Analyse des données génomiques 2020
Importation de données d'expression et contrôle qualité

Sandrine Lagarrigue et David Causeur

Nom Prénom :

Dans la suite, on reprend les étapes, mises en oeuvre grâce au package *limma* de *R*, de l'importation et du contrôle qualité des données d'expression des gènes de poulets dans le foie. **A partir des données disponibles sur la plateforme moodle, modifiez les commandes ci-après afin de les adapter au sujet de votre étude de cas.** En particulier,

- les données d'expression pouront concerner un autre tissu que celui du foie (muscle ou tissu adipeux) ;
- de même, les facteurs de variation des profils d'expression pourront ne pas être limités au régime et au génotype. En effet, ils pourront aussi être choisis dans le tableau de données supplémentaires sur les poulets également disponible (profils d'acide gras, ...).

1 Préparation de la session de travail

La commande d'installation ci-après du package *limma* à partir de la plateforme *bioconductor* ne doit être exécutée que si le package n'est pas déjà installé.

```
if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")

BiocManager::install("limma", update = FALSE)
```

Une fois installé, le package doit être chargé dans la session de travail :

```
require(limma)
```

2 Importation des données

2.1 Dispositif expérimental

Toute l'information sur le dispositif expérimental est contenue dans le fichier 'targetsF.txt'. On peut importer ce fichier grâce à la fonction *readTargets* du package *limma* :

```
targets = readTargets("targetsF.txt")
head(targets)  # Display the first 6 rows
```

```
  ArrayName Factor
1      F_01   M_BL
2      F_02   G_BL
3      F_03   M_HL
4      F_04   G_HL
5      F_05   G_HL
6      F_06   M_HL
                                                                ArrayFile
1 ./Data/Foie/F01_US10453848_254200410001_S01_GE1_107_Sep09_1_3.txt
2 ./Data/Foie/F02_US10453848_254200410001_S01_GE1_107_Sep09_2_1.txt
3 ./Data/Foie/F03_US10453848_254200410001_S01_GE1_107_Sep09_2_4.txt
```

```
4 ./Data/Foie/F04_US10453848_254200410001_S01_GE1_107_Sep09_1_2.txt
5 ./Data/Foie/F05_US10453848_254200410001_S01_GE1_107_Sep09_2_2.txt
6 ./Data/Foie/F06_US10453848_254200410001_S01_GE1_107_Sep09_1_4.txt
```

En particulier, la colonne *Factor* contient toute l'information sur les deux facteurs expérimentaux du dispositif, à savoir le régime et le génotype.

```
dietgenotype = factor(targets$Factor)
table(dietgenotype)
```

```
dietgenotype
G_BL G_HL M_BL M_HL
  12   12   12   12
```

La fonction *substring* (extrait une sous-chaîne de caractères) permet d'extraire les informations relatives au régime et au génotype à partir des chaînes de caractères stockées dans *dietgenotype* :

```
genotype = factor(substring(dietgenotype, first = 1, last = 1))
diet = factor(substring(dietgenotype, first = 3, last = 4))
```

La commande suivante produit la table de contingence des deux facteurs expérimentaux, ce qui permet de s'assurer de l'équilibre du dispositif expérimental :

```
table(diet, genotype)
```

```
    genotype
diet  G  M
  BL 12 12
  HL 12 12
```

