7 - Differential expression analysis

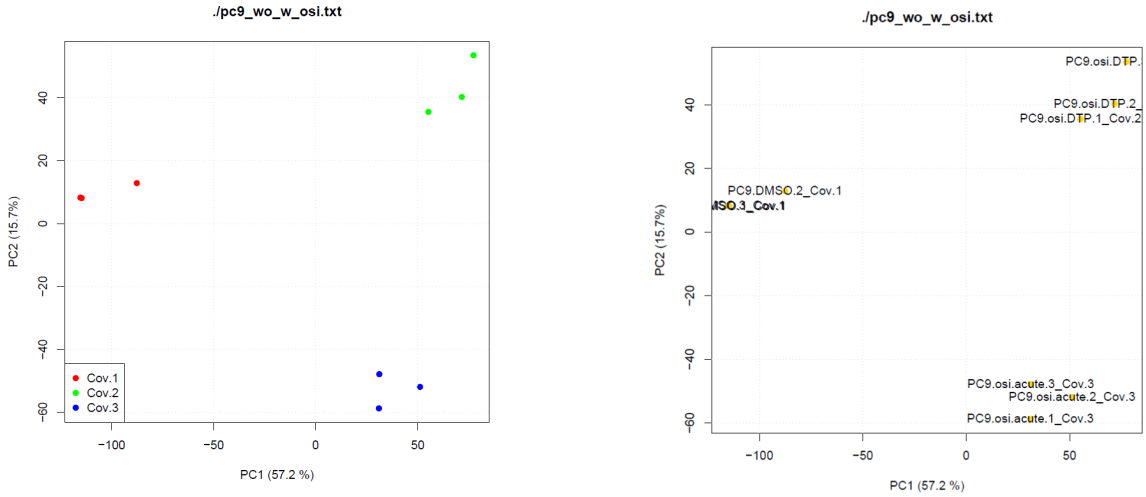
TOPICS	Differential expression analysis
DATE	@April 27, 2022 2:00 PM
LAST EDITED BY	
LAST EDITED TIME	@May 15, 2022 5:05 PM
MADE BY	
PROF	Calogero
Recording	
STATUS	☑

Solution of last time exercises

We had to made the PCA for bulkRNAseq data using the PCA function of Docker4Seq.

```
# in this case you don't give the name to the dots because the sampleNames = FALSE -->left image
library(docker4seq) #first you have to load and recall the library that we are going to use
pca(
  experiment.table = "./pc9_wo_w_osi.txt", #name of the file with tha data
  type = c("counts"),
  covariatesInNames = TRUE,
  samplesName = FALSE,
  principal.components = c(1, 2),  # 1 and 2 for PC1 and PC2
  legend.position = c("bottomleft"),
  pdf = TRUE,
  output.folder = getwd()
)
#you need to rename the file otherwise it gets overwritten
file.rename("pca.pdf", "nameofchoice.pdf")

#in this case you give a name to the dots because the sampleNames=TRUE-->right image
library(docker4seq)
pca(
  experiment.table = "./pc9_wo_w_osi.txt",
  type = c("counts"),
  covariatesInNames = FALSE,
  samplesName = TRUE,
  principal.components = c(1, 2),
  legend.position = c( "bottomleft"),
  pdf = TRUE,
  output.folder = getwd()
)
#also here you can give the output file the name you like with
file.rename("pca.pdf", "nameofchoice2.pdf")
```

Running the function above you get a two .pdf file that contain a plot as we can see below.



Looking at the plot is visible that there is a big difference between the untreted and the two treated group (this is what we want).

The difference among the two treatments (acute and chronic) is small bacause the main difference is given by PC2 rather than PC1 (red dots =DMSO, green dots=chronic treatment, blue dots=acute treatment).

> ⚠️ For a more detailed description of this function, see lesson 6.

What we see before was done for bulkRNAseq data, now we have to focus on scRNAseq data that use a different apporach (tSNE or UMAP). We want to evaluate the characteristics of single-cell data looking at the difference between acute treatment vs no-treatment. First, we have to look at the structure of the untreated cell line (no treatment) and we ask if it is homogeneous or if it has subpopulations.

Imputation concept. The single-cell data are 0 inflated (dataset that contain a lots of 0 values) for this reason do not really represent the real RNA expression. This is something stochastical, during the extraction of polyA-RNA some molecules may get lost. This problem can be partially moderated using a statistical tool called **SAVER**. It guesses the expected behavior of the expression of the data looking gene by gene and imputes the value for those data. Imputation helps in improving the quality of the analysis. The data provided to us already had SAVER imputed data. We needed to convert them in *log2cpm* (it is a sort of normalization for visualization purposes) before running tSNE e UMAP function.

Function for the conversion in log2cpm

```
# Create the function counts2cpm
counts2cpm <- function(file, sep = ","){
  tmp <- read.table(file, sep=sep, header=T, row.names=1)
  col.sum <- apply(tmp, 2, sum)
  tmp1 <- t(tmp)/col.sum
  tmp1 <- t(tmp1)
  tmp1 <- tmp1 * 1000000
  write.table(tmp1, "cpm.csv", sep=",",
              col.names=NA)
  write.table(log2(tmp1 + 1), "log2cpm.csv",
              sep=",", col.names=NA)
}

# Recall the function and run it on your ctrl file
counts2cpm(file="saver_ctrl.csv", sep=",")
file.rename(from="log2cpm.csv", to="ctrl_log2CPM.csv")

# Recall again the function and run it on your osi file
counts2cpm(file="saver_osi.csv", sep=",")
file.rename(from="log2cpm.csv", to="osi_log2CPM.csv")
```

The output is a list with the first element being a dataframe of 2 columns. This is a dataframe for visualization of the data. You can also change the dimensions you want to represent, but we will stick to 2D only.

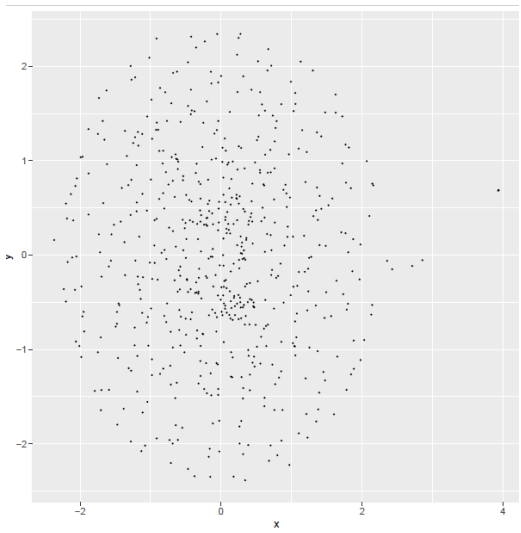
We decide to use two different tools: tSNE and UMAP

tSNE function and plot

What simply tSNE does is to traspose the matrix with `t` function. To run tSNE we are using the vary basic parameter that are put as default. The tSNE results is a list (contaneir that containt different types of data). The first part of the element of the list is a data frame of two coloums that represent the x-y cooridinate that are used to plot tSNE and have a image for the visualization. Using tSNE you can change some parameter such as the number of dimension that you want to represent (you can decide to have more dimension not only 2).

What the function does is to create a data frame contaning the two colums (x-y cooridnates) of the first element of the list geenreted by tSNE and plot the data frame using the `ggplot` function.

```
# Open an R session an type the following commands
# Do tSNE on ctrl samples first
library(Rtsne)
library(ggplot2)
tmp <- read.table("ctrl_log2CPM.csv", sep=",", header=T, row.names=1)
tmp.labels <- sapply(strsplit(names(tmp), '\\.'), function(x)x[2])
cell_line <- as.factor(tmp.labels)
set.seed(111)

# We do not want any PCA step before doing tSNE -> set pca parameter pca =FALSE
tsne_out <- Rtsne(as.matrix(t(tmp)), pca=FALSE, perplexity=30,
                  theta=0.0)
f=data.frame(x = as.numeric(tsne_out$Y[,1]),y = tsne_out$Y[,2])

# plotting tSNE
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("ctrl_noPCA.pdf")
print(sp)
dev.off()
```



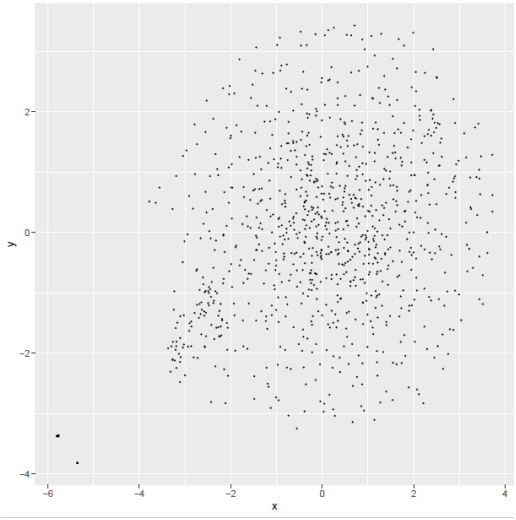
The image represent the tSNE plot for the non-treated single-cell data. What you see is not a separation and the cell line seams to be very homogenuos.

⚠️ There is no reason to put the color parameter beacause there are no color. The geometry point (geom_point) only indicate the kinf of dots that you want and in this case is pch=19 that indicate full circle dots and the cex parameter indicate the size of the dots and cex=0.3 is one third of the standard size. Instead to have directly the plot, the function generate a pdf containing the plot. pch=types of symbol, cex=sixe of the symbol.

Then, you have to run again all the function for the acute treatment group of the single-cell data changing the name of the input and the output file using the same parameter.

```
# Then we do the same for Osimertinib treatment
library(Rtsne)
library(ggplot2)
tmp <- read.table("osi_log2CPM.csv", sep=",", header=T, row.names=1)
tmp.labels <- sapply(strsplit(names(tmp), '\\.'), function(x)x[2])
cell_line <- as.factor(tmp.labels)
set.seed(111)

# Again no PCA
tsne_out <- Rtsne(as.matrix(t(tmp)), pca=FALSE, perplexity=30,
                  theta=0.0)
f=data.frame(x = as.numeric(tsne_out$Y[,1]),y = tsne_out$Y[,2])

# plotting tSNE
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("osi_noPCA.pdf")
print(sp)
```

Again you get a .pdf file that contain the tSNE plot. In this plot you see a little extension from the large homogenous set of cells, seems a little separation between the cells compare to the control tSNE plot.



