Module 3 - Analyse statistique avec R - Séance 1

DUBii 2019

Hugo Varet, Frédéric Guyon, Olivier Kirsh et Jacques van Helden

2019-02-05

R en quelques mots

Langage de programmation qui permet de :

- manipuler des données : importer, transformer, exporter
- faire des analyses statistiques plus ou moins complexes : description, exploration, modélisation...
- créer des (jolies) figures

Disponible sur Windows, MacOS, Linux

Historique :

- 1993 : début du projet R
- 2000 : sortie de R 1.0.0
- 2018 : R 3.5.1

Avantages et inconvénients

Avantages :

- Souplesse d'utilisation pour réaliser des analyses statistiques
- R est libre et gratuit, même s'il existe maintenant des versions payantes de RStudio (shiny et/ou server)
- Reproductibilité des analyses en écrivant/sauvegardant les commandes R dans des scripts
- Très largement utilisé par la communauté
- Très largement enrichi par la communauté : système de packages

Inconvénients :

Analyse de données vs langage de programmation

- Lire un tableau : `read.table()`
- Fusionner deux tableau : `merge()`
- Sélectionner des colonnes : `mydata[ , c("col1","col2")]`
- Calculer une moyenne : `mean(x)`
- Exporter un tableau de données : `write.table()`
- Régression linéaire : `lm(y ~ x)`
- Tester une hypothèse : `t.test()`
- Dessiner un histogramme : `hist()`
- Convertire des données : `as.data.frame()`
- Tracer une courbe : `plot()`
- Réaliser une ACP : `prcomp()`

Un script R ne doit pas être une boîte noire !

Modes d'utilisation (liste non exhaustive)

- Localement via le terminal : pas très convivial
- Localement via RStudio : utilisation classique
- Sur un serveur distant via le terminal et une connexion ssh : cluster de calculs de l'IFB
- Sur un serveur via un nagivateur pour accéder à RStudio server : cf slide suivante

Se connecter au serveur ou ouvrir RStudio

Les travaux pratiques seront réalisés sur le serveur RStudio sur IFB core cluster.

https://rstudio.cluster.france-bioinformatique.fr/

Identifiez-vous avec votre login du cluster IFB core. Ceci vous permettra d'accéder à votre dossier personnel à partir de l'interface de RStudio.

Définir et créer son dossier de travail pour ce TP

Définir une variable qui indique le chemin du dossier de travail

```
work.dir <- "~/intro_R"
```

S'il n'existe pas encore, créer le dossier de travail. (Commande Unix équivalente: "mkdir -p ~/intro_R")

```
dir.create(work.dir, recursive = TRUE, showWarnings = FALSE
```

Alternative : créer le répertoire `intro_R` en utilisant les fonctionnalités de RStudio

Explorer son dossier de travail

Aller dans ce dossier de travail. (Commande Unix équivalente: “cd ~/intro_R”)

```
setwd(work.dir)
```

Alternative : cliquer sur “Session/Set Working Directory/Choose Directory”

Où suis-je ? (Commande Unix équivalente: “pwd”)

```
getwd()
```

Qu’y a-t-il par ici ? (Commande Unix équivalente: “ls”)

```
list.files()
```

R vu comme une calculatrice

```
2 + 3
4 * 5
6 / 4
```

Notion de variable/objet

```
a <- 2          ## Assigner une valeur à une variable
print(a)        ## Afficher la valeur de la variable a

b <- 3          ## Assigner une valeur à une seconde variable
c <- a + b      ## Effectuer un calcul avec 2 variables
print(c)        ## Afficher le contenu de la variable c

a <- 7          ## Changer la valeur de a
print(c)        ## Note: le contenu de c n'est pas modifié
```

Les types de données élémentaires

- les nombres (réels par défaut): `numeric`, `double`

```
x <- 3.5
class(x)
is.double(x)
is.numeric(x)
```

- les nombres entiers: `integer`

```
x <- 3.5
x <- as.integer(x)
is.integer(x)
```

- les caractères: `character`

```
x <- "a"
x <- "toto"
```

Les types de données

- les booléens: `boolean`

```
x <- FALSE
x <- TRUE
as.integer(x)
```

- les facteurs: `factor`

très fréquent en R : on verra plus tard

Les types de données : les vecteurs

- tout est vecteur
- vecteur: ensemble de valeurs du même type et indexées

```
x <- c(1, 3, 4, 7)
x[1]
x[4]
length(x)
```

- listes: ensemble de valeurs de types différents et indexées

```
x <- list(names=c("toto", "titi", "tutu"),
          values=c(1, 2, 4))
print(x)
x[[2]]
x$values
```

Les types de données : les tableaux de données

- type de données typique de l'analyse de données
- data.frame: tableaux dont les colonnes sont de même types

```
A <- data.frame(names=c("toto", "titi", "tutu"),
                values=c(1, 2, 4))
print(A)
```

Manipulation des vecteurs

```
x <- rnorm(26)
print(x)
```

- indiçage des vecteurs: []

```
x[3]
```

- vecteur d'indices

```
x[c(3,5)]
x[1:6]
x[seq(1,100,by=2)]
```

- vecteur de booléens=on prend ou pas sous condition

```
x[x>=0]
```

Manipulation des vecteurs

- accés aux éléments de vecteur par le nom

```
names(x) <- letters # assigner des noms aux entrées du vec
head(x) # les premiers éléments, noter les noms associés a
x[c("a","t","z")] # imprimer des valeurs sélectionnées par
```

Listes

Une liste combine plusieurs sous-structures désignées par des noms ou des indices. Les sous-structures peuvent être de différents types (vecteurs, tableaux à deux dimensions, ou objets plus complexes).

