

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: 2017-01-06

Contents

The Snf2 dataset	1
Loading the dataset	2
Phenotypic data	2
Descriptive statistics	3
Basic statistics	3
Distributions	6
Scatter plots	8
Eliminating undetected genes	10
Selecting randomly samples	13
Differential analysis with DESeq2	13
Creating a DESeqDataSet dataset	13
Normalization	14
How is the scaling factor computed ?	14
Count variance is related to mean	15
Modeling read counts	19
What is the negative binomial ?	19
Modelling read counts through a negative binomial	22
Performing differential expression call	23
Volcano plot	24
Check the expression levels of the most differentially expressed gene	25
Looking at the results with a MA plot	25
Hierarchical clustering	26
Functional enrichment	27
Assess the effect of sample number on differential expression call	30

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 were consistent with theoretical 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).

```
# Download data files from the Web site (only if not done yet)
url.counts <- "http://jvanheld.github.io/stats_avec_RStudio_EBA/practicals/yeast_2x48_replicates/data/c

## Local paths
dir.snf2 <- ("~/ASG/practicals/rnaseq_snf2_Schurch_2015")
dir.counts <- file.path(dir.snf2, "data")
file.counts <- file.path(dir.counts, "counts.txt")
file.expDesign <- file.path(dir.counts, "expDesign.txt")

## Create a directory to download the dataset if it does not exist
dir.create(dir.counts, showWarnings = FALSE, recursive = TRUE)

## Download the data files if required
if (!file.exists(file.counts)) {
  message("Downloading count table from ", url.counts)
  download.file(url=file.path(url.counts, "counts.txt"), destfile = file.counts)
}
if (!file.exists(file.expDesign)) {
  message("Downloading design table from ", url.counts)
  download.file(url=file.path(url.counts, "expDesign.txt"), destfile = file.expDesign)
}

# Load the count table
count.table <- read.table(file=file.counts, sep="\t", header=TRUE, row.names=1)
# View(count.table)
```

Phenotypic data

The dataset contains RNA-Seq count data for a wild type strain (**WT**) and a **Snf2** mutant, with 48 biological replicates for each genotype.

All phenotypic informations are enclosed in a dedicated file. Note that the produced data.frame encodes the ‘strains’ columns as a factor¹.

```
# Load experimental design file
expDesign <- read.table(file=file.expDesign, sep="\t", header=TRUE)
#View(expDesign)

# Check the first and last line of the phenotypic data
head(expDesign)
```

	label	strain
1	WT1	WT
2	WT2	WT
3	WT3	WT

¹A factor is a vector with levels (categories), which permits an efficient storage and indexing, but can in some cases lead to misleading effects. To circumvent this, we will sometimes have to convert the factor to a vector, with the R command **as.vector()**.

```

4   WT4      WT
5   WT5      WT
6   WT6      WT
tail(expDesign)

  label strain
91 Snf43    Snf
92 Snf44    Snf
93 Snf45    Snf
94 Snf46    Snf
95 Snf47    Snf
96 Snf48    Snf

## Count the number of sample in each class
table(expDesign$strain)

```

```

Snf  WT
48  48

## Define a strain-specific color for each sample,
## and add it as a supplementary column to the phenotypic data
col.strain <- c("WT"="green","Snf"="orange") # Choose one color per strain
expDesign$color <- col.strain[as.vector(expDesign$strain)]

```

- Draw a barplot showing the number of reads in each sample. Use either the `colSums()` or the `apply()` function (run `help(colSums())` if you don't know this 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=expDesign$color,
#        names.arg = "",
        las=1, horiz=TRUE,
        ylab="Samples", cex.names=0.5,
        xlab="Million counts")

```

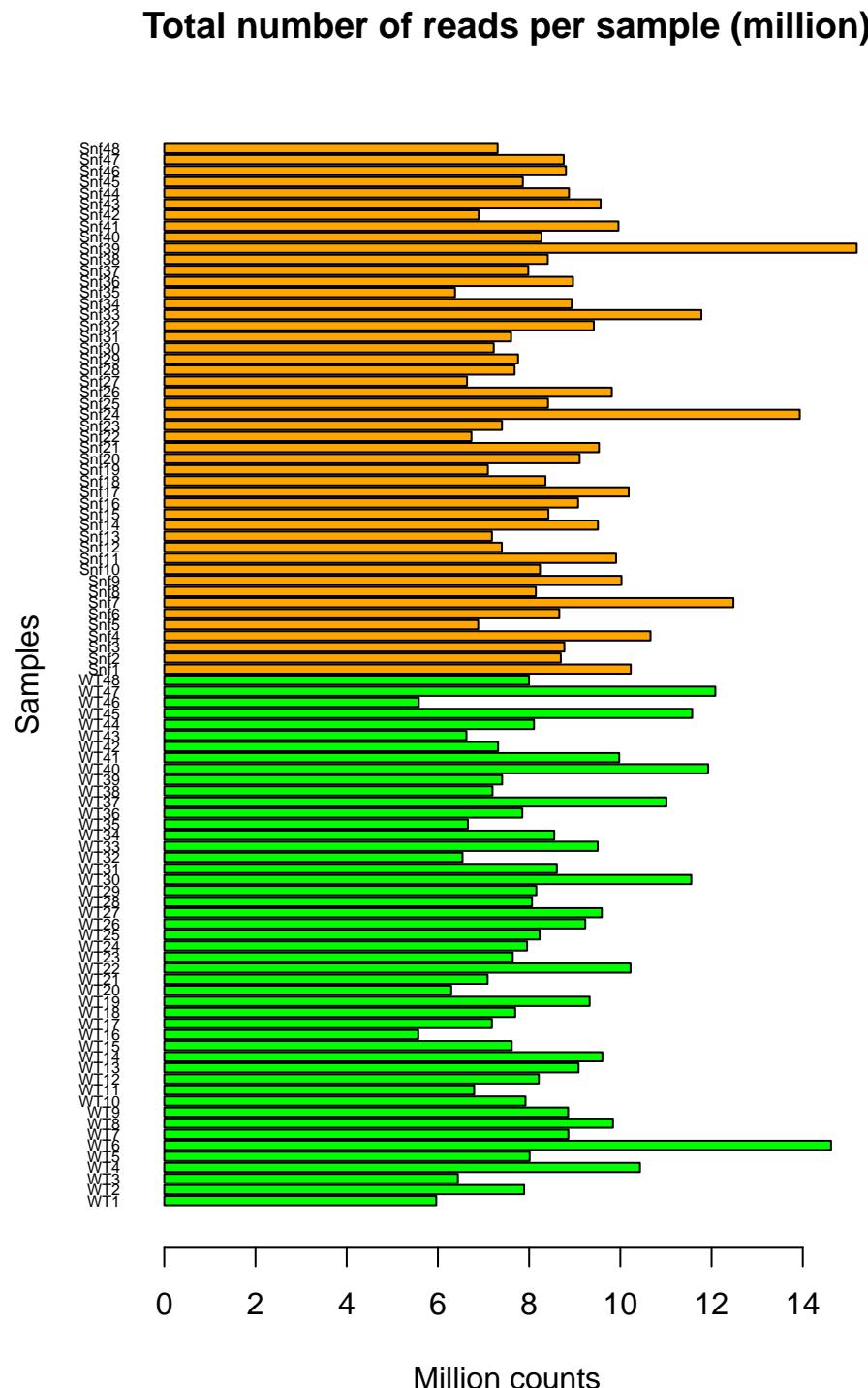
The barplot indicates the library sizes (total number of reads) for each sample. We can see important differences, ranging from 5.57 to 15.2 million reads per sample.

This has important consequences: any read count should be interpreted relative to the sequencing depth of the corresponding sample. We will thus need to **normalise** the counts. Consequently, it makes not much sense to compute summary statistics per gene (mean counts, standard deviation, ...) before having normalized the data.

Descriptive statistics

Basic statistics

Before going further in the analysis, we will compute some descriptive statistics on the dataset. At this stage we only compute statistics per sample, since statistics per gene are meaningful only after library-wise normalization of the counts.



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

```
## A magic trick to convert column-wise summaries into a data.frame.
## The do.call() function produces a data frame with one col per sample,
## we transpose it to obtain one row per sample and one column per statistics.
stats.per.sample <- data.frame(t(do.call(cbind, lapply(count.table, summary))))
head(stats.per.sample)
```

	Min.	X1st.Qu.	Median	Mean	X3rd.Qu.	Max.
WT1	0	49.0	224	837	561	189000
WT2	0	88.0	371	1110	853	197000
WT3	0	74.2	297	903	670	172000
WT4	0	104.0	430	1460	1020	329000
WT5	0	84.0	353	1120	819	225000
WT6	0	153.0	667	2050	1600	357000

```
## We can now add some columns to the stats per sample
stats.per.sample$libsum <- apply(count.table, 2, sum) ## libsum
# Add some percentiles
stats.per.sample$perc05 <- apply(count.table, 2, quantile, 0.05)
stats.per.sample$perc10 <- apply(count.table, 2, quantile, 0.10)
stats.per.sample$perc90 <- apply(count.table, 2, quantile, 0.90)
stats.per.sample$perc95 <- apply(count.table, 2, quantile, 0.95)
stats.per.sample$zeros <- apply(count.table==0, 2, sum)
stats.per.sample$percent.zeros <- 100*stats.per.sample$zeros/nrow(count.table)

