TP R sur l'ACP

Emmanuel Rachelson and Matthieu Vignes

25 septembre & 2 octobre 2013, SupAero - ISAE

1 Introduction

Ce doc est conçu comme un tutoriel pour faciliter l'analyse d'un jeu de données dites transcriptomiques. Les lignes de commande sont précisées mais aucune raison de se limiter à celles-ci. Il est cependant essentiel de bien comprendre leur syntaxe évidemment mais surtout la nature des sorties ! Ne pas hésiter à consulter l'aide des fonctions (`help("function")`) et en particulier les options/paramètres disponibles.

2 Les données

2.1 Descriptif

Données fournies par T. Pineau et P. Martin du laboratoire *Toxalim* de l'INRA Toulouse. Elles proviennent d'une étude de nutrition chez la souris. Pour $n = 40$ souris, nous disposons de :

- données d'expression de $p = 120$ gènes,
- (mesures de $q = 21$ acides gras hépatiques).

Par ailleurs, les 40 souris sont réparties selon 2 facteurs:

- génotypes (2 modalités) sauvage (wt) ou génétiquement modifiées ($PPAR$): 20 souris dans chaque cas,
- régimes (5 modalités) notés *ref*, *efad*, *dha*, *lin* et *tournesol*; 4 souris de chaque génotype sont soumis à chaque régime alimentaire.

Les questions auxquelles on se propose de répondre peuvent être:

- quels sont les gènes *différentiellement* exprimés ?
- Quelle est l'influence des facteurs 'génotypes' et 'régime' sur l'expression des gènes ?
- Peut-on effectuer des regroupements de gènes ?
- Existe-t-il des corrélations entre certains gènes et acides gras ? Entre des groupes de ceux-ci ?

2.2 Importation sous R

Les données sont disponibles dans deux fichiers texte (un pour les gènes `genes.csv`, l'autre pour les acides gras `lipides.csv`): la première ligne contenant les titres (nom des gènes ou noms des acides gras) et la première colonne l'identifiant des individus souris.

Les souris sont en lignes, les variables phénotypiques et les gènes en colonnes. Le fichier contient sur la première ligne (par défaut, `Header=TRUE`) le nom de chaque variable ou gène mais pas pour la première colonne qui est donc automatiquement considérée comme celle des identifiants des lignes : le numéro de chaque souris.

```
>lipides <- read.table("lipides.csv",sep=",") # Lecture du fichier
des covariables dans un objet de type data frame
>Exprs <- read.table("genes.csv",sep=",") # Lecture des expressions
des gènes
```

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 23 colonnes pour `lipides` (21 acides gras + 1 colonne régime + 1 colonne génotype) :

```
>dim(Exprs) ;dim(lipides) # Dimensions des objets de type data frame
ou matrice
```

ou encore d'utiliser la commande `summary` qui fournit un premier aperçu quantitatif des données.

```
>summary(Exprs)
>summary(lipides)
```

Quelques manipulations élémentaires :

```
>genotype=as.factor(lipides[,23]) # Extraction de la colonne 23
décrivant les génotypes avec le type "factor"
>regime=as.factor(lipides[,22]) # Extraction de la colonne 22 décrivant
les régimes
>lipides=lipides[,1 :21] # Extraction des variables lipidiques
>Exprs=Exprs[,-3] # Suppression du troisième gène dont la mesure
a été contaminée
```

3 Statistiques élémentaires

3.1 Description univarée

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.

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

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

```
>boxplot(Exprs) # Application directe de la commande sur un data
frame
```

Distributions des expressions des gènes pour chaque "variable" souris après transposition :

```
>boxplot(data.frame(t(Exprs)),horizontal=TRUE) # Re-création d'un
data frame après transposition qui crée un type matrix
```

Distributions des expressions des gènes après centrage :

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

Distributions des expressions des gènes après centrage puis réduction qui s'avère inutile :

```
>boxplot(data.frame(scale(Exprs))) # La fonction scale perd également
le type data frame
```

Le centrage pour chaque souris est inutile :

```
>boxplot(data.frame(scale(t(Exprs),scale=FALSE)),horizontal=TRUE)
```

Extraction des identificateurs des gènes et des souris :

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

3.2 Description bivariée

Quelques calculs de covariance :

```
>cov(Exprs[,1],Exprs[,2]) # entre les 2 premiers gènes
>cov(lipides$C22.6n.3,lipides$C18.1n.9) # entre les acides C22.6n.3
et C18.1n.9
```

