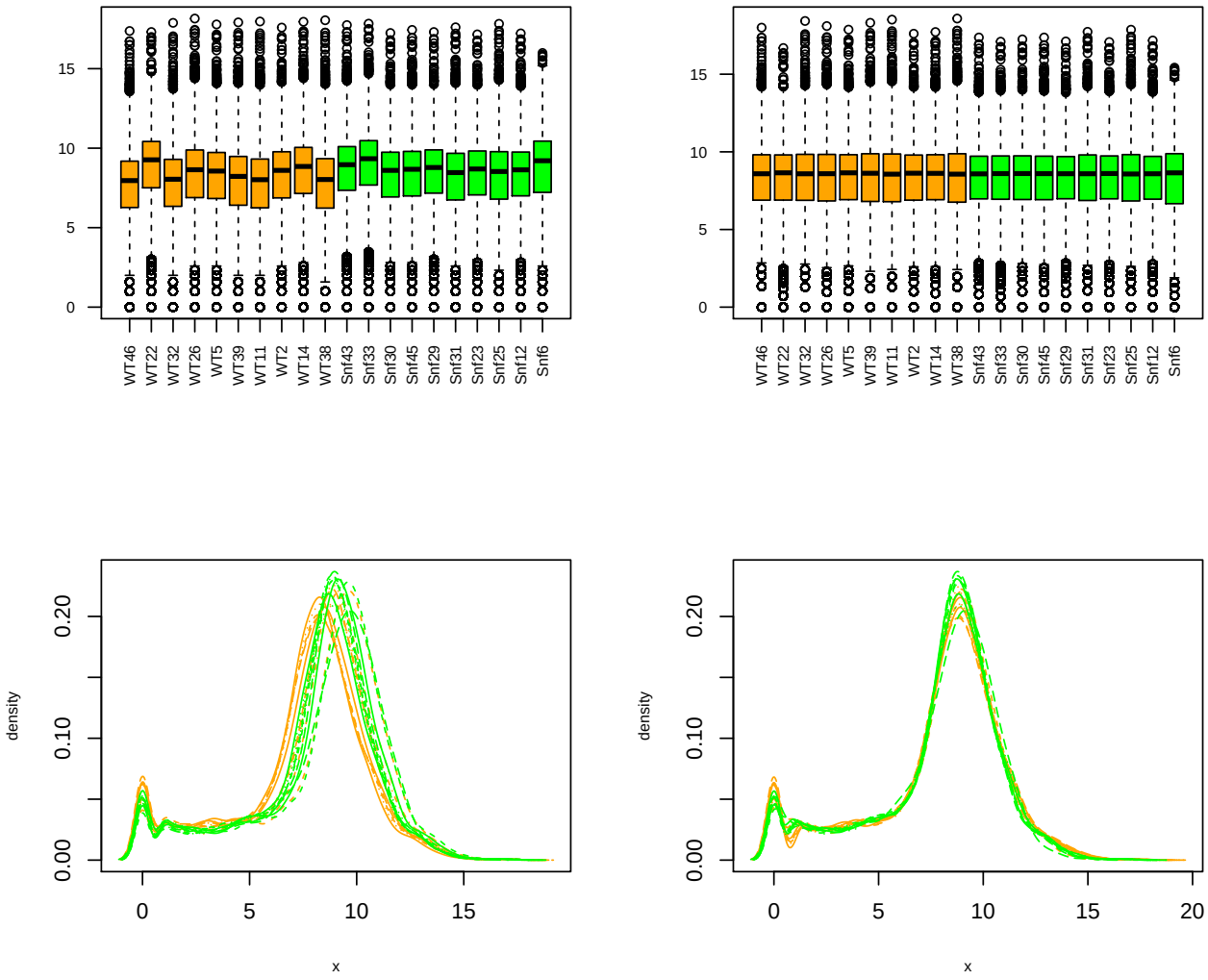
```
## Now get the scaling factor with our homemade function.cds.norm
head(estimSf(dds0))
```

```
 WT46  WT22  WT32  WT26   WT5  WT39
0.646 1.536 0.686 1.036 0.940 0.759
```

```
all(round(estimSf(dds0),6) == round(sizeFactors(dds.norm), 6))
```

```
[1] TRUE
```

```
## Checking the normalization
par(mfrow=c(2,2),cex.lab=0.7)
boxplot(log2(counts(dds.norm)+1),  col=col.pheno.selected, cex.axis=0.7, las=2)
boxplot(log2(counts(dds.norm, normalized=TRUE)+1),  col=col.pheno.selected,, cex.axis=0.7, las=2)
plotDensity(log2(counts(dds.norm)+1),  col=col.pheno.selected, cex.lab=0.7)
plotDensity(log2(counts(dds.norm, normalized=TRUE)+1),  col=col.pheno.selected, cex.lab=0.7)
```



Modeling read counts

Let us imagine 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, the measured expression values for each genes are expected to fluctuate more importantly, due to the combination of biological and technical factors: inter-individual variations in gene regulation, sample purity, cell-synchronization issues or reponses to environment (e.g. heat-shock).