2.2 Données supplémentaires sur les poulets

On importe les données contenues dans le fichier au format *.csv* disponible sur la plateforme (il a été généré en enregistrant le fichier Excel au format CSV, séparateur : point-virgule), grâce à la fonction *read.csv2* :

```
external_data = read.csv2("44variables_poulet9S_V1.csv")
str(external_data)
```

```
'data.frame':   48 obs. of  44 variables:
 $ No_ordre             : int  1 2 3 4 5 6 7 8 9 10 ...
 $ No_Bague_droite      : int  3696 3931 3459 4081 3905 3560 3864 3634 3711 4054 ...
 $ Lignée               : Factor w/ 2 levels "G","M": 2 1 2 1 1 2 1 2 2 1 ...
 $ Régime               : Factor w/ 2 levels "BL","HL": 1 1 2 2 2 2 1 1 1 1 ...
 $ Lot                  : Factor w/ 4 levels "G-BL","G-HL",..: 3 1 4 2 2 4 1 3 3 1 ...
 $ PoidsVif_S9_g        : int  2185 2593 2400 2102 2631 2116 2458 2495 2414 2273 ...
 $ GrasAbdo_g           : num  43 104.7 42.7 40.6 68.6 ...
 $ Gras_arrière2cuisse_g: num  8.39 15.48 6.54 4.65 3.88 ...
 $ Foie_g               : num  42.1 51.8 51.7 35.4 43.9 38 52.5 53.9 53.8 51.6 ...
 $ Cœur_g               : num  14.4 12.7 12.5 15.5 16 16.5 15.3 14.3 13.8 14.8 ...
 $ PectMajor_g          : num  87.9 105.2 100.6 85.7 111.1 ...
 $ plasma_AcÜrique      : num  114.8 112.4 48.2 77.4 73.9 ...
 $ plasma_B.OH          : num  1016 345 780 452 318 ...
 $ plasma_Urée          : num  37.3 32.3 40.6 25.1 25.4 ...
 $ plasma_Chol_total    : num  1486 1298 1031 1329 1149 ...
 $ plasma_PPL           : num  2.61 2.12 1.84 2.01 1.8 1.59 2.13 1.68 2.21 2.13 ...
 $ plasma_TG            : num  702 729 644 599 537 ...
 $ plasma_T3            : num  2.44 1.36 2.08 2.19 2.08 ...
 $ plasma_T4            : num  20.9 15.8 18.7 20.8 18.7 ...
 $ plasma_Insuline      : int  123 77 235 53 93 121 153 154 63 101 ...
 $ foie_C14             : num  0.47 0.68 0.55 0.37 0.28 0.26 0.67 0.15 0.77 0.52 ...
```

```
 $ foie_C16             : num  24.3 24.5 23.5 20 18.1 ...
 $ foie_C16.1           : num  2.49 2.8 1.71 0.62 0.55 0.59 3.39 2.69 2.52 2.59 ...
 $ foie_C18             : num  20 16.6 19.3 24.8 25.1 ...
 $ foie_C18.1           : num  25.5 35.9 31.5 19.2 19.3 ...
 $ foie_C18.2           : num  15.7 11.5 14.4 20 21 ...
 $ foie_C18.3           : num  0.77 0.46 0.73 0.97 0.94 1.7 0.44 0.38 0.42 0.45 ...
 $ foie_C20             : num  0.18 0.26 0.11 0.18 0.17 0.16 0.11 0.13 0.19 0.19 ...
 $ foie_C20.1           : num  0.22 0.31 0.38 0.31 0.26 0.38 0.32 0.34 0.4 0.34 ...
 $ foie_C20.4           : num  7.42 5.03 5.47 9.74 10.2 9.19 2.95 5.58 7.06 7.86 ...
 $ foie_C20.5           : num  0.75 0.54 0.39 0.52 0.4 0.53 0.29 0.38 0.56 0.53 ...
 $ foie_C22.5           : num  0.71 0.37 0.41 0.62 0.62 0.66 0.18 0.33 0.42 0.38 ...
 $ foie_C22.6           : num  1.58 1.13 1.51 2.68 3.14 2.78 0.68 0.18 1.87 1.99 ...
 $ foie_AGS             : num  44.9 42 43.5 45.3 43.6 ...
 $ foie_AGMI            : num  28.2 39 33.6 20.1 20.1 ...
 $ foie_AGPI            : num  26.9 19 22.9 34.5 36.3 ...
 $ foie_n.6             : num  23.1 16.5 19.8 29.7 31.2 ...
 $ foie_n.3             : num  3.81 2.5 3.04 4.79 5.1 5.67 1.59 1.27 3.27 3.35 ...
 $ foie_n6ONn3          : num  6.06 6.61 6.52 6.21 6.12 ...
 $ foie_LCn3            : num  3.04 2.04 2.31 3.82 4.16 3.97 1.15 0.89 2.85 2.9 ...
 $ foie_PERC.lipid      : num  3.35 6.35 6.29 3.29 5.35 4.92 5.82 5.99 4.02 2.4 ...
 $ foie_glycogen        : num  420 232 454 246 238 ...
 $ foie_Lactate         : num  5.18 4.98 5.12 5.75 7.19 6.83 3.77 6.47 7.52 6.41 ...
 $ foie_PG              : num  844 468 914 498 483 ...
```

La commande ci-dessus donne un aperçu des 44 variables contenues dans ces données supplémentaires. En particulier, la 1ère colonne est un identifiant des lignes, dont l'ordre doit être cohérent avec le numéro figurant dans l'identifiant des lignes du tableau *targets*. Pour s'en assurer, on commence par extraire ce numéro des chaînes de caractères *ArrayName* :

```
# Extracts the microarray number from ArrayName in targets
ArrayNumber = substring(targets$ArrayName, first = 3, last = 4)
ArrayNumber = as.numeric(ArrayNumber)
```

On peut maintenant vérifier l'exacte correspondance entre les identifiants des deux tableaux :

```
# Checks the consistency with the numbers in the 1st column of external_data
all(ArrayNumber == external_data[, 1])
```

```
[1] TRUE
```

La valeur *TRUE* retournée par la commande ci-dessus nous assure d'une égalité terme à terme parfaite entre les vecteurs d'identifiants des tableaux *targets* et *external_data*.