# View(stats.per.sample)
kable(stats.per.sample[sample(1:ncol(count.table), size = 10),],
      caption = "##Table: statistics per sample. ** We only display a random selection of 10 samples. ")
```

Table 1: **Table: statistics per sample.** We only display a random selection of 10 samples.

	Min.	X1st.Qu.	Median	Mean	X3rd.Qu.	Max.	libsum	perc05	perc10	perc90	perc95
Snf15	0	80	345	1180	815	252000	8422845	0	1	1976	3661
Snf9	0	115	495	1410	1110	211000	10025640	0	2	2530	4496
Snf38	0	88	382	1180	878	215000	8408882	0	2	2084	3724
WT14	0	102	432	1350	1020	253000	9606909	0	2	2464	4349
Snf22	0	85	338	945	751	140000	6735821	0	2	1730	3066
WT31	0	83	369	1210	912	211000	8608070	0	2	2160	3892
Snf18	0	107	439	1170	985	160000	8358221	0	2	2204	3772
Snf27	0	75	317	932	727	143000	6638966	0	1	1698	2998
WT45	0	100	457	1620	1120	339000	11575895	0	2	2752	5098
WT32	0	60	248	918	601	242000	6540347	0	1	1576	2827

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",
     main="", xlab="log2(counts)", ylab="Number of genes")

## Boxplots
boxplot(log2(count.table + epsilon), col=expDesign$color, pch=".",
        horizontal=TRUE, cex.axis=0.5,
        las=1, ylab="Samples", xlab="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 + epsilon), lty=1, col=expDesign$color, lwd=2)
grid()
legend("topright", legend=names(col.strain), col=col.strain, lwd=2)
```

Beware: the R function *plotDensity()* does not display the actual distribution of your values, but a polynomial

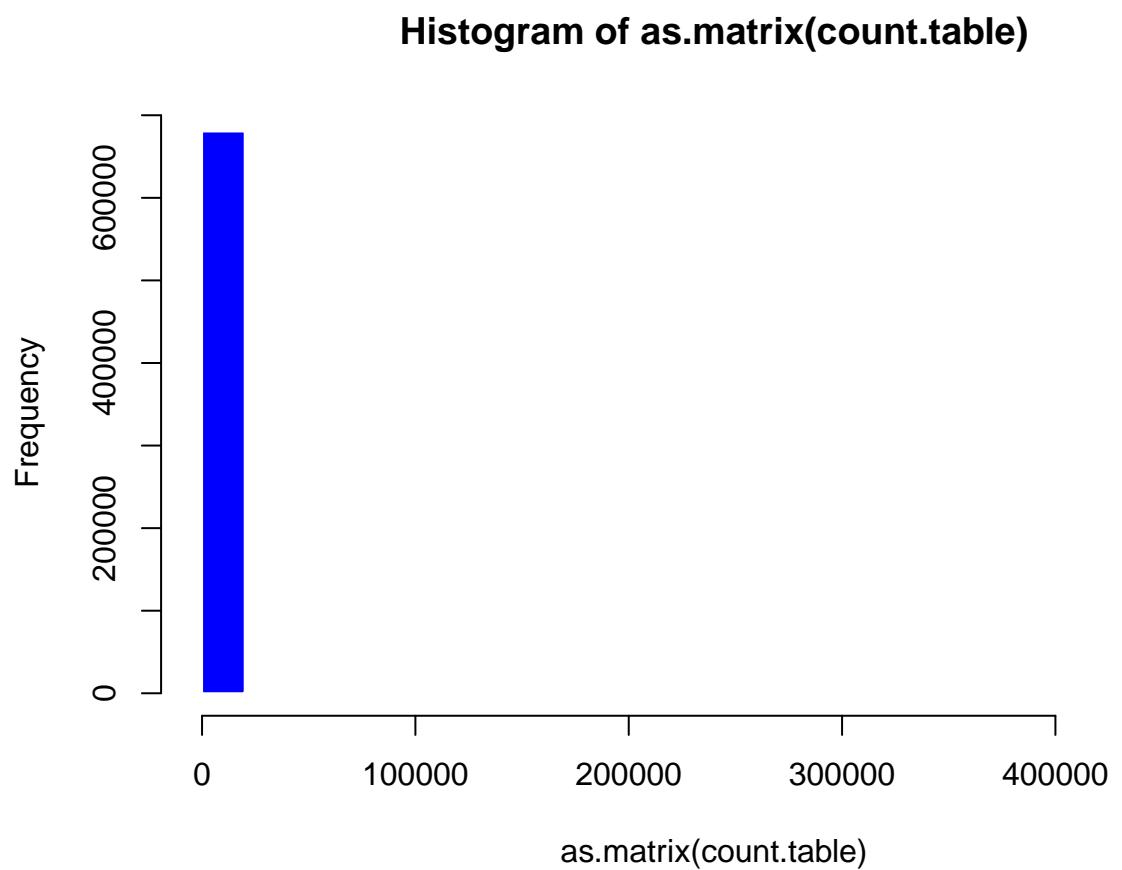


Figure 2: Histogram of raw counts. The scale is determined by the gene with the highest count, which is apparently an outlier.

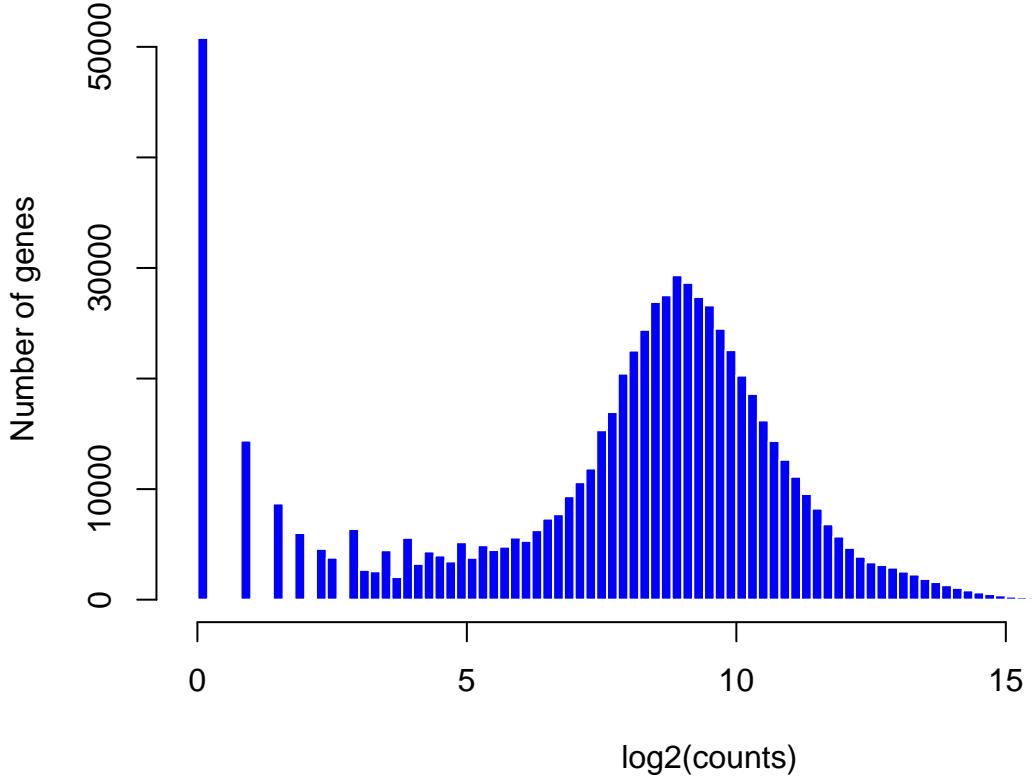


Figure 3: Histogram of log2-transformed counts per gene.