The Poisson distribution has only one parameter indicating its expected mean : λ. The variance of the distribution equals its mean λ. Thus in most cases, the Poisson distribution is not expected to fit very well with the count distribution in biological replicates, since we expect some over-dispersion (greater variability) due to biological noise.

As a consequence, when working with RNA-Seq data, many of the current approaches for differential expression call rely on an alternative distribution: the *negative binomial* (note that this holds 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 give us the probability of observing x failures before a target number of succes n is obtained. As we will see later the negative binomial can also be used to model over-dispersed data (in this case this overdispersion is relative to the poisson model).

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).

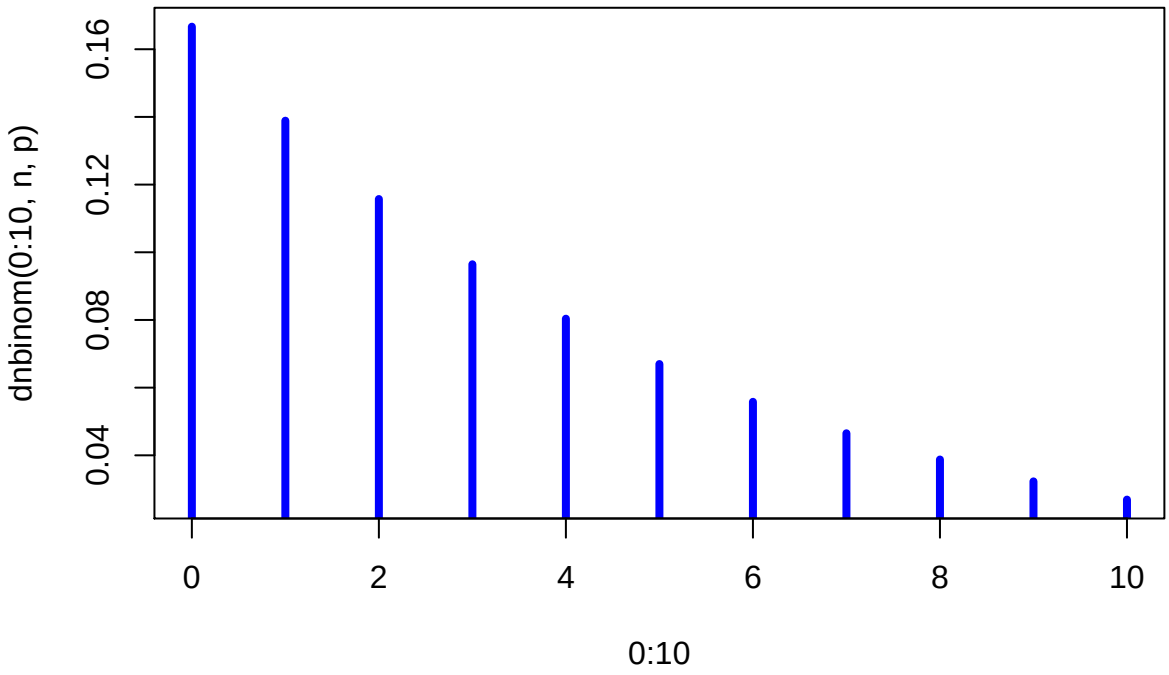
$$P_{NegBin}(x; n, p) = \binom{x+n-1}{x} \cdot p^n \cdot (1-p)^x = C^x_{x+n-1} \cdot p^n \cdot (1-p)^x$$

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: $P_{NegBin}(x; 1, p) = P_{geom}(x; p) = p \cdot (1-p)^x$.

```
par(mfrow=c(1,1))

## Some intuition about the negative binomiale 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=4)
```




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

```
[1] 0.167
```

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

```
[1] 0.665
```

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

```
[1] 0.865
```

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

```
[1] 0.135
```

```
## 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",
     ylab="P(x; n,p)",
     xlab=paste("x = number of failures before", n, "successes"))
```