Création d'une liste

```
x <- list(names = letters, values = 1:26)
print(x)
```

Accés aux éléments éléments de liste par le nom

```
x$names
x$values
```

Data frame

- data.frame = tableau à deux dimensions

```
A <- data.frame(names = c("toto", "titi", "tutu"), values =
A[1, 2]
A[2, 1]
A[1:2, ]
A[A[,2] >= 3, ]
```

- data.frame = liste de vecteur colonne

```
A$names
A$values
```

- ou bien

```
A[,"names"]
A[,"values"]
```

Les fonctions de base

- `+,-,*,\,**`
- `cos,sin,log,log10,exp`
- les fonctions sont toutes vectorielles

```
x <- runif(10)
x <- x+1
print(x)
cos(x)
y <- seq(0, pi, len=10)
x+y
x*y
```

Télécharger un fichier

La commande `download()` permet de télécharger un fichier à partir d'un serveur, et `dir.create()` permet de créer un nouveau dossier dans l'espace de travail:

```
dir.create("data")
download.file(url = "https://tinyurl.com/dubii-expressions"
# URL complete : https://raw.githubusercontent.com/DU-Bii/
```

```
download.file(url = "https://tinyurl.com/dubii-annotation",
# URL complete : https://raw.githubusercontent.com/DU-Bii/
```

Chargement des données

Charger le contenu du fichier “expression.txt” dans une variable nommée “exprs”.

```
exprs <- read.table(file = "data/expression.txt",
                    header = TRUE,
                    sep = "\t")
```

Question : à quoi servent les options `header` et `sep` ?

Réponse : appelez à l’aide (diapo suivante)

Afficher l'aide d'une fonction

```
help(read.table)
```

Notation alternative

```
?read.table
```

Affichage de l'objet "exprs"

La fonction `print()` imprime l'ensemble des valeurs d'une variable.

Quand on travaille avec un tableau de données omiques comportant des milliers de lignes, ce n'est pas forcément très utile d'afficher toutes les valeurs d'une table de données.

```
print(exprs)
```

	id	WT1	WT2	KO1	KO2
1	ENSG00000034510	235960	94264	202381	91336
2	ENSG00000064201	116	71	64	56
3	ENSG00000065717	118	174	124	182
4	ENSG00000099958	450	655	301	472
5	ENSG00000104164	4736	5019	4845	4934
6	ENSG00000104783	9002	8623	7720	7142
7	ENSG00000105229	1295	2744	1113	2887
8	ENSG00000105723	3353	7449	3589	7202

Affichage des premières lignes de l'objet

```
head(exprs)
```

	id	WT1	WT2	KO1	KO2
1	ENSG00000034510	235960	94264	202381	91336
2	ENSG00000064201	116	71	64	56
3	ENSG00000065717	118	174	124	182
4	ENSG00000099958	450	655	301	472
5	ENSG00000104164	4736	5019	4845	4934
6	ENSG00000104783	9002	8623	7720	7142

Un peu plus de lignes

```
head(exprs, n = 15)
```

	id	WT1	WT2	KO1	KO2
1	ENSG00000034510	235960	94264	202381	91336
2	ENSG00000064201	116	71	64	56
3	ENSG00000065717	118	174	124	182
4	ENSG00000099958	450	655	301	472
5	ENSG00000104164	4736	5019	4845	4934
6	ENSG00000104783	9002	8623	7720	7142
7	ENSG00000105229	1295	2744	1113	2887
8	ENSG00000105723	3353	7449	3589	7202
9	ENSG00000116199	2044	4525	2604	4902
10	ENSG00000118939	7022	2526	6269	3068
11	ENSG00000119285	15783	17359	18591	20077
12	ENSG00000121680	3133	2775	2045	2796
13	ENSG00000125384	1380	3079	869	2419

Caractéristiques d'un tableau

Dimensions

```
dim(exprs)     ## Dimensions
ncol(exprs)    ## Nombre de colonnes
nrow(exprs)    ## Nombre de lignes
```

Noms des lignes et colonnes

```
colnames(exprs)
rownames(exprs)
```

Résumé rapide des données par colonne

```
summary(exprs)
```

```
                id             WT1               WT2              
 ENSG00000034510: 1   Min.   :    31   Min.   :   43.0   Mi
 ENSG00000064201: 1   1st Qu.:   264   1st Qu.:  203.2   1s
 ENSG00000065717: 1   Median :  1338   Median : 1903.0   Me
 ENSG00000099958: 1   Mean   :  9358   Mean   : 6498.6   Me
 ENSG00000104164: 1   3rd Qu.:  3730   3rd Qu.: 4727.2   3r
 ENSG00000104783: 1   Max.   :235960   Max.   :94264.0   Ma
 (Other)        :44
```

Sélection de colonnes d'un tableau

Valeurs stockées dans la colonne nommée "WT1"

```
exprs$WT1
```

Notation alternative

```
exprs[, "WT1"]  ## Sélection de la colonne WT1
```

Sélection de plusieurs colonnes.

