Scénario: souris, régimes, transcrits et acides gras

Résumé

Introduction à l'étude de données génomiques élémentaires (120 gènes) décrivant les expressions des gènes et des concentrations d'acides gras lors d'une expérience de nutrition de souris. Exploration des transcrits par analyse en composantes principales (ACP), classification et MDS ; recherche des gènes différentiellement exprimés (modèle linéaire et tests multiples) ; comparaison et intégration des données transcriptomiques et phénotypiques d'acides gras du foi par régression PLS et spase PLS.

1 Les données

1.1 Descriptif

Les données (Martin et al. 2007 [3]) proviennent d'une étude de nutrition chez la souris. Pour 40 souris, nous disposons :

- des données d'expression de 120 gènes recueillies sur membrane nylon avec marquage radioactif,
- des mesures de 21 acides gras hépatiques.

Par ailleurs, les 40 souris sont réparties selon deux facteurs :

- Génotype (2 modalités) : les souris sont soit de type sauvage (wt) soit génétiquement modifiées (PPAR) ; 20 souris dans chaque cas.
- Régime (5 modalités) : les 5 régimes alimentaires sont notés `ref`, `efad`, `dha`, `lin`, `tournesol` ; 4 souris de chaque génotype sont soumises à chaque régime alimentaire.

Les questions auxquelles nous nous proposons de répondre sont les suivantes :

1. Quels sont les gènes différentiellement exprimés ?
2. Peut-on effectuer des regroupements de gènes ?
3. Quelle est l'influence des facteurs "génotype" et "régime" sur l'expression des gènes ?
4. Existe-t-il des corrélations entre certains gènes (ou groupe de gènes) et certains acides gras (ou groupes d'acides gras) ?

1.2 Importation sous R

1.2.1 Prise en charge des données

Les données sont disponibles dans la librairie `mixOmics` pour servir d'exemple à différentes méthodes sous forme d'une liste de *data frames*.

```
# chargement de la librairie
library(mixOmics)
# déclaration des données
data(nutrimouse)
# concentrations des lipides
lipides=nutrimouse$lipid
# expressions des gènes
Exprs=nutrimouse$gene
# facteur régime
regime=nutrimouse$diet
# facteur génotype
genotype=nutrimouse$genotype
# nom des gènes
nomgene = colnames(Exprs)
```

Afin de s'assurer du bon déroulement de l'importation des données, il est utile de vérifier la dimension des objets ainsi créés : 40 lignes et 120 colonnes pour `Exprs` ; 40 lignes et 21 colonnes pour `lipides` (21 acides gras) :

```
dim(Exprs);dim(lipides)
summary(Exprs)
summary(lipides)
```

2 Description univariée

Une première étude descriptive élémentaire est indispensable pour se faire une première idée des données et de leur cohérence. Elle donnera aussi des

indications sur les éventuelles transformations à opérer, en général un logarithme.

Remarque préalable : il est important de bien contrôler les types d'objets qui sont manipulés. C'est la première source d'erreur dans l'utilisation des fonctions de R.

```
# Vérifie le type : data frame
class(Exprs)
# Le type devient matrix après transposition
class(t(Exprs))
```

Distributions des expressions pour chaque variable gène. Certains gènes s'expriment systématiquement plus que d'autres. Un centrage s'avère nécessaire.

```
# Commande sur un data frame
boxplot(Exprs)
# Distributions des expressions des gènes pour
# chaque ``variable'' souris après transposition:
boxplot(data.frame(t(Exprs)),horizontal=TRUE)
```

Distributions des expressions des gènes après centrage :

```
# scale() centre et réduit les colonnes par défaut
boxplot(data.frame(scale(Exprs,scale=FALSE)))
```

Distributions des expressions des gènes après centrage puis réduction qui s'avère inutile (pourquoi ?) :

```
# scale() perd également le type data frame
# Le centrage pour chaque souris est inutile :
boxplot(data.frame(scale(Exprs)))
boxplot(data.frame(scale(t(Exprs),scale=FALSE)),
   horizontal=TRUE)
```

Extraction des identificateurs des gènes et des souris :

```
# Deuxième élément d'un objet de type liste
genes=dimnames(Exprs)[[2]]
# Premier élément
namsour=dimnames(Exprs)[[1]]
```

3 Analyse en Composantes Principales

Différentes approches sont abordées pour explorer les données et obtenir des représentations lisibles malgré le nombre de gènes.

3.1 Différentes ACPs

Deux fonctions (`princomp, prcom`) de R calculent l'ACP classique. Seule `prcomp` accepte un nombre de colonnes supérieur à celui des lignes mais les objets (résultats) créés par ces deux fonctions n'ont pas tout à fait la même structure ! La fonction `plot` trace l'éboulis des valeurs propres tandis que `biplot` donne le graphique usuel des individus et des variables dans le premier plan principal par défaut. Une visite à l'aide en ligne (`help.start()`) fournit la description de tous les paramètres.

L'objectif est donc de construire la représentation la plus pertinente.

La première ACP avec les gènes en lignes et les souris en colonnes avec `princomp` dans laquelle les variables "souris" sont centrées n'est pas la plus adaptée. Elle fait apparaître un très fort effet taille sans grand intérêt.

```
# Il est nécessaire de transposer le tableau
souracp=princomp(t(Exprs))
# Eboulis des valeurs propres
plot(souracp)
biplot(souracp)
```

Fort effet taille qui confirme le fait que ce sont les gènes plutôt que les souris qui doivent être centrés mais pas nécessairement réduits.