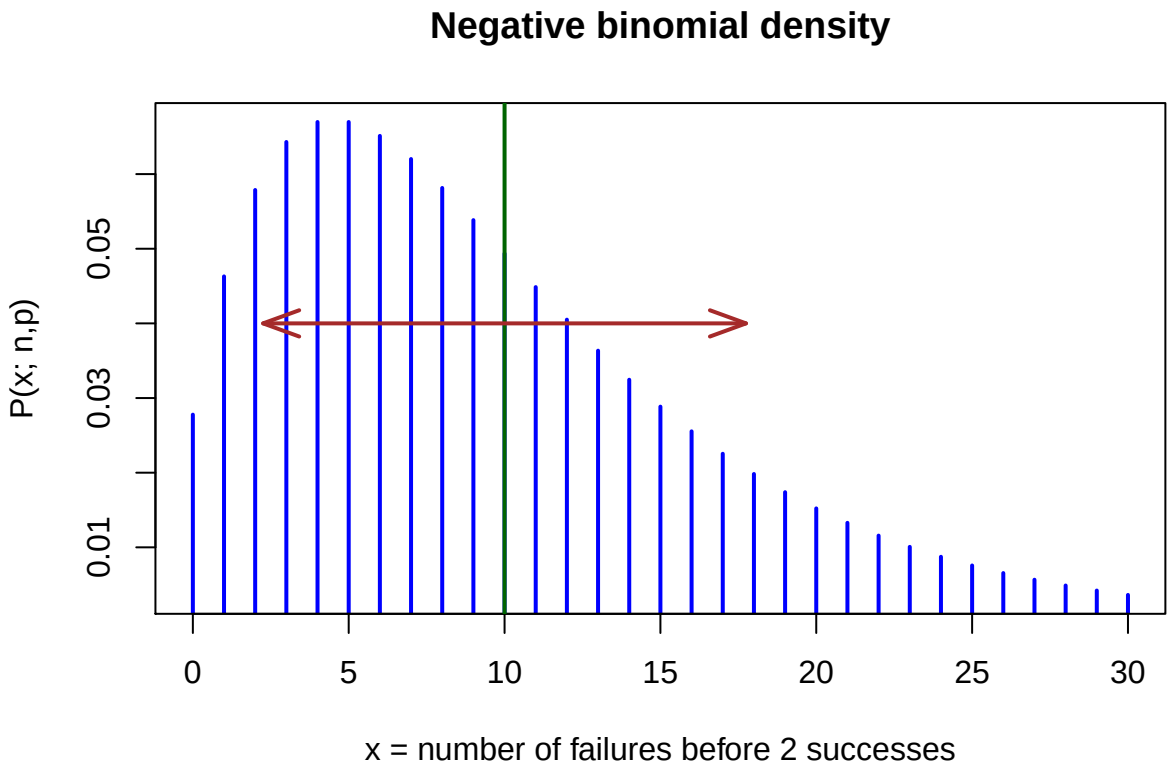
```
# Expected value
q <- 1-p
(ev <- n*q/p)
```

```
[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 $m + m^2/r$.

Note that m can be deduced from n and p.