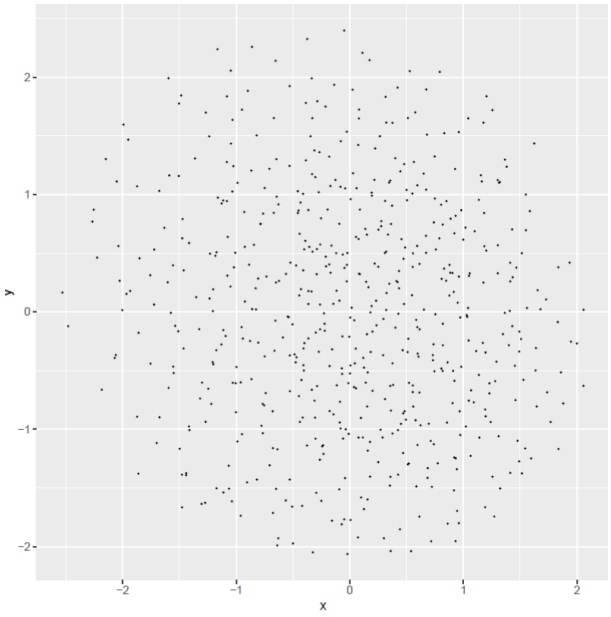
> ❗ Adapt the code to what you have to do. You can adjust the parameter and since we only one cell line we can plot the cell with only one colour. So, we can take out the colour parameter from the script.

UMAP

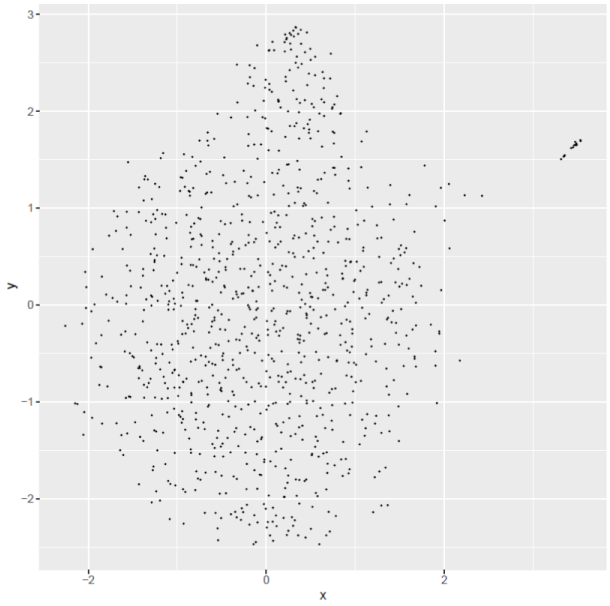
Then you can run the UMAP function to get UMAP plot.

```
# Open an R session an type the following commands
library(umap)
library(ggplot2)

#CTRL samples
ctrl <- read.table("ctrl_log2cpm.csv", sep=",", header=T, row.names=1)
ctrl.labels <- sapply(strsplit(names(ctrl), '\\.'), function(x)x[2])
cell_line <- as.factor(ctrl.labels)
ctrl.umap <- umap(t(ctrl), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(ctrl.umap$layout[,1]),y=as.numeric(ctrl.umap$
                                                             layout[,2]))
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("ctrlUMAP.pdf")
```

```
print(sp)
dev.off()

#OSI samples
osi <- read.table("osi_log2cpm.csv", sep=",", header=T, row.names=1)
osi.labels <- sapply(strsplit(names(osi), '\\.'), function(x)x[2])
cell_line <- as.factor(osi.labels)
osi.umap <- umap(t(osi), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(osi.umap$layout[,1]),y=as.numeric(osi.umap$
                                                            layout[,2]))
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("osiUMAP.pdf")
print(sp)
dev.off()
```



A) UMAP of the ctrl dataset



B) UMAP of the OSI

📌 Remember tSNE looks to a more local picture, while UMAP looks to a more complete picture.

From the UMAP of the control you see that cells are pretty homogeneous. If you look at the UMAP of OSI treatment a group of cell is separated and a subset of cells start to get different. Thi image above are got only using the standart parameter.

UMAP has two main parameters that are the driving force: the **near neighbors** (n_neighborns) this parameter is the number of near neighbors and **minimal distance** (min_dist) the distance used to consider near neighbors.

To get a better separation in the UMAP plot you can play changing the value of some parameters. You can play with the number of **near neighbors** (n_neighborns)**, minimal distance** (min_dist) and **number of epochs** (n_epochs) to see if you can better define the differences between groups.

- min_dist→ for example change the minimal distance from 0.01 to 0.05;
- n_neighbors→ if you reduce the number to 10 the picture change drammatically and you can see only the big difference.
- n_epochs = is the number of times you repeat the analysis until to have the final result. Normally n_epochs = 200 is set as default but it is usually too little and you have to set to 1000 or 10000 but you have to wait longer.

The parameter are inside file called “umap configuration parameter”. To change the parameter you simply have to add the parameter and specify it in the function otherwise the function consider the default parameter.

```
#part ot the UMAP function above
osi.umap <- umap(t(osi), random_state=111, n_epochs = 1000, min_dist=0.05, n_neighbors= 10)
```

We can also decide to use `intersect` function to focus on the elements that are in a certain position (ex. out of the big cloud). We give a value for the x and for the y axis and with intersect you can combine and see how many single cells are shifting their expression from the bulk.

```
#for example select the cells that have value higher than 20 in the x axis
length(which(f$x > 20)) # 10

#select the cells that have a value of y axis < -30 and the x axis < -50; it select the rettangle
length(intersect(which(f$y < -30), which(f$x < -50)) # 4
```

You see which cells are starting to modify their expression and get a more resistance profile. If you use information that comes from gene that are linked to the disease you produce a much better picture (use genes from the DE); if you use information coming from all the genes you obtain a lot of noise.

Exercise for next week

Exercise

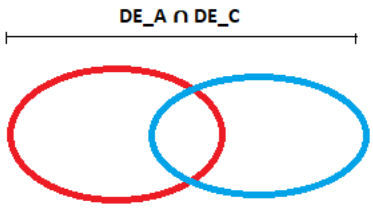
- Install in your Rstudio Bioconductor DESeq2 package.
- Following DESeq2 vignette: detect the genes called differential expressed comparing:
  - DMSO versus acute osi,
    - Thresholds: $|logFC| \geq 1$ & adj-p-val ≤ 0.05
  - DMSO versus cronic osi
    - Thresholds: $|logFC| \geq 1$ & adj-p-val ≤ 0.05
- Filter osi_log2CPM.csv using DE from acute osi and plot UMAP
  - There is any difference with respect to what can be observed using all genes?
- Filter osi_log2CPM.csv using DE from cronic osi and plot UMAP
  - There is any difference with respect to what can be observed using all genes?
- Filter osi_log2CPM.csv using combined DE from cronic/acute osi and plot UMAP
  - There is any difference with respect to what can be observed using all genes?

Instead of using all the genes for single cell, select only the subset of genes that are differentially expressed in the bulkRNA seq to get a better representation on the UMAP plot.

For bulkRNAseq dataset, you have to compare acute treatment (A→24h Osimertinib treatment) vs MOCK (M→only DMSO) and chronic treatment (C→21 days Olimertinib treatment) vs MOCK (M→only DMSO) . Comparing this two groups you find the list of differentially expressed genes (DE).

A(acute treatment)/M → DE_A (Differential expression genes for acute treatement)

C(chronic treatment)/M → DE_C (Differential expression genes for chronic treatment)



⚠️ We are not really interest in the intersection between the DE genes. We are interest to have the two independent groups (acute and chronic) and the group that get all toghether (combining acute and chronic).

Then you take the full list of genes from the single cell data treadted with osi (because are the only one in which you can see a difference). Looking at the image above to have a better clear concept. In the red circle you have all the DE genes in the acute treatment, in the blue circle you have all the DE genes following the chronic treatment and in the intersection you have the DE genes that are in common between the acute and the chronic treatment. If we start to select the only the DE genes in the response to the drug and we use this selected genes to perform UMAP on single cell dataset, we can grab a better separation of the cells that are acquiring resistance. In other words, we can better distinguish subpopulation of cells on single cell data. In practice, we want to use bulkRNAseq data to grab more information on the single-cell data.

Before doing the comparison and the selection of the DE genes we have to solve one problems.

📌 AIM of the exercise. First do the DE analyis, then use the list of gene form the DE analysis to subset the list of cell on the single cell data; in order to have a smaller list of cells to find genes that are resistance to osinertinib. At this point you use this cell, represented by those gene that give resistance to build UMAP. The question is: are we able to improve the subpopulation of cell that are aquiring resistance.

Annotation problem: for scRNAseq data the annotation of the genes is ensembleID:symbol while for bulkRNAseq data the annotation is symbol:ensembleID. To solve this problem we have to use the `strsplit` function to have the same annotation to perform intersection.

```
#strsplit function

strsplit(element_to_be_split, symbol_for_split)
```

The first element of the `strsplit` function is the element that has to be splitted (in our case is the rownames of the dataset that is a vector conaining the symbol:ensembleID) and the second element is the symbol used for the splitting (in our case is ":").

