

A la découverte de R

Mots-clés : Importation de données, objets, fonction, automatisation des calculs, Rcmdr.

David Causeur (david.causeur@agrocampus-ouest.fr)

Table des matières

1	Préambule	3
2	Premiers pas dans R	3
2.1	Ouverture d'une session R	3
2.2	Principes de base des commandes	4
2.3	Liste des objets	5
2.4	Environnement de programmation	5
2.5	Chargement de bibliothèques externes	6
3	Importation de données	7
3.1	Lancement de Rcmdr	8
3.2	Importation	9
3.3	Coup d'oeil rapide sur les données	10
4	Premières analyses statistiques	11
4.1	Transformation de variables	11
4.2	Boîtes de dispersions	11
4.3	Un exemple d'opération sur des données : le centrage	14
5	Automatisation d'un traitement	16
5.1	Principes généraux de la création d'une fonction	17
5.2	Un exemple d'analyse statistique : le test de comparaison de deux moyennes	18
5.3	Automatisation d'une analyse statistique	19

1 Préambule

R est un logiciel libre de la famille GNU (pour plus d'informations sur le projet GNU, voir <http://www.gnu.org>). Malgré l'intention du projet GNU de développer un système d'exploitation alternatif aux solutions commerciales existantes, ce logiciel dédié au traitement statistique de données est disponible sous les principaux systèmes d'exploitation, dont Windows.

Parmi les propriétés souvent mises en avant par ses utilisateurs, R est disponible gratuitement, par téléchargement à partir de plusieurs sites miroirs de la page officielle <http://www.r-project.org/>. C'est pourtant une autre de ses propriétés, directement héritées des principes fondateurs des logiciels libres, qui vaut à R d'être devenu aussi populaire dans de nombreux secteurs d'activités : l'accès libre aux codes sources du logiciel et, par conséquent, l'évolution du logiciel par la collaboration active et non concertée de tous ses utilisateurs. Une des ambitions des pilotes du projet R (the R core team) est de le faire évoluer de manière plus structurée en encourageant la constitution de communautés d'utilisateurs sur des thématiques ciblées. Ainsi est né par exemple le projet `bioconductor` (voir <http://www.bioconductor.org/>) qui fédère les contributions des utilisateurs dans le domaine du traitement des données génomiques et post-génomiques.

2 Premiers pas dans R

De manière générale, le traitement statistique dans R se traduit par une suite d'opérations sur des objets, à savoir les données elle-même ou les résultats d'une précédente opération. La mise en œuvre de ces opérations passe donc par l'écriture de commandes, ce qui suppose la connaissance d'une syntaxe spécifique. Le principe de base des commandes est aussi intuitif que possible :

$$> \text{sortie} = \text{opération} (\text{entrée}) \quad (1)$$

où `sortie` et `entrée` sont les noms d'objets et `opération` est le nom d'une fonction. La commande ci-dessus affecte dans `sortie` le résultat de l'application de `opération` sur `entrée`. Les plus attentifs parmi les lecteurs s'étonnent sans doute du signe `>` au début de la ligne de commande. Ce signe est une invite, présent en début de chaque ligne comme une invitation à travailler, et n'est pas un élément actif de la commande.

Remarque : dans la commande (1), le signe `=` est parfois remplacé par son équivalent `<-`.

2.1 Ouverture d'une session R

Au démarrage de l'application R, le logiciel ouvre une session de travail par le message suivant : [Sauvegarde de la session précédente restaurée]. En effet, le logiciel indique ainsi qu'il charge les objets créés lors de la session précédente et sauvegardés dans un

fichier dit `image`. A la fermeture de la session actuelle, R demandera donc à l'utilisateur s'il souhaite sauvegarder sa session.

La seule fenêtre ouverte au démarrage est dite *fenêtre de commandes* : c'est le cœur de la session de travail, le lieu de toutes les opérations. De nombreuses opérations y sont disponibles, des plus simples opérateurs arithmétiques jusqu'aux traitements statistiques les plus courants.