```
n <- 10
p <- 1/6
q <- 1-p
mu <- n*q/p

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

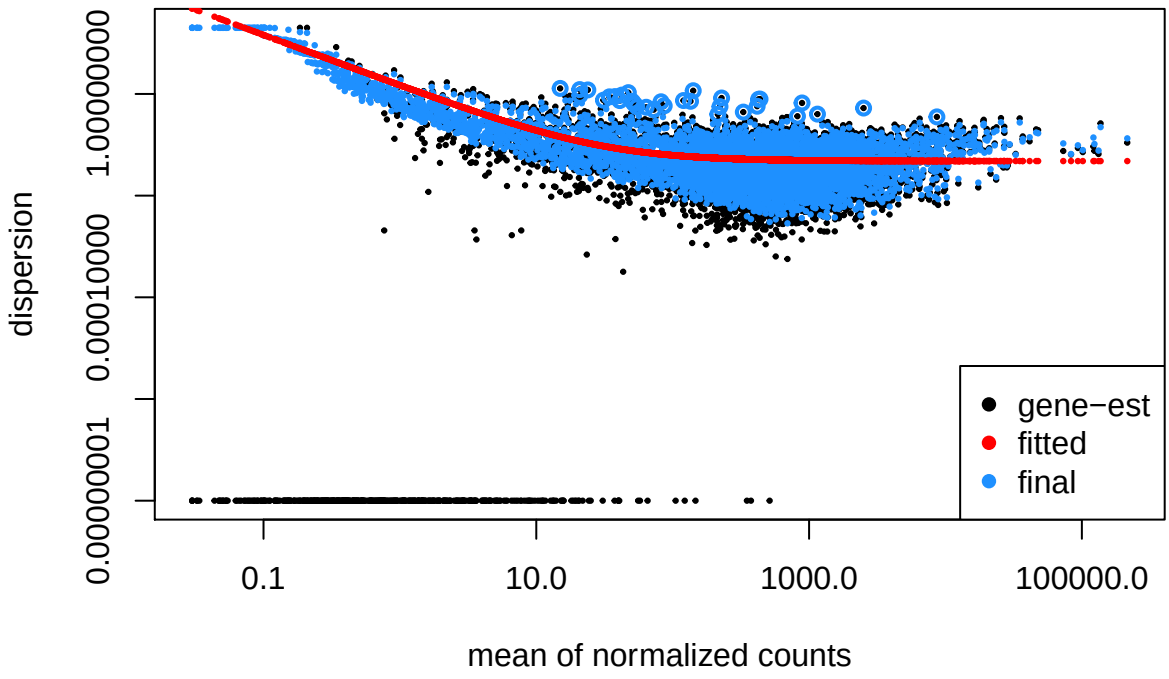
```
[1] TRUE
```

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
 dds.disp <- estimateDispersions(dds.norm)

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



Performing differential expression call

Now that a negative binomial model has been fitted for each gene, the **nbinomWaldTest** 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).

```
alpha <- 0.0001
wald.test <- nbinomWaldTest(dds.disp)
res.DESeq2 <- results(wald.test, alpha=alpha, pAdjustMethod="BH")

## What is the object returned by nbinomTest()
is(res.DESeq2) # a data.frame
```

```
[1] "DESeqResults"    "DataFrame"       "DataTable"       "SimpleList"
[5] "DataTableORNULL" "List"            "Vector"          "Annotated"
```

```
head(res.DESeq2)
```

```
log2 fold change (MAP): strain WT vs Snf
Wald test p-value: strain WT vs Snf
DataFrame with 6 rows and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
15s_rrna     20.87          0.189     0.404     0.469    0.6392     0.764
21s_rrna    135.14          0.187     0.384     0.486    0.6267     0.755
hra1          2.17          0.762     0.394     1.933    0.0532     0.117
icr1        128.78         -0.190     0.117    -1.625    0.1042     0.199
lsr1        197.55         -0.313     0.286    -1.095    0.2736     0.418
nme1         21.84         -0.251     0.299    -0.838    0.4020     0.551
```