Coefficients de corrélation pour les mêmes variables

```
>cor(Exprs[,1],Exprs[,2]) # coefficient de corrélation linéaire
(de Pearson)
>cor(Exprs[,1],Exprs[,2],method="spearman") # coefficient de corrélation
de Spearman (sur les rangs)
>cor(lipides$C22.6n.3,lipides$C18.1n.9)
```

Représentations graphiques en nuage de points:

```
>plot(Exprs[,1],Exprs[,2])
>plot(Exprs[,1],Exprs[,2], xlab=colnames(Exprs)[1],ylab=colnames(Exprs)[2])
# avec les noms des axes
>plot(lipides$C22.6n.3,lipides$C18.1n.9)
```

Le calcul des corrélations peut être généralisé sur l'ensemble des données

```
>correl.lip = cor(lipides) # la fonction cor() calcule les corrélations
entre les colonnes d'une matrice prises deux à deux
>correl.lip # La lecture globale de la matrice de corrélation n'est
pas évidente
>round(correl.lip,2) # même en diminuant le nombre de chiffres après
la virgule
```

Une représentation sous forme d'image est souvent plus lisible pour un aperçu global des corrélations entre les différentes variables:

```
>image(correl.lip)
```

L'inconvénient de cette représentation réside dans le fait que l'origine de l'image (élément [1,1] de la matrice) est placé en bas à gauche

```
>image(t(correl.lip[dim(lipides)[2] :1,]),axes=F)
```

Une manipulation de la matrice permet d'y remédier:

```
>mtext(colnames(lipides),side=3,adj=0, at=seq(0,1,l=21),cex=0.5,las=3)
>mtext(colnames(lipides)[dim(lipides)[2] :1],side=2,adj=0,
+at=seq(0,1,l=21),cex=0.5,las=1,line=2)
```

To do: représenter la légende en jouant avec la fonction `legend()`.

Le positionnement des étiquettes des variables dans les marges de la représentation graphique facilite l'interprétation.

4 Analyse en composantes principales

Cette section donne déjà un résumé intéressant des gènes et de leurs expressions. 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 (surprise ?). La fonction `plot` trace l'éboulis des valeurs propres tandis que `biplot` donne le graphique "usuel" des individus et des variables dans, par défaut, le premier plan principal. Une visite à l'aide en ligne fournit la description de tous les paramètres.

L'objectif est donc de trouver 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.

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

```
>souracp=princomp(t(scale(Exprs, scale=FALSE))) # La fonction scale
pour centrer sans réduction (false) les gènes avant l'ACP
```

```
>plot(souracp) # éboulis des valeurs propres
>biplot(souracp) # biplot gènes et souris
```

Il est en fait plus "légitime" et aussi 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. La gestion de l'objet créé contenant les résultats demande plus de doigté avec en plus un tracé des diagrammes en boîtes des composantes principales.

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

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 :

```
>pt=23+as.integer(genotype) # calcul du code symbole de chaque génotype
>plot(souracp$x,type="p",pch=pt,cex=2) # Graphe en utilisant ces
symboles et en fixant leur taille
>text(30*souracp$rotation,genes,col="blue") # ajout des libellés
des gènes en bleu
```

Une couleur est en plus utilisée pour identifier chaque régime :

```
>plot(souracp$x,type="p",pch=pt,cex=2,col=as.integer(regime)) #
le vecteur 'col' contient les codes des couleurs de 1 à 4
>text(30*souracp$rotation,genes,col="blue") # ajout des libellés
des gènes en bleu
```

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

Le nombre important de gènes rend toujours difficile la lecture des graphiques alors qu'une grande partie de ceux-ci n'interviennent que très peu dans la définition des axes. En faire la sélection par leur contribution sur le premier plan (ou le sous-espace de dimension q retenu) améliore les choses. Ceci nécessite quelques calculs intermédiaires qui montrent la flexibilité de R par rapport un logiciel fermé guidé par menus.