Ceux qui veulent abandonner ici la lecture de ce document et s'aventurer seuls dans l'apprentissage de R peuvent le faire à partir de la commande suivante, qui provoque l'ouverture d'une page web d'aide en ligne :

```
> help.start() # S'il te plaît, R, aide-moi à démarrer !
```

La présence de parenthèses dans la commande précédente peut sembler étrange. Il faut simplement se rappeler du format général (1) d'une commande de R : l'opération invoquée ici est `help.start`, qui s'applique sans nécessité d'objet *entrée* et qui ne produit pas d'objet *sortie*. La commande (1) est donc réduite ici à sa plus simple expression.

Le symbole `#` indique le début d'une zone de commentaires, non interprétés par R.

2.2 Principes de base des commandes

Examinons la séquence de commandes suivante, en rouge :

```
> x = 2 # affectation de la valeur 2 dans l'objet appelé x
> y = sqrt(x) # affectation dans y du résultat de la fonction sqrt appliquée à x
> y # lecture de l'objet y
[1] 1.414214
```

Comme le lecteur averti aura compris, la fonction `sqrt` (pour square root) calcule la racine carrée. Lorsque l'on veut obtenir de l'aide à propos d'une commande, il suffit d'invoquer la fonction `help` :

```
> help(sqrt) # S'il te plaît, R, dis moi tout sur la fonction sqrt
```

La commande précédente provoque l'ouverture d'une fenêtre contenant un texte d'explication sur la fonction `sqrt`.

Dans l'affichage du résultat, l'invite `>` est remplacée par une valeur entière entre crochets. Cette valeur prend tout son sens lorsque le résultat de la fonction est une collection de valeurs : elle indique le rang dans la collection de la première valeur de chaque ligne.

Illustration :

```
> x = 1:20 # affectation dans x des valeurs entières entre 1 et 20
> y = sqrt(x) # affectation dans y du résultat de la fonction sqrt appliquée à x
> y # lecture de y
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[9] 3.000000 3.162278 3.316625 3.464102 3.605551 3.741657 3.872983 4.000000
[17] 4.123106 4.242641 4.358899 4.472136
```

Comme le suggère la séquence de commandes précédente, il est très facile d'automatiser un calcul dans R car les fonctions `y` sont en général conçues pour opérer sur des collections de valeurs.

2.3 Liste des objets

La commande `objects` permet d'interroger R pour connaître la liste des objets de la session :

```
> objects() # dis moi, R, quels sont les noms des objets de la session?
[1] "x" "y"
```

Comme la plupart des fonctions de R, l'appel de `objects` peut être mieux précisé à l'aide d'arguments. Par exemple, l'argument `pattern` sert à restreindre la liste des objets à ceux dont le nom contient une chaîne de caractères particulière :

```
> objects(pattern="x") # dis moi, R, quels sont les noms des objets de la session
# contenant la lettre x?
[1] "x"
```

2.4 Environnement de programmation

À l'exception de quelques opérations ponctuelles, l'utilisation directe de la fenêtre de commandes comme espace d'édition des programmes est déconseillée. En effet, dans cette fenêtre se mêlent commandes et résultats sans fonctionnalités simples pour gérer l'édition des programmes et leur sauvegarde. Une meilleure organisation du travail est possible grâce à une fenêtre dédiée à l'édition des commandes, la fenêtre `script`. Pour accéder à un tel environnement, choisir la rubrique `Fichier` dans le menu déroulant, puis la sous-rubrique `Nouveau script`.

Lorsque la fenêtre `script` est active, quelques nouvelles icônes apparaissent sous le menu déroulant dont une, appelée icône `Run`, permet de basculer tout ou partie des commandes dans la fenêtre de commandes pour exécution (en passant lentement le curseur de la souris sur ces icônes, une bulle donne des informations sur leurs fonctions). Par exemple, supposons que la fenêtre de `script` contienne les lignes suivantes :

```
x = 2      # affectation de la valeur 2 dans l'objet appelé x
y = sqrt(x) # affectation dans y du résultat de la fonction sqrt appliquée à x
```

Positionner le curseur de la fenêtre `script` sur une des lignes et cliquer sur l'icône `Run` exécute la ligne dans la fenêtre de commandes. Sélectionner l'ensemble des deux lignes et cliquer sur l'icône `Run` exécute d'un seul coup les deux commandes.