⚠️ This part written is only to solve the annotation problem!!!

```
1. #start with the bulkRNAseq file p9_wo_w_osi.txt and setwd(/wherethefileis)
2. #insert the file as a variable in R
x <- read.table("pc9_wo_w_osi.txt", header=TRUE, sep = "\t", row.names=1)
# with header=TRUE it considers the first line as a header (colum names)
# with row.names=1 it means that the 1st column of the table becomes the rowname of the dataframe
#these 2 options are conceptually the same but one for columns and the other for rows1
3. #to check the file to see if it's correct
head(x)
4. #use strsplit function to create a large list containing the symbol and the ensemble IDs separeted
y <- strsplit(row.names(x), ":")
#the strsplit function take the rownames of the dataframe and separate the name for the ":" character
5. #convert the list to a matrix
matrix <- matrix(unlist(y), ncol = 2, byrow = TRUE)
# at this point you have a list that as to be converted in  a matrix with the unlist function that use the number of colum=2 and unlist b
y row.
6. #use the paste function to create a vector with the new annotation as in the single cell dataset  ensembleID:symbol
z <- paste(matrix[,2], matrix[,1], sep=":")
```

```
#the function paste, paste the two colum of the matrix in the order that we want separeted by the character ":"
7. #put the new row names in the original dataframe (x) where we had all the data
row.names(x) <- z
```

DE analysis

▼ **DESeq vignette**

DESeq2 vignette

The object class used by the DESeq2 package to store the read counts and the intermediate estimated quantities during statistical analysis is the *DESeqDataSet*, which will usually be represented in the code here as an object `dds`.

The DESeqDataSet has an associated “design formula”. The design is specified at the beginning of the analysis, as this will inform many of the DESeq2 functions how to treat the samples in the analysis (one exception is the size factor estimation – adjustment for differing library sizes – which does not depend on the design formula). The design formula tells which variables in the column metadata table (colData) specify the experimental design and how these factors should be used in the analysis.

The simplest design formula for differential expression would be `~ condition`, where condition is a column in colData(dds) which specifies which of two (or more groups) the samples belong to.

There are different ways of constructing a *DESeqDataSet*, depending on what pipeline was used upstream of DESeq2 to generate counts or estimated counts.

The function *DESeqDataSetFromMatrix* can be used if you already have a matrix of read counts prepared from another source. **The count values must be raw counts of sequencing reads.** This is important for DESeq2’s statistical model to hold, as only the actual counts allow assessing the measurement precision correctly. Hence, please do not supply other quantities, such as (rounded) normalized counts, or counts of covered base pairs as this will only lead to nonsensical results.

To use *DESeqDataSetFromMatrix*, the user should provide the counts matrix, the information about the samples (the columns of the count matrix) as a *DataFrame* or *data.frame,* and the design formula.

Remember: in the count table, each row represents an Ensembl gene, each column a sequenced RNA library, and the values give the raw numbers of sequencing reads that were mapped to the respective gene in each library.

To do the differential expression part with the DESeq function, first you have to install library DESeq2 packages and then you have to create a input data file organized in a specific way.

Since you have to perform DE on Acute treatment vs Control and on Chronic treatment vs Control, it is necessary to split the table given by the professor (file pc9_wo_w_osi.txt) in two different matrices. One matrix contains the three ctrl replicates plus three acute treatment replicates and the second matrix contains the three crtl replicates plus three chronic treatment replicates (each replicate is a column). Acute and chronic replicates are analysed independently in two different matrices and the untreated (crtl) is the reference.

```
1. #to make the two matrix
acute<-x[,c(1,2,3,7,8,9)]
chronic<-x[,c(1,2,3,4,5,6)]
#create the two matrices starting on the dataframe (x) and selecting the coloums of interest and put this in two new variable one contaning

2. #save the file as txt to be used in future
write.table(acute, file="acute_dataframe.txt", sep="\t", row.names = TRUE, col.names = TRUE)
write.table(chronic, file="chronic_dataframe.txt", sep="\t", row.names = TRUE, col.names = TRUE)
```

Since this two matrices derive from a dataframe, are dataframe, but we need matrix for the `DESeqDatasetFromMatrix` function and so we need to convert the dataframe in matrices. So you do `read.table` to have them in R and then you perform the convertion of the dataframe in a matrix with `as.matrix`.

```
#read the files
acute_df<-read.table("acute_dataframe.txt",header = TRUE, sep="\t", row.names = 1)
chronic_df<-read.table("chronic_dataframe .txt",header = TRUE, sep="\t", row.names = 1)

#convert the dataframe in a matrix
```

```
acute_matrix<-as.matrix(acute_df, rownames.force=NA)
chronic_matrix<-as.matrix(chronic_df, rownames.force=NA)
```

To continue our analysis with the `DESeqDatasetFromMatrix` function as is explained in the DESeq vignette wwe need to create a new dataframe called **coldata** containing the information related to the experimental group.

In practice we need to create a table with the name of the samples (name of the columns) and the condition (control or treatment). To create this coldata table, you have to build a dataframe starting with the creation of two vectors containing the names of the samples (one for the acute called samples_acute and one for the chronic called samples_chronic) and the conditions vector (which is equal for both chronic and acute). The samples vector is a vector with the column names of the marix created above, while the conditions is a vector containing the conditions of the treatment, which are always the same(ctrl,ctrl,ctrl,treated,treated,treated).

Then, it is necessary to convert the condition vector as a factor variables with the `factor` function.

Factor is a label. The factor is a numerical representation with a label: factor has been designed to have labels but to give the system some number to work with. c=1 t=2 but if you make t vs untreated you will have other numbers. **riscrivere meglio**

The labels are assigned on the basis of the lexicographical order, so first ctrl=1 and then treated=2.

This are the command for what is explained above.

```
1. #create the two vectors containing the samples and the conditions
samples_acute <- colnames(acute_matrix)
samples_chronic <- colnames(chronic_matrix)
#take the colnames of the two matrix and put it in a vector
conditions<-c(rep("ctrl",3), rep("treated",3))
#create a new vector that contain three rows contaning ctrl and then three row with treated

2. #create the dataframe combining the two vectors. The names of the vectors will be considered automatically as header
coldata_acute <- data.frame(samples_acute, conditions)
coldata_chronic <- data.frame(samples_chronic, conditions)

3. #put the rownames in the new dataframe taking them from the vectors we created before
rownames(coldata_acute)<-samples_acute
rownames(coldata_chronic)<-samples_chronic

4. #factor the conditions column
coldata_acute$conditions<-factor(coldata_acute$conditions)
coldata_chronic$conditions<-factor(coldata_chronic$conditions)

5. #control that effectively the conditions column it is a factor
class(coldata_acute$conditions)
class(coldata_chronic$conditions)
```

Now, we have all the element for running the `DESeqDatasetFromMatrix` function.

`DESeqDatasetFromMatrix` parameter:

- **countData** (are our acute_matrix and chronic_matrix contaning the counts of our genes)
- **coldata** (coldata_acute and coldata_chronic)
- **design** is equal the name of the parameter you would like to use for DESeq (conditions).

The output of the `DESeq` function is a datastructure called dds_* containing everything you need. Then is necessary to put the results in a variable for looking at them with the function `results`. The results are put in a dataframe called res_* and in thiese daatframes you will have all the genes and the statistics related to each genes (log2FC, p-value and adjp-value).

⚠️ It is possible that some genes get NA as results in some of the statistical parameter, this means that this genes were not robust enough in expression for the differential expression analysis.

```
1. # now we are ready to do DESeq on acute since deSEQ wants 3 parameters:
# 1. countData (the matrix with the data)
# 2. colData (our dataframe containing the samples and the conditions)
# 3. design = the parameter for which we have to perform differential expression. Here we put our condition vector with the 2 conditions to

library("DESeq2")# to load the library with all the function needed
```

```
#DESeqDatasetFromMatrix generate a Dataset on which the DESeq can be run.
dds_acute <- DESeqDatasetFromMatrix(countData = acute_matrix, colData = coldata_acute, design=~conditions)
dds_chronic <- DESeqDataSetFromMatrix(countData = chronic_matrix, colData = colata_chronic, design=~conditions)

2. # Use the DESeq function to perform the DE analysis
dds_acute <- DESeq(dds_acute)
dds_chronic <- DESeq(dds_chronic)

3. # Save the results in a variable
res_acute <- results(dds_acute_study)
res_chronic <- results(dds_chronic_study)
```

Now, we have to filter our results data of the DE analysis for adj p-value ≤0.05 and a |log2FC|≥1.

```
1. # Filter the data for adjusted pValue <=0.05 e |log2FoldChange|>=1.
res_acute_filter<-res_acute[which(res_acute[,6]<=0.05 & abs(res_acute[,2])>=1),]
res_chronic_filter<-res_chronic[which(res_chronic[,6]<=0.05 & abs(res_chronic[,2])>=1),]

# This command creates a new variable taking all the rows from the res_* dataframe that have a value in the $padj column <=0.05 and the abs
# All the columns of the res_* dataframe are selected and maintained.
# You can see that which is followed by [..., ] the "..." are the rows, the space means all columns.

2. #Save the filtered data into a file
write.table(res_acute_filter, file="DESeq_results_acute_filter.txt", sep="\t", row.names=TRUE,col.names=TRUE)
write.table(res_chronic_filter, file="DESeq_results_chronic_filter.txt", sep="\t", row.names=TRUE,col.names=TRUE)
```

3. Filter osi_log2cpm.csv using DE of acute RNAseq data and plot UMAP

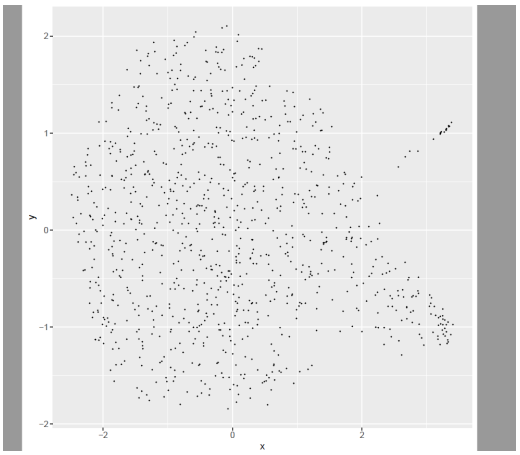
At this point we have done the DE analysis on the bulkRNAseq data and we can use the filtered data to filter the single cell dataset.