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)
>coord=souracp$x[,1 :2] # extraction des coordonnées dans le premier
plan
>coord2=coord^2 # élevées au carré
>contrib=apply(coord2,1,sum)/sum(souracp$sdev[1 :2]) # somme divisée
par la somme des valeurs propres
>hist(contrib) # distribution de ces contributions
>selec=contrib>0.5 # vecteur logique identifiant les plus fortes
contributions
>sum(selec) # Nombre de ces gènes
```

Les 13 gènes de plus forte contribution :

```
>genes[selec]
```

sont positionnés sur le graphique identifiant génotypes et régimes :

```
>plot(souracp$rotation,type="p",pch=pt,cex=2,col=as.integer(regime))
>text(0.2*souracp$x[selec,],genes[selec],col="blue")
```

ou simplement sur le biplot :

```
>biplot(souracp$x[selec,],souracp$rotation) # on spécifie les individus-gènes
à représenter dans le biplot
```

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.

```
>boxplot(CYP4A14 ~ genotype,data=Exprs) # diagrammes boîtes par
modalité de la variable génotype
>boxplot(CYP3A11 ~ genotype,data=Exprs)
>boxplot(THIOL ~ genotype,data=Exprs)
>boxplot(PMDCI ~ genotype,data=Exprs)
```

5 Positionnement multidimensionnel (MDS)

Cette technique factorielle particulière est utilisée pour construire d'autres représentations des gènes précédemment sélectionnés. Pour ce faire, on calcule différentes distances entre ces gènes. Deux, parmi d'autres, on été sélectionnées sur la base de la corrélation linéaire : un moins la corrélation et racine de un moins la corrélation au carré.

5.1 Calcul des distances

```
>rS = cor(Exprs) # calcul de la matrice des corrélations
>dS=1-rS # calcul de la distance basée sur la corrélation
>dS2=sqrt(1-rS^2) # calcul de la distance entre gènes basée sur
leur corrélation au carrée
>dN=dimnames(Exprs)[[2]] # extraction des noms ou codes des gènes
```

5.2 MDS avec la corrélation linéaire

```
>mdsour= cmdscale(dS, k=2) # calcul du MDS
>plot(mdsour, type="n", xlab="", ylab="", main="1-correlation des
genes") # tracé des axes à l'échelle mais sans les points (type="n")
>text(mdsour,dN) # positionnement des libellés sur le graphe précédent
>abline(v=0,h=0) # tracé des axes à l'origine
```

5.3 MDS avec le carré de la corrélation

Les mêmes calculs sont réalisés avec la distance alternative.

```
>mdsour= cmdscale(dS2, k=2)
>plot(mdsour, type="n", xlab="", ylab="", main="MDS : 1- correlation
carree des genes")
>text(mdsour,dN)
>abline(v=0,h=0)
```

6 Classification

6.1 Classification ascendante hiérarchique

La mise en oeuvre d'une méthode de classification vise à rechercher une partition des individus en classes sans *a priori* sur le nombre de classes. Toute méthode de ce type est basée sur deux critères de distance : l'un entre individus, l'autre entre groupe d'individus. 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.

```
>tExprs <- t(Exprs) # transposition de la matrice des données
```

Maintenant que nous disposons des données au format désiré, nous pouvons calculer la matrice des distances entre gènes :

```
>dist.tExprs <- dist(tExprs) # calcul des distances euclidiennes
inter-individus
>length(dist.tExprs) # renvoie la longueur du vecteur des distances
```

Par défaut, la fonction `dist` utilise la distance euclidienne. La commande précédente renvoie un vecteur de taille 7021 qui contient les éléments situés dans la partie triangulaire supérieure de la matrice de distance inter-gènes ($7021 = (119 \times 118)/2$).

Remarque : pas besoin de stocker les $119 \times 119 = 14161$ éléments de la matrice pleine par symétrie ($d(x; y) = d(y; x)$) et les éléments diagonaux sont nuls ($d(x; x) = 0$). Pour une matrice carrée de taille n, le nombre total d'éléments à conserver est: $(n^2 - n)/2 = n(n - 1)/2$. Pour effectuer la classification hiérarchique ascendante, on préfèrera le critère de Ward ou le critère moyen `average`. Le recours à tout autre critère est plutôt à reserver à certains situations particulières ou à des questions spécifiques.

```
>hc.tExprs <- hclust(dist.tExprs,method="ward") # appel à la fonction
hclust avec le critère de Ward
```

Pour produire le graphique, il suffit alors d'utiliser la fonction `plot` :

```
>plot(hc.tExprs) # tracé du dendrogramme
```

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

Un utilisateur avancé de R trouvera naturel de regrouper certaines étapes pour accéder directement à la représentation graphique :

```
>plot(hclust(dist(t(Exprs)),method="ward"))
```

La lecture d'une telle ligne de commande s'effectue de "l'intérieur vers l'extérieur".

Un choix très important est celui du nombre de classes. Il est facilité par le graphique ci-dessous.