Ce que réalise l'analyse suivante (gènes centrés en ligne, souris en colonnes) :

```
# La fonction scale pour centrer sans réduction
# (false) les gènes avant l'ACP
souracp=princomp(t(scale(Exprs, scale=FALSE)))
# Eboulis des valeurs propres
plot(souracp)
# Biplot gènes et souris
biplot(souracp)
```

Il est plus lisible de considérer les gènes/variables en colonnes. Cela nécessite d'utiliser l'autre fonction R pour calculer l'ACP car le nombre de colonnes est plus grand que le nombre de lignes.

```
# Il n'est plus nécessaire de transposer
souracp=prcomp(Exprs)
# L'élément x de la liste des résultats
# est la matrice des composantes principales
boxplot(data.frame(souracp$x))
# Eboulis des valeurs propres
plot(souracp)
# Construction du biplot
biplot(souracp)
```

La même chose avec une réduction.

```
souracp=prcomp(Exprs,scale=TRUE)
boxplot(data.frame(souracp$x))
biplot(souracp)
```

Une fois la "bonne" représentation construite, il est intéressant d'y projeter d'autres informations. Ainsi, en représentant chaque génotype par un symbole particulier :

```
# Calcul du code symbole de chaque génotype
pt=23+as.integer(genotype)
# Graphe en utilisant ces symboles
plot(souracp$x,type="p",pch=pt)
# Ajout des libellés des gènes en bleu
text(30*souracp$rotation,genes,col="blue")
```

Une couleur est en plus utilisée pour identifier chaque régime :

```
# Le vecteur col contient les codes de 1 à 4
plot(souracp$x,type="p",pch=pt,
   col=as.integer(regime))
# Ajout des libellés des gènes en bleu
text(30*souracp$rotation,genes,col="blue")
legend(9,1.5,c("dha","efad","lin","ref","tourn"),
   col =c(1,2,3,4,5),pch="+",title="Régime")
```

Si les génotypes se discriminent d'eux mêmes sur le premier plan, ce n'est pas le cas des régimes.

3.2 Représentation plus "lisible"

L'objectif est de restreindre le nombre de gènes représentés afin de se limiter aux plus représentatifs c'est-à-dire à ceux contribuant le plus à l'ACP. On revient sur l'ACP des gènes centrés considérés comme des individus dont on peut calculer les contributions à l'inertie du premier plan. Cette fois les variables souris sont également centrées introduisant une modification : un léger effet taille étant supprimé (certaines souris s'expriment en moyenne plus que d'autres), la discrimination des génotypes apparaît maintenant sur le premier axe.

```
souracp=prcomp(t(scale(Exprs,scale=FALSE)))
plot(souracp)
biplot(souracp)
# Extraction des coordonnées dans le premier plan
coord=souracp$x[,1:2]
# Elevées au carré
coord2=coord^2
# Somme divisée par la somme des valeurs propres
contrib=apply(coord2,1,sum)/sum(souracp$sdev[1:2])
# Distribution de ces contributions
hist(contrib)
# Vecteur identifiant les plus fortes contributions
selec=contrib>0.5
# Nombre de ces gènes sélectionnés
sum(selec)
```

Représentation des gènes de plus forte contribution sur le graphique identifiant génotypes et régimes :

```
genes[selec]
plot(souracp$rotation,type="p",pch=pt,
   col=as.integer(regime))
text(0.2*souracp$x[selec,],genes[selec],col="blue")
legend(9,1.5,c("dha","efad","lin","ref","tourn"),
   col =c(1,2,3,4,5),pch="+",title="Régime")
```

ou simplement sur le biplot :

```
biplot(souracp$x[selec,],souracp$rotation)
```

Ainsi, on tombe "naturellement" sur des gènes dont les expressions permettent de discriminer facilement les génotypes mais la discrimination des régimes est un problème plus difficile qui doit être abordé avec des outils spécifiques.

3.3 *sparse* PCA

La librairie `mixOmics` poursuit le même objectif de recherche de représentation lisible de l'ACP par une autre voie. Il s'agit de rechercher une version parcimonieuse des composantes principales en limitant le nombre de variables dans les combinaisons. Cela revient à ajouter une contrainte de type Lasso dans la décomposition en valeur singulière de la matrice.

L'ACP sans contrainte conduit évidemment aux mêmes résultats :

```
library(mixOmics)
res.spca=spca(Exprs,ncomp=2,scale.=FALSE)
# graphe des individus 2D
plotIndiv(res.spca,ind.names=FALSE, pch=pt ,
   cex=1 ,col=as.integer(regime))
legend(0.2,0.3,c("dha","efad","lin","ref","tour"),
   col =c(1,2,3,4,5),pch =16, title = "Régime")
# graphe des variables 2D
plotVar(res.spca, comp = c(1,2),var.label =
   TRUE, cex = 0.7)
```

Tandis que la contrainte ci-dessous sur le nombre de paramètres non nuls des composantes restreint l'analyse et les représentations aux 10 gènes les plus "importants" par composante.

```
res.spca=spca(Exprs,ncomp=2,scale.=FALSE,
   keepX=c(10,10))
# graphe des individus 2D
plotIndiv(res.spca,ind.names=FALSE, pch=pt ,
   cex=1 ,col=as.integer(regime))
legend(0.2,0.3,c("dha","efad","lin","ref","tour"),
   col =c(1,2,3,4,5),pch =16, title = "Régime")
# graphe des variables 2D
plotVar(res.spca, comp = c(1,2),var.label =
   TRUE, cex = 0.7)
```

Comparer cette sélection "automatique" ou "sous contrainte" avec celle précédente.