To filter the data we can use the `intersect` function. The intersect function require only two parameter; needs only the two element (vectors) in which the analysis has to be done. The output of the intersection function is a vector containing the name of al the genes in the intersection. The column names is “x” and the row names are number starting from 1.

In the `intersect` function you give rowname of the single cell osi dataframe and the list of genes from the DE analysis on the bulkRNAseq and it will subset only the differentially expressed genes in the single cell dataset. We are using bulk expression to find the expression of gene that changes upon treatment and then we use this list to subset the population of cells in the single cell population on the basis of this results.

```
#to put the single cell data of the osimertinib treatment in a variable using read.table
osi <- read.table("osi_log2CPM.csv", sep=",", header=T, row.names=1)

#perform the filtering of the single cell data using the DE genes of RNAseq acute treatment
intersection_osi_acute <- intersect(row.names(res_acute_filter), row.names(osi))

#select the row of the data frame (osi) of the single cell experiment containing the DE genes in common between the single cell data and th
osiUMAP_acute<-osi[which(row.names(osi)%in%intersection_osi_acute), ]

#save the data in a csv file needed to run the UMAP function
write.csv(osiUMAP_acute, "osiUMAP_acutelog2cpm.csv", sep=",")

#load the library necessary to run UMAP
library(umap)
library(ggplot2)

#run UMAP function
osiUMAP_acute <- read.table("osiUMAP_acutelog2cpm.csv", sep=",", header=T, row.names=1)
osi.umap <- umap(t(osiUMAP_acute), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(osi.umap$layout[,1]),y=as.numeric(osi.umap$layout[,2]))

#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("osiUMAP_acute.pdf")
print(sp)
dev.off()
```

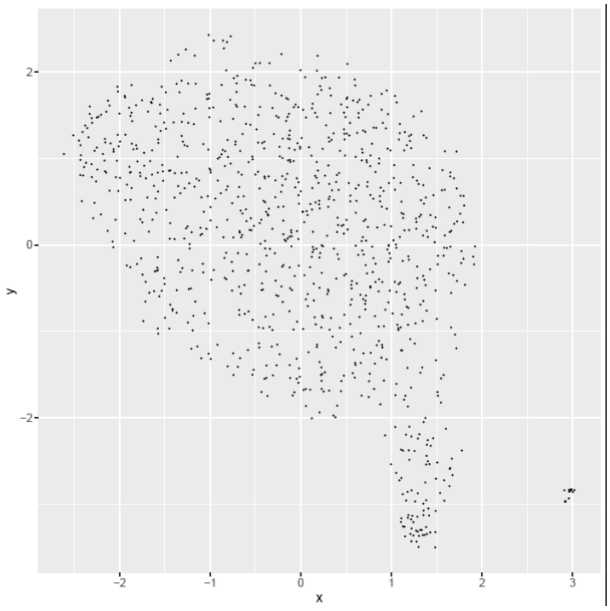
At the end you get the UMAP plot and is different from the one directly done on the single cell data.



4. Filter osi-log2cpm.csv using DE of chronic RNAseq data and plot UMAP

```
#perform the filtering of the single cell data using the DE genes of RNAseq chronic treatment
intersection_osi_chronic <- intersect(row.names(res_chronic_filter), row.names(osi))

#select the row of the data frame (osi) of the single cell experiment containing the DE genes in common between the single cell data and th
osiUMAP_chronic<-osi[which(row.names(osi)%in%intersection_osi_chronic), ]

#save the data in a csv file needed to run the UMAP function
write.csv(osiUMAP_chronic,"osiUMAP_chroniclog2cpm.csv", sep=",")

#run UMAP function
osiUMAP_chronic <- read.table("osiUMAP_chroniclog2cpm.csv", sep=",", header=T, row.names=1)
osi.umap <- umap(t(osiUMAP_chronic), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(osi.umap$layout[,1]),y=as.numeric(osi.umap$layout[,2]))

#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("osiUMAP_chronic.pdf")
print(sp)
dev.off()
```

At the end you get the UMAP plot and is different from the one directly done on the single cell data.

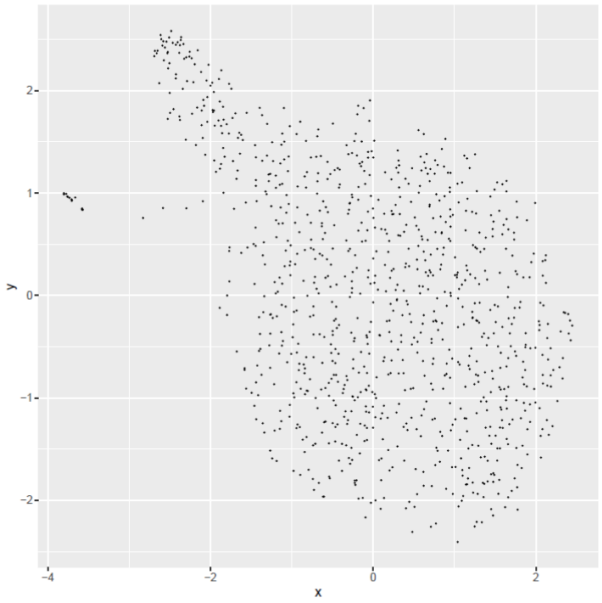


5. Filter osi-log2cpm.csv using combined DE of acute and chronic RNAseq data and plot UMAP

For this point you have to use the function `union` to combine and take all the data

```
#perform first the union on the RNAseq data combining acute and chronic DE genes to have all.
union_acute_chronic <- union(row.names(res_chronic_filter),row.names(res_acute_filter))

#filter the scRNAseq osi dataset with the genes from chronic and acute united
osiUMAP_union<-osi[which(row.names(osi)%in%union_acute_chronic), ]

#save the dataset into a csv table
write.csv(osiUMAP_union, "osiUMAP_unionlog2cpm.csv", sep=",")
#run UMAP function
osi_union <- read.table("osiUMAP_unionlog2cpm.csv", sep=",", header=T, row.names=1)
osi.umap <- umap(t(osi_union), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(osi.umap$layout[,1]),y=as.numeric(osi.umap$layout[,2]))

#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("osiUMAP_union.pdf")
print(sp)
dev.off()
```

At the end you get the UMAP plot and is different from the one directly done on the single cell data, but is equal to the one done at the point 4 of the exercise.



Why log2FC?

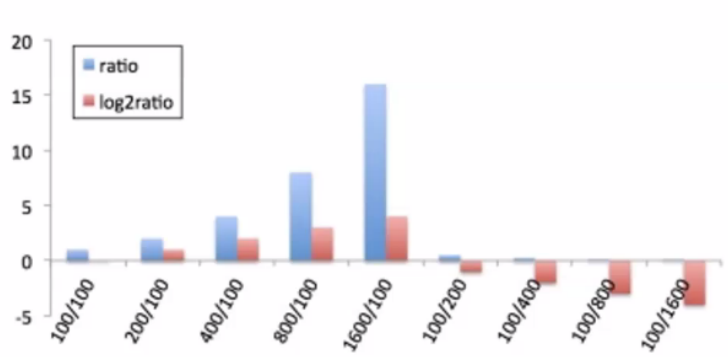


Figure 19: log2FC

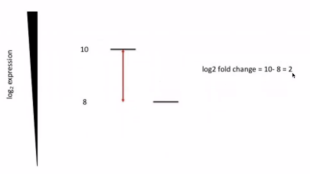
We are talking about bulk now. We have done linear dimensionality reduction for bulk, while a more sophisticated dimensionality reduction, thus not linear anymore, is necessary for single cell. On the bulk, we have to identify the differential expressed genes comparing two conditions and establish if the Δ expression is significant. To do this, we represent the difference by using the **log2FC (fold change) = $\log_2$(expression of genes in condition A / expression of the same genes in condition B like reference).** If the ratio increases so that the value is greater than 1, the expression is higher. The problem is that if it goes below the expression level of the reference, all the values will range between 1 and 0. Instead, if this is bigger, they go from 1 to anything. If you plot the ratio of the same value but simply changing denominator with denominator, the values of downregulation are not seen very well. Everything changes when you put the log2 in front of this ratio. Without the log 2, it will have a value from 0 to 1. With the log 2 you can write it as a subtraction: positive value for upregulation and negative number for downregulation; the size of the change given the same number has the same value but different sign. This is the correct way to represent data.

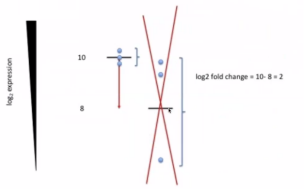
Other people do **$\log_2$FC filtered for ≥ 1**. This is what is usually considered significant: knowledge of the noise present in bulk experiments, values below this FC are not precise: there are much more false positives. In this case, the differential expression is hidden from the noise. This kind of threshold is safe for bulk analysis. If you have a very big dataset and your statistics are solid, you can shrink the noise of the dataset. Even if you use other experimental techniques to come to expression data, the noise is sometimes just too big.

$$|log_2 FC| \geq 1$$

Is log2FC enough for our analysis? (SPOILER = NO)

In reality, if you look at the image below, there is a very significant difference in expression (log2FC=2); however the data generating the value has a high dispersion and make the mean difference not significant. We need to add a **p-value threshold** to the FC value.





Sometimes the differential expression is big, some other times the dispersion of the data is so big that the difference is much less. This is an indication that this difference may not be significant.

We have to provide a p-value to our analysis. Normally, the adjusted-p-value considered is equal to 0.01, 1 over 100 of genes may be there by chance and not really be differentially expressed. Less conservative approaches are 0.05 and 0.1, 5% possibility and 10% possibility that a gene is in the list of differentially expressed genes by chance. The approach is dependent on what you want to do with your data and extract from your data. With very noisy data you use 10%. If the data is actually representing a difference that is very big, 0.01 is more than enough.