2.3 Données d'expression de gènes

La colonne *ArrayFile* du tableau *targets* contient les noms des fichiers dans la forme sous laquelle ils seront importés.

```
head(targets)  # Display the first 6 rows
```

```
  ArrayName Factor
1      F_01   M_BL
2      F_02   G_BL
3      F_03   M_HL
4      F_04   G_HL
5      F_05   G_HL
6      F_06   M_HL
                                                                ArrayFile
1 ./Data/Foie/F01_US10453848_254200410001_S01_GE1_107_Sep09_1_3.txt
2 ./Data/Foie/F02_US10453848_254200410001_S01_GE1_107_Sep09_2_1.txt
3 ./Data/Foie/F03_US10453848_254200410001_S01_GE1_107_Sep09_2_4.txt
```

```
4 ./Data/Foie/F04_US10453848_254200410001_S01_GE1_107_Sep09_1_2.txt
5 ./Data/Foie/F05_US10453848_254200410001_S01_GE1_107_Sep09_2_2.txt
6 ./Data/Foie/F06_US10453848_254200410001_S01_GE1_107_Sep09_1_4.txt
```

La commande ci-dessus montre que cette colonne doit être modifiée de telle sorte que n'y figurent que les noms des fichiers, en supprimant la mention au chemin vers ces fichiers. Pour cela, on extrait des noms actuels des fichiers la sous-chaîne de caractères commençant au 13ème caractère :

```
arrayfile = targets$ArrayFile
targets$ArrayFile = substring(arrayfile, first = 13, last = nchar(arrayfile))
```

La fonction *read.maimages* du package *limma* permet à la fois d'importer les données d'expression et de contrôler leur qualité, selon des règles que l'utilisateur se donne via une fonction qui associe à chaque spot sur chaque microarray la valeur 0 ou 1, 0 si la mesure sur ce spot est de mauvaise qualité, 1 sinon. La fonction que l'on propose ci-après impose trois conditions pour qu'un 'spot' soit considéré comme donnant une mesure utile et de bonne qualité :

- le 'spot' doit correspondre à un gène ;
- l'expression du gène doit être bien supérieure au bruit de fond ;
- l'image du spot doit être de bonne qualité (pas de signalement par Agilent, répartition spatiale uniforme des pixels)

```
MyFiltering <- function(x) {
    okType = x$ControlType == 0  # Probe=gene
    okExpress = x$gIsWellAboveBG == 1  # Gene expression >> background
    okSpotQuality = (x$IsManualFlag == 0) & (x$gIsFeatNonUnifOL == 0)
    # No flag + no spatial aggregates
    as.numeric(okType & okExpress & okSpotQuality)
}
```

L'importation peut commencer avec la fonction *read.maimages*, en spécifiant bien qu'il s'agit de données d'hybridation mono-couleur (*green.only=TRUE*) et que les mesures d'expression et de bruit de fond sont des niveaux de couleurs médians (*columns=list(E = "gMedianSignal", Eb = "gBGMedianSignal")*) :

```
G = read.maimages(files = targets$ArrayFile, names = targets$ArrayName, source = "agilent"
    green.only = TRUE, wt.fun = MyFiltering, columns = list(E = "gMedianSignal",
        Eb = "gBGMedianSignal"))
```

L'objet *G* créé par la commande ci-dessus est une liste à 6 composantes :

```
typeof(G)  # G is a list \t\t
```

```
[1] "list"
```

```
names(G)  # G has 6 components
```

```
[1] "E"       "Eb"      "weights" "targets" "genes"   "source"
```

Une de ces composantes, *G$genes* est elle-même une liste donnant des informations spécifiquement sur les sondes (position sur la puce, type, etc.) :

```
str(G$genes)  # G$genes gives information on probes
```

```
'data.frame':   62669 obs. of  5 variables:
 $ Row           : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Col           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ ControlType   : int  1 1 1 0 0 0 0 0 0 0 ...
 $ ProbeName     : chr  "GE_BrightCorner" "DarkCorner" "DarkCorner" "seq_RIGG02544" ...
 $ SystematicName: chr  "GE_BrightCorner" "DarkCorner" "DarkCorner" "seq_RIGG02544" ...
```