Spécificités des distributions de certains des gènes sélectionnés :

```
boxplot(CYP4A14  ~  genotype,data=Exprs)
boxplot(CYP3A11  ~  genotype,data=Exprs)
boxplot(THIOL  ~  genotype,data=Exprs)
boxplot(PMDCI  ~  genotype,data=Exprs)
boxplot(S14  ~  regime,data=Exprs)
```

Réfléchir à l'interpétation des plans factoriels en notant la bonne discrimination des génotypes alors que celle des régimes est difficile pour les souris sauvages voire très difficiles pour les souris mutantes.

4 Classification et représentation des classes

La recherche de classes ou groupes de gènes présentant des caractéristiques communes et donc probalement des fonctions communes est très fréquente. La question principale réside dans la façon de comparer deux gènes et plus précisément dans la façon de calculer une distance ou une dissimilarité entre eux. Connaissant les expressions des gènes, il est facile d'en déduire une distance euclidenne classique mais aussi des distances liées à leur corrélation.

En fonction du choix de la distance et une fois une classification obtenue, il est utile de représenter ces classes afin de s'assurer de leur plus ou moins bonnes séparation. Ce peut être fait par une ACP (distance euclidienne basée des vecteurs d'expression) ou par positionnement multidimensionnel (MDS) lorsque d'autres distances sont utilisées. La recherche de classes est approfondie dans le cas de la distance euclidienne tandis que les représentations sont comparées pour différents choix de distance.

4.1 Distances

Distance euclidienne

La première étape consiste à transposer la matrice de données relatives aux données d'expression. En effet, les individus que nous souhaitons classer sont les gènes et non pas les souris.

```
# Transposition de la matrice des données
tExprs=t(scale(Exprs))
# calcul des distances par défaut euclidiennes
dist.e=dist(tExprs)
# Longueur du vecteur des distances
length(dist.e)
```

La commande précédente renvoie un vecteur de taille 7140 qui contient les éléments situés dans la partie triangulaire supérieure de la matrice de distance inter-gènes ($7140 = (120 \times 119)/2$).

Remarque : Il est inutile de stocker les $120 \times 120 = 14400$ éléments de la matrice pleine car la matrice est symétrique ($d(x, y) = d(y, x)$) et les éléments diagonaux sont nuls ($d(x, x) = 0$). Pour une matrice carrée de taille n, le nombre total d'éléments est n^2 auquel on retranche les n termes diagonaux et on divise le tout par 2 pour ne conserver que la partie tringulaire supérieure : $(n^2 - n)/2 = n(n - 1)/2$.

Distances basées sur la corrélation

La corrélation entre deux gènes est un indice de similarité à partir duquel il est facile de construire deux distances :

```
# Calcul de la matrice des corrélations
rS = cor(Exprs)
# Calcul de la distance basée sur la corrélation
dist.r=as.dist(1-rS)
# Calcul de la distance entre gènes basée sur
# la corrélation au carrée
dist.r2=as.dist(sqrt(1-rS**2))
# Extraction des noms ou codes des gènes
dN=dimnames(Exprs)[[2]]
```

4.2 Classification ascendante hiérarchique

La mise en œuvre d'une méthode de classification vise à rechercher une partition des individus en classes sans a priori sur le nombre de classes. La CAH dépend de deux options ou deux critères de distance : l'un entre individus, l'autre entre groupe d'individus.

Dendrogramme

Le critère de Ward ou le critère moyen `average` est préféré pour définir la distance entre groupes. Le recours à tout autre critère est à réserver à des cas très particuliers ou à des questions spécifiques.

```
# Appel à la fonction hclust avec critère de Ward
hc.e=hclust(dist.e,method="ward")
# Tracé du dendrogramme
plot(hc.e)
```

La fonction `plot` est codée pour réagir à un objet de classe `hclust` et tracer un dendrogramme.

Nombre de classes

Un choix très important est celui du nombre de classes. Il est facilité par le graphique ci-dessous des hauteurs respectives des nœuds du dendrogramme.

```
plot(hc.e$height[118:100],
   xlab="Nb de classes",ylab="Hauteur")
```

L'élément `height` de l'objet `hc.e` contient les hauteurs des nœuds du dendrogramme classées dans l'ordre croissant. En précisant `height[118:100]`, nous prenons les 19 dernières valeurs dans l'ordre décroissant.

Couper l'arbre et extraire 4 classes de gènes :

```
classif.4G=cutree(hc.e,k=4)
# répartition par classe
sort(classif.4G)
```

Le résultat de `cutree` est un vecteur de taille le nombre de gènes contenant les valeurs de 1 à 4 indiquant l'appartenance à l'un des 4 groupes. Les éléments

sont identifiés par les noms donnés aux gènes dans le tableau initial.