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.

```
## Define a function to draw a scatter plot for a pair of variables (samples) with density colors
plotFun <- function(x,y){
  dns <- densCols(x,y);
  points(x,y, col=dns, pch=". ", panel.first=grid());
  # abline(a=0, b=1, col="brown")
}

## Plot the scatter plot for a few pairs of variables selected at random
set.seed(123) # forces the random number generator to produce fixed results
pairs(log2(count.table[,sample(ncol(count.table),5)] + epsilon),
      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 the 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

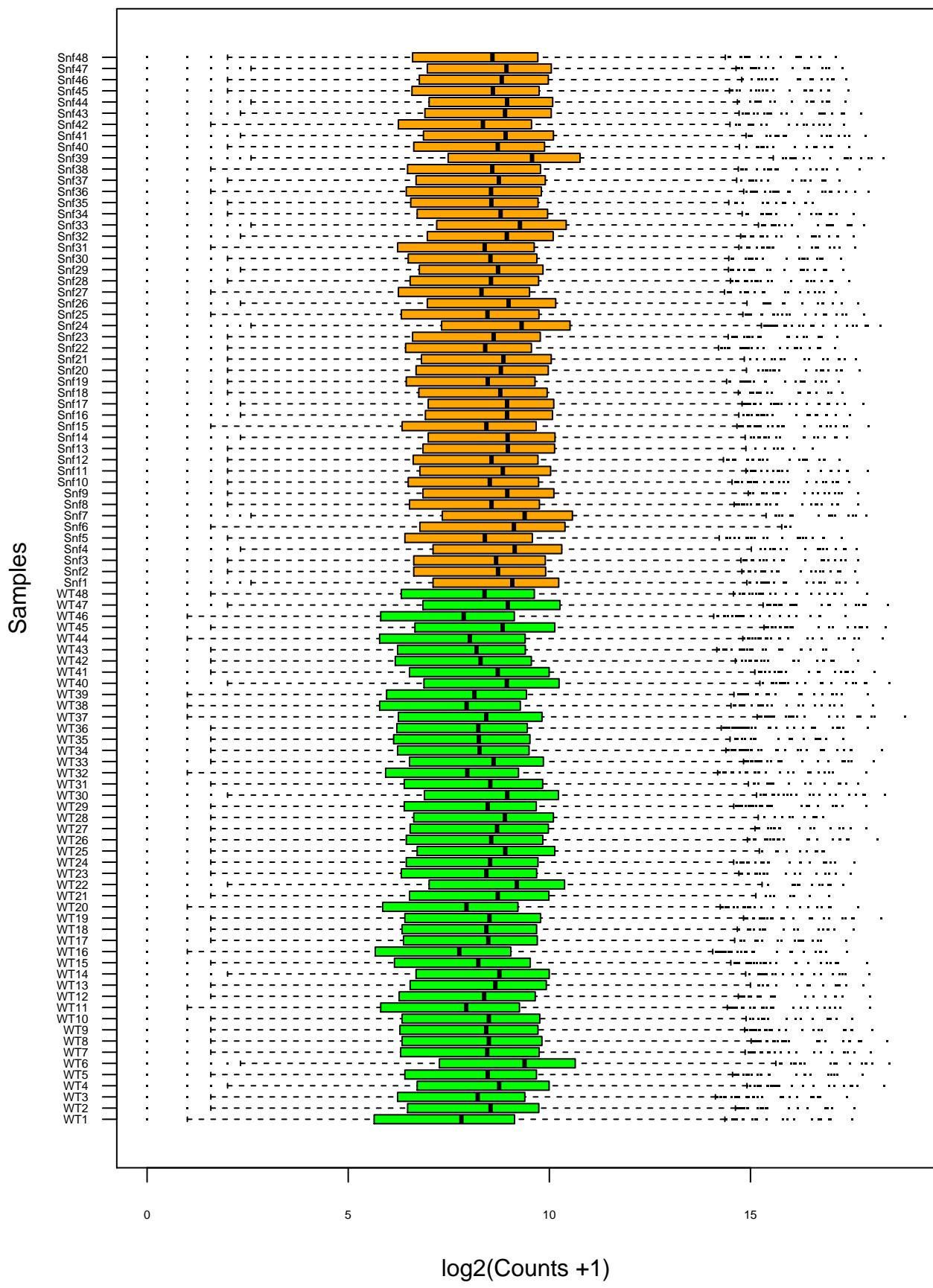


Figure 4: Box plots of non-normalized $\log_2(\text{counts})$ per sample.

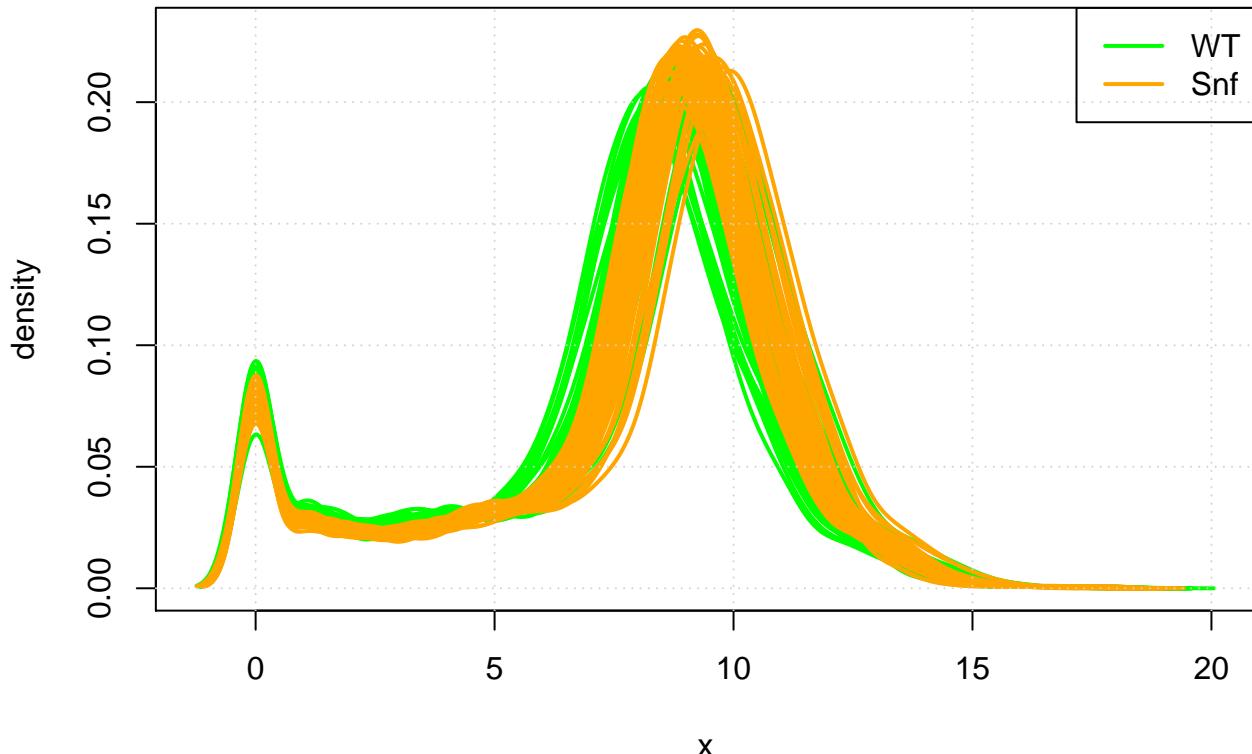


Figure 5: Densities of $\log_2(\text{counts})$. Each curve corresponds to one sample.

from the diagonal.

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",
        horiz=TRUE, cex.names=0.5, las=1,
        col=expDesign$color, ylab='Samples', xlab='% of null counts')
```

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

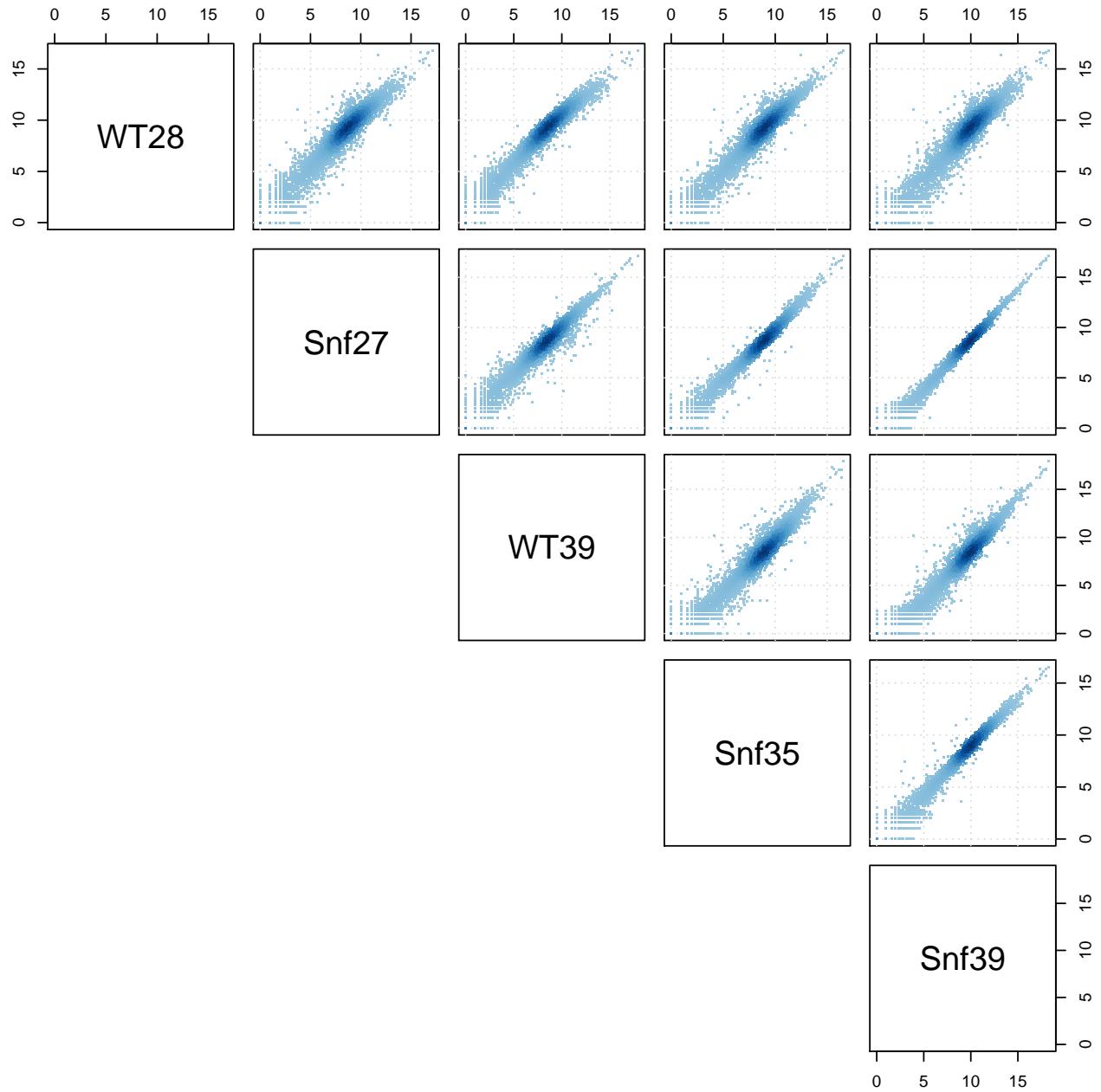


Figure 6: Scatter plot of log2-counts for a random selection of samples.