Le tableau *G$E* est une matrice dont les colonnes sont les profils complets d'expression sur chaque microarray :

```
dim(G$E)  # G$E is the probe x microarray matrix of gene expressions
```

```
[1] 62669    48
```

```
head(G$E[, 1:10])  # Display the first 6 row and 10 columns
```

```
        F_01    F_02  F_03    F_04  F_05    F_06    F_07    F_08  F_09
[1,] 47929.5 34561.0 46665 45209.0 17968 21164.0 18489.0 27677.0 21499
[2,]    29.0    41.0    31    33.0    33    23.0    32.0    27.0    26
[3,]    29.0    35.0    30    30.0    29    31.0    26.0    28.0    27
[4,] 15955.5 11706.0 11007 13934.0 15008 13743.0 12784.0 15633.5 16467
[5,]    92.5    88.5    79    96.5    77    63.0   104.5    80.5    97
[6,]    40.0    40.0    38    62.0    32    33.5    36.5    36.0    32
        F_10
[1,] 22092.0
[2,]    31.0
[3,]    31.0
[4,] 12245.0
[5,]   107.0
[6,]    53.5
```

Dans un premier temps, on va restreindre les données aux spots porteurs d'une mesure d'expression d'un gène. Ils sont identifiables par le fait que *G$genes$ControlType* prend la valeur 0 :

```
G = G[G$genes$ControlType == 0, ]
dim(G)
```

```
[1] 61350    48
```

3 Contrôle de la qualité des données d'expression

La définition d'une mesure de bonne qualité sur un spot est donnée par la fonction de filtrage utilisée lors de la procédure d'importation des données. Le résultat prend la forme d'une matrice *G$weights* dont les dimensions sont les mêmes que *G$E* et dont l'élement (i,j) vaut 1 si le $i$ème spot sur la $j$ème microarray est de bonne qualité :

```
dim(G$weights)
```

```
[1] 61350    48
```

```
head(G$weights[, 1:10])  # G$weights is a probe x microarray matrix of boolean values
```

```
     F_01 F_02 F_03 F_04 F_05 F_06 F_07 F_08 F_09 F_10
[1,]    1    1    1    1    1    1    1    1    1    1
[2,]    1    1    1    1    1    1    1    1    1    1
[3,]    0    0    0    1    0    0    0    0    0    1
[4,]    0    0    0    1    0    0    0    0    0    1
[5,]    1    1    1    1    1    1    1    1    1    1
[6,]    1    1    1    1    1    1    1    1    1    1
```

```
# 0 = bad spot, 1 = good spot
```

3.1 Qualité des microarrays

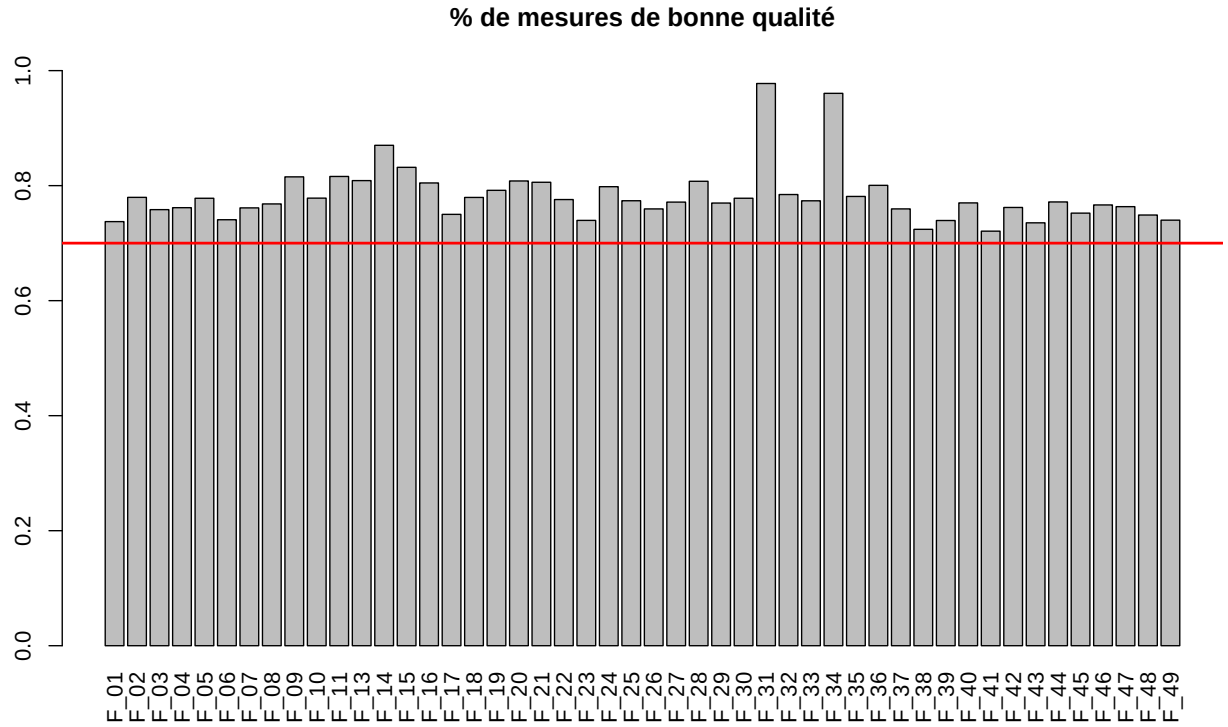
La proportion des mesures de bonne qualité par microarray s'obtient facilement en calculant les moyennes par colonne des valeurs de la matrice *G$weights* :

```
arrayquality = colMeans(G$weights)  # Proportions of good spots on each microarray
summary(arrayquality)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.721   0.759   0.773   0.782   0.799   0.978
```