On peut également afficher séparément les noms des gènes qui appartiennent à chacun des groupes :

```
# Extraction des noms des éléments du tableau
# classif.4G qui sont égaux à 1
names(classif.4G[classif.4G==1])
names(classif.4G[classif.4G==2])
names(classif.4G[classif.4G==3])
names(classif.4G[classif.4G==4])
```

Représentation par MDS

L'interprétation de chacun des groupes est alors l'affaire du praticien qui peut s'aider d'une représentation des classes dans un graphique obtenu par ACP ou MDS. L'ACP est naturellement utilisée pour représenter des classes d'individus tandis que le MDS est plus adapté à représenter des classes de variables (les gènes) en utilisant la même matrice de distances.

```
mds.e=cmdscale(dist.e, k=2)
col=cutree(hc.e,k=4)
plot(mds.e, type="n", xlab="cp1", ylab="cp2")
text(mds.e,dN,col=col)
```

Démarche identique avec les autres distances :

```
## avec la corrélation
mds.r=cmdscale(dist.r, k=2)
hc.r=hclust(dist.r,method="ward")
col=cutree(hc.r,k=4)
# Ouvrir une autre fenêtre graphique
x11()
plot(mds.r, type="n", xlab="cp1", ylab="cp2")
text(mds.r,dN,col=col)
## avec la corrélation carrée
mds.r2=cmdscale(dist.r2, k=2)
hc.r2=hclust(dist.r2,method="ward")
col=cutree(hc.r2,k=4)
# Ouvrir une autre fenêtre graphique
x11()
plot(mds.r2, type="n", xlab="cp1", ylab="cp2")
text(mds.r2,dN,col=col)
```

Comparer les graphiques obtenus. De grandes similarités entre les deux premiers tandis que le choix de la corrélation au carré, qui rapproche des gènes corrélés positivement ou négativement, introduit des modifications.

4.3 Double classification

Il est aussi possible de construire un graphique souvent réalisé sur ce type de données et issu d'une double classification ascendante hiérarchique à la fois sur les gènes et sur les échantillons biologiques ; c'est le rôle de la fonction `heatmap`.

```
# Spécification de la méthode de Ward
lf = function(d) hclust(d, method="ward")
# Appel à heatmap avec un objet de type matrix
heatmap(as.matrix(Exprs),hclustfun=lf)
# Appel identique après centrage des gènes
heatmap(scale(as.matrix(Exprs),scale=FALSE),
   hclustfun=lf)
```

4.4 Classification par réallocation dynamique

La fonction `kmeans` exécute un algorithme de réallocation dynamique. L'option à fixer est cette fois le nombre de classes suggéré par exemple par la classification ascendante hiérarchique précédente.

```
km.genes=kmeans(tExprs,centers=4)
```

Pour illustrer les changements intervenus par rapport à la classification hiérarchique, on peut construire la table de contingence croisant les résultats des deux classifications.

```
table(classif.4G,km.genes$cluster,dnn=c("tree",
   "kmeans"))
```

Les algorithmes de type kmeans dépendent de leur initialisation ; pour optimiser, on peut fixer les centres initiaux de l'algorithme aux centres des classes

issues de la classification hiérarchique.

```
# Initialisation matrice des centres des classes
mat.init.km.genes=matrix(nrow=4,ncol=40)
for (i in 1 :4)
# Affectation à la ligne i de la matrice
# des centres des classes...
   mat.init.km.genes[i,]=
# ...de la moyenne des expressions des
# gènes de la classe i
   apply(tExprs[classif.4G==i,],2,mean)
km.genes.init=kmeans(tExprs,
   centers=mat.init.km.genes)
```

On peut à nouveau comparer les classifications dans une table de contingence.

```
table(classif.4G,km.genes.init$cluster,
   dnn=c("tree","kmeans"))
```

4.5 Réallocation autour des "medoïds"

PAM (Partionning Around Medoïds) est un autre algorithme de réallocation dynamique disponible dans la librairie spécialisée `cluster`.

```
library(cluster)
pamv=pam(dist.r2,4)
# Calcul du mds pour la distance considérée
mds.r2= cmdscale(dist.r2, k=2)
# Code couleur défini par les numéros des classes
col=pamv$clustering
plot(mds.r2, type="n", xlab="cp1", ylab="cp2")
text(mds.r2,dN,col=col)
```

Cet algorithme propose encore d'autres regroupements. Seule une "expertise biologique" permettrait de déterminer laquelle est la plus pertinente.

Une fonction (`clusplot`) avec pas mal d'options permet d'obtenir directement ces graphiques.

```
clusplot(dist.r2,pamv$clustering,diss=TRUE,
   labels=2,color=TRUE,col.txt=pamv$clustering)
```

5 Modèle linéaire et tests multiples

Ces outils sont systématiquement utilisés pour l'analyse des données transcriptomiques.

5.1 Influence du génotype sur le gène AOX

On peut effectuer un test de Student à l'aide de la fonction `t.test` ou une analyse de variance (ANOVA) sur ce gène en utilisant la fonction `lm()`.

```
t.test(Exprs[,"AOX"] ~ genotype,var.equal=T)
anov.AOX = lm(Exprs[,"AOX"] ~ genotype)
summary(anov.AOX)
```

Comparer les deux analyses et vérifier que ce sont les mêmes résultats (même statistique et même p-value).

```
names(summary(anov.AOX))
summary(anov.AOX)$coefficients
# P-value associé au l'effet génotype
summary(anov.AOX)$
coefficients["genotypeppar","Pr(>|t|)"]
# autre paramétrisation sans le terme constant
anov.AOX.1 = lm(Exprs[,"AOX"] ~ genotype-1)
summary(anov.AOX.1)
```

En considérant ce modèle (`anov.AOX` ou `anov.AOX.1`) comme le modèle de référence (modèle complet), le modèle réduit est celui ne comportant qu'une seule moyenne pour expliquer l'effet du gène.

```
anov1=lm(Exprs[,"AOX"] ~ 1)
summary(anov1)
# test de Fisher comparaison de modèles emboités
anova(anov.AOX,anov1)
# Verification des hypothèses du modèle
plot(anov.AOX)
```

Vérifier à nouveau que l'on obtient encore la même p-value qu'avec le test de Student. Remarquez que les statistiques sont différentes, mais les p-valeurs

sont les mêmes : les tests sont identiques. Pour les statistiques, on retrouve que $F = t^2$ puisque le facteur `genotype` a 2 niveaux.