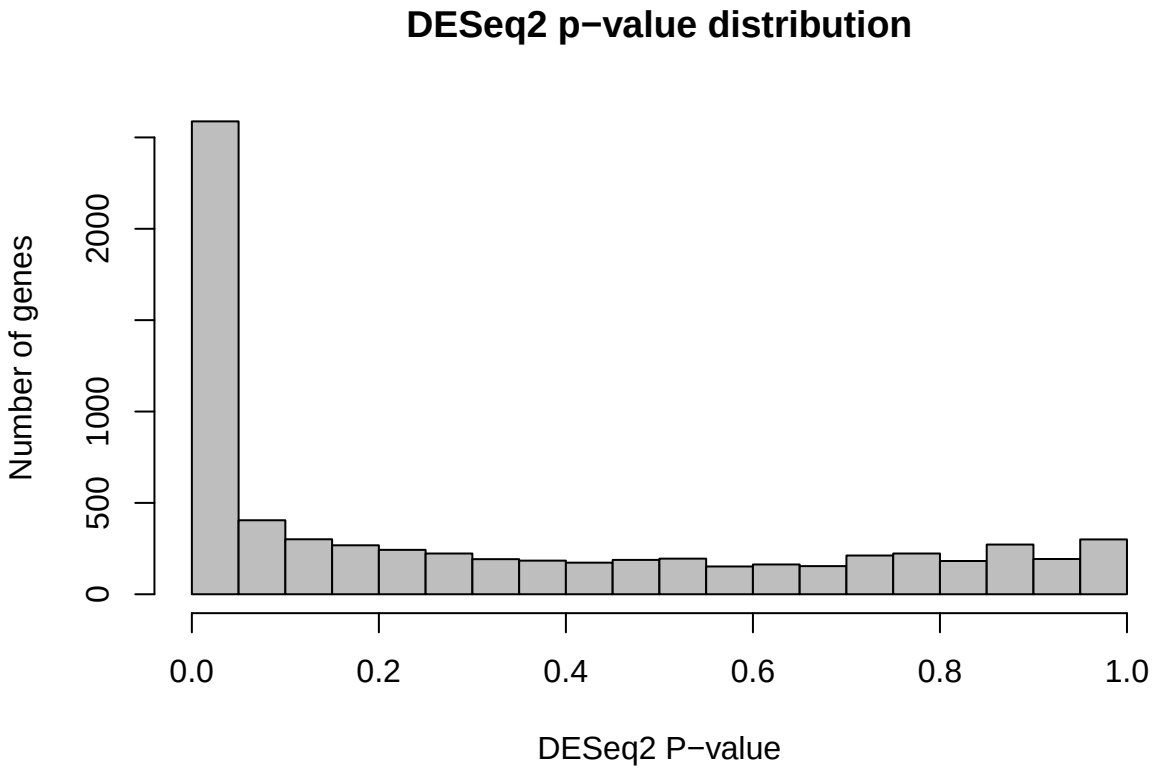
```
## The column names of the data.frame
## Note the column padj
## contains FDR values (computed Benjamini-Hochberg procedure)
colnames(res.DESeq2)
```

```
[1] "baseMean"       "log2FoldChange" "lfcSE"          "stat"
[5] "pvalue"         "padj"
```

```
## Order the table by decreasing p-valuer
res.DESeq2 <- res.DESeq2[order(res.DESeq2$padj),]
head(res.DESeq2)
```

```
log2 fold change (MAP): strain WT vs Snf
Wald test p-value: strain WT vs Snf
DataFrame with 6 rows and 6 columns
         baseMean log2FoldChange     lfcSE      stat    pvalue      padj
        <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
ygr234w      2697           4.07    0.1061      38.4  0.00e+00  0.00e+00
yml123c      4676           4.54    0.1170      38.8  0.00e+00  0.00e+00
yor290c       753           6.98    0.1422      49.1  0.00e+00  0.00e+00
yar071w      1547           4.08    0.1171      34.8 1.06e-265 1.81e-262
yhr215w      1200           4.17    0.1324      31.5 4.90e-218 6.67e-215
ycl064c      3000           1.38    0.0456      30.3 7.83e-202 8.89e-199
```

```
## Draw an histogram of the p-values
hist(res.DESeq2$padj, breaks=20, col="grey", main="DESeq2 p-value distribution", xlab="DESeq2 P-value",
```



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 :

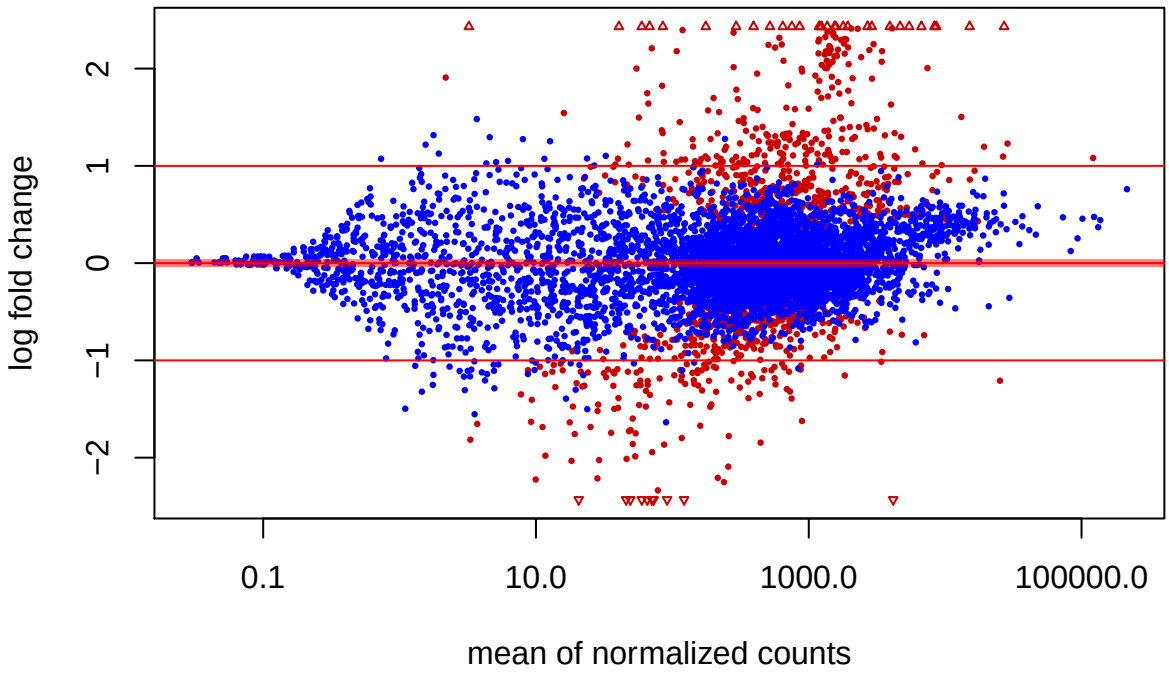
- M (Minus) is the log ratio of counts calculated for any gene.

