2: Rapport d'analyse des données Affimetrix

Jerome Ambroise

2 mars 2017

Dans le document, ci dessous, nous allons analyser des données microarray générées avec une plateforme Agilent.

Les données proviennent de l'étude "MicroArray Quality Control (MAQC) project" dont un des papier vous est fourni. Dans ce document, nous analysons 5 échantillons A (A1, A2, A3, A4 et A5) et 5 échantillons B (B1, B2, B3, B4 et B5) et l'objectif est d'identifier les gènes qui sont modulés.

1: Chargement des packages et importation des données.

Nous commençons par charger le package Limma. Limma est un package bioconductor qui dispose d'un "user guide" très complet.

Le package 'gcrma' sera utilisé pour le prétraitement des données.

Le package 'hgu133plus2.db' sera utilisé pour annoter les sondes (ou "probes").

```
rm(list=ls())
library(limma)
```

```
## Warning: package 'limma' was built under R version 3.2.4
```

```
library(gcrma)
```

```
## Loading required package: affy

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following object is masked from 'package:limma':
##
##     plotMA

## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, do.call, duplicated, eval, evalq, Filter, Find, get,
##     grep, grepl, intersect, is.unsorted, lapply, lengths, Map,
##     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, Position, rank, rbind, Reduce, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unlist, unsplit

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
library(hgu133plus2.db)
```

```
## Loading required package: AnnotationDbi

## Loading required package: stats4

## Loading required package: IRanges

## Loading required package: S4Vectors

## Loading required package: org.Hs.eg.db

## Loading required package: DBI

## Warning: package 'DBI' was built under R version 3.2.5

##

##
```

Nous importons les données qui sont situées dans le répertoire 1-data/AGL. Il s'agit de 10 fichiers '.CEL'.

```
files <- list.files('1-data/AFFX/')
files <- paste0('1-data/AFFX/',files)
rawfiles <- ReadAffy(filenames=files)
```

2: Pré-traitement des données.

Les données de microarray sont généralement pré-traitées en 3 étapes:

1: la correction du background

2: la normalisation

3: la transformation (généralement log2)

Avec les données de affymetrix, ces trois étapes sont généralement réalisée avec une seule fonction. GCRMA est une méthode standard pour réaliser ces trois étapes.

```
expressionset <- gcrma(rawfiles)
```

```
## Warning: replacing previous import by 'utils::tail' when loading
## 'hgu133plus2cdf'
```

```
## Warning: replacing previous import by 'utils::head' when loading
## 'hgu133plus2cdf'
```

```
##
```

```
## Adjusting for optical effect.........Done.
## Computing affinities.Done.
## Adjusting for non-specific binding.........Done.
## Normalizing
## Calculating Expression
```

```
print(expressionset)
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 54675 features, 10 samples
##   element names: exprs
## protocolData
##   sampleNames: AFX_1_A1.CEL AFX_1_A2.CEL ... AFX_1_B5.CEL (10
##     total)
##   varLabels: ScanDate
##   varMetadata: labelDescription
## phenoData
##   sampleNames: AFX_1_A1.CEL AFX_1_A2.CEL ... AFX_1_B5.CEL (10
##     total)
##   varLabels: sample
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation: hgu133plus2
```