5.2 Recherche des gènes régulés par le génotype

Effectuer autant d'analyses de variance que de gènes en appliquant la fonction `apply()`.

```
pval = apply(Exprs,2,function(x)
summary(lm(x ~ genotype))$
    coefficients["genotypeppar","Pr(>|t|)"])
# autant de p-valeurs que de gènes testés
# Histogramme des p-valeurs
hist(pval)
```

Mise en œuvre des corrections pour les tests multiples en utilisant la librairie `multtest`. La difficulté avec la détection de gènes réside dans le nombre de comparaisons qui nécessite d'ajuster le seuil de chaque comparaison deux à deux afin de contrôler le risque global de l'ensemble des comparaisons. Il existe de nombreuses méthodes permettant d'ajuster les p-valeurs individuelles ; une des plus connues est due à Bonferroni, elle consiste à diviser le risque de première espèce par le nombre de populations dont on veut comparer la moyenne (nombre de tests). Nous allons voir comment mettre en œuvre un certain nombre de ces méthodes dont celle de Benjamini Hochberg (BH) contrôlant le FDR (false discovery rate), la plus utilisée pour des données d'expression. Il est à noter cependant, qu'aucune d'entre elles n'est réellement satisfaisante dans notre contexte car les hypothèses théoriques nécessaires pour ces tests ne sont en général pas vérifiées par les données d'expression. Il convient donc d'interpréter les résultats de ces analyses avec prudence et de les confronter avec d'autres types d'analyse. La mise en œuvre les méthodes de comparaisons multiples de moyennes utilise le package `multtest` intégré à la librairie `Bioconductor` et qui nécessite une installation spécifique.

```
# chargement de la librairie avec
# Bioconductor déjà installé
library(multtest)
# liste des méthodes de correction
help(mt.rawp2adjp)
```

```
# choix de deux méthodes
procs = c("Bonferroni", "BH")
# calculer les p-valeurs ajustées
tmp = mt.rawp2adjp(pval, procs)
# Rangement des gènes dans l'ordre initial
adjp = tmp$adjp[order(tmp$index), ]
res.adjp = round(adjp, 2) # Arrondi
# Affectation des noms des gènes aux lignes
dimnames(res.adjp)[[1]] =  names(Exprs)
```

La fonction `mt.plot()` du package `multtest` fournit des représentations graphiques permettant de comparer les différentes méthodes. Par exemple, l'option `plottype = "pvsr"` permet de représenter les p-valeurs ajustées (éventuellement aussi les valeurs brutes) rangées dans l'ordre croissant.

```
mt.plot(cbind(pval,adjp[, procs]), statt,
   plottype = "pvsr",proc = c("pval",procs),
   lty = c(1,2,3),col = c("black", "blue", "red"),
   leg=c(50,0.8),lwd=2)
```

Dans les lignes précédentes, les 2 dernières règlent certains paramètres de présentation du graphique :

- `lty` : type de ligne (trait plein ou pointillé)
- `col` : couleur des lignes
- `leg` : position du coin haut-gauche de la légende
- `lwd` : épaisseur des traits

Pour faciliter la lecture du graphique, on peut y ajouter des lignes horizontales pour chaque niveau de risque.

```
abline(h=c(0.01,0.05,0.1),lty=c(1,2,3))
```

Liste des gènes qui sont différentiellement exprimés pour un FDR à 5%

```
listgenes= names(Exprs)[res.adjp[,"BH"]<0.05]
listgenes
```

5.3 Influence du régime sur le gène AOX

Une analyse de variance pour voir si le gène AOX est différentiellement exprimé selon les régimes ; attention, comme la variable régime contient 5 niveaux, un test de Fisher est calculé.

```
anov.AOX.R = lm(Exprs[,"AOX"] ~ regime)
summary(anov.AOX.R)
names(summary(anov.AOX.R))
names(summary(anov.AOX.R))
# statistique de test et degrés de liberté
summary(anov.AOX.R)$fstatistic
```

Considérant ce modèle (anov.AOX.R) comme le modèle de référence (modèle complet), le modèle réduit est celui ne comportant qu'une seule moyenne pour expliquer l'effet du gène.

```
anov1=lm(Exprs[,"AOX"] ~ 1)
summary(anov1)
# comparaison de modèles emboîtés
anova(anov.AOX.R,anov1)
```

5.4 Recherche des gènes régulés par le régime

Une analyse de variance est calculée sur l'ensemble des gènes pour rechercher les gènes différentiellement exprimés selon le régime. Chaque modèle complet est comparé au modèle réduit.

```
pval = rep(NA,120)
for (i in 1:120) {
   lm.complet = lm(Exprs[,i] ~ regime)
         lm.reduit = lm(Exprs[,i] ~ 1)
         anova(lm.complet,lm.reduit)
         pval[i] = anova(lm.complet,lm.reduit)[2,
         "Pr(>F)"]
         }
# histogramme des p-valeurs
hist(pval)
```

Correction pour les tests multiples comme précédemment.

```
procs = c("Bonferroni", "BH")
tmp = mt.rawp2adjp(pval, procs)
adjp = tmp$adjp[order(tmp$index), ]
res.adjp = round(adjp, 2)

dimnames(res.adjp)[[1]] =  names(Exprs)
mt.plot(cbind(pval,adjp[, procs]),
   statt,plottype = "pvsr",
   proc = c("pval",procs),lty = c(1,2,3),
   col = c("black", "blue", "red"),
   leg=c(50,0.8),lwd=2)

abline(h=c(0.01,0.05,0.1),lty=c(1,2,3))
```