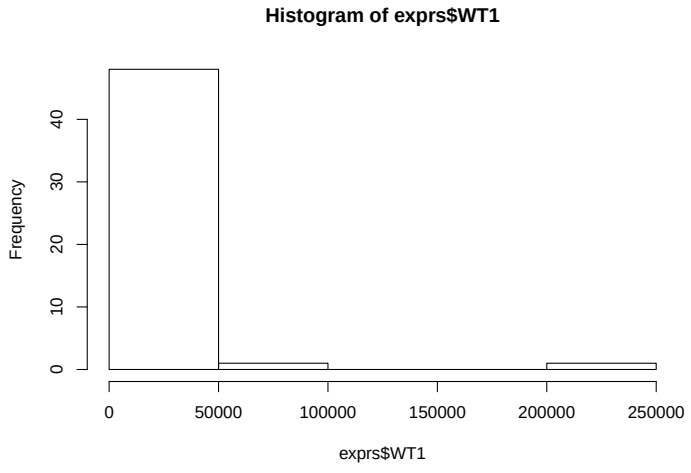
```
exprs[, c("WT1", "WT2")]
```

Sélection de colonnes par leur indice

```
exprs[, 2]
exprs[, c(2, 3)]
```

Histogramme des valeurs d'expression pour WT1

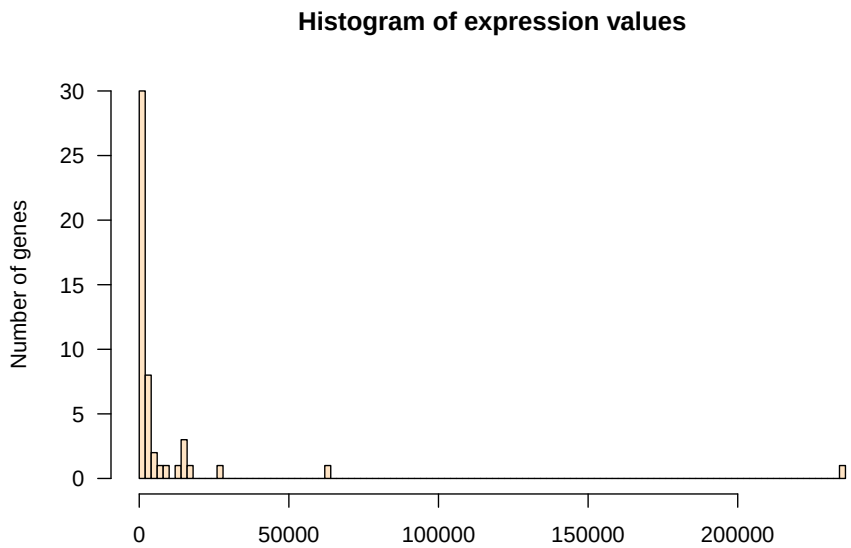
```
hist(exprs$WT1)
```



Exercice : améliorer ce graphique en modifiant la couleur de l'histogramme, en ajoutant un titre et des noms aux axes des abscisses et ordonnées.

Histogramme avec quelques options esthétiques

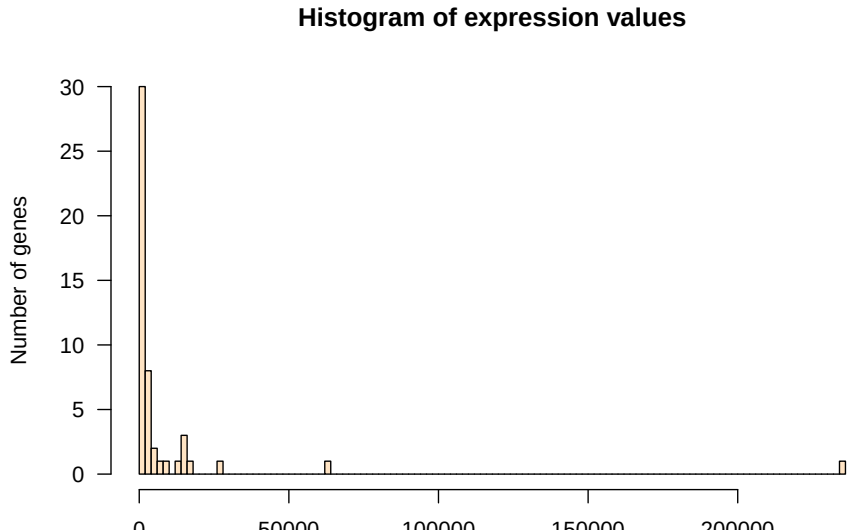
```
hist(exprs$WT1,
     breaks = 100, # class intervals
     main = "Histogram of expression values",
     xlab = "Counts per gene", # X label
     ylab = "Number of genes", # Y label
     las = 1, # Plot axis labels horizontally
     col = "bisque" # filling color
     )
```



Histogramme avec quelques options esthétiques

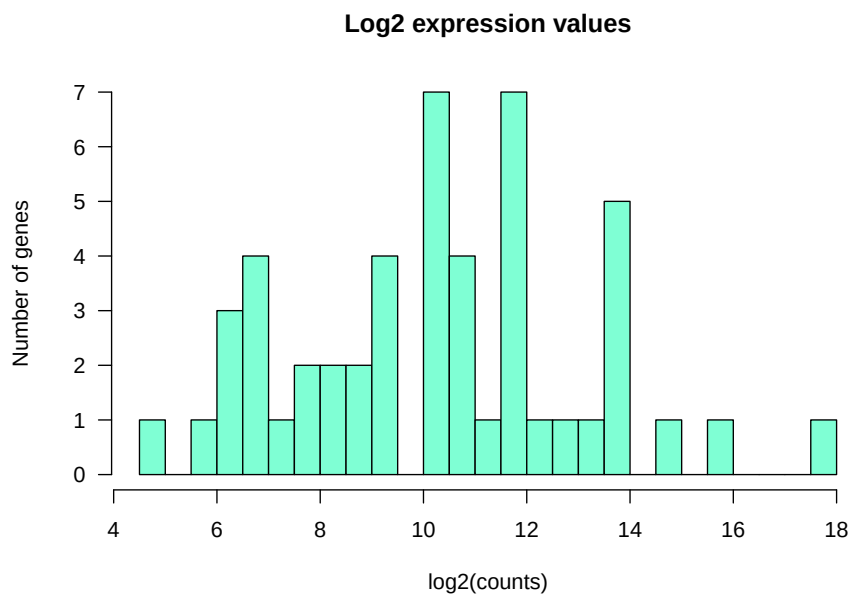
Remarques

- La distribution sur l'abcsisse est déséquilibrée: les valeurs les plus réfquentes sont "collées au mur" (concentrées sur la gauche) du fait d'une valeur aberrante (1 gène avec un très grand nombre de reads).
- L'histogramme n'est pas représentatif car pour ce tutoriel nous avons séletionné un tout petit nombre de gènes ($n = 50$). Nous traiterons un jeu de données complet à titre d'exercice.