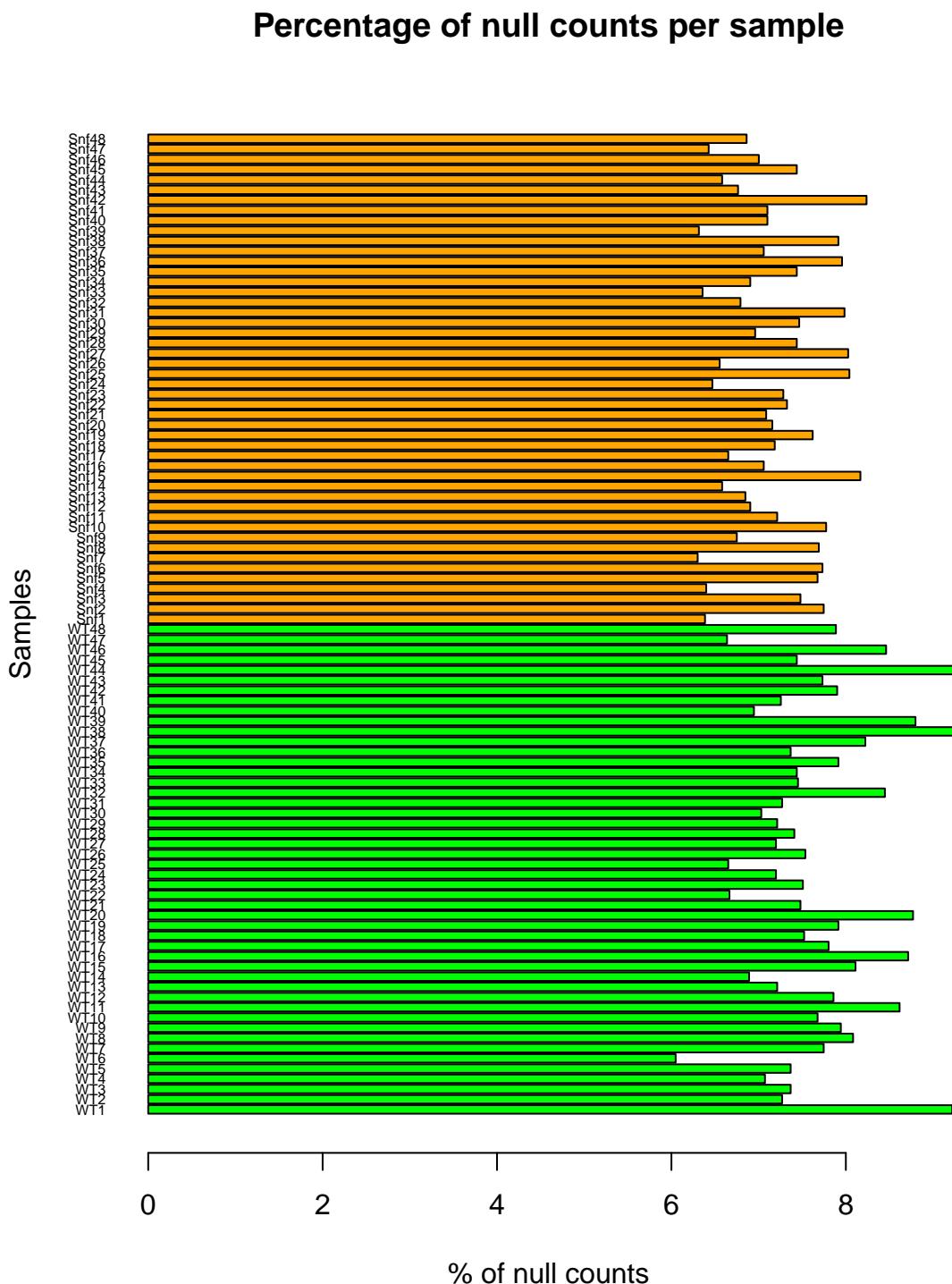


Figure 7: Percentage of null counts per sample.

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 <- expDesign$color[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")){
  install.packages("lazyeval")
  install.packages("ggplot2")

  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[selected.samples,], design = ~genotype)
print(dds0)

class: DESeqDataSet
dim: 6887 20
metadata(1): version
assays(1): counts
rownames(6887): 15s_rrna 21s_rrna ...
rowData names(0):
```

```

colnames(20): WT3 WT25 ... Snf40 Snf26
colData names(3): label strain color
## 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:

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

```

### 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 subsequent division 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)

```

```

WT3  WT25  WT42  WT47  WT21  WT46  WT20  WT28  WT23  WT5  Snf44  Snf12
0.771 1.218 0.820 1.296 1.075 0.619 0.644 1.197 0.889 0.903 1.283 0.994
Snf2  Snf15  Snf42  Snf39  Snf30  Snf27  Snf40  Snf26
1.109 0.924 0.869 2.004 0.962 0.843 1.098 1.339

```

```

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

```

```

WT3  WT25  WT42  WT47  WT21  WT46
0.771 1.218 0.820 1.296 1.075 0.619

```

```

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)+epsilon), col=col.pheno.selected, cex.axis=0.7,
        las=1, xlab="log2(counts)", horizontal=TRUE, main="Raw counts")
boxplot(log2(counts(dds.norm, normalized=TRUE)+epsilon), col=col.pheno.selected, cex.axis=0.7,
        las=1, xlab="log2(normalized counts)", horizontal=TRUE, main="Normalized counts")
plotDensity(log2(counts(dds.norm)+epsilon), col=col.pheno.selected,
            xlab="log2(counts)", cex.lab=0.7, panel.first=grid())
plotDensity(log2(counts(dds.norm, normalized=TRUE)+epsilon), col=col.pheno.selected,
            xlab="log2(normalized counts)", cex.lab=0.7, panel.first=grid())

```

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
norm.counts <- counts(dds.norm, normalized=TRUE)
mean.counts <- rowMeans(norm.counts)
variance.counts <- apply(norm.counts, 1, var)

```

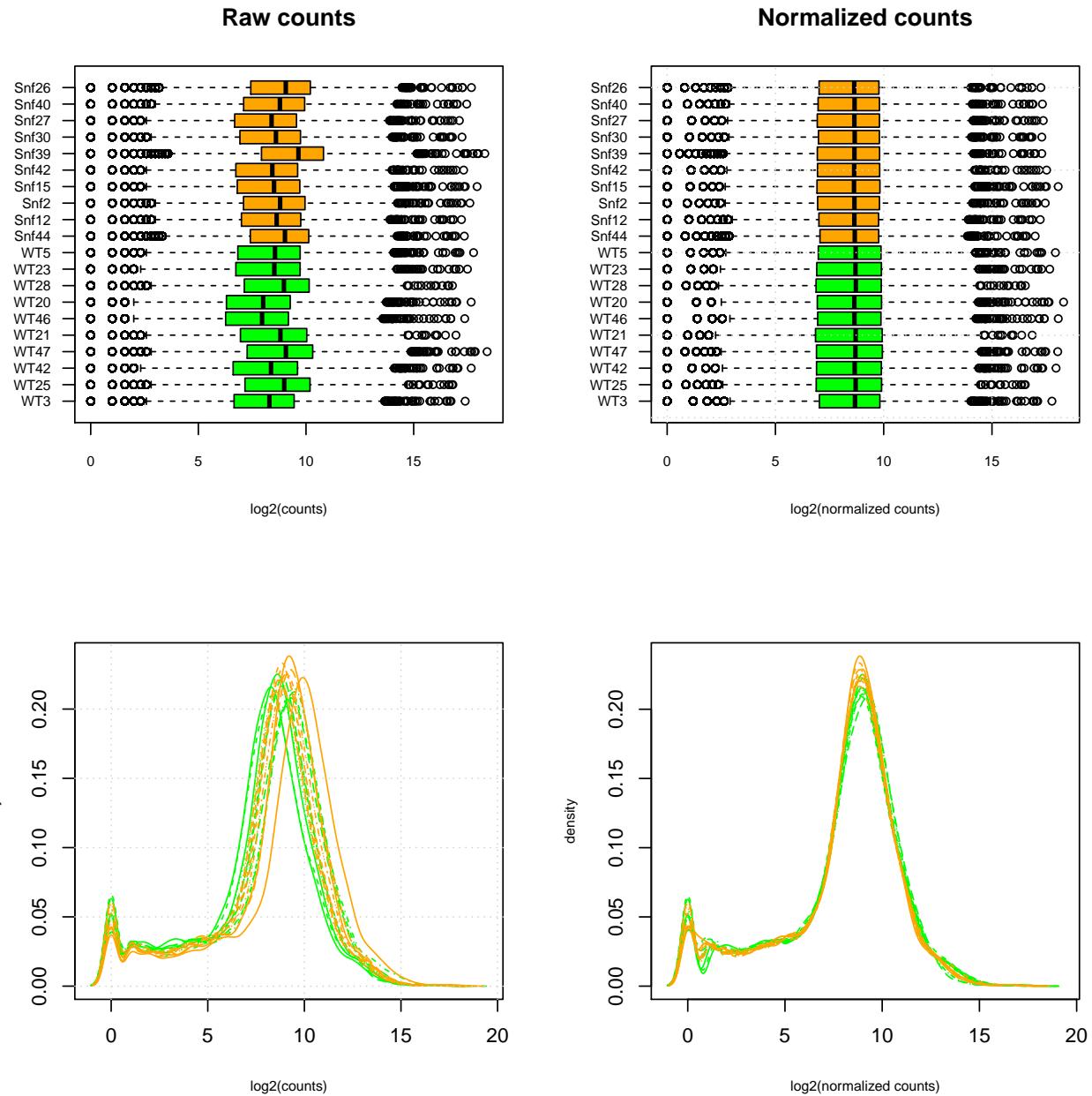


Figure 8: Impact of the count normalization.