Normally the p-value and log2FC are two values that are put together to select the subset of genes that you would like to bring ahead for further analysis in the lab.

Actually, we will use mainly the FC threshold to filter the single cell data.

To find differential expressed genes various steps are needed:

1. **DATA DISTRIBUTION**: find a good data distribution to fit our statistical model.

   In general, the typical distribution that is used for biological tests is the **t-test distribution**; but because the data are sampled in RNAseq, **Poisson distribution** is also very widely employed.

   → A **negative binomial distribution** embeds both Poisson- and t-distribution and so is the most widely used: you take into account also the biological variability of your sample together with the way you are subsampling data from the RNAseq.

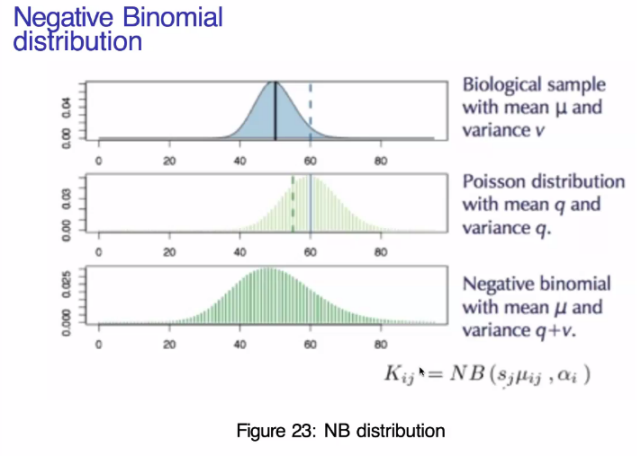


Figure 23: NB distribution

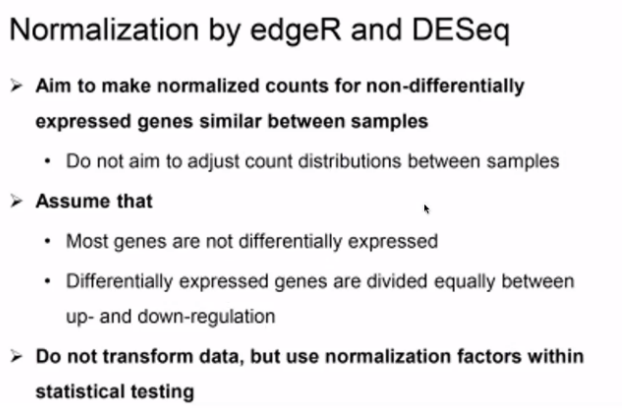
2. **NORMALIZATION**: with the micro-array data this step is done before doing the analysis. In this case, we will use a statistical model that embeds in itself all these types of information.
   You give to the statistical model the list of your samples, e.g. DMSO+acute treatment.

   Normalization → for comparing gene expression between (groups of) samples, normalize for:

   - library size (number of reads obtained);
   - RNA composition effect

   You have to normalize differences due to the sampling of your experiment. When doing sequencing, libraries have different sizes and the number of sampling is not identical. You have to select a way of scaling your data so that the overall counts for all the samples are the same.

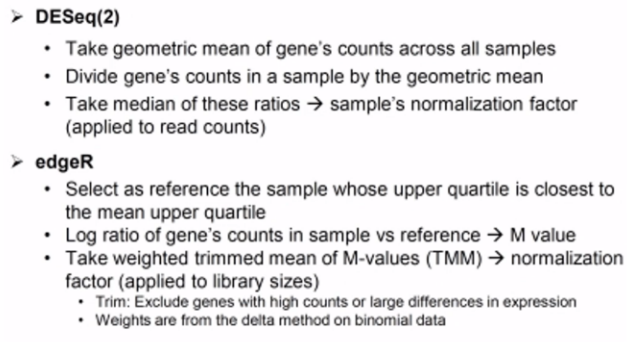
   Example: 9 million vs 11 million samples experiments, I have to do something between, about 10 million. The experiments after the normalization are comparable.



We are not adjusting the distribution of the samples. We are realigning the mean expression and we are assuming the most of the genes are not differentially expressed. This is one of the strongest assumptions: if you KO histone deacetylases and you use this kind of approach you will loose more data because you are inducing heavy changes in the chromatin and hence there will be many big changes in the expression. If you are do only a little perturbation, this type of differential expression analysis can be applied. Whereas, if the changes are referred to a big subset of genes, this assumption is wrong.

How to do differential expression analysis with DESeq and edgeR?

Go to Biocunductor and install DESeq2 in Rstudio.

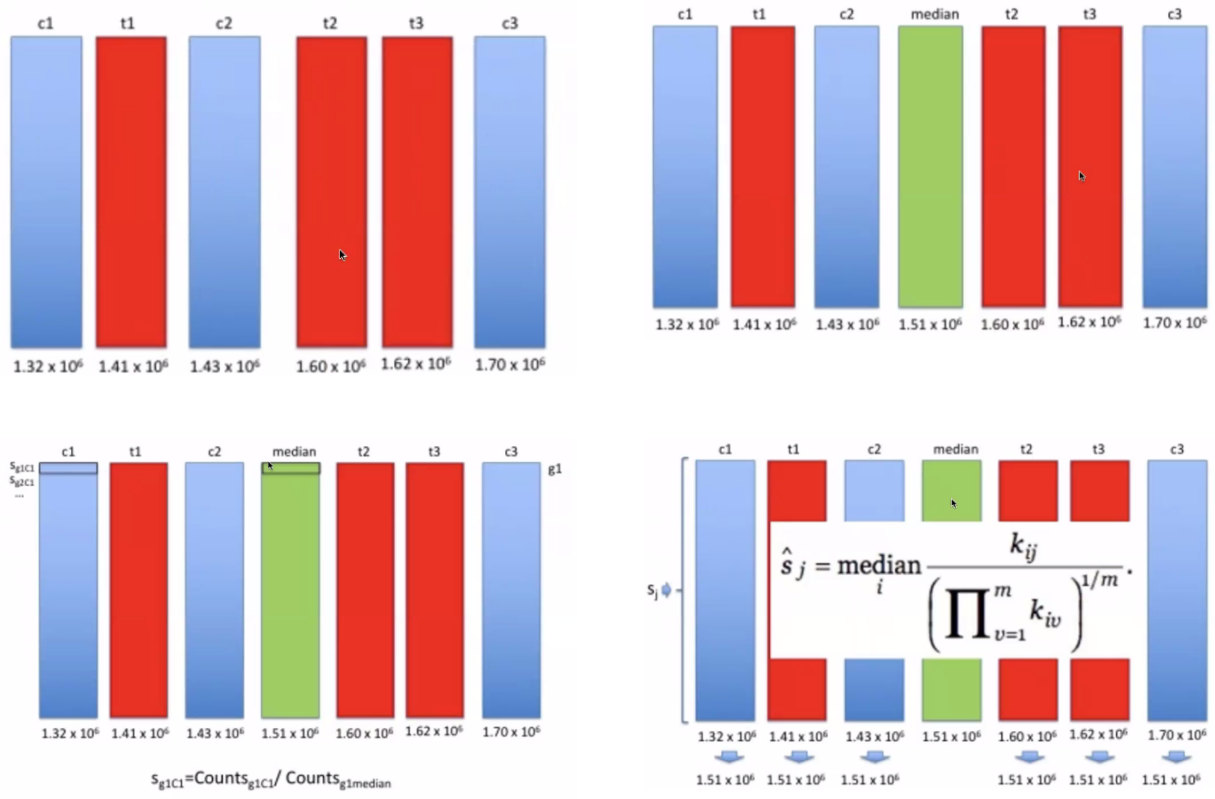


DESeq takes a mean of the genes for all the samples and create a virtual experiment that is a mean experiment of everything → imagine to have 6 samples, you will have a mean experiment that is located in between (the median). The idea is to scale all the other experiments to have a total number of genes that is coherent.

It takes the geometric mean and divides each gene counts in each sample by this mean referring to that gene and then does the normalization.

At the end you will obtain a list of ratio and all the ratio values are used to make another mean → *scaling factor* to be applied to convert the gene in the median. The global scaling value brings all the samples to the median value, the overall size of each sample is goin to be identical to the median because of the global scaling → eliminate the effect due to the difference sampling coming from the library size.

Formula $\hat{S}_J$: the expression of the gene i in the sample j divided by the geometric mean of the corresponding gene in all the samples. Once you have collect all these values, you make the median of the gene i and you get the sample normalization value for all the experiments.



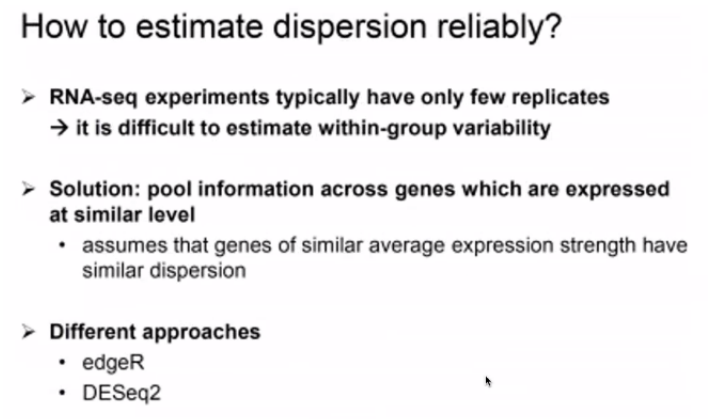
3. **DISPERSION ESTIMATION**: it is the most important thing that is needed to calculate to evaluate what genes that are differential expressed. If genes are differential expressed, you expected that they change very much from one sample to the other, but that means that the dispersion that exists comparing the treatment and the control is very big.

   Here, the idea is to try to estimate the dispersion for the genes that are not differentially espressed in the way that you can have the ones that are differentially expressed that are considered as a sort of outliers compared to the rest.