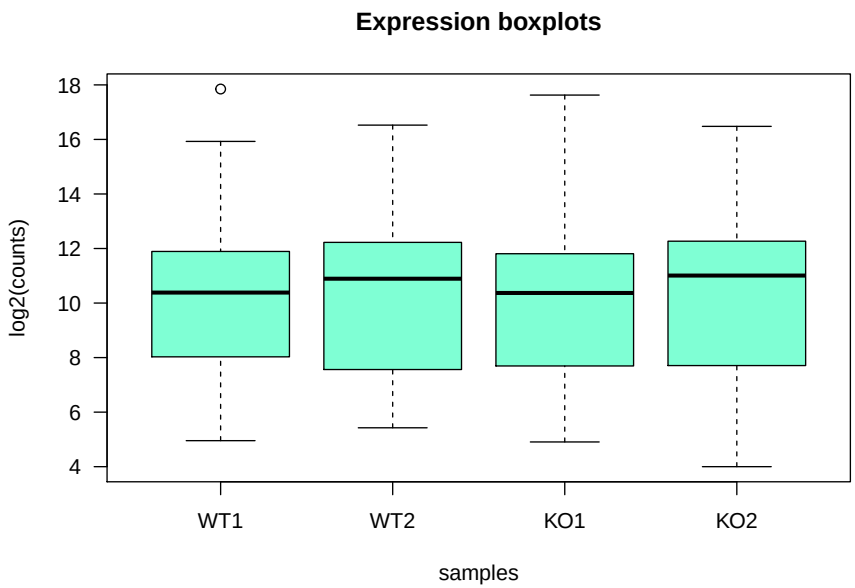
Histogramme du logarithme de ces valeurs

```
hist(log2(exprs$WT1), breaks = 20,
     main = "Log2 expression values",
     xlab = "log2(counts)", # X label
     ylab = "Number of genes", # Y label
     las = 1, # Plot axis labels horizontally
     col = "aquamarine" # filling color
     )
```



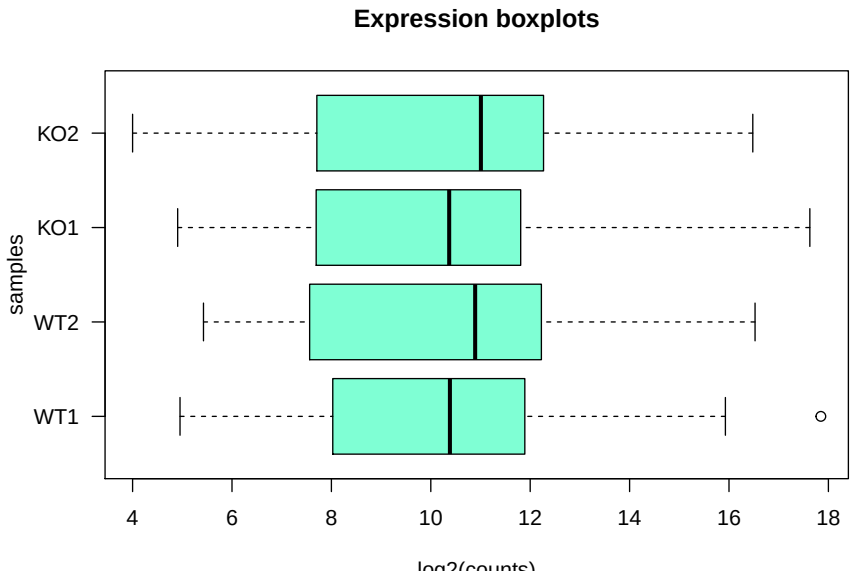
Boîtes à moustaches

```
boxplot(log2(exprs[, c("WT1", "WT2", "KO1", "KO2")]),
      main = "Expression boxplots",
      xlab = "samples", # X label
      ylab = "log2(counts)", # Y label
      las = 1, # Plot axis labels horizontally
      col = "aquamarine" # filling color
      )
```