Les commandes ci-dessus révèlent que la proportion moyenne de mesures de bonne qualité par microarray est satisfaisante. Le graphique suivant permet de visualiser la répartition de ces proportions de mesures de bonne qualité :

```
barplot(arrayquality, ylim = c(0, 1), las = 3, main = "% de mesures de bonne qualité")
abline(h = 0.7, col = "red", lwd = 2)
```



Le graphique ci-dessus met en évidence deux microarrays dont les proportions de mesures de bonne qualité sont très différentes de celles des autres microarrays. Cette différence en fait des microarrays suspectes, que l'on suggère de retirer de l'étude :

```
out = which(arrayquality > 0.9)  # Suspicious microarrays
G = G[, -out]  # Two microarays are removed
dietgenotype = dietgenotype[-out]  # Removed from the design
diet = diet[-out]  # Removed from the diet variable
genotype = genotype[-out]  # Removed from the genotype variable
targets = targets[-out, ]  # Removed from targets
external_data = external_data[-out, ]  # Removed from external_data
```

3.2 Qualité des sondes (gènes)

De la même manière que précédemment, on peut calculer des proportions de mesures de bonne qualité par sonde, au sein de chaque lot expérimental (une lignée x un régime):

```
genesqualityGBL = rowMeans(G$weights[, dietgenotype == "G_BL"])
genesqualityMBL = rowMeans(G$weights[, dietgenotype == "M_BL"])
genesqualityGHL = rowMeans(G$weights[, dietgenotype == "G_HL"])
genesqualityMHL = rowMeans(G$weights[, dietgenotype == "M_HL"])
```

On propose de considérer qu'une sonde doit être conservée si, au moins dans un lot expérimental, la proportion de mesures de bonne qualité pour cette sonde dépasse 75%:

```
okgenes = (genesqualityGBL >= 0.75) | (genesqualityMBL >= 0.75) | (genesqualityGHL >=
    0.75) | (genesqualityMHL >= 0.75)
```

Le vecteur *okgenes* créé ci dessus contient autant de valeurs booléennes que de sondes : *TRUE* si la sonde peut être conservée, *FALSE* sinon. Au total, le nombre de sondes à conserver est donné par la commande suivante :

```
sum(okgenes)
```

```
[1] 46741
```

Finalement, on réduit encore le tableau des données d'expression en ne conservant que les sondes ayant passé avec succès le contrôle qualité :

```
G = G[okgenes, ]
dim(G)  # 46741 probes x 46 microarrays
```

```
[1] 46741    46
```

4 Normalisation des données

On applique dans la suite plusieurs étapes de normalisation des données d'expression :

- transformation logarithmique,
- gestion du bruit de fond,
- identification d'écarts de répartition entre microararys et correction,
- gestion des réplicats.