Il est fortement recommandé de sauvegarder régulièrement l'ensemble des commandes de la fenêtre `script` par la rubrique `Fichier` du menu déroulant et la sous-rubrique `Sauver . . .`. Les fichiers `script` sont suffixés `.R`.

Une autre manière de lancer l'exécution d'un fichier de commandes `R`, très utile si le fichier est volumineux, est d'utiliser la fonction `source` :

```
> source("C:/chezmoi/ilfaitsoleil.R") # lance les commandes de ilfaitsoleil.R
                                     # qui se trouve dans le répertoire C:/chezmoi.
```

Attention, lorsque des commandes `R` impliquent des adresses de répertoire, comme ici, les traditionnels `backslash` `\` font place à des `slash` `/`. Par ailleurs, si l'on est amené à utiliser à de nombreuses reprises l'adresse d'un même répertoire contenant par exemple les fichiers de données, les fichiers de résultats ou les fichiers de commandes, on peut le définir comme répertoire de travail pour l'ensemble de la session :

```
> setwd("C:/chezmoi/") # définit le répertoire C:/chezmoi
                       # comme répertoire de travail
> source("ilfaitsoleil.R") # lance les commandes du fichier ilfaitsoleil.R
```

2.5 Chargement de bibliothèques externes

Il est toujours possible d'enrichir la session de nouvelles fonctions contenues dans des bibliothèques externes (on utilise dans la suite le terme anglais `package`). La liste de certains `packages`, parmi les plus utilisés, est accessible par le menu déroulant, rubrique `Packages`, sous-rubrique `Charger le package . . .`. La boîte de dialogues `Select one` qui s'ouvre alors donne accès au chargement de `packages` qui ont déjà été installés en complément de la version originelle de `R`. Une description des `packages` installés est disponible par le lien hypertexte `packages` de la page web d'aide en ligne (voir section 2.1).

Si le `package` souhaité n'apparaît pas dans la liste proposée par la boîte de dialogues `Select one`, c'est que celui-ci n'est pas installé. Pour installer le `package` désiré, deux solutions sont possibles :

- si l'ordinateur sur lequel on travaille est connecté à internet, il suffit alors de télécharger le `package` à partir de la page web du projet `R`. Pour cela, choisir la rubrique `Packages` dans le menu déroulant, puis la sous-rubrique `Installer le(s) package(s) . . .`. Une boîte de dialogue s'ouvre alors, vous demandant de choisir un site miroir pour le téléchargement. Une fois le miroir choisi, la liste exhaustive des `packages` disponibles sur le site apparaît dans une boîte de dialogue `Packages` : il suffit de sélectionner la bibliothèque souhaitée.

- si l'ordinateur sur lequel on travaille n'est pas connecté à internet, il faut disposer d'un fichier .zip contenant le package (ces fichiers compressés sont téléchargeables à partir de la page web du projet R). Choisir alors la rubrique Packages dans le menu déroulant, puis la sous-rubrique Installer le(s) package(s) depuis des fichiers zip Une boîte de dialogue s'ouvre, vous demandant d'indiquer l'emplacement du fichier zip, ce qui lance la procédure d'installation.

Cas particulier des packages du projet bioconductor

Le projet bioconductor fédère un grand nombre de packages, tous dédiés à l'analyse des données génomiques ou post-génomiques. Pour éviter un long travail de recherche, de sélection et de téléchargement individuels de chacun des packages, un fichier de commandes, réalisant le téléchargement automatique de l'ensemble des packages concernés, est accessible sur le site du projet bioconductor :

```
> source("http://www.bioconductor.org/biocLite.R")
# Chargement du fichier de commandes biocLite.R
# disponible sur la page web http://www.bioconductor.org.
> biocLite()
# La fonction biocLite est définie dans biocLite.R. Elle commande le
# téléchargement automatique des packages (version allégée de Bioconductor).
```

Un package particulièrement intéressant : Rcmdr (prononcer "Rcommander")

Le package Rcmdr permet de produire des commandes R sans les éditer directement grâce à un menu déroulant assez complet. C'est donc un moyen très simple d'appréhender le logiciel lorsque l'on est débutant. Toutefois, le menu déroulant de Rcmdr étant limité à certaines applications, l'utilisation de ce package prive l'utilisateur de tout une palette de méthodes statistiques.