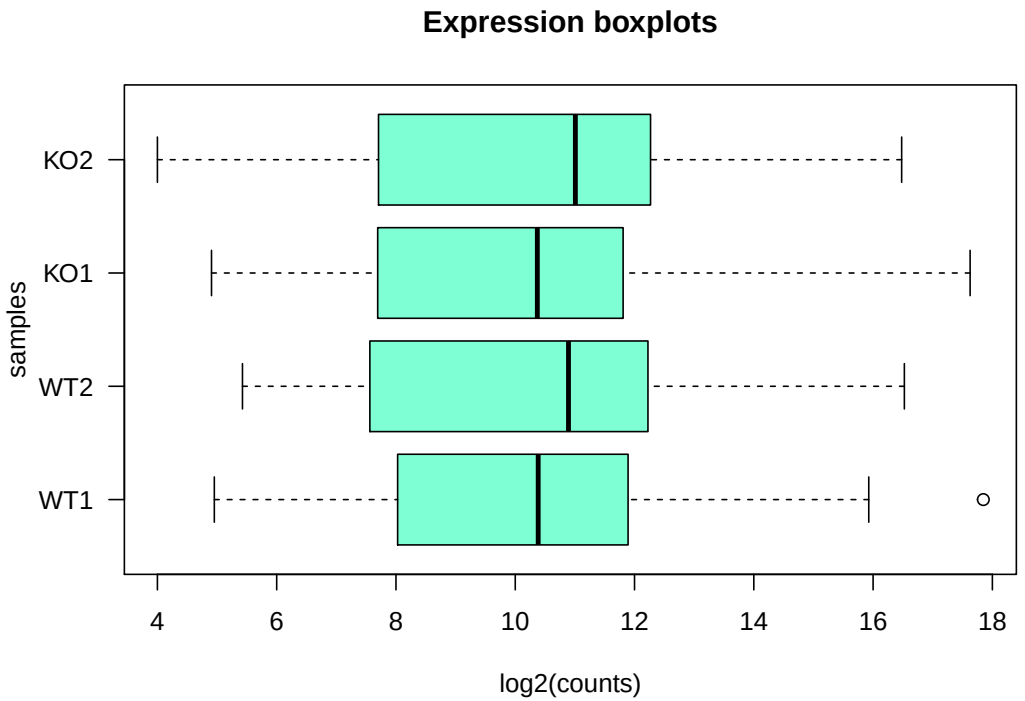
Boîtes à moustaches horizontales

```
boxplot(log2(exprs[, c("WT1", "WT2", "KO1", "KO2")]),
      main = "Expression boxplots",
      xlab = "log2(counts)", # X label
      ylab = "samples", # Y label
      las = 1, # Plot axis labels horizontally
      horizontal = TRUE, # plot boxplots horizontally
      col = "aquamarine" # filling color
      )
```



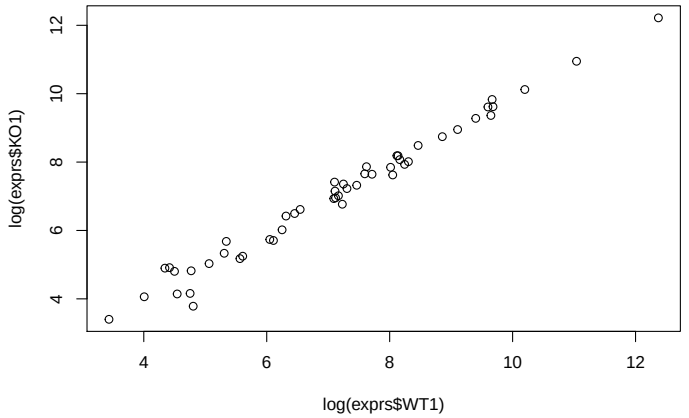
Pourquoi les boîtes à moustaches apparaissent-elles décalées ?

Remarque : le décalage entre boîtes nous indique que les librairies de comptage ne sont pas normalisées. Les méthodes de normalisation seront vues dans un cours ultérieur.



Nuages de points : expressions KO1 vs WT1

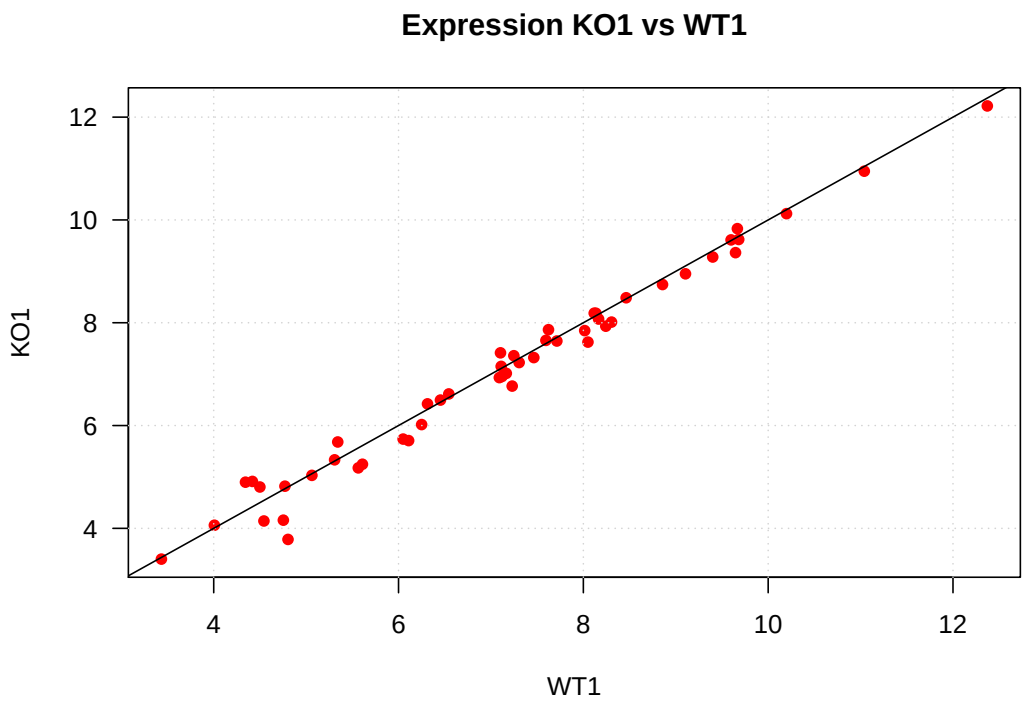
```
plot(x = log(exprs$WT1), y = log(exprs$KO1))
```



Exercice : améliorer ce graphique selon vos envies !