L'expressionset contient de nombreux "slots" dont la matrice d'expression. Il est intéressant de noter que pour le moment le 'slot' featureData est vide. Il s'agit du slot dans lequel doit être stocké la significaiton de chacune des sondes. Il est possible d'extraire cette matrice d'expression avec la fonction 'exprs'.

```
expressionmatrix <- exprs(expressionset)
dim(expressionmatrix)
```

```
## [1] 54675    10
```

```
expressionmatrix[1:10,1:10]
```

```
##           AFX_1_A1.CEL AFX_1_A2.CEL AFX_1_A3.CEL AFX_1_A4.CEL AFX_1_A5.CEL
## 1007_s_at     8.147886     8.250589     8.322309     8.155026     8.099427
## 1053_at       8.736785     8.835875     8.757296     8.819973     8.858737
## 117_at        3.868678     3.940337     3.716728     3.720480     3.779149
## 121_at        4.955711     4.866018     5.037398     5.377712     4.847162
## 1255_g_at     2.245347     2.245347     2.245347     2.245347     2.245347
## 1294_at       5.295410     5.498915     5.237261     5.175809     5.302495
## 1316_at       3.319834     3.331560     3.416571     3.408511     3.331045
## 1320_at       2.305168     2.311859     2.342989     2.305168     2.305168
## 1405_i_at     3.981935     3.815307     3.258160     3.627365     4.060098
## 1431_at       2.962549     2.987215     2.758345     2.931044     2.987215
##           AFX_1_B1.CEL AFX_1_B2.CEL AFX_1_B3.CEL AFX_1_B4.CEL AFX_1_B5.CEL
## 1007_s_at     9.279285     9.329932     9.298593     9.375831     9.277640
## 1053_at       6.120688     6.290722     5.877382     6.034217     6.102481
## 117_at        4.442848     4.294477     4.012228     4.069823     4.150946
## 121_at        4.722144     4.840412     4.783354     4.764420     4.834662
## 1255_g_at     2.263149     2.245347     2.248260     2.245347     2.245347
## 1294_at       3.574560     3.926068     3.926068     4.006317     3.637207
## 1316_at       4.448794     3.751927     4.322197     3.934547     4.069936
## 1320_at       2.306510     2.305168     2.305168     2.305168     2.392275
## 1405_i_at     2.295635     2.295635     2.295635     2.415916     2.386517
## 1431_at       5.349734     5.451178     5.156364     5.384927     5.500888
```

Nous allons determiner l'annotation de chacune des sondes à partir du package 'hgu133plus2.db'

```
probename <- rownames(expressionset)

columns(hgu133plus2.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"
##  [5] "ENSEMBLTRANS" "ENTREZID"     "ENZYME"       "EVIDENCE"
##  [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"         "ONTOLOGY"
## [17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
## [21] "PROBEID"      "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

```
keytypes(hgu133plus2.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"
##  [5] "ENSEMBLTRANS" "ENTREZID"     "ENZYME"       "EVIDENCE"
##  [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MAP"          "OMIM"         "ONTOLOGY"
## [17] "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
## [21] "PROBEID"      "PROSITE"      "REFSEQ"       "SYMBOL"
## [25] "UCSCKG"       "UNIGENE"      "UNIPROT"
```

```
annotation <- select(hgu133plus2.db,keytype='PROBEID', keys=probename, columns = c("SYMBOL"))
```

```
## 'select()' returned 1:many mapping between keys and columns
```

```
head(annotation,n=10)
```

```
##      PROBEID  SYMBOL
## 1  1007_s_at    DDR1
## 2  1007_s_at MIR4640
## 3    1053_at    RFC2
## 4     117_at   HSPA6
## 5     121_at    PAX8
## 6  1255_g_at  GUCA1A
## 7    1294_at    UBA7
## 8    1294_at MIR5193
## 9    1316_at    THRA
## 10   1320_at  PTPN21
```

```
dim(annotation)
```

```
## [1] 58616     2
```

```
idx <- match(probename,annotation$PROBEID)
annotation <- annotation[idx,]
dim(annotation)
```

```
## [1] 54675     2
```

```
all.equal(annotation$PROBEID,probename)
```

```
## [1] TRUE
```

```
fData(expressionset)
```

```
## data frame with 0 columns and 54675 rows
```

```
fData(expressionset) <- annotation
```

3: Création des modèles statistiques pour détecter les gènes modulés

Pour rappel, le but est de comparer les niveaux d'expression de chacun des gènes entre les 5 échantillons 'A' et les 5 échantillons 'B'. Nous commencçons par spécifier à R le design de notre expérience

```
mycondition <- c(rep('A',5),rep('B',5))
mydesign <- model.matrix(~mycondition )
print(mydesign)
```

```
##    (Intercept) myconditionB
## 1            1            0
## 2            1            0
## 3            1            0
## 4            1            0
## 5            1            0
## 6            1            1
## 7            1            1
## 8            1            1
## 9            1            1
## 10           1            1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$mycondition
## [1] "contr.treatment"
```

Une fois que le modèle est spécifié, il est possible de créer les modèles pour chacun des gènes et de calculer les statistiques de test et les p-valeurs associées à chacun des gestes. Cela se fait en deux étapes avec les fonctions 'lmFit' et 'eBayes' du package 'limma'.