Liste des gènes qui sont différentiellement exprimés selon le régime avec un FDR à 5%.

```
listgenes = names(Exprs)[res.adjp[,"BH"]<0.05]
listgenes
```

5.5 Influence des deux facteurs sur le gène AOX

Analyse de variance à 2 facteurs (régime, génotype) croisés pour savoir si le gène AOX est différentiellement exprimé selon le génotype, le régime ou les 2 à la fois.

```
# modèle avec interaction
anov1.AOX =lm(Exprs[,"AOX"] ~ genotype +
   regime + genotype:regime)
summary(anov1.AOX)
# comparaison avec le modèle réduit
anova(anov1.AOX,lm(Exprs[,"AOX"] ~ 1))
# modèle sans interaction
anov2.AOX =lm(Exprs[,"AOX"] ~ genotype+regime)
summary(anov2.AOX)
# l'interaction est-elle significative ?
anova(anov1.AOX,anov2.AOX)
```

Même opération que précédemment mais sur l'ensemble des gènes.

```
pval = rep(NA,120)
for (i in 1:120) {
   lm.complet = lm(Exprs[,i]~genotype+regime+
      genotype:regime)
   lm.reduit = lm(Exprs[,i]~1)
   anova(lm.complet,lm.reduit)
   pval[i] = anova(lm.complet,lm.reduit)[2,
   "Pr(>F)"] }
hist(pval)
procs = c("Bonferroni", "BH")
tmp = mt.rawp2adjp(pval, procs)
adjp = tmp$adjp[order(tmp$index), ]
res.adjp = round(adjp, 2)

dimnames(res.adjp)[[1]] =  names(Exprs)
mt.plot(cbind(pval,adjp[, procs]), statt,
   plottype = "pvsr",
   proc = c("pval",procs),lty = c(1,2,3),
   col = c("black", "blue", "red"),
   leg=c(50,0.8),lwd=2)

abline(h=c(0.01,0.05,0.1),lty=c(1,2,3))
```

Liste des gènes qui sont différentiellement exprimés soit selon le génotype, soit selon le régime ou soit les 2 à la fois.

```
listgenes= names(Exprs)[res.adjp[,"BH"]<0.05]
```

Rechercher :

1. les gènes différentiellement exprimés selon à la fois le régime et le génotype et avec une interaction significative (`list1`)
2. la liste des gènes différentiellement exprimés selon à la fois le régime et le génotype mais dont l'interaction est non significative (`list2`)
3. la liste des gènes différentiellement exprimés selon uniquement le génotype (`list3`)
4. la liste des gènes différentiellement exprimés selon uniquement le régime (`list4`)

```
pval.interaction=rep(NA,120)
pval.regime=rep(NA,120)
pval.genotype=rep(NA,120)
for (gene in listgenes) {
   i=match(gene,names(Exprs))
   lm.complet = lm(Exprs[,i] ~ genotype+ regime+
      genotype:regime)
   lm.additif = lm(Exprs[,i] ~ genotype+ regime)
   lm.genotype=lm(Exprs[,i]  ~ genotype)
   lm.regime =lm(Exprs[,i]   ~ regime)
   pval.interaction[i]=anova(lm.complet,
      lm.additif)[2, "Pr(>F)"]
   pval.regime[i]=anova(lm.additif,
      lm.regime)[2, "Pr(>F)"]
   pval.genotype[i]=anova(lm.additif,
      lm.genotype)[2, "Pr(>F)"]}
```

Liste des gènes qui sont différentiellement exprimés avec un FDR à 5%.

```
# interaction significative
list1 = nomgene[pval.interaction < 0.05]
list1 =  list1[!is.na(list1)];list1
# interaction non significative mais un effet
# du genotype et du regime.
list2 = nomgene[(pval.interaction > 0.05) &
   (pval.regime <0.05) & (pval.genotype <0.05)]
list2 =  list2[!is.na(list2)];list2
# genes uniquement régulés par le seul génotype
list3 = nomgene[(pval.interaction > 0.05) &
   (pval.regime >0.05) & (pval.genotype <0.05)]
list3 =  list3[!is.na(list3)];list3
# genes uniquement régulés par le seul régime
list4 = nomgene[(pval.interaction > 0.05) &
   (pval.regime <0.05) & (pval.genotype >0.05)]
list4 =  list4[!is.na(list4)];list4
# dernière vérification
pval.total = data.frame(nomgene=nomgene,
```

```
    fdr=res.adjp[,"BH"],
    interaction=round(pval.interaction,5),
    regime=round(pval.regime,5),
    genotype=round(pval.genotype,5))
pval.total
```

6 Recherche de gènes discriminants

Les hypothèses (indépendances) nécessaires au contrôle des tests multiples ne sont pas vérifiées. D'autres approches globales (*wrapper methods*) consistent à rechercher des gènes discriminants. Ces techniques seraient également utilisées pour le recherche de biomarqueurs à fin prévisionnelle mais ce n'est pas l'objectif ici.

L'analyse factorielle discriminante (AFD) pourrait être utilisée pour cet objectif mais le nombre de variables (gènes) au regard de la taille de l'échantillon l'en empêche ; la matrice de variance 120×120 n'est pas inversible car au plus de rang 39. D'autres méthodes sont abordées. La première est une adaptation de la régression PLS pour calculer l'analyse discriminante, la deuxième fait appel à un modèle d'agrégation d'arbres.