4.1 Transformation logarithmique

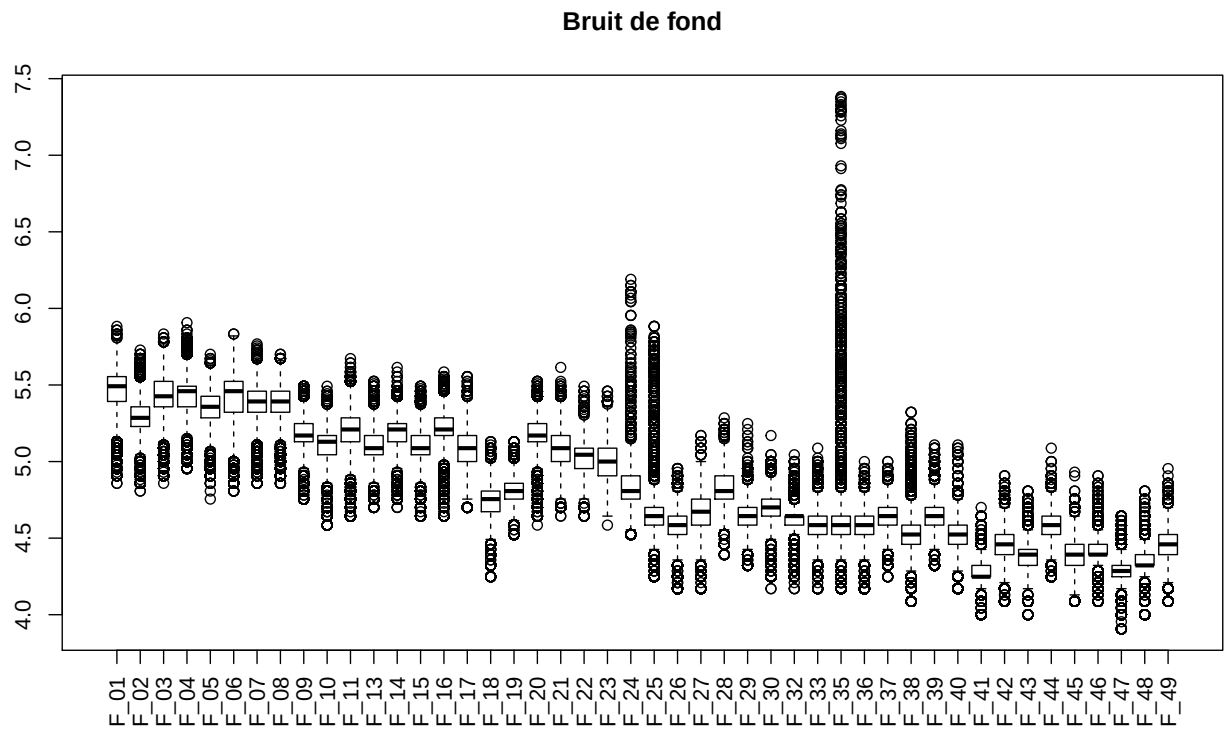
Les données d'expression et de bruit de fond sont transformées par la fonction log_2 :

```
G$E = log2(G$E)  # log-transformation of expression data
G$Eb = log2(G$Eb)  # log-transformation of background values
```

4.2 Gestion du bruit de fond

Le graphique suivant permet de comparer les niveaux de bruit de fond par microarray :

```
boxplot(G$Eb, main = "Bruit de fond", names = targets$ArrayName, las = 3)
```



Il montre des variations évidentes des niveaux du bruit de fond. De plus, ces variations ne semblent pas aléatoire, mais décrivant une tendance progressive à la baisse dans le temps, ce qui laisse supposer qu'elles résultent d'actions correctives des opérations ayant généré les données. Ces différences de bruit de fond peuvent remettre en cause l'équité d'une étude comparative des signaux mesurés par microarray (en effet, un signal est en réalité la somme d'une expression et d'un bruit de fond).

Toutefois les différences entre microarrays étant jugées faibles, on décide de faire l'hypothèse qu'elles n'ont pas d'impact sur la comparaison des signaux d'expression :

```
G = backgroundCorrect(G, method = "none")
# Another possibility is method='subtract'
names(G)
```