```
fit <- lmFit(expressionset, design=mydesign)
efit <- eBayes(fit)
```

```
## Warning: Zero sample variances detected, have been offset
```

Une fois les modèles construits, on peut aller chercher les informations qui nous intéressent et les regrouper dans une data.frame. Il s'agit de 3 éléments

1: Le coefficient d'intérêt du modèle construit pour chacun des gènes. Ce coefficient correspond à la différence moyenne entre le niveau d'expression des échantillons A et B, dans l'échelle log2. Il s'agit donc d'un 'log2 fold-change'.

2: la p-valeur d'intérêt du modèle construit pour chacun des gènes.

3: l'annotation (çad le nom) de chacun des gènes

```
coefficient.affx <- fit$coefficients
head(coefficient.affx)
```

```
##           (Intercept) myconditionB
## 1007_s_at    8.195047  1.117208790
```

```
## 1053_at      8.801733 -2.716635279
## 117_at       3.805074  0.388990012
## 121_at       5.016800 -0.227801949
## 1255_g_at    2.245347  0.004142885
## 1294_at      5.301978 -1.487934001
```

```
coefficient.affx <- coefficient.affx[,2]
pvalue.affx <- efit$p.value
head(pvalue.affx)
```

```
##            (Intercept) myconditionB
## 1007_s_at 5.675008e-18 2.211156e-09
## 1053_at   1.692487e-16 6.654111e-11
## 117_at    1.482463e-12 1.800570e-03
## 121_at    3.326560e-13 4.356789e-02
## 1255_g_at 3.152984e-22 2.942450e-01
## 1294_at   2.783024e-13 2.180080e-07
```

```
pvalue.affx <- pvalue.affx[,2]
annotation <- efit$genes
annotation <- annotation[,'SYMBOL']
```

```
sum(is.na(annotation))
```

```
## [1] 9907
```

Nous allons supprimer les sondes qui ne sont pas annotées. Elles sont positionnées dans une localisation du génome qui n'est pas codante (çad en dehors d'un gène).

```
resultat.affx <- data.frame(annotation,coefficient.affx,pvalue.affx)
dim(resultat.affx)
```

```
## [1] 54675     3
```

```
resultat.affx <- data.frame(na.omit(resultat.affx))
dim(resultat.affx)
```

```
## [1] 44768     3
```

4: Classement des résultats et ajustement des p-valeurs

Nous allons maintenant classer les gènes selon les p-valeurs. Nous allons également calculer les p-valeurs ajustées selon la méthode de 'Benjamini-Hochberg'.

```
resultat.affx <- resultat.affx[sort.list(resultat.affx$pvalue.affx),]
adj.pvalue.affx <- p.adjust(resultat.affx$pvalue.affx,method='BH')
resultat.affx <- data.frame(resultat.affx,adj.pvalue.affx)
head(resultat.affx,n=10)
```