Dans la suite, pour quelques opérations classiques, on privilégie l'utilisation du package Rcmdr lorsqu'elle simplifie la procédure.

3 Importation de données

L'objectif est ici d'importer des données contenues dans un fichier nommé poulets.txt au format texte. Dans ce fichier, les colonnes sont délimitées par des tabulations et les données manquantes sont signalées par NA (de l'anglais Not Available, NA est aussi le code d'une donnée manquante dans R).

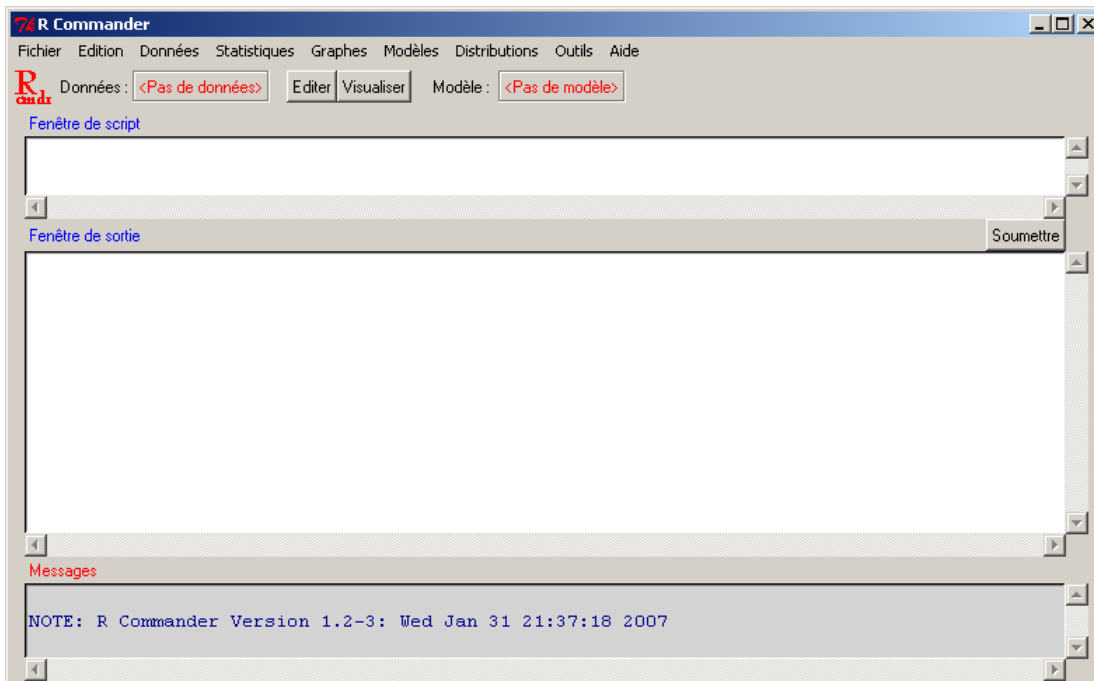


FIG. 1: Fenêtre Rcmdr.