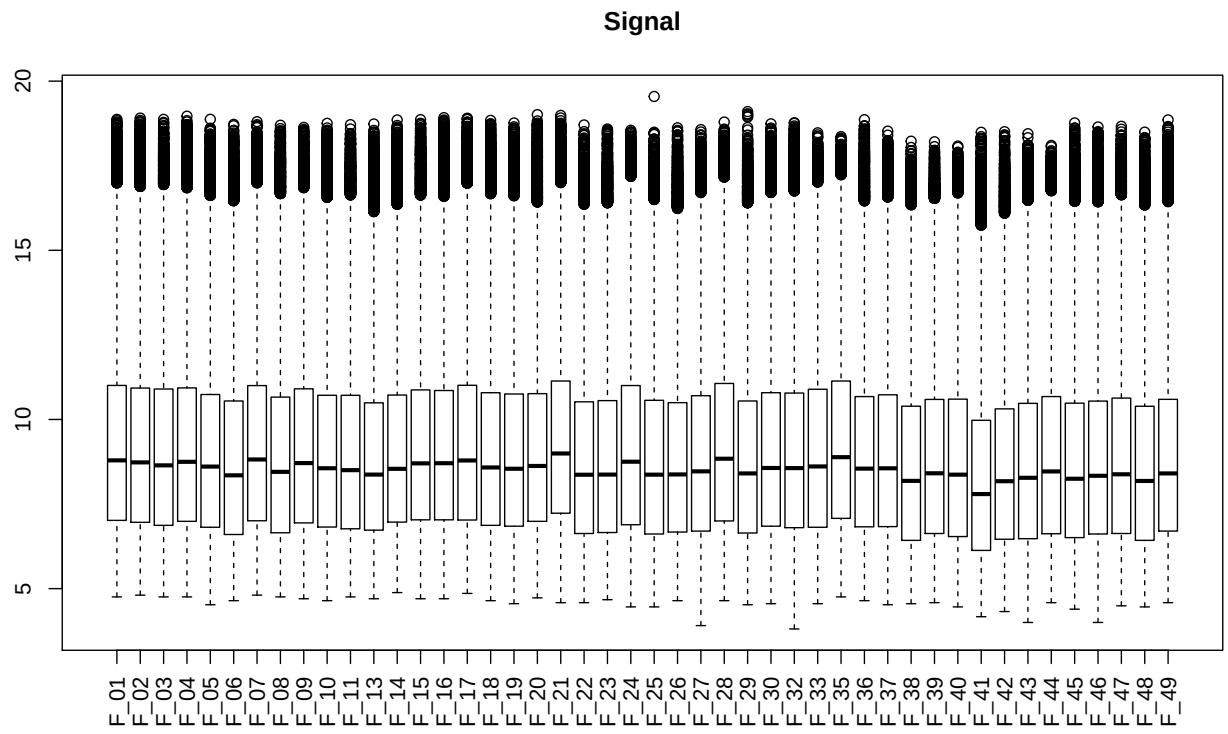
```
[1] "E"       "weights" "targets" "genes"   "source"
```

```
# The background component has been removed. The matrix E is unchanged.
```

4.3 Différences de répartition entre microarrays

De la même manière que pour le bruit de fond, le graphique suivant décrit les variations de répartition des signaux d'expression par microarray :

```
boxplot(G$E, main = "Signal", names = targets$ArrayName, las = 3)
```