```

## sum(mean.counts==0) # Number of completely undetected genes

norm.counts.stats <- data.frame(
  min=apply(norm.counts, 2, min),
  mean=apply(norm.counts, 2, mean),
  median=apply(norm.counts, 2, median),
  max=apply(norm.counts, 2, max),
  zeros=apply(norm.counts==0, 2, sum),
  percent.zeros=100*apply(norm.counts==0, 2, sum)/nrow(norm.counts),
  perc05=apply(norm.counts, 2, quantile, 0.05),
  perc10=apply(norm.counts, 2, quantile, 0.10),
  perc90=apply(norm.counts, 2, quantile, 0.90),
  perc95=apply(norm.counts, 2, quantile, 0.95)
)

kable(norm.counts.stats)

```

	min	mean	median	max	zeros	percent.zeros	perc05	perc10	perc90
WT3	0	1213	407	223341	286	4.15	1.298	6.49	2162
WT25	0	981	416	95177	235	3.41	0.821	5.75	2016
WT42	0	1296	404	254884	324	4.71	1.219	6.10	2300
WT47	0	1353	413	268933	234	3.40	0.771	6.17	2323
WT21	0	957	417	118777	294	4.27	0.930	5.58	2097
WT46	0	1309	400	273555	364	5.29	0.000	6.46	2280
WT20	0	1419	399	324143	386	5.61	0.000	4.66	2431
WT28	0	978	419	95311	289	4.20	0.835	4.18	2029
WT23	0	1248	414	208093	296	4.30	1.125	5.62	2240
WT5	0	1289	417	249736	286	4.15	1.108	6.65	2163
Snf44	0	1005	405	130568	230	3.34	0.780	7.02	1848
Snf12	0	1081	399	153508	253	3.67	1.006	7.04	1940
Snf2	0	1138	400	180318	313	4.54	0.902	5.41	2066
Snf15	0	1323	395	272840	343	4.98	1.082	5.41	2219
Snf42	0	1152	397	188354	348	5.05	0.000	5.75	2103
Snf39	0	1100	400	161407	211	3.06	0.998	5.99	2034
Snf30	0	1091	404	161522	293	4.25	1.040	6.24	1985
Snf27	0	1143	400	169171	333	4.83	1.186	5.93	2076
Snf40	0	1094	402	164662	267	3.88	0.911	5.47	2016
Snf26	0	1064	398	157066	228	3.31	0.747	5.97	1951

```

## Mean and variance relationship
mean.var.col <- densCols(x=log2(mean.counts), y=log2(variance.counts))
plot(x=log2(mean.counts), y=log2(variance.counts), pch=16, cex=0.5,
  col=mean.var.col, main="Mean-variance relationship",
  xlab="Mean log2(normalized counts) per gene",
  ylab="Variance of log2(normalized counts)",
  panel.first = grid())
abline(a=0, b=1, col="brown")

```

Mean–variance relationship

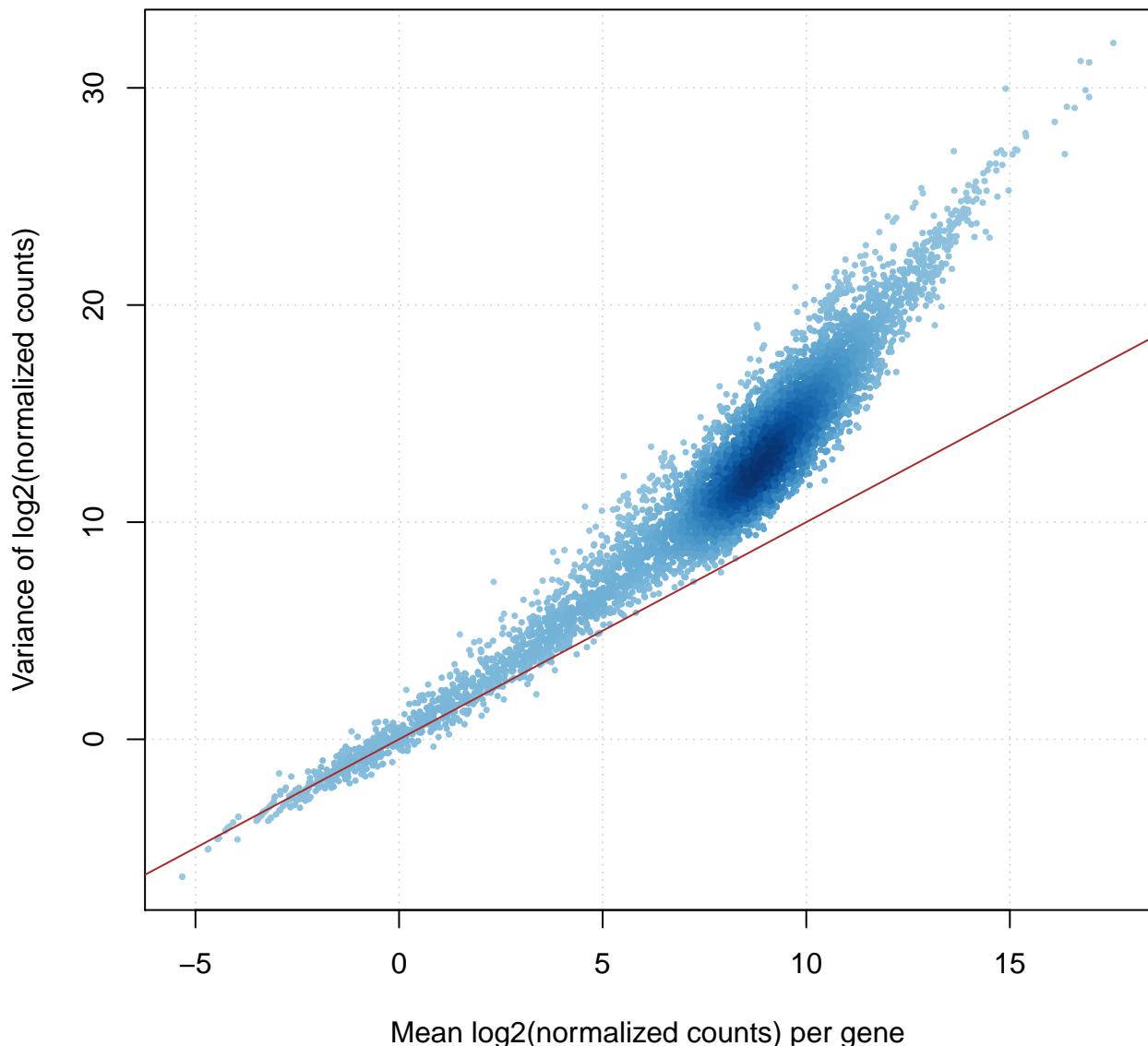


Figure 9: **Figure: variance/mean plot.** The brown line highlights $x = y$, which corresponds to the expected relationship between mean and variance for a Poisson distribution.

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 slightly 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 responses 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 successes 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 Bernoulli 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 **dnbino $m()$** function of R, n and p are denoted by arguments **size** and **prob** respectively).

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

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+n-1}^x 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 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
```

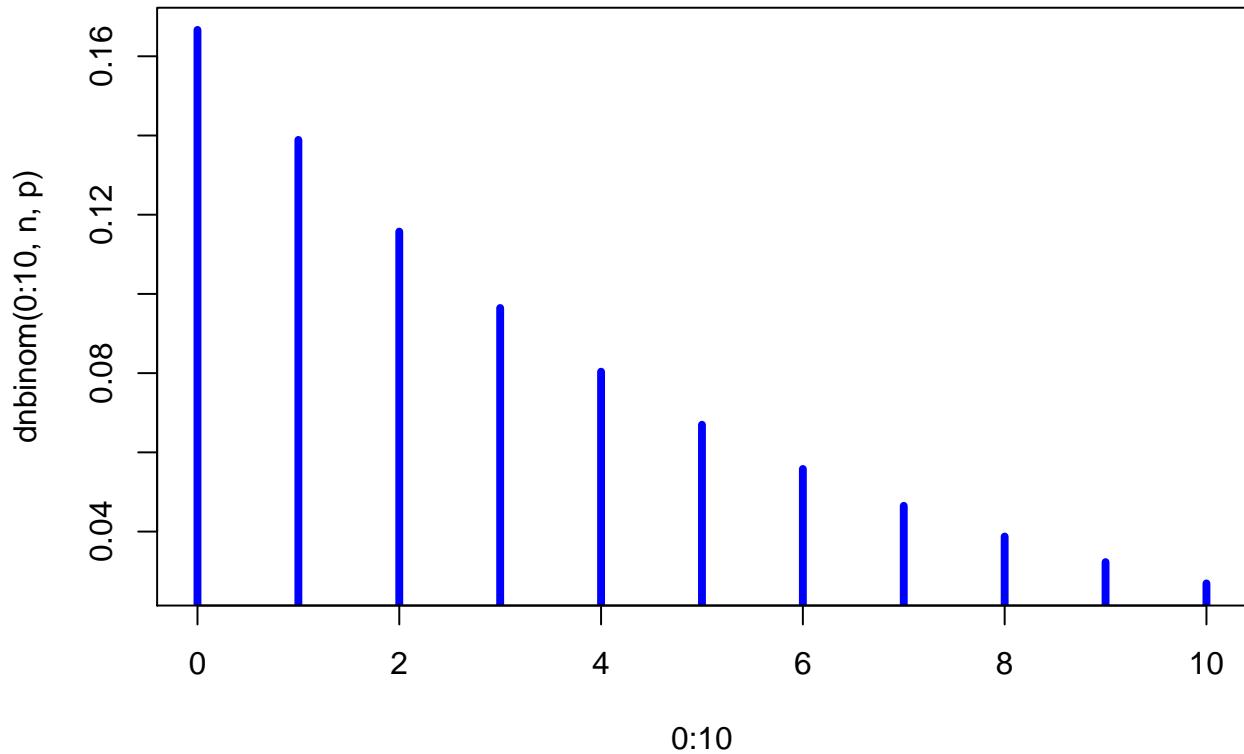


Figure 10: Negative binomial distribution.

```
# 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)",
```