Dispersion

- **When comparing gene's expression levels between groups, it is important to know also its within-group variability**
- **Dispersion = (BCV)²**
  - BCV = gene's biological coefficient of variation
  - E.g. if gene's expression typically differs from replicate to replicate by 20% (so BCV = 0.2), then this gene's dispersion is $0.2^2 = 0.04$
- **Note that the variability seen in counts is a sum of 2 things:**
  - Sample-to-sample variation (dispersion)
  - Uncertainty in measuring expression by counting reads

Dispersion is the square of the gene's biological coefficient of variation : BCV^2. In principle, it contains two information: the sample to sample variation (remember the majority of the genes are not differentially expressed) and the uncertainty in measuring expression by counting reads, due to the noise present in the system.



The genes that are very low expressed and do not have enough information are normally discarded. Low expressed genes are not considered in terms of differential expression because if the reads are few, the variability/random fluctuation may play a significant role in determining the output of the expression level of the gene. These genes are removed from the analysis (NA = not applicable since not enough) → the DESeq shrinks the data referring to those samples.

Genes that are very low expressed are not good. With very small numbers, the differences in expression are not reliable.

Example of three scenarios:

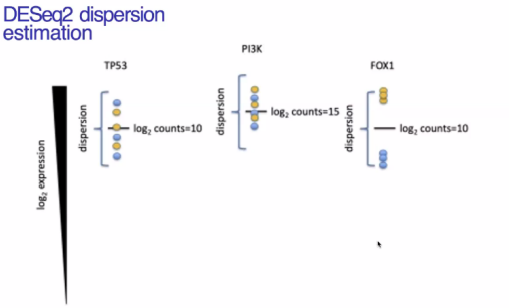
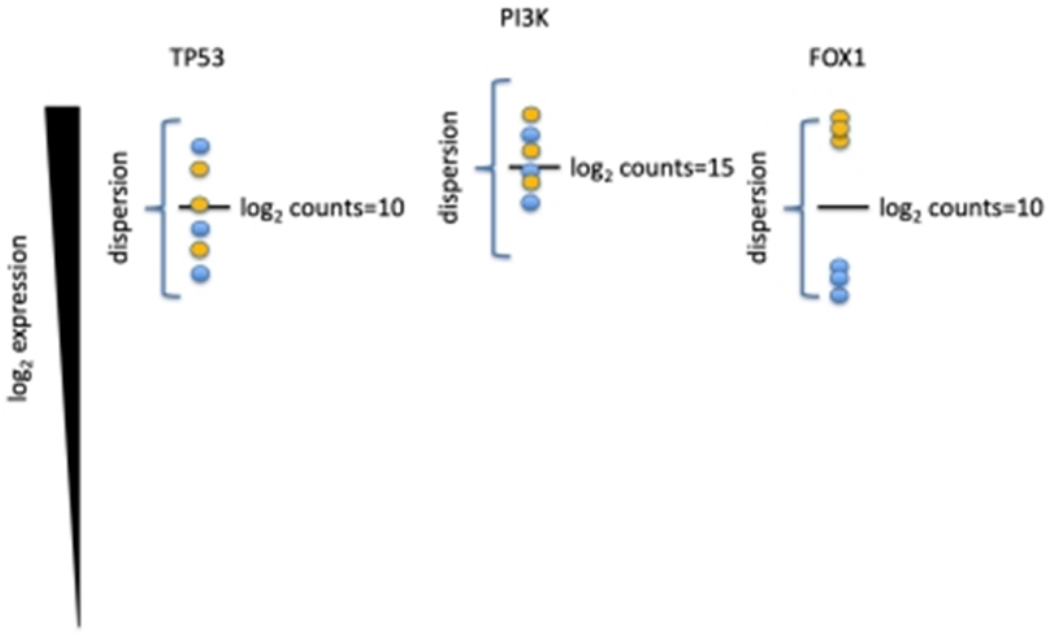
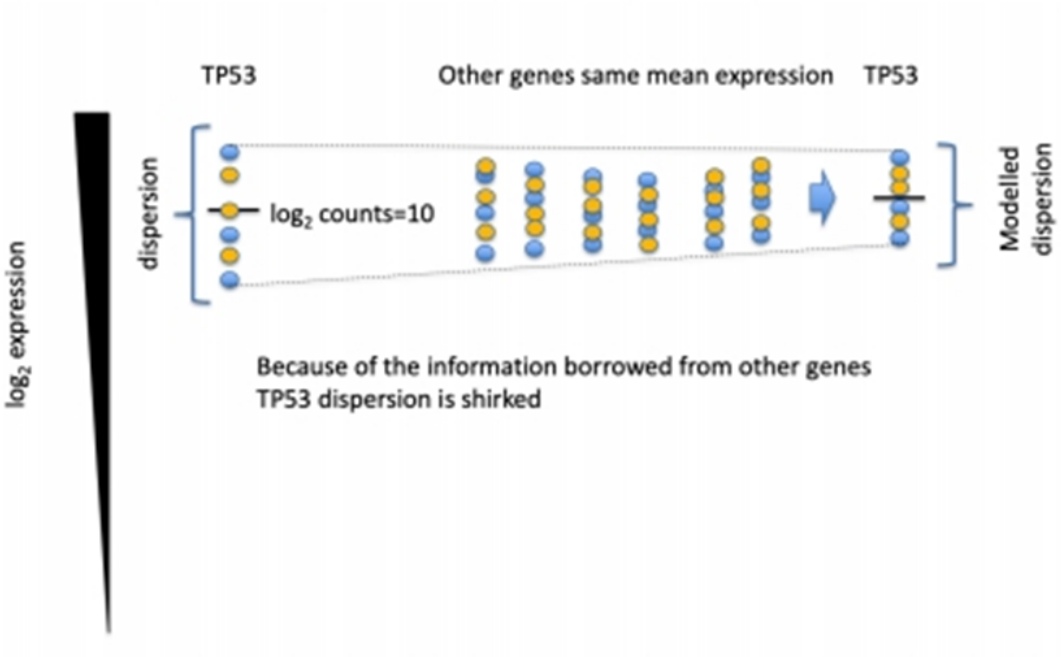


Figure 28: scenarions

There is a situation in which we have two not differentially expressed genes (TP53 and PI3K) and one differentially expressed (FOX1).

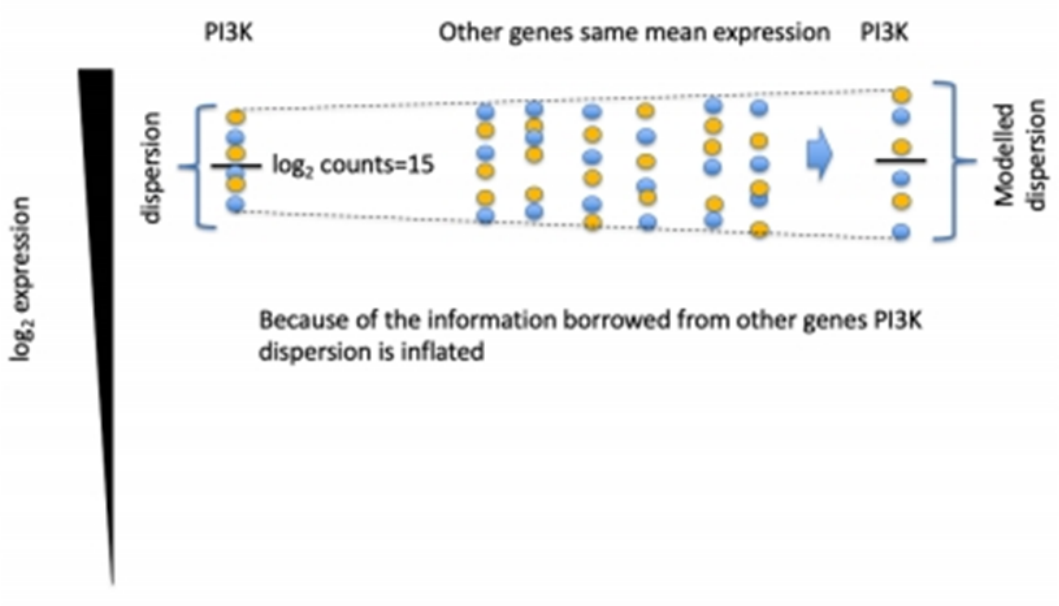


1. Let’s talk about the example of TP53.



We will never have enough samples to cancel uncertainty out and correctly estimate the dispersion. All the genes having the same expression share a very similar uncertainty. If, for example, the expression for TP53 is log2FC= 10, you can take other genes with log2FC expression of 10 to see how they are distributed, if they are differentially expressed. Since the majority of the genes are not differentially expressed and since the dispersion depends on the noise in the measurement, if you borrow the information coming from other genes that have the same mean expression, it should improve the way how represent the dispersion of TP53 and you shrink it. You simply take your dispersion and on the basis of this extra information you shrink it. So, the **modelled dispersion** is used and it tells you that, since all the genes that are characterized by the same mean expression value of TP53 have a smaller dispersion than TP53, it is evident that TP53 was not correctly measured and so its dispersion will be shrinked.

2. Let's look the opposite situation for PI3K.



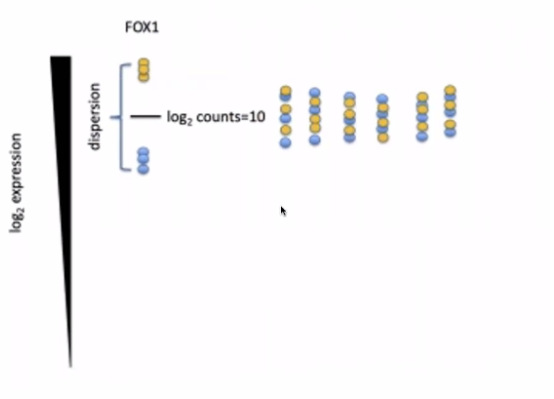
PI3K log2FC is equal to 15 and the majority of the genes are characterized by a greater dispersion than the one measured for the sample. In this case, you have underestimated the dispersion of your sample and so, by borrowing the information from the other genes, the dispersion for PI3K is expanded.