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

- 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 adjusted p-values below 1% are shown
plotMA(res.DESeq2, colNonSig = "blue")
abline(h=c(-1:1), col="red")
```



Hierarchical clustering

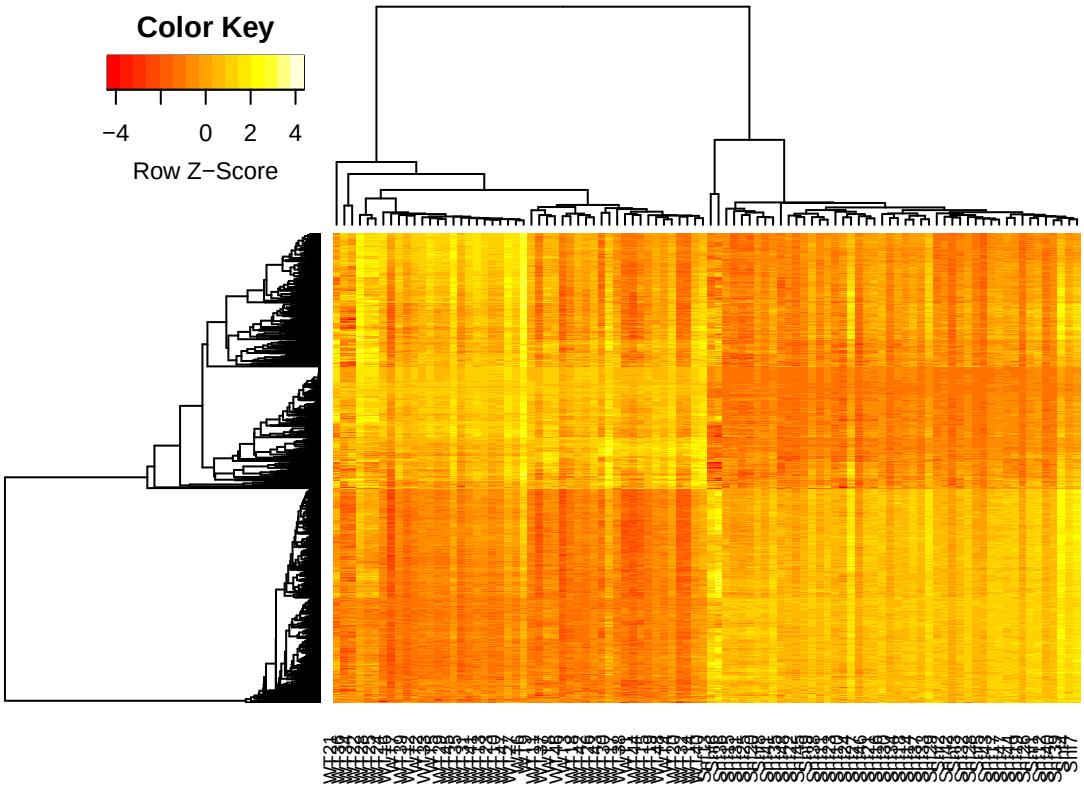
To ensure that the selected genes distinguish well between "treated"" and "untreated" condition we will perform a hierachical clustering using the **heatmap.2()** function from the gplots library.

```
## We select gene names based on FDR (1%)
gene.kept <- rownames(res.DESeq2)[res.DESeq2$padj <= alpha & !is.na(res.DESeq2$padj)]

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

```
[1] 1079   96
```

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



Assess the effect of sample number on differential expression call

Using a loop, randomly select 10 times 2,5,10,15..45 samples from WT and Snf2 KO. Perform differential expression calls and draw a diagram showing the number of differential expressed genes.