Negative binomial density

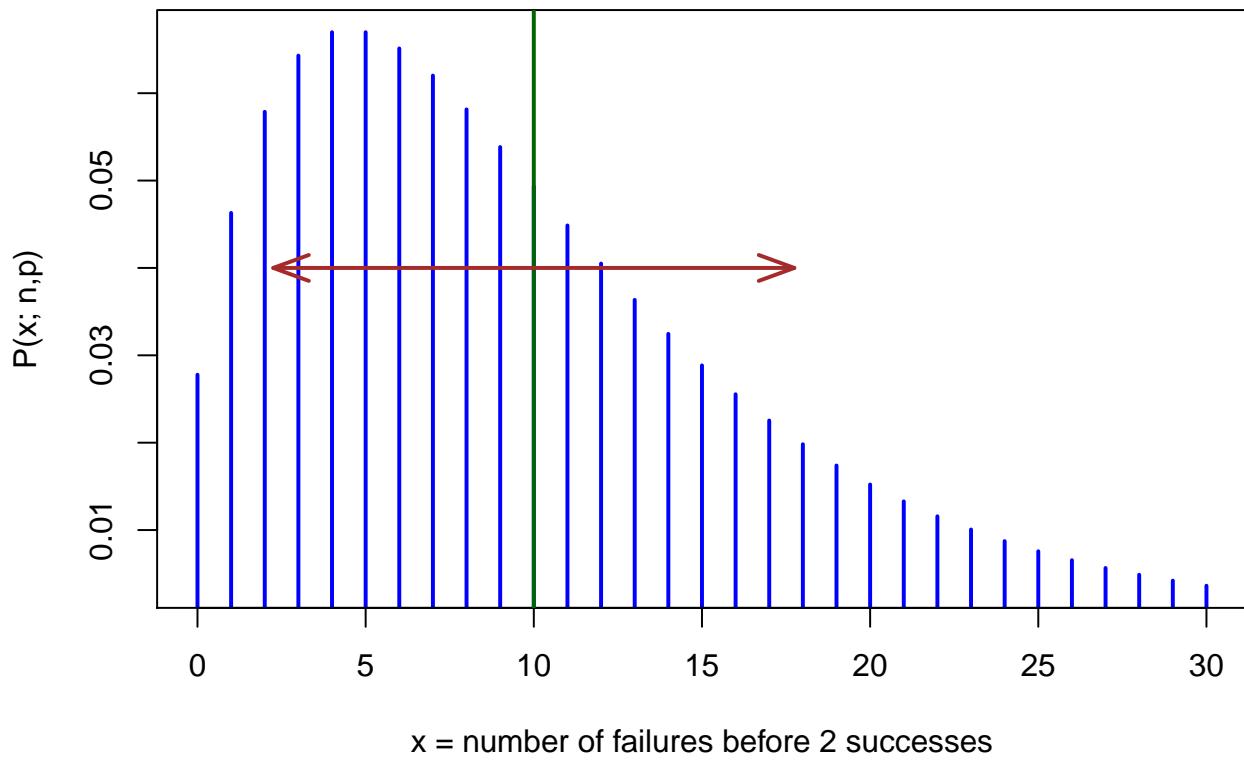


Figure 11: Negative binomial distribution.

```

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=90)

```

Using mean and dispersion

The second way of parametrizing the distribution is using the mean value m and the dispersion parameter r (in `dtnbinom()` 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 differential 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.

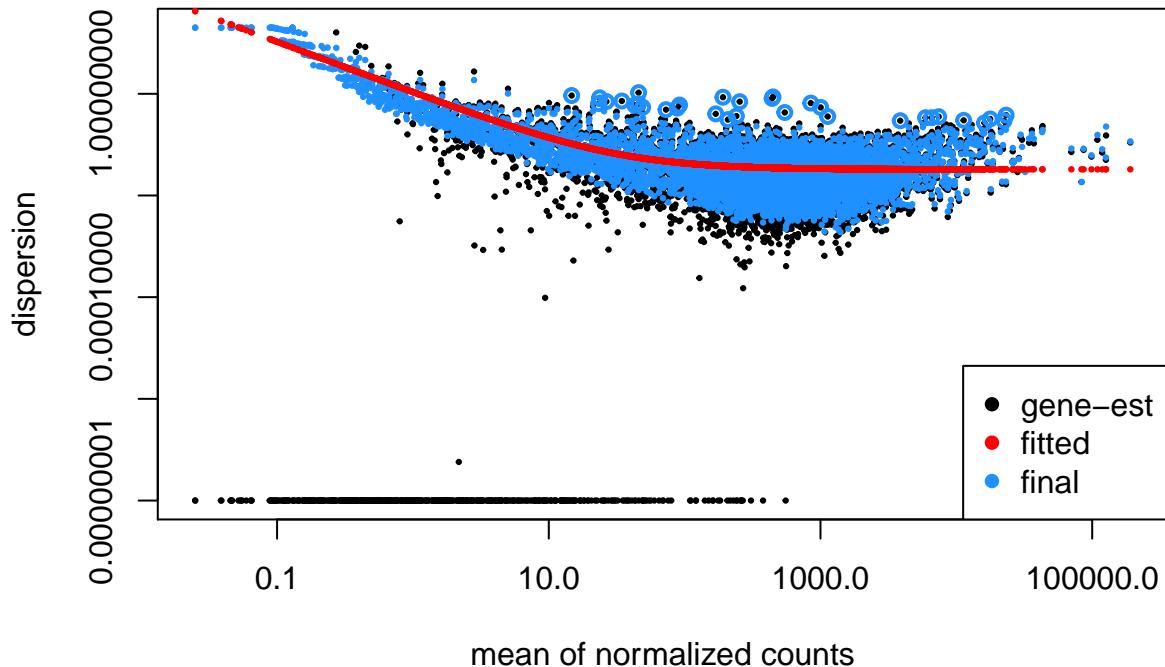
- The mean will be estimated from the observed normalized counts in both conditions.
- The first step will be to compute a gene-wise dispersion. When the number of available samples is insufficient to obtain a reliable estimator of the variance for each gene, DESeq will apply a **shrinkage** strategy, 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
plotDispEts(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()
class(res.DESeq2)

[1] "DESeqResults"
attr(,"package")
[1] "DESeq2"

is(res.DESeq2) # a data.frame

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

slotNames(res.DESeq2)

[1] "rownames"         "nrows"           "listData"         "elementType"
[5] "elementMetadata"  "metadata"

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      14.74       0.896    0.4289     2.09  0.036655  0.0705
21s_rrna      91.73       0.979    0.3858     2.54  0.011192  0.0249
hra1          2.75        0.681    0.3860     1.77  0.077515  0.1340
icr1          142.71      -0.189   0.0911    -2.07  0.038038  0.0728
lsr1          202.49       0.851    0.2522     3.37  0.000743  0.0022
nme1          24.92        0.298    0.2207     1.35  0.176897  0.2688

## 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-value
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>
```

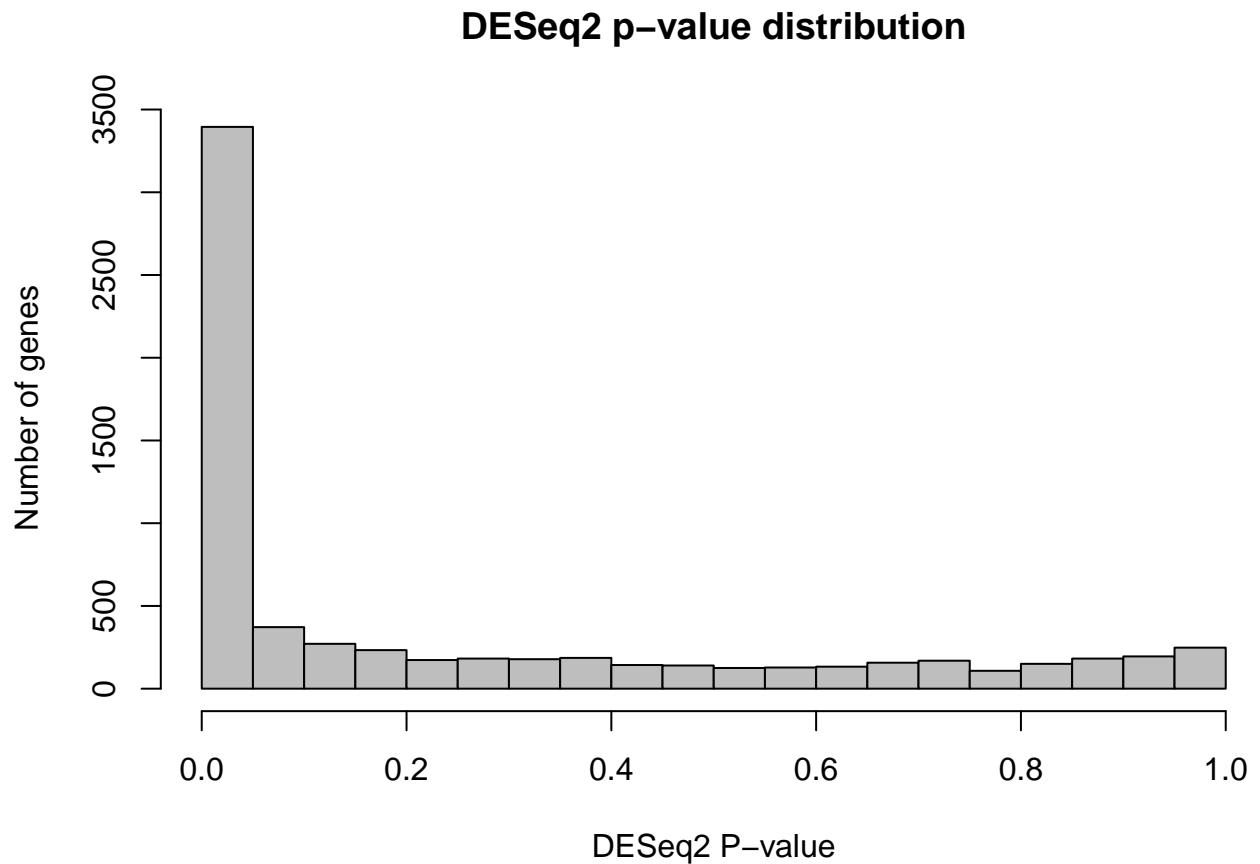


Figure 12: Histogram of the p-values reported by DESeq2.

yar071w	1483	4.00	0.0973	41.1	0	0
yer001w	670	2.35	0.0620	37.9	0	0
ygr234w	2823	4.10	0.0984	41.6	0	0
yhr215w	1169	4.16	0.1090	38.1	0	0
yil121w	860	2.79	0.0570	49.0	0	0
yor290c	878	7.16	0.1806	39.7	0	0

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

Volcano plot