3.1 Lancement de Rcmdr

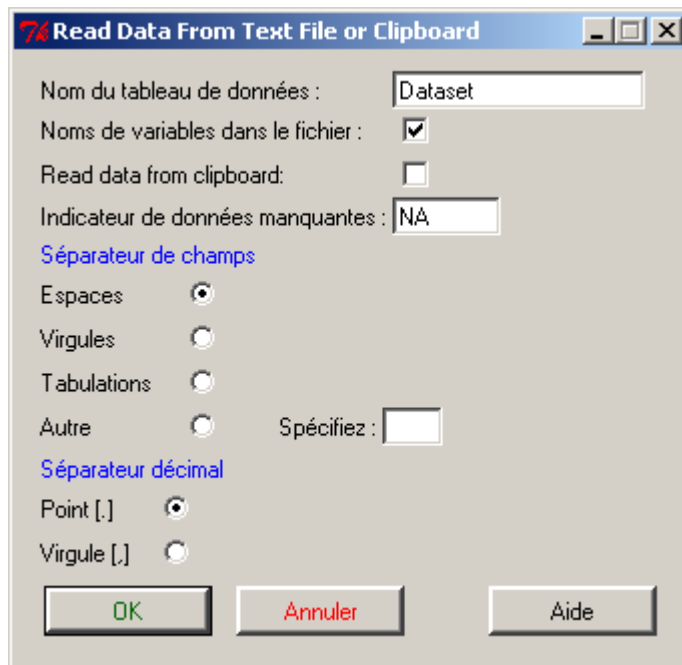
Charger le package Rcmdr. Une fenêtre R Commander s'ouvre alors, identique à celle de la figure 1, composée de quatre parties reproduisant l'environnement de travail de R :

- un menu déroulant.
Il est beaucoup plus riche que celui de R, donnant accès à de nombreuses rubriques d'édition et de traitement de données.
- une fenêtre de script.
On peut l'utiliser comme la fenêtre de script de R, l'icône Run étant remplacée par un bouton Soumettre placé sous la fenêtre. Cependant, son grand intérêt réside dans le fait que s'y affiche la commande générée lors d'une action par le menu déroulant.
- une fenêtre de sortie.
A l'instar de la fenêtre de commandes de R, les commandes et les résultats provenant de la fenêtre de script s'y exécutent.
- une fenêtre de messages.

Elle donne accès aux informations relatives au déroulement des opérations.

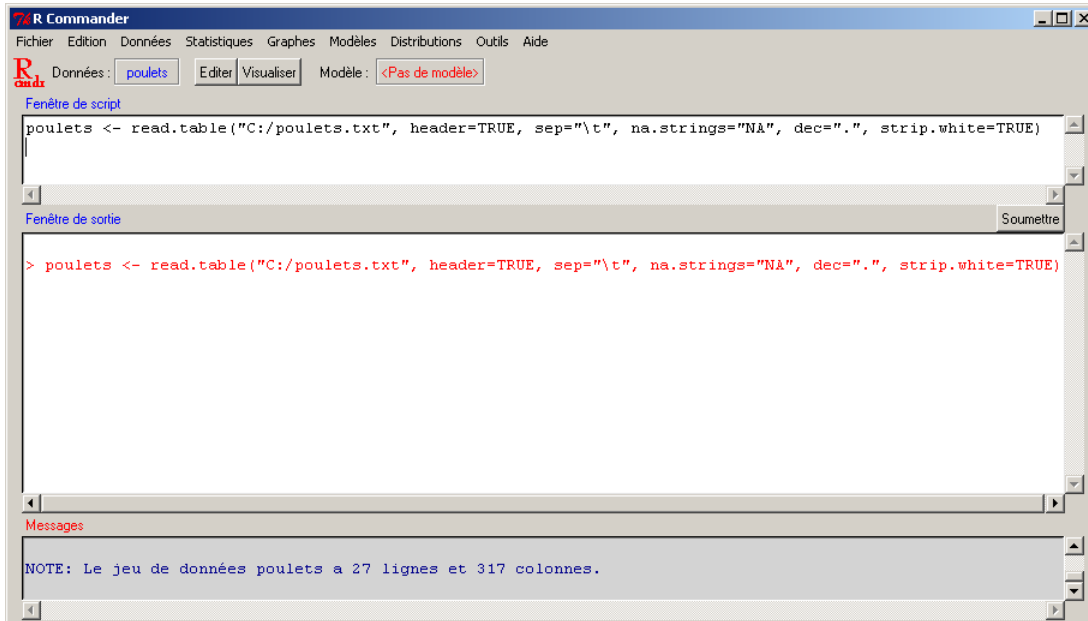
3.2 Importation

Dans le menu déroulant de Rcmdr, choisir la rubrique Données, puis la sous rubrique Importer des données et enfin From text file or clipboard S'ouvre alors la boîte de dialogue suivante :



On propose ici de remplacer, dans la rubrique Nom du tableau de données, le nom générique Dataset par un nom plus descriptif du contenu des données, par exemple poulets. Il faut aussi cocher le choix Tabulations dans la rubrique Séparateur de champs. Après lance-

ment de la procédure d'importation, la fenêtre Rcmdr apparaît ainsi :



La commande générée par la procédure d'importation est automatiquement éditée dans la fenêtre script: `poulets <- read.table(...)` et exécutée dans la fenêtre de sortie. La fenêtre de messages annonce le résultat: Le jeu de données poulets a 27 lignes et 317 colonnes.