6.1 Analyse discriminante par régression PLS

Cette étude élémentaire se limite ici aux souris sauvages dont les régimes se différencient plus facilement. La régression PLS explique un ensemble de variables Y, les indicatrices des classes dans le cas de l'analyse discriminante PLS (PLS-DA), par des combinaisons linéaires des variables X, les expressions des gènes.

```
X=Exprs[1:20,]
Y=regime[1:20]
res.plsda=plsda(X,Y,ncomp=3)
plotIndiv(res.plsda, comp = 1:2, ind.names = FALSE,
   col = regime, cex = 1, pch = 16)
plot3dIndiv(res.plsda, cex=2,col = regime)
```

Comme pour l'ACP, la librairie `mixOmics` propose une version parcimonieuse de la PLS-DA afin d'obtenir des représentations plus faciles à interpréter.

```
res.plsda=splsda(X,Y,ncomp=3,keepX=c(5,5,5))
plotIndiv(res.plsda, comp = 1:2, ind.names = FALSE,
   col = regime, cex = 1, pch = 16)
plotVar(res.plsda, comp=1:2, var.label=TRUE,cex=.8)

plot3dIndiv(res.plsda, cex=.8,col = regime)
plot3dVar(res.plsda,var.label=TRUE,cex=.8)
```

6.2 Random forest

Le modèle d'agrégation d'arbres cherche à prévoir le régime à partir des expressions. Les modèles obtenus sont de qualité médiocres mais les indicateurs d'importance calculés par cet algorithme fournissent des indications utiles sur les gènes considérés les plus discriminants. La discrimination est recherchée dans les régimes, conditionnellement au génotype.

```
library(randomForest)
# séparation des données par génotype
Exprs1=data.frame(Exprs[1:20,],
   regime=regime[1:20])
Exprs2=data.frame(Exprs[21:40,],
   regime=regime[21:40])
# recherche des modèles
rf.res1 = randomForest(regime ~ .,data=Exprs1,
   ntree=1000, importance=TRUE)
rf.res2 = randomForest(regime ~ .,data=Exprs2,
   ntree=1000, importance=TRUE)
# discrimination plus facile des souris sauvages
print(rf.res1)
print(rf.res2)
# Importances des gènes
imp1=rf.res1$importance[,6]
impt1=sort(imp1,decreasing=TRUE)
imp2=rf.res2$importance[,6]
impt2=sort(imp2,decreasing=TRUE)
# Gènes importants discriminant les souris sauvages
```

```
plot(impt1,type="n")
text(impt1,names(impt1),cex=.8)
# Gènes importants discriminants les souris mutantes
plot(impt2,type="n")
text(impt2,names(impt2),cex=.8)
# comparaison par génotype
plot(imp1,imp2,type="n")
text(imp1,imp2,names(imp1),cex=.8)
# Un gène parmi d'autres
boxplot(CYP2c29~regime[1:20],data=Exprs[1:20,])
boxplot(CYP2c29~regime[21:40],data=Exprs[1:20,])
```

7 Intégration des données

Un des aspects de cette étude est l'intégration des deux ensembles de données avec toujours la contrainte forte que le nombre de variables est très supérieur à celui des observations.

Représentation graphique de la structure de corrélation des variables.

```
X = lipides
Y = Exprs
imgCor(X, Y, X.names = FALSE, Y.names = FALSE)
```

7.1 Analyse canonique pénalisée

Pour des raisons évidentes de dimension, les matrices de projection indispensables en analyse canonique ne sont pas définies aussi Gonzalez et al. (2009)[1] proposent d'utiliser une version régularisée (r-cca ou *ridge canonical correlation analysis*) en faisant jouer un rôle symétrique aux deux ensembles de données.

Cette approche nécessite d'optimiser la pénalisation en maximisant la corrélation canonique estimée par validation croisée.

```
grid1 = seq(0, 0.2, length = 5)
grid2 = seq(0.0001, 0.2, length = 5)
# validation croisée M-folds
estim.Mfold <- estim.regul(X, Y, grid1 = grid1,
   grid2 = grid2, validation = "Mfold")
image(estim.Mfold)
# validation croisée leave-one-out
estim.loo = estim.regul(X, Y, grid1=grid1,
   grid2=grid2,validation="loo",plt = FALSE)
image(estim.loo)
```

Même si le choix n'est pas simple, fixer les paramètres et lancer l'analyse `r-cca` et représenter les graphiques dans les deux premières dimensions.

```
# calcul
rcca.res = rcc(X, Y, ncomp = 3, lambda1 = 0.06,
   lambda2 = 0.01)
rcca.res
# représentation des souris avec la
# couleur en fonction du génotype
col.nutri = as.numeric(genotype)
col.nutri[col.nutri == 1] = "blue"
col.nutri[col.nutri == 2] = "red"
## et le nom en fonction du régime
plotIndiv(rcca.res, comp = 1:2, ind.names =
   regime, col = col.nutri)
# représentation des variables
plotVar(rcca.res, comp = 1:2, cutoff = 0.5,
   X.label=TRUE,Y.label=TRUE,cex=c(0.8, 0.8))
```