```
alpha <- 0.01 # Threshold on the adjusted p-value
cols <- densCols(res.DESeq2$log2FoldChange, -log10(res.DESeq2$pvalue))
plot(res.DESeq2$log2FoldChange, -log10(res.DESeq2$padj), col=cols, panel.first=grid(),
     main="Volcano plot", xlab="Effect size: log2(fold-change)", ylab="-log10(adjusted p-value)",
     pch=20, cex=0.6)
abline(v=0)
abline(v=c(-1,1), col="brown")
abline(h=-log10(alpha), col="brown")

gn.selected <- abs(res.DESeq2$log2FoldChange) > 2 & res.DESeq2$padj < alpha
```

Volcano plot

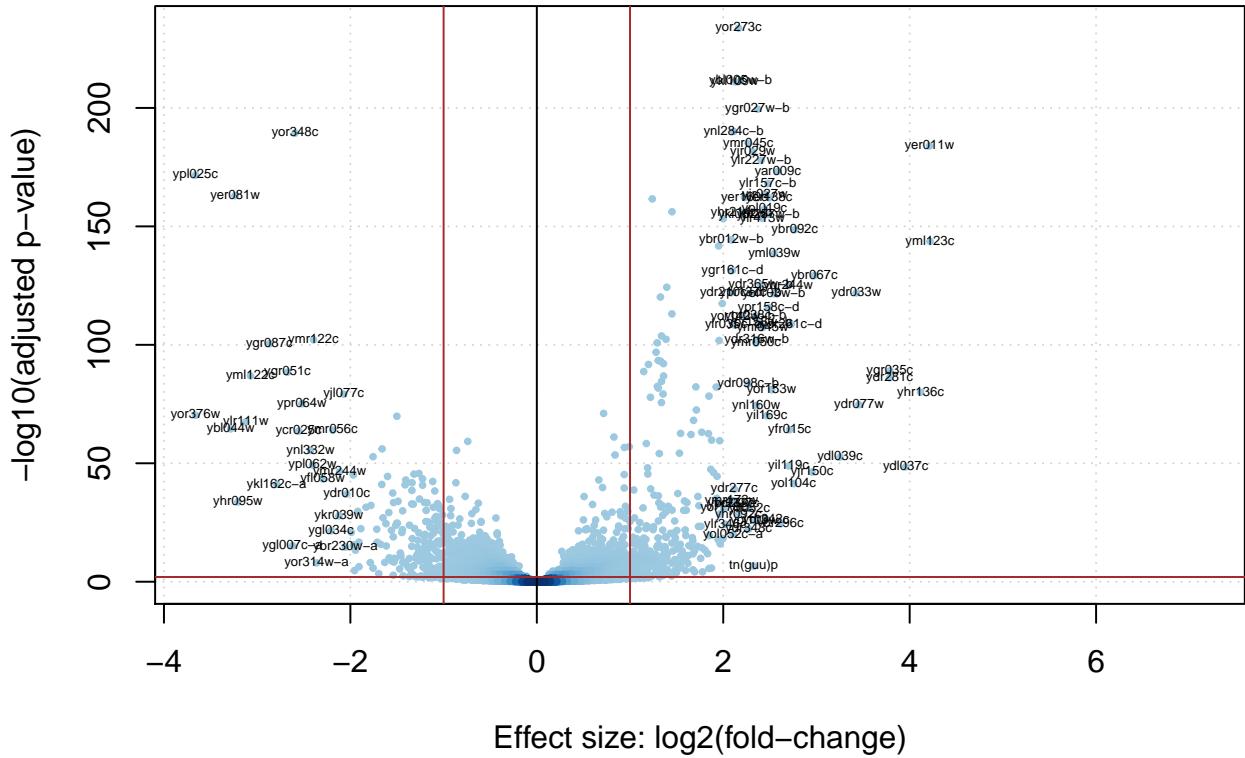


Figure 13: Volcano plot of DESeq2 results. Abcsissa: $\log_2(\text{fold-change})$. Ordinate: significance ($-\log_{10}(P - \text{value})$).

```
text(res.DESeq2$log2FoldChange[gn.selected],
     -log10(res.DESeq2$padj)[gn.selected],
     lab=rownames(res.DESeq2)[gn.selected], cex=0.4)
```

Check the expression levels of the most differentially expressed gene

It may be important to check the validity of our analysis by simply assessing the expression level of the most highly differential gene.

```
gn.most.sign <- rownames(res.DESeq2)[1]
gn.most.diff.val <- counts(dds.norm, normalized=T)[gn.most.sign,]
barplot(gn.most.diff.val, col=col.pheno.selected, main=gn.most.sign, las=2, cex.names=0.5)
```

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.

$$M_g = \log_2(\bar{x}_{g,a}) - \log_2(\bar{x}_{g,b})$$

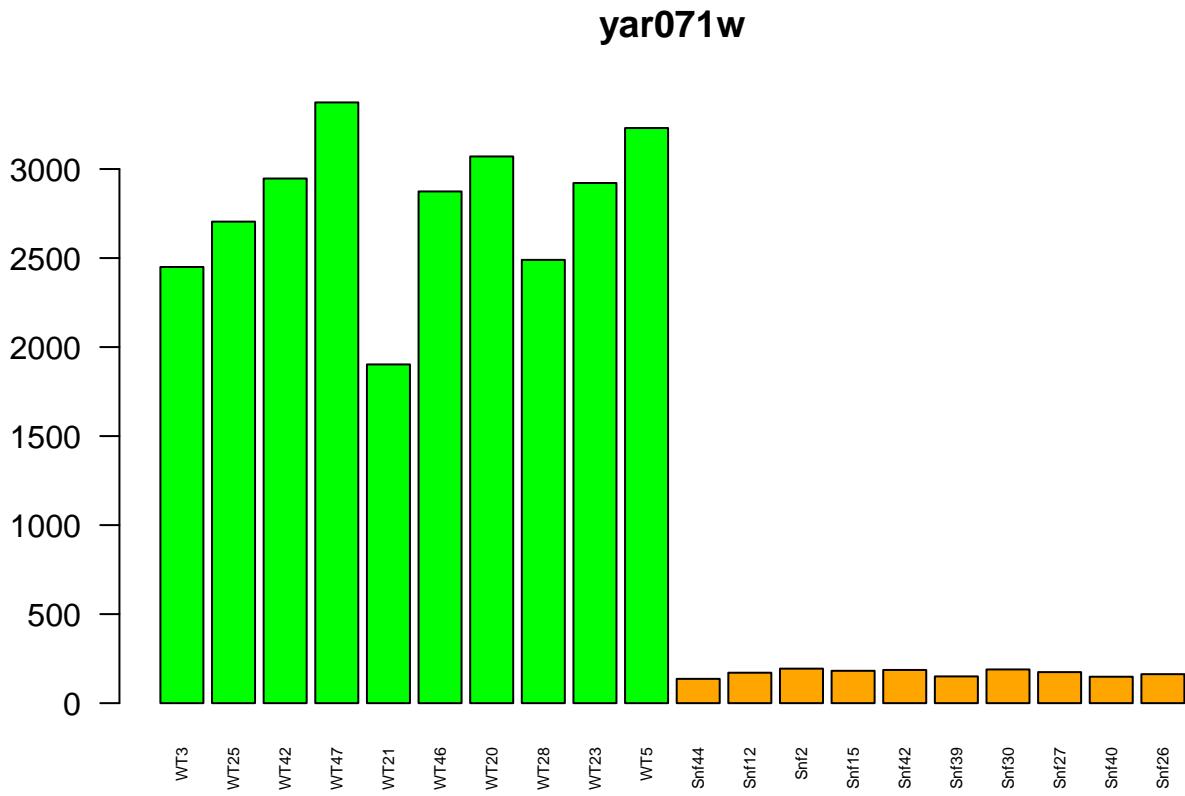


Figure 14: Barplot of the counts per sample fr a selected gene.

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

$$A_g = \frac{1}{2}(\log_2(\bar{x}_g, a) + \log_2(\bar{x}_g, b))$$