So, dispersion is gonna be shrinked or expanded on the basis of the value that you get from the other results.

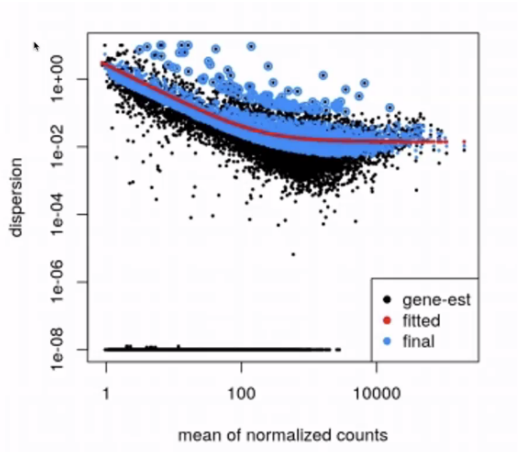
3. When I have differentially expressed genes, we have a too big dispersion to use the modelled dispersion. We cannot shirk and normalize it. We consider it as an outlier= too much difference between dispersions; practically, all the outliers are the genes that are differentially expressed.

Why we need to have few genes that are differentially expressed?

If the majority of the genes are differentially expressed, we should shrink everything and then, we will lose some differential expressed genes. This is why we must be sure that the majority of the genes are not changing.

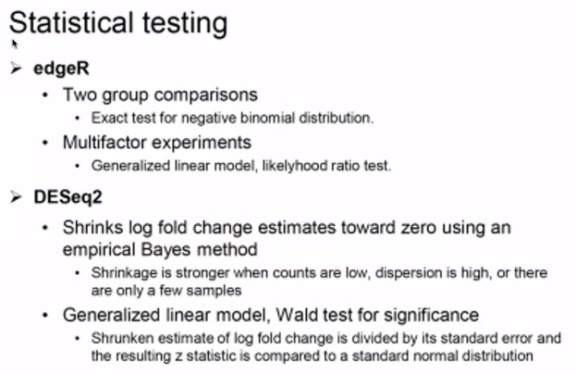


Dispersion vs mean of the normalized counts plot:



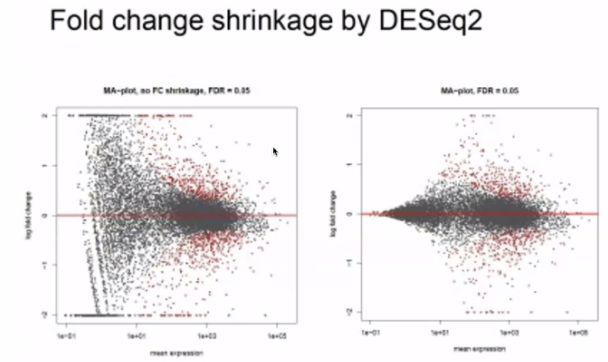
- black dots: the values of the real dispersion
- blue dots: represent the shrinked dispersion
- red curve: the trend of what is expected for the dispersion → you can see a very big dispersion for low expression and it gets stabilized when the level of expression reaches a value of more than 100 counts. Few counts more uncertainty, higher counts less uncertainty and dispersion is obtained.
- blue circles with black dots inside (above the cloud): dispersion outliers, the genes that are differentially expressed. They have a dispersion that is bigger of what is expecting for the genes that are not differentially expressed. We can try to use this info to extract them out from the noise. Dispersion is the most important part to find the outliers.

4. **STATISTICAL TESTING**



DESeq2 shrinks logFC estimates toward zero using an empirical Bayes method. It t takes out all the genes that are supported by few reads and for which the logFC is not reliable.

If you look at the FC ratio that you can obtain in a normal experiment, it looks like the one shown in the figure on the left. The black dots at very low expression level are practically all false. DESeq2 simply removes these genes because they are not enough significant and leaves what are the rest of the genes from which the differential expression is estimated.



Wald statistic test

The statistics used by DESeq is Wald test for significance. It is very similar to a t-test although you are estimating something slightly different: you are doing a sort of transformation of the data in Z score (distance of the data VS expected mean of the data). It is estimated if the Z score is far away in the normal distribution of the Z score that you have (on the tails). You are actually looking at the fact that the genes that are significantly differentially expressed if their Z score drops on the tail of the normal distribution of the Z score. It is not anymore the mean value of the t-test, here there is the Z score representation that is used to evaluate in a normal distribution of the Z test if the genes are differentially expressed or not, thus we are evaluating the differential expression → we obtain a **row p-value** that gives an indication of the differential expression.

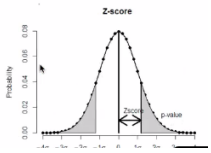
If the hypothesis involves only a single parameter restriction, then the Wald statistic takes the following form:

$$W = \frac{(\hat{\theta} - \theta_0)^2}{\operatorname{var}(\hat{\theta})}$$

The square root of the single-restriction Wald statistic can be understood as a (pseudo) *t*-ratio, following an asymptotic *z* distribution.

$$\sqrt{W} = \frac{\hat{\theta} - \theta_0}{\operatorname{se}(\hat{\theta})}$$

Deseq statistics



There is another thing that is done by DESeq that is the correction given by the **Cook's distance**.

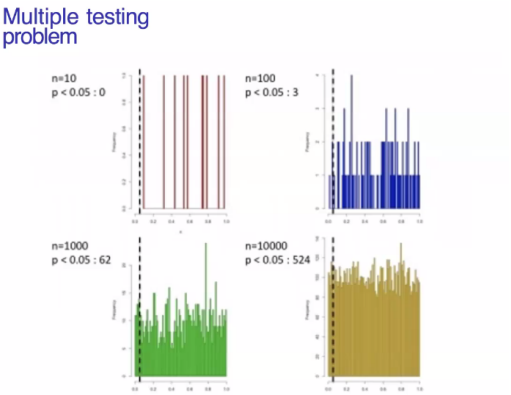
- RNA-Seq data sometimes contain isolated instances of very large counts that are apparently unrelated to the experimental or study design, and which may be considered outliers.
- There are many reasons why outliers can arise, including rare technical or experimental artifacts, read mapping problems, etc.
- For every gene and for every sample there is a diagnostic test for outliers called Cook's distance
- Cook's distance is a measure of how much a single sample is infuencing the fitted coefficients for a gene, and a large value of Cook's distance is intended to indicate an outlier count.
- DESeq2 automatically flags genes with Cook's distance above a cutoff and sets their p-values and adjusted p-values to NA

If there is a situation as the one shown in the figure above, where samples make the mean very different. Here, in particular, the mean is mainly given by the single sample and this sample is removed because not enough robust to be used to evaluate the mean, since the majority of the information comes from one single sample.

Let's go back to the p-value again. You are actually using about 20 thousands genes and test them for evaluating if the p-value indicates that they are differentially expressed.

However, p-value distribution is uniform, so each single value has the same probability of the others to come out. The reason why it comes out is due to how many times you are acutally sampling your data.

In this example:



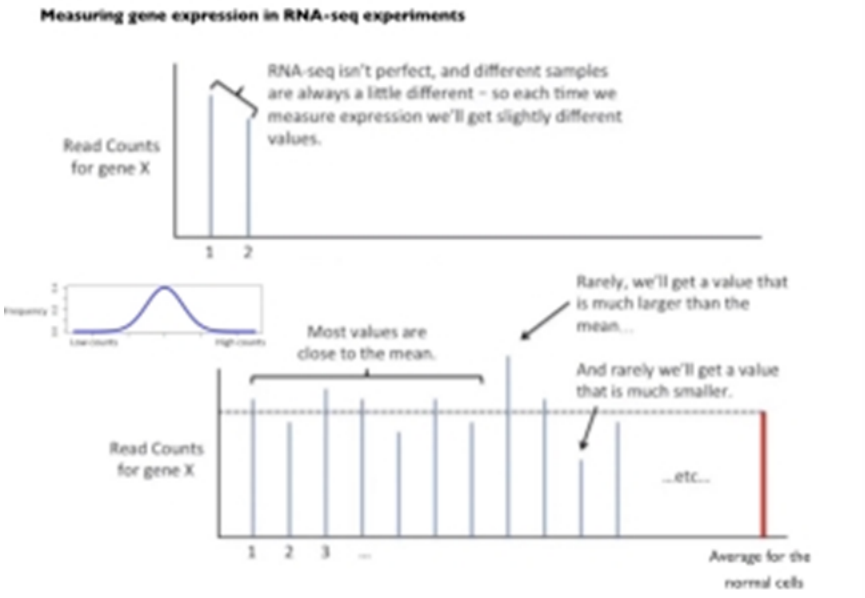
If you sample data 10 times and the threshold is 0.05 → if you do 10 samples, there won't be any p-values below the value of threshold by chance. If I increase the n° of sampling, this n° will increase proportionally.

You have to sample enough to obtain the expected p-value given an uniform distribution.

If you take 20 thousands genes, the expected p-values that will be significant only by chance will be high ($p < 0.05$ → 5% of 20k genes only by chance are differentially expressed): we have to solve this issue.