Personnalisation des paramètres graphiques

```
plot(x = log(exprs$WT1), y = log(exprs$KO1), main = "Expres
     xlab = "WT1", ylab = "KO1",  pch = 16, las = 1, col =
grid()                    # add a grid
abline(a = 0, b = 1) # add a diagonal line
```



Sélection de lignes d'un tableau

Sélection des lignes 4 et 11 du tableau des expressions

```
exprs[c(4, 11), ]
```

Indices des lignes correspondant aux IDs ENSG00000253991 et ENSG00000099958

```
which(exprs$id %in% c("ENSG00000253991", "ENSG00000099958")
```

Afficher les lignes correspondantes

```
gene.indices <- which(exprs$id %in% c("ENSG00000253991", "E
exprs[gene.indices, ]
```

Sélection de lignes et colonnes

On peut sélectionner à la fois des lignes et des colonnes en combinant les méthodes vues ci-dessus.

```
exprs[10:15, 1:5]
```

	id	WT1	WT2	KO1	KO2
10	ENSG00000118939	7022	2526	6269	3068
11	ENSG00000119285	15783	17359	18591	20077
12	ENSG00000121680	3133	2775	2045	2796
13	ENSG00000125384	1380	3079	869	2419
14	ENSG00000129562	12089	7958	10708	7683
15	ENSG00000129932	1744	2247	1513	3104

On peut également désigner les lignes ou les colonnes par leur nom.

Calculs sur des colonnes

Calcul de moyennes par ligne (`rowMeans`) pour un sous-ensemble donné des colonnes (WT1 et WT2).

```
rowMeans(exprs[,c("WT1","WT2")])
```

Ajout de colonnes avec les expressions moyennes des WT et des KO.

```
exprs$meanWT <- rowMeans(exprs[,c("WT1","WT2")])
exprs$meanKO <- rowMeans(exprs[,c("KO1","KO2")])

head(exprs) ## Check the result
```

Fold-change KO vs WT

```
exprs$FC <- exprs$meanKO / exprs$meanWT
head(exprs) ## Check the result
```

Moyenne de tous les échantillons

MA-plot: log2FC vs intensité

M est le logarithme en base 2 du rapport d'expression.

$$M = log_2(\text{FC}) = log_2\left(\frac{\text{KO}}{\text{WT}}\right) = log_2(\text{KO}) - log_2(\text{WT})$$

```
exprs$M <- log2(exprs$FC)
```

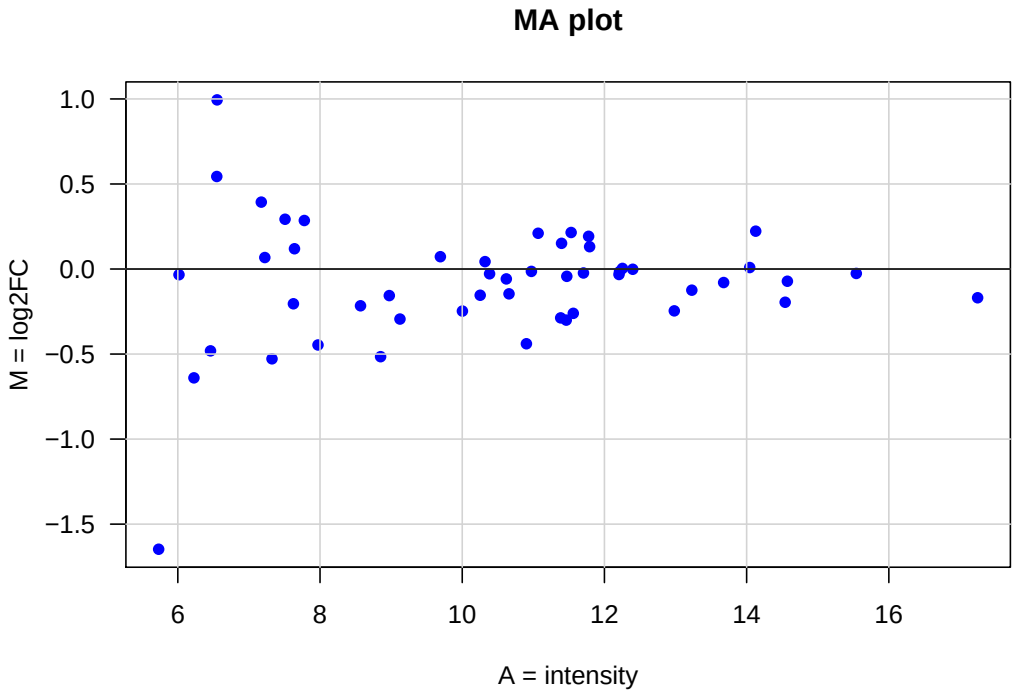
A (average intensity) est la moyenne des logarithmes des valeurs d'expression.

$$A = \frac{1}{2} log_2(\text{KO} \cdot \text{WT}) = \frac{1}{2}\left(log_2(\text{KO}) + log_2(\text{WT})\right)$$

```
exprs$A <- rowMeans(log2(exprs[,c("meanWT", "meanKO")]))
```

MA-plot : log2FC vs intensité

```
plot(x = exprs$A, y = exprs$M, main = "MA plot", las = 1,
     col = "blue", pch = 16, xlab = "A = intensity", ylab =
grid(lty = "solid", col = "lightgray")
abline(h = 0)
```



Charger les annotations des gènes

```
annot <- read.table(file = "data/annotation.csv", header =
dim(annot)    ## Vérifier les dimensions
head(annot)   ## Afficher quelques lignes
```