```
## 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 + epsilon)[gene.kept, ]
dim(count.table.kept)
```

[1] 2753 96

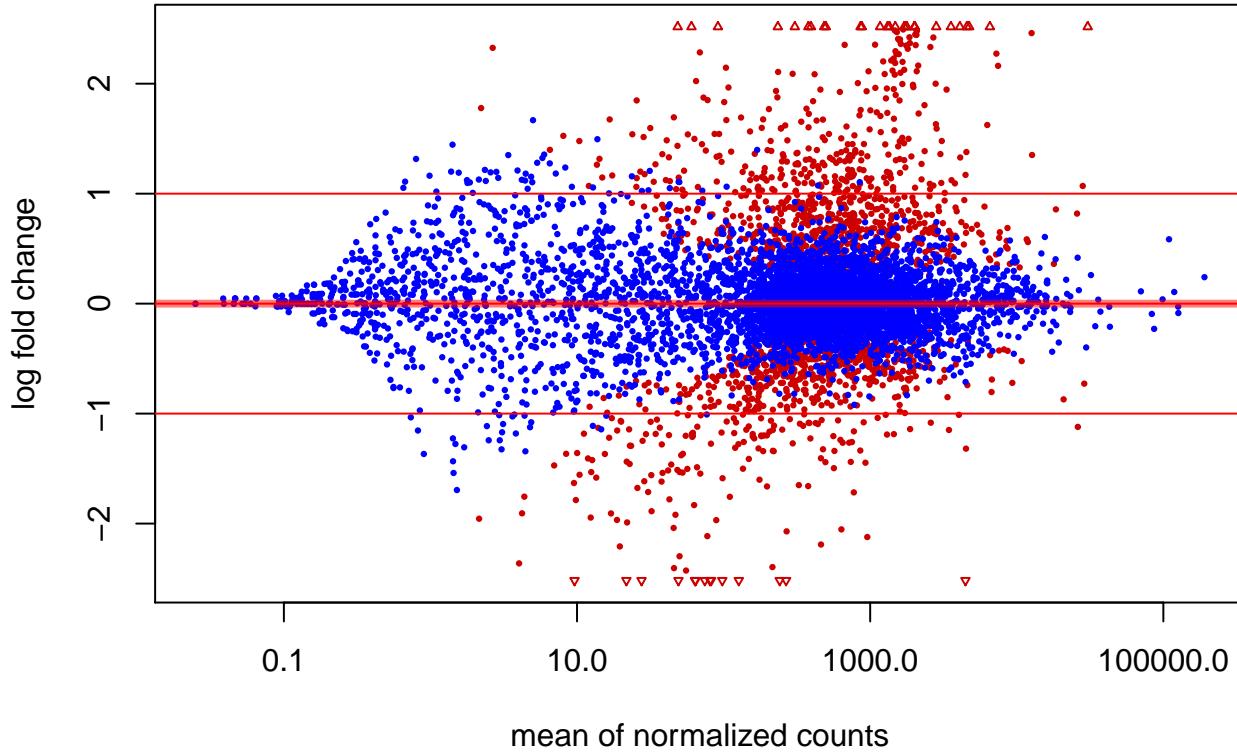


Figure 15: MA plot. The abcsissa indicates the mean of normalized counts; the ordinate the $\log_2(\text{fold-change})$.

```
## 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)
```

Functional enrichment

We will now perform functional enrichment using the list of induced genes. This step will be performed using the gProfileR R library.

```
library(gProfileR)

res.DESeq2.df <- na.omit(data.frame(res.DESeq2))
```

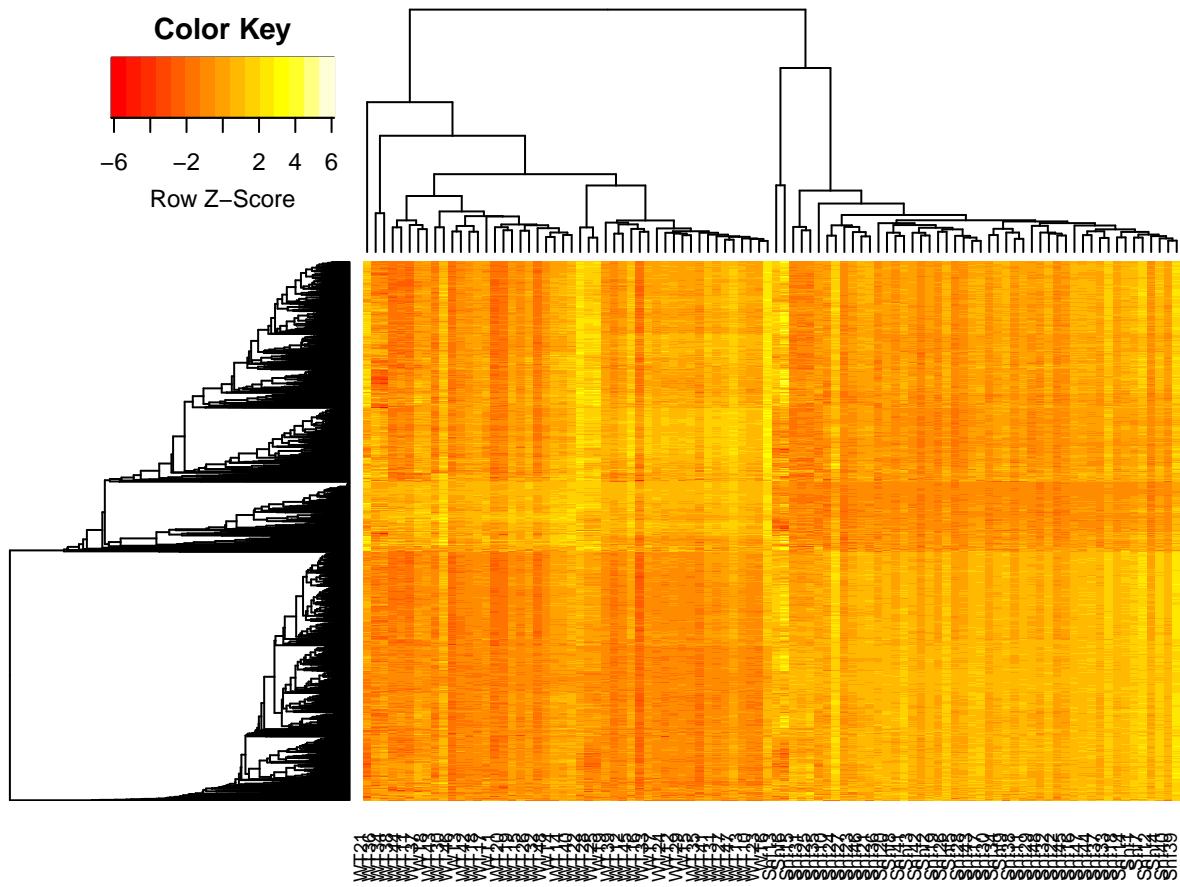


Figure 16: Heatmap of the genes declared significant with DESeq2. Rows correspond to genes, columns to samples.

```

induced.sign <- rownames(res.DESeq2.df)[res.DESeq2.df$log2FoldChange >= 2 & res.DESeq2.df$padj < alpha]
# head(induced.sign)
# names(term.induced)

term.induced <- gprofiler(query=induced.sign, organism="scerevisiae")
term.induced <- term.induced[order(term.induced$p.value),]
# term.induced$p.value
kable(term.induced[1:10,c("term.name",
                           "term.size",
                           "query.size",
                           "overlap.size",
                           "recall",
                           "precision",
                           "p.value",
                           "intersection")],
      format.args=c(engineer=TRUE, digits=3), caption="**Table: functional analysis with gProfiler. **")

```

	term.name
38	RNA-DNA hybrid ribonuclease activity
6	DNA integration
29	RNA-directed DNA polymerase activity
41	aspartic-type peptidase activity
43	aspartic-type endopeptidase activity
37	endoribonuclease activity, producing 5'-phosphomonoesters
28	DNA-directed DNA polymerase activity
27	DNA polymerase activity
35	endonuclease activity, active with either ribo- or deoxyribonucleic acids and producing 5'-phosphomonoesters
36	endoribonuclease activity

And now using the list of repressed genes.

```

res.DESeq2.df <- na.omit(data.frame(res.DESeq2))
repressed.sign <- rownames(res.DESeq2.df)[res.DESeq2.df$log2FoldChange <= -2 & res.DESeq2.df$padj < alpha]
head(repressed.sign)

[1] "yor348c" "ypl025c" "yer081w" "ymr122c" "ygr087c" "ygr051c"

term.repressed <- gprofiler(query=repressed.sign, organism="scerevisiae")
term.repressed <- term.repressed[order(term.repressed$p.value),]
kable(head(term.repressed[,c("p.value", "term.name", "intersection")]), 10))

```

	p.value	term.name
38	0	RNA-DNA hybrid ribonuclease activity
6	0	DNA integration
29	0	RNA-directed DNA polymerase activity
41	0	aspartic-type peptidase activity
43	0	aspartic-type endopeptidase activity
37	0	endoribonuclease activity, producing 5'-phosphomonoesters
28	0	DNA-directed DNA polymerase activity
27	0	DNA polymerase activity
35	0	endonuclease activity, active with either ribo- or deoxyribonucleic acids and producing 5'-phosphomonoesters

	p.value	term.name
36	0	endoribonuclease activity

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.

```
## Create a directory to store the results that will be obtained below
dir.results <- file.path(dir.snf2, "results")
dir.create(dir.results, showWarnings = FALSE, recursive = TRUE)

## Export the table with statistics per sample.
write.table(stats.per.sample, file=file.path(dir.results, "stats_per_sample.tsv"),
            quote=FALSE, sep="\t", col.names =NA, row.names = TRUE)

# Export the DESeq2 result table
DESeq2.table <- file.path(dir.results, "yeast_Snf2_vs_WT_DESeq2_diff.tsv")
write.table(res.DESeq2, file=DESeq2.table, col.names = NA, row.names = TRUE, sep="\t", quote = FALSE)
```