```
>plot(hc.tExprs$height[118 :100], xlab="Nb de classes",ylab="Hauteur")
# tracé des hauteurs des noeuds du dendrogramme
```

L'élément `height` de l'objet `hc.tExprs` contient les hauteurs des noeuds du dendrogramme classées dans l'ordre croissant. En précisant `height[118:100]`, nous prenons les 9 dernières valeurs dans l'ordre décroissant.

Pour couper le graphique en 4 classes, on peut utiliser la fonction `cutree`:

```
>classif.4G <- cutree(hc.tExprs,k=4) # élagage de l'arbre en 4 classes
```

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. Pour accéder à ces noms:

```
>names(classif.4G)
```

Pour trier les gènes en fonction de leur groupe, il suffit d'ordonner le vecteur issu de la fonction `cutree`:

```
>sort(classif.4G) # rangement par ordre croissant du numéro de groupe
```

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

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

Idée: utiliser une boucle `for(){...}`

Pour faciliter l'exploitation des résultats, on peut rediriger la sortie vers un fichier et compléter un peu les sorties :

```
   >sink('groupes.txt') # redirige l'affichage vers le fichier 'groupe.txt'
>for (i in 1 :4){
+eff <- length(classif.4G[classif.4G==i]) # affecte à la variable
eff le nombre de valeurs égale à i dans le vecteur classif.4G
+print(paste("*** Groupe ",i," :  ", eff," gènes ***")) # affiche
(fonction print()) le collage (fonction paste()) de différentes
```

```
informations :  soit des chaînes de caractères en l'état (" "),
soit la valeur des variables (i et eff)
+print(names(classif.4G[classif.4G==i])) # liste le nom des gènes
du groupe i}
>sink() # renvoie l'affichage vers l'écran
```

L'interprétation de chacun des groupes est alors l'affaire du praticien.

6.2 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. Pour cela, on fait appel à la fonction `heatmap` :

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

6.3 Agrégation autour des centres mobiles

Nous mettons en oeuvre un algorithme d'agrégation autour de centres mobiles par la fonction `kmeans`. Ici la seule information fournie à l'algorithme est le nombre de classes, suggéré par exemple par la classification ascendante hiérarchique.

```
>km.genes=kmeans(tExprs,centers=4) # appel à la fonction kmeans
avec 4 classes
```

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"))
```

Pour optimiser la classification par nuées dynamiques, on peut fixer les centres initiaux de l'algorithme aux centres des classes issues de la classification hiérarchique.

```
>mat.init.km.genes=matrix(nrow=4,ncol=40) # initialisation vide
de la matrice des centres des classes
>for(i in 1:4){
+mat.init.km.genes[i,]= # affectation à la ligne i de la matrice
des centres des classes...
+apply(tExprs[classif.4G==i,],2,mean) # ...de la moyenne des expressions
des gènes de la classe i
>km.genes.init=kmeans(tExprs,centers=mat.init.km.genes) # appel
à la fonction kmeans avec initialisation des centres}
```

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

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

6.4 Représentation d'une classification

Cette section croise classification et MDS. Techniquement, elle décrit quelques commandes conduisant à la représentation d'une partition des gènes dans les coordonnées d'une ACP ou de celles du MDS. Elle permet de se faire rapidement une idée sur la plus ou moins bonne séparation des classes.

On considère une classification par réallocation dynamique ; ici un PAM (*Partionning Around Medoids*), disponible dans la librairie spécialisée `cluster`. D'autres exemples seront repris par la suite.

```
>library(cluster)
>pamv=pam(dS2,6) # classification par réallocation dynamique
>mds= cmdscale(dS2, k=2) # calcul du MDS pour la distance considérée
>col=pamv$clustering # code couleur défini par les numéros des classes
>plot(mds, type="n", xlab="cp1", ylab="cp2")
>text(mds,dN,col=col)
```

Une fonction (`clusplot`, avec pas mal d'options) permet de faire ça directement. Elle est également disponible dans la librairie `cluster` :

```
>clusplot(dS2,pamv$clustering,diss=TRUE,labels=2,color=TRUE,
+col.txt=pamv$clustering)
```

NB: on pourra revenir sur ces données lors de l'étude du modèle linéaire...