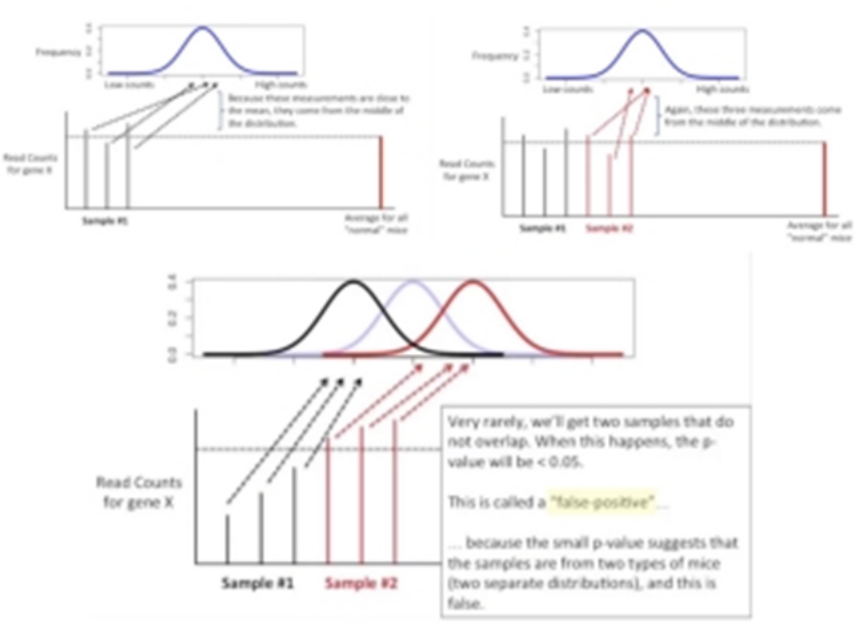
You accept a certain number of false positives contaminating the true positives, but by solving that issue there are going to be a certain number of false negatives → do again the same experiment increasing the number of samples to make those false negatives to drop.

> ! RNAseq is usually a preliminary experiment to find some putative targets in an unbiased way: then more biological validation is required.



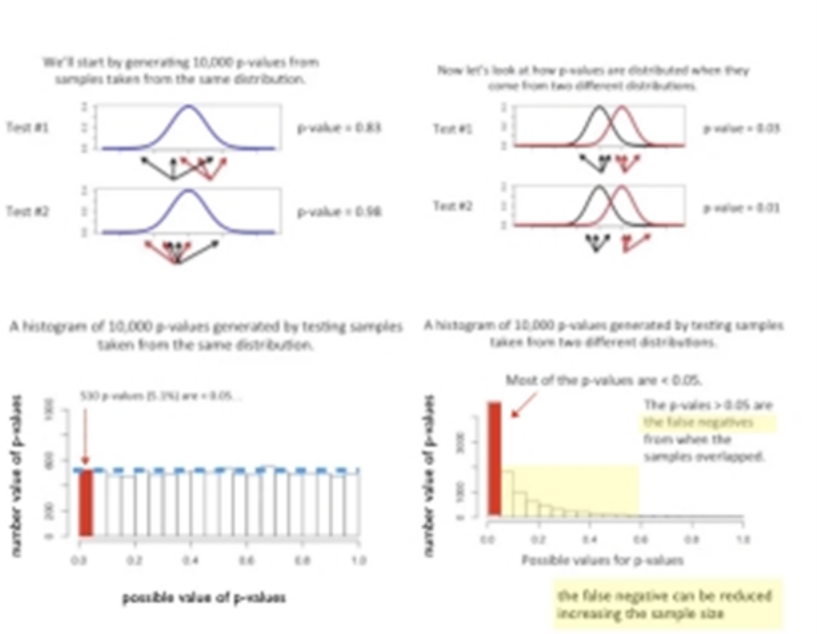
One sample in which no differential expression is observed, no changes, so genes are more or less uniformly distributed and their expression follows the normal distribution all around the mean. Of course, some of the genes will be slightly differentially expressed and these fall within the tails of the normal distribution.

The majority is still staying in the middle of the normal distribution, but some of the genes stay in the tails. The changes in gene expression are only due by chance.

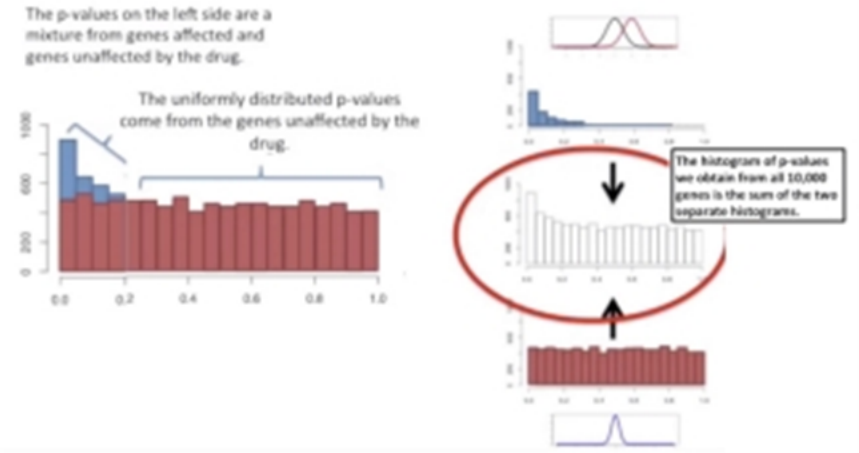


If I have two set of samples and the genes are not differentially expressed. In both situations there are outliers and there will be some downregulated genes and some upregulated genes. It is a consequence of the distribution that these genes are differentially expressed, but it is only a matter of chance, due to the distribution of the curve.

Those set of genes not differentially expressed are the ones that we try to eliminate when we look at the correction of the p-value. One of the methods that is used to do this correction is the **Benjamini-Hochberg test**. You have to identify the upper banding of the distribution.

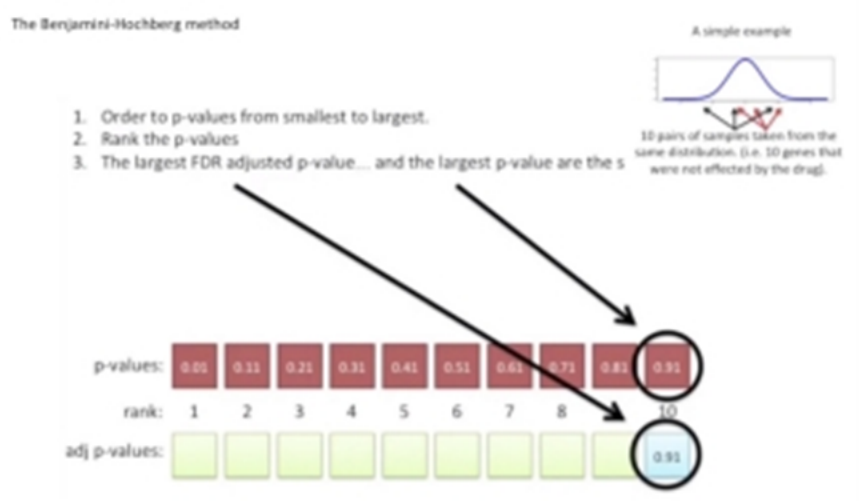


We are combining the uniform distribution of the genes obtained by chance and the distribution of the differentially expressed genes. We have to identify the region which is changing and dividing the two groups (blue region in the image below).

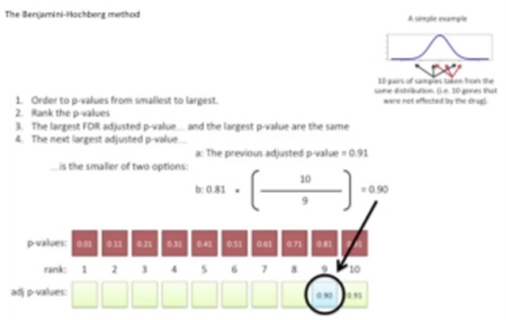


We rank the p-values from the smallest to the highest and associate the ranking to the gene.

Then, the adjusted p-value for the top ranked one (the one that is not differentially expressed, the highest p-value) initally is going to be the same.



Instead, for the second last gene, we take the normal value of the p-value divided by the ranked p-value and then multiplied for the current p-value that is present.



We do this for all the samples.

With this system, simply by ordering the p-values from the smallest to the highest and by considering the ranking of the data, you can identify the threshold that is separating the subset of the true positives with respect to the others. We see only the true differentially expressed genes and we are loosing a subset of genes which are not detectable because discarded before the analysis.

DESeq2 vignette

The object class used by the DESeq2 package to store the read counts and the intermediate estimated quantities during statistical analysis is the *DESeqDataSet*, which will usually be represented in the code here as an object `dds`.

The DESeqDataSet has an associated "design formula". The design is specified at the beginning of the analysis, as this will inform many of the DESeq2 functions how to treat the samples in the analysis (one exception is the size factor estimation – adjustment for differing library sizes – which does not depend on the design formula). The design formula tells which variables in the column metadata table (colData) specify the experimental design and how these factors should be used in the analysis.

The simplest design formula for differential expression would be `~ condition`, where condition is a column in colData(dds) which specifies which of two (or more groups) the samples belong to.

There are different ways of constructing a *DESeqDataSet*, depending on what pipeline was used upstream of DESeq2 to generate counts or estimated counts.

The function *DESeqDataSetFromMatrix* can be used if you already have a matrix of read counts prepared from another source. **The count values must be raw counts of sequencing reads.** This is important for DESeq2's statistical model to hold, as only the actual counts allow assessing the measurement precision correctly. Hence, please do not supply other quantities, such as (rounded) normalized counts, or counts of covered base pairs as this will only lead to nonsensical results.

To use *DESeqDataSetFromMatrix*, the user should provide the counts matrix, the information about the samples (the columns of the count matrix) as a *DataFrame* or *data.frame,* and the design formula.

Remember: in the count table, each row represents an Ensembl gene, each column a sequenced RNA library, and the values give the raw numbers of sequencing reads that were mapped to the respective gene in each library.

```
head(cts,2)
```

```
##             untreated1 untreated2 untreated3 untreated4 treated1 treated2
## FBgn0000003          0          0          0          0        0        0
## FBgn0000008         92        161         76         70      140       88
##             treated3
## FBgn0000003        1
## FBgn0000008       70
```

cts or cpm matrix (as you want to name it)

```
coldata
```

```
##              condition        type
## treated1fb     treated single-read
## treated2fb     treated  paired-end
## treated3fb     treated  paired-end
## untreated1fb untreated single-read
## untreated2fb untreated single-read
## untreated3fb untreated  paired-end
## untreated4fb untreated  paired-end
```

coldata