Représentation en 3 dimensions après avoir sélectionné une liste de gènes bien connus du biologiste.

```
# triangles pour acides gras et
# sphères pour le sgènes
pch  <- c("t", "s")
col.lip = rep("black", 21)
col.gen = rep("darkgray", 120)
names(col.gen) = rcca.res$names$Y
genes1 = c("CAR1")
genes2 = c("GSTpi2", "CYP3A11", "CYP2c29")
genes3 = c("S14","ACC2","cHMGCoAS","HMGCoAred")
```

```
genes4 = c("ACBP","AOX","BIEN","CPT2","CYP4A10",
   "HPNCL","L.FABP","PECI","PMDCI","THIOL",
   "mHMGCoAS")
genes5 = c("FAS")
genes6 = c("PLTP")

col.gen[genes1] = "darkviolet"
col.gen[genes2] = "darkgreen"
col.gen[genes3] = "red"
col.gen[genes4] = "orange"
col.gen[genes5] = "cyan"
col.gen[genes6] = "magenta"
col = list(col.lip, col.gen)

plot3dVar(rcca.res,cutoff=0.6,col=col,pch=pch,
    cex = c(1.5, 1.5), label.axes.box = "axes")
# Interactive = TRUE limite la représentation des
# variables au delà d'un seuil de corrélation
```

La librairie `mixOmics` offre d'autres types de graphiques comme des réseaux ou une *heat map* pour illustrer les liens entre les variables.

```
network(rcca.res, comp = 1:3, interactive = TRUE,
   threshold = 0.55)
cim(rcca.res, comp = 1:3, xlab = "genes",
   ylab = "lipids", margins = c(5,6))
```

L'interprétation détaillée de cette analyse est décrite dans Gonzalez et al. (2009)[1].

7.2 Régession PLS

Dans cet exemple, il paraît plus légitime de tenter d'expliquer globalement les variables phénotypiques (concentration en acides gras) en fonction des expressions des gènes. Cet objectif est typiquement celui de la régression PLS de type 2 ou plusieurs variables Y sont linéairement expliquées par un ensemble d'autres variables X.

De façon classique le nombre de composantes peut être optimisé par validation croisée.

```
X=Exprs
Y=lipides
pt=23+as.integer(genotype)
pls.res=pls(X,Y,ncomp=3,mode="regression")
error.pls = valid(pls.res,validation = "loo",
   criterion = "MSEP")
error.pls$MSEP
plotIndiv(pls.res, comp = 1:2, ind.names = FALSE,
   col = regime, cex = 1,pch=pt)
plotVar(pls.res, comp = 1:2, Y.label = TRUE,
   X.label = TRUE, cex = c(0.5, 0.8))
```

Déterminer un nombre optimal de dimensions est difficile, il semble clairement plus grand que 3 mais l'objectif exploratoire va restreindre le choix à trois ainsi que le nombre de variables avec une version *sparse*) de la PLS (Lê cao et al. 2008 [2])dans le but d'obtenir des représentations interprétables.

```
# estimation de la régression sparse pls
# sous contrainte
spls.res=spls(X,Y,ncomp=3,mode="regression",
   keepX = rep(10,3), keepY = rep(5,3))
# représentation des individus
plotIndiv(spls.res, comp = 1:2, ind.names = FALSE,
   col = regime, cex = 1,pch=pt)
# représentation des variables
plotVar(spls.res, comp = 1:2, Y.label = TRUE,
   X.label = TRUE, cex = c(0.8, 0.8))
# Représentations 3d
pt3d=rep(c("c", "s"),c(20,20))
plot3dIndiv(spls.res, cex=.8,col = regime,pch=pt3d)
plot3dVar(spls.res, Y.label = TRUE, X.label = TRUE,
   cex = c(0.5, 0.8))
```

Comme précédemment avec l'analyse canonique, le même type de représentation plus fouillé peut être obtenu.

```
network(spls.res, comp = 1:3, interactive = TRUE,
```

```
   threshold = 0.55)
cim(spls.res, comp = 1:3, xlab = "genes",
   ylab = "lipids", margins = c(5,6))
```

Reste à savoir quelle représentation graphique, quelle méthode est la plus pertinente pour le biologiste !

Conclusion

Cette étude de cas permet de mettre en exergue le fait qu'il n'existe pas une meilleure méthode permettant de traiter des données génomiques. A chaque question : exploration et validation des données, recherche de classes de gènes ou d'échantillons biologiques, recherche de gènes différentiellement exprimés, de gènes discriminants ou biomarqueurs, intégration de données... correspond souvent plusieurs méthodes apportant des représentations et résultats complémentaires. Seule une collaboration étroite entre d'une part un statisticien connaissant bien les méthodes utilisées, leurs options, leurs limites et, d'autre part, un biologiste donnant du sens aux résultats trouvés, permet d'élaborer conjointement une stratégie pertinente d'analyse de ces données dans toute leur complexité.

Un autre scénario propose des exemples plus élaborés d'utilisation des versions *sparse* de la PLS en version régression ou analyse discriminante.

Références

[1] I. González, S. Dejean, P.G.P. Martin, O. Gonçalvez, P. Besse et A. Baccini, *Highlighting Relationships Between Heteregeneous Biological Data Through Graphical Displays Based On Regularized Canonical Correlation Analysis*, Journal of Biological Systems **17** (2009), n^o 2, 173–199.

[2] K. A. Lê Cao, D. Rossouw, C. Robert-Granié et P. Besse, *A sparse PLS for variable selection when integrating Omics data*, Statistical Applications in Genetics and Molecular Biology **7** (2008), n^o 35.

[3] P.G. Martin, H. Guillou, F. Lasserre, S. Dejean, A. Lan, J.M. Pascussi, M. Sancristobal, P. Legrand, P. Besse et T. Pineau, *Novel aspects of PPARalpha-mediated regulation of lipid and xenobiotic metabolism revealed through a nutrigenomic study*, Hepatology **45** (2007), 767–777.