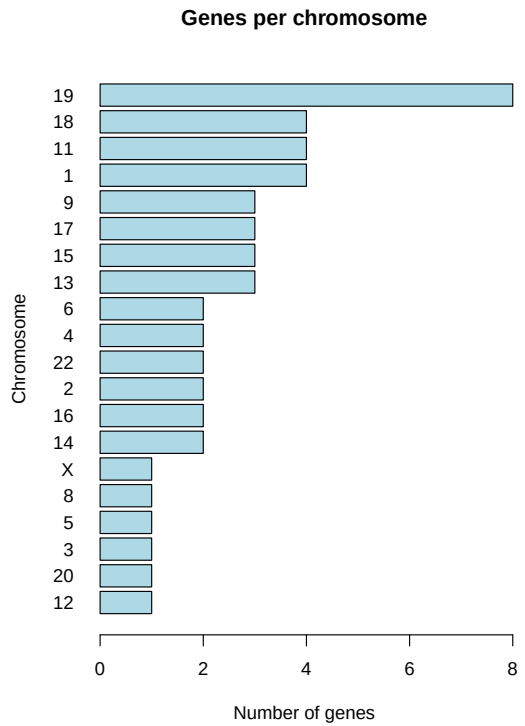
Combien de gènes par chromosome ?

```
table(annot$chr)
```

Question : combien de gènes sur le chromosome 8 ? Et sur le X ?

Diagramme en bâtons – gènes par chromosomes

```
barplot(sort(table(annot$chr)), horiz = TRUE, las = 1,
        main = "Genes per chromosome", ylab = "Chromosome",
        col = "lightblue", xlab = "Number of genes")
```



Sélectionner les données du chromosome 8

1ere étape: fusionner les deux tableaux exprs et annot

```
exprs.annot <- merge(exprs, annot, by = "id")
head(exprs.annot)
```

2eme étape: sous-ensemble des lignes pour lesquelles chr vaut 8

```
exprs8 <- exprs.annot[which(exprs.annot$chr == 8),]
print(exprs8)
```

Exporter exprs8 dans un fichier

```
write.table(x = exprs8, file = "exprs8.txt", sep = "\t",
            row.names = TRUE, col.names = NA)
```

Les graphiques en R

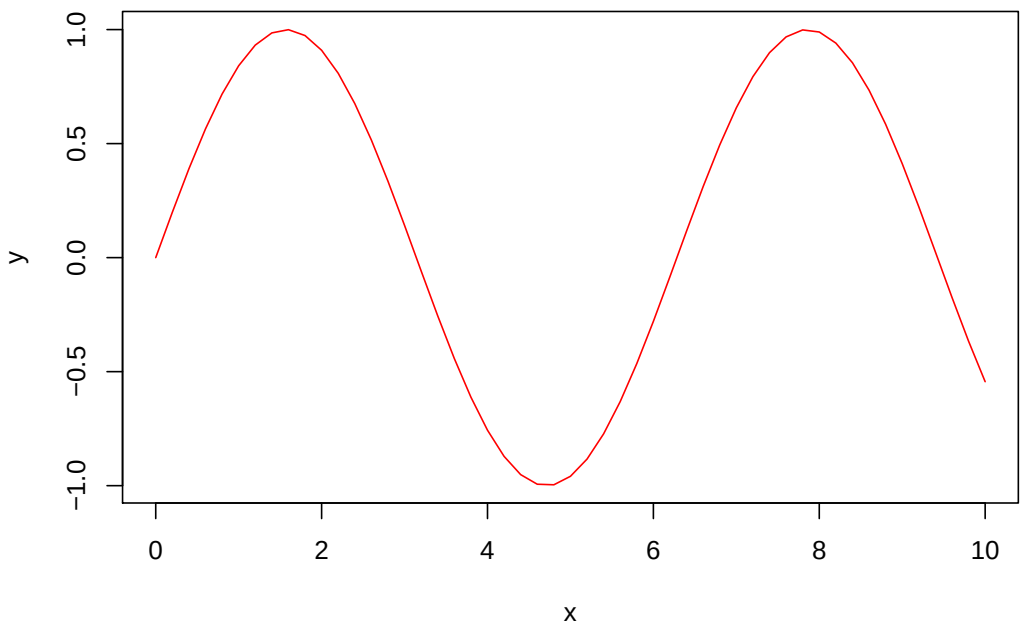
- Fonction générique pour le graphisme : *plot*

```
plot(x, y,
     xlim = range(x),
     ylim = range(y),
     type = "p",
     main, xlab, ylab, ...)
```

Les graphiques en R

- Exemple

```
x <- seq(0,10,0.2)
plot (x, sin(x), type="l",col="red", xlab="x", ylab="y")
```