```
##              annotation coefficient.affx  pvalue.affx adj.pvalue.affx
## 206647_at           HBZ        -9.778036 1.552200e-20    3.507633e-16
## 209773_s_at        RRM2        -9.187942 1.567027e-20    3.507633e-16
## 211298_s_at         ALB       -11.415371 8.131817e-20    1.016181e-15
## 240433_x_at       CADM2         7.181440 1.209417e-19    1.016181e-15
## 209138_x_at       IGLC1        -9.037074 1.385960e-19    1.016181e-15
## 229039_at          SYN2         8.780131 1.465968e-19    1.016181e-15
## 219466_s_at       APOA2       -10.581004 1.588918e-19    1.016181e-15
## 211276_at        TCEAL2         9.023983 4.584497e-19    2.565485e-15
## 1556057_s_at    NEUROD1         8.967740 8.822418e-19    4.388467e-15
## 206427_s_at       MLANA        -8.710560 1.020810e-18    4.411199e-15
```

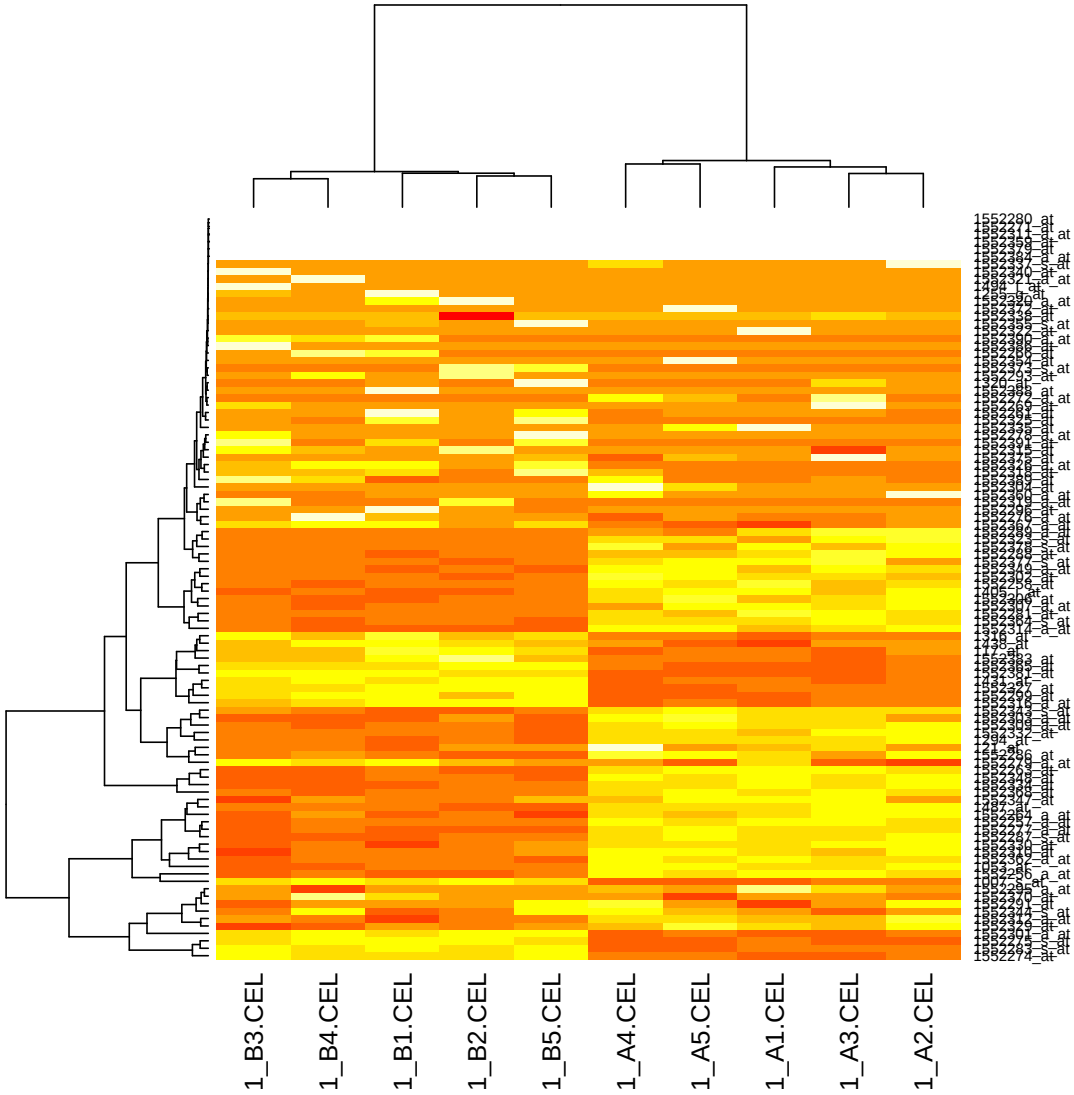
Finalement, nous écrivons tous les résutlats dans un fichier '.txt' dans le répertoire '2-result'.

```
write.table(resultat.affx,'2-result/1-fold-change-affx.txt',sep='\t',row.names=F)
```

4: Quelques graphiques

Nous allons représenter une heatmap des 100 premiers gènes (choix arbitraire des gènes).