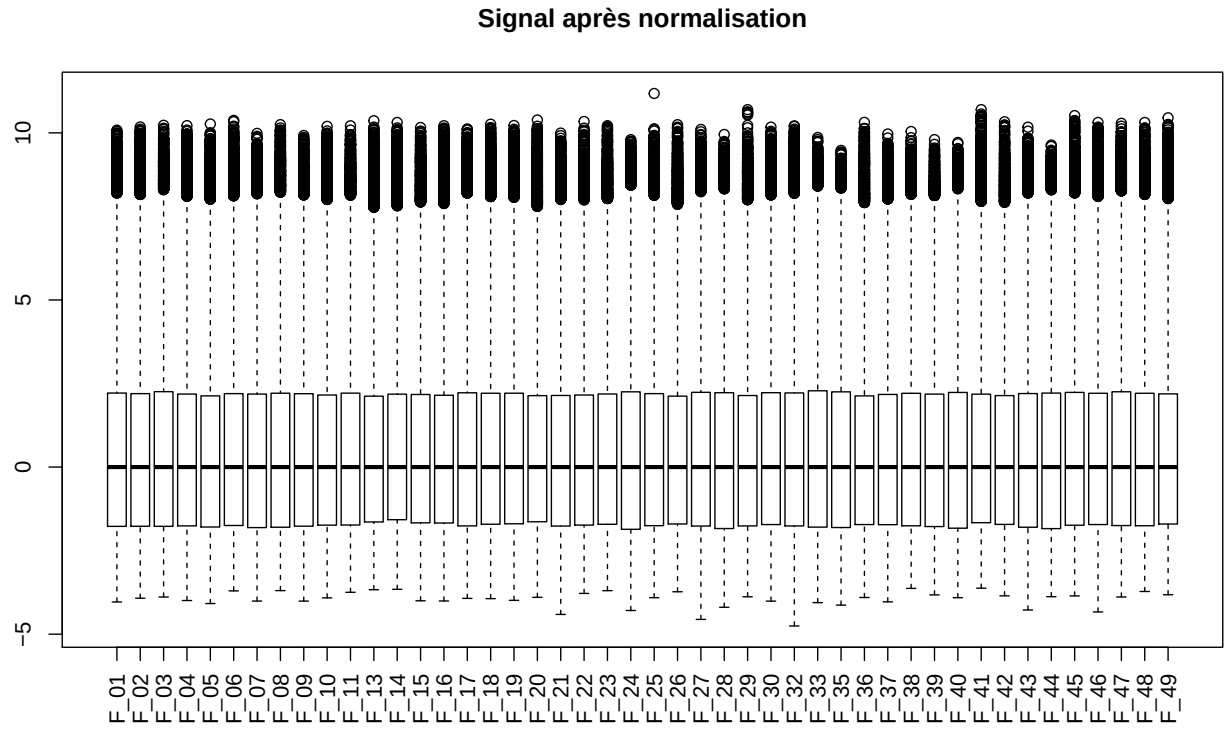
Le graphique montre des différences assez faibles entre les signaux médians par microarray. On décide néanmoins de gommer ces différences en soustrayant à chaque mesure d'expression la valeur médiane de la microarray :

```
medians = apply(G$E, 2, median)  # Medians for each microarray
G$E = sweep(G$E, MARGIN = 2, FUN = "-", STATS = medians)
# Subtract medians for each microarray (column)
apply(G$E, 2, median)
```

```
F_01 F_02 F_03 F_04 F_05 F_06 F_07 F_08 F_09 F_10 F_11 F_13 F_14 F_15 F_16
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
F_17 F_18 F_19 F_20 F_21 F_22 F_23 F_24 F_25 F_26 F_27 F_28 F_29 F_30 F_32
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
F_33 F_35 F_36 F_37 F_38 F_39 F_40 F_41 F_42 F_43 F_44 F_45 F_46 F_47 F_48
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
F_49
   0
```

L'impact de cette étape de la normalisation est visible sur le graphique comparant les répartitions des signaux d'expression par microarray :

```
boxplot(G$E, main = "Signal après normalisation", names = targets$ArrayName,
    las = 3)
```



4.4 Gestion des réplicats

Les sondes peuvent parfois être présentes en plusieurs exemplaires sur chaque microarray. Pour s'en rendre compte, on peut par exemple compter le nombre d'exemplaires de chaque sonde :

```
head(G$genes$ProbeName)  # First 6 probe names
```

```
[1] "seq_RIGG02544"                  "A_87_P009088"
[3] "A_87_P113528"                   "T001527_G001030_SCAMP4_420081"
[5] "A_87_P023872"                   "T041025_G024289"
```

```
names_counts = table(G$genes$ProbeName)  # Numbers of probes for each probe name
table(names_counts)  # 681 probe names are replicated
```

```
names_counts
    1     2
45379   681
```

On constate donc que 681 sondes sont présentes en deux exemplaires. Pour ces sondes, on propose de remplacer les deux mesures par leur moyenne, en utilisant la fonction *avereps* :

```
G$E = avereps(G$E, ID = G$genes$ProbeName)  # Replace replicated probes by average
dim(G$E)  # Dimensions of the normalized expression data
```

```
[1] 46060    46
```

5 Création des fichiers de données

5.1 Données d'expression

On commence par exporter les données d'expression dans le format sondes x microarrays :

```
write.table(G$E, "foie.txt")
```

5.2 Données du dispositif expérimental

De la même manière, on crée et on exporte des données sur les facteurs régime et génotype :

```
covariates = data.frame(Diet = diet, Genotype = genotype)
rownames(covariates) = targets$ArrayName
str(covariates)
```

```
'data.frame':   46 obs. of  2 variables:
 $ Diet    : Factor w/ 2 levels "BL","HL": 1 1 2 2 2 2 1 1 1 1 ...
 $ Genotype: Factor w/ 2 levels "G","M": 2 1 2 1 1 2 1 2 2 1 ...
```

```
write.table(covariates, "covariates.txt")
```

5.3 Données supplémentaires

Enfin, on exporte les données apportant des informations supplémentaires sur les poulets :

```
write.table(external_data, "external.txt")
```