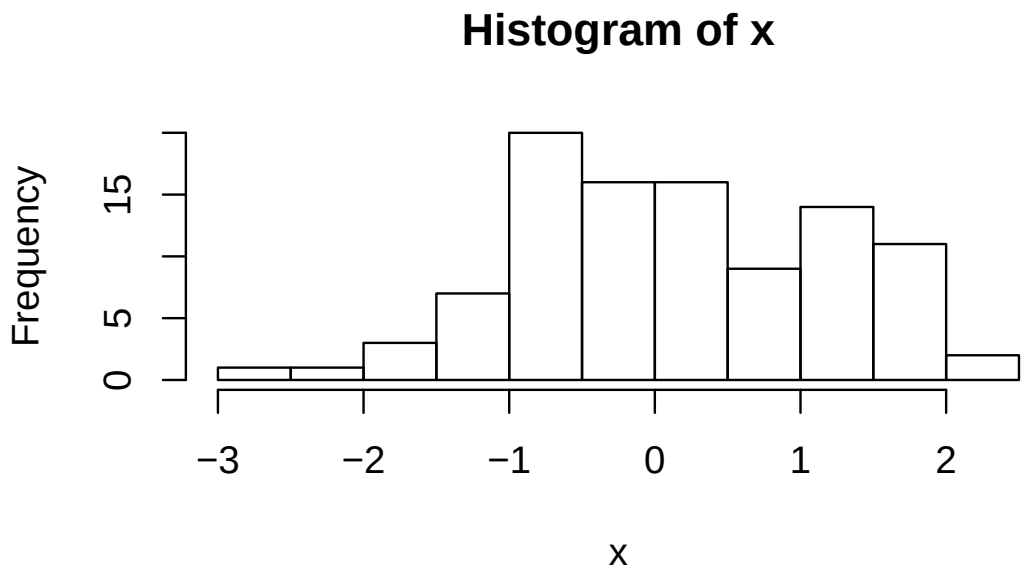
Les graphiques en R

Autres fonctions graphiques de haut-niveau :

- histograms: `hist()`
- bar plot with vertical or horizontal bars: `barplot()`
- contour plots or level plots: `contour()`
- images: `image()`
- surfaces: `persp()`
- 3D: `plot3d()`

Les graphiques en R

```
x <- rnorm(100)
hist(x)
```



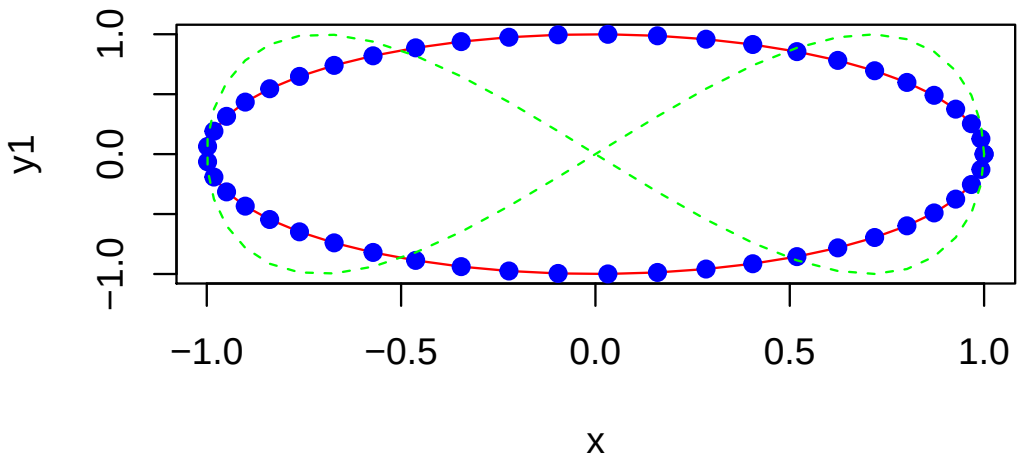
Les graphiques en R

Fonctions graphiques de bas niveau : permettent d'ajouter des éléments à un graphique déjà ouvert.

- des points: `points()`
- un titre: `title()`
- une légende: `legend()`
- des droites: `abline()`
- des lignes: `lines`

Les graphiques en R: ajout d'éléments

```
theta <- seq(0,2*pi,len=50)
x <- cos(theta)
y1 <- sin(theta)
y2 <- sin(2*theta)
plot(x, y1, type="l", col="red")
points(x, y1, pch=19, col="blue")
lines(x, y2, lty=2, col="green")
```



Les fonctions en R

```
ma_fonction <- function(nom_arg1, nom_arg2){
# des calculs...
return(valeur_retour)
}
```

- `ma_fonction`: nom de la fonction (variable comme les autres)
- `valeur1, valeur2`: arguments de la fonction
- `valeur_retour`: la dernière ligne donne la valeur retournée par la fonction
- exemple simple:

```
add1 <- function(x) {
   return(x+1)
}

add1(2)
```

Les fonctions en R

- les arguments ont un nom et peuvent avoir une valeur par défaut
- à l'appel de la fonction:
  - les arguments sont dans l'ordre
  - dans le désordre si ils sont nommés
  - peuvent être absents, si valeur par défaut
- exemple simple:

```
add2 <- function(x, y, z=0) {
   x+2*y+3*z
}
add2(1, 2)
add2(y=1, x=2)
add2(1, 2, 3)
add2(y=1, x=2, 3)
```

Pourquoi documenter son code ?

Quel que soit le langage de programmation utilisé, il est crucial de documenter son code.

- pour le rendre compréhensible pour d'autres personnes,
- pour s'y retrouver quand on devra le modifier quelques mois plus tard.

Comment documenter son code ?

En R, le caractère # marque le début d'un commentaire. Le texte qui suit est ignoré, jusqu'à la fin de la ligne.

- Avant un bloc de code, annoncer à quoi il sert.
- Vous pouvez également ajouter un commentaire en fin de ligne (par exemple pour décrire les variables)

```
# Calcul de l'espérance d'un coup de dé
p <- 1/6            # Probabilité de chaque face
valeurs <- 1:6      # valeurs associées aux  faces

# L'éspérance est la moyenne attendue au terme d'un nombre
# On la calcule par la somme des produits des valeurs par
sum(p * valeurs)
```

Take home messages

- Tout est faisable avec R
- Définir et comprendre l'opération mathématique/statistique avant de chercher la fonction R correspondante
- R est un langage :
  - plusieurs types et structures de données
  - énormément de commandes à connaître
  - Google est votre ami
- Une infinité de :
  - ressources en ligne
  - tutoriels pour des analyses spécifiques (e.g. DESeq2 pour le RNA-Seq)