```
expression <- expressionmatrix[1:100,]
heatmap(expression)
```



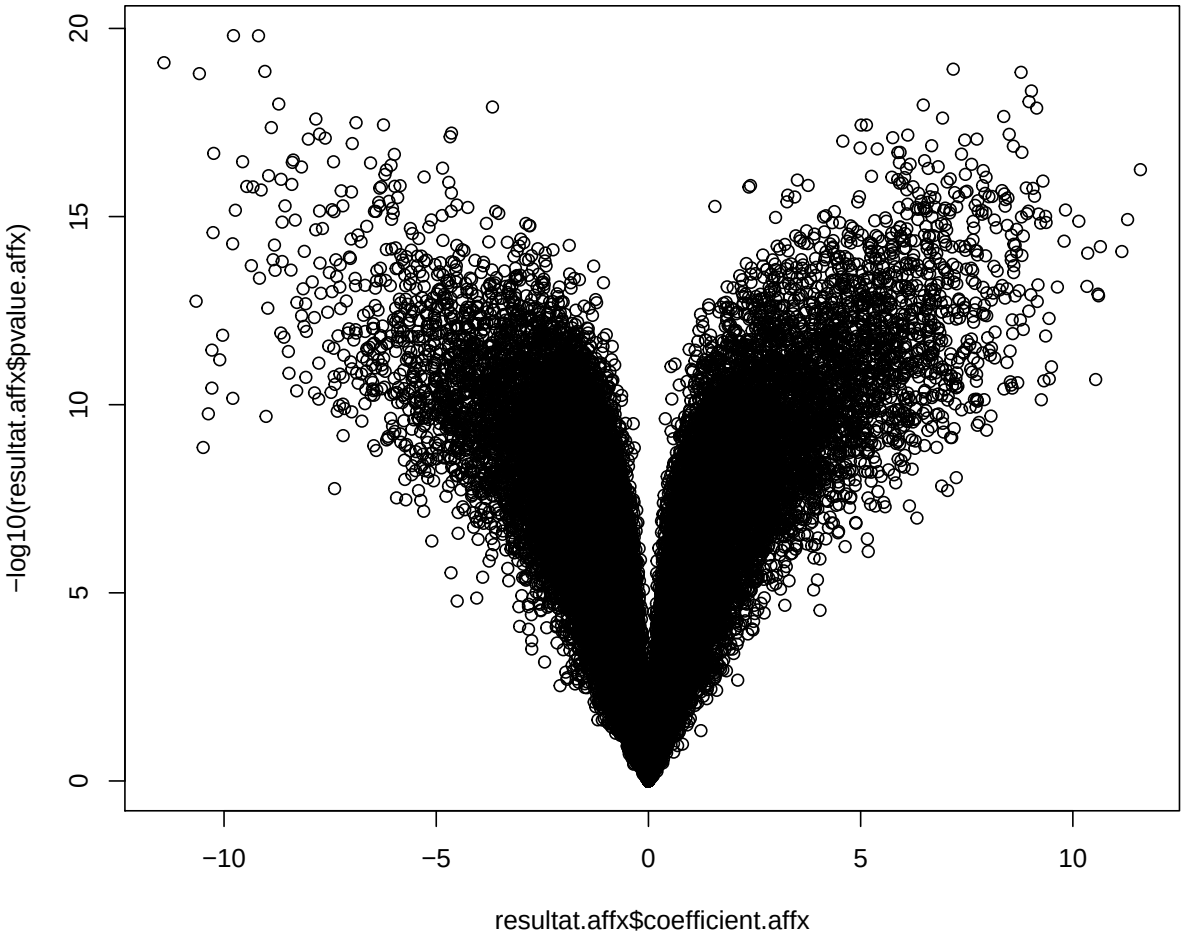
Nous pouvons afficher ce graphique dans une fichier pdf.

```
pdf('2-result/2a-affx-heatmap.pdf',width=8,height=8)
expression <- expressionmatrix[1:100,]
heatmap(expression)
dev.off()
```

```
## pdf
##   2
```

Nous allons représenter un volcano-plot.

```
plot(resultat.affx$coefficient.affx,-log10(resultat.affx$pvalue.affx))
```



Nous pouvons afficher ce graphique dans une fichier pdf.

```
pdf('2-result/2b-affx-volcano-plot.pdf',width=8,height=8)
plot(resultat.affx$coefficient.affx,-log10(resultat.affx$pvalue.affx))
dev.off()
```

```
## pdf
##   2
```