3.3 Coup d'oeil rapide sur les données

Le bouton Données, au dessus de la fenêtre script, indique que le tableau de données sur lequel les actions opéreront, sauf mention contraire, est poulets. Un clic sur le bouton Éditer, à droite du bouton Données, permet de voir l'intégralité des données sous forme de table. Notons que ce jeu de données contient 314 variables quantitatives et 3 variables qualitatives placées en fin de tableau.

Il est très conseillé de précéder toute analyse d'une synthèse des données, variable par variable: cette synthèse permet souvent d'identifier des problèmes d'importation, comme le non-respect de la nature d'une variable. Pour cela, dans le menu déroulant, choisir la rubrique Statistiques, la sous-rubrique Résumés et enfin Jeu de données actif.

Pour chaque variable quantitative, la synthèse apparaissant dans la fenêtre de sortie est constituée de statiques élémentaires: valeur minimale, 1er quartile, médiane, moyenne, 3ème quartile et valeur maximale. Pour les variables qualitatives, la synthèse consiste en un décompte des effectifs par modalité.

4 Premières analyses statistiques

Le jeu de données sur lequel on travaille est `poulets`. Il s'agit pour R d'une grande armoire contenant autant de tiroirs qu'il y a de variables. Pour accéder au tiroir `G1.6E.3.ddrt` de l'armoire `poulets`, la commande est la suivante :

```
poulets$G1.6E.3.ddrt # Valeurs de la variable G1.6E.3.ddrt du tableau poulets
```

à éditer dans la fenêtre script et à exécuter en cliquant sur le bouton Soumettre.

De manière équivalente, comme `G1.6E.3.ddrt` est le nom du premier tiroir de `poulets`, on aurait pu éditer la commande suivante :

```
poulets[,1] # Valeurs de la 1ère variable du tableau poulets
```

La virgule entre les crochets sépare à gauche une sélection éventuelle sur les lignes du tableau (ce n'est pas le cas ici) et à droite une sélection éventuelle sur les colonnes (ici, on sélectionne la 1ère colonne).

Si l'on souhaite maintenant voir les valeurs des 5 premières variables du tableau `poulets`, la commande s'écrit de la façon suivante :

```
poulets[,1:5] # Valeurs des 5 1ères variables du tableau poulets
```

Dans la commande ci-dessus, `1:5` est la collection des entiers entre 1 et 5.

4.1 Transformation de variables

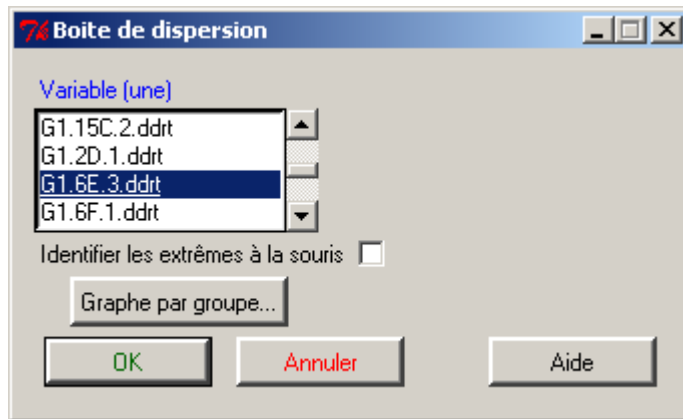
On souhaite ici remplacer toutes les données quantitatives du jeu de données `poulets` par leur logarithme en base 2. Pour cela, on exécute la commande suivante :

```
poulets[,1:314] = log2(poulets[,1:314]) # Remplace les 314 1ères variables  
# du tableau poulets par leur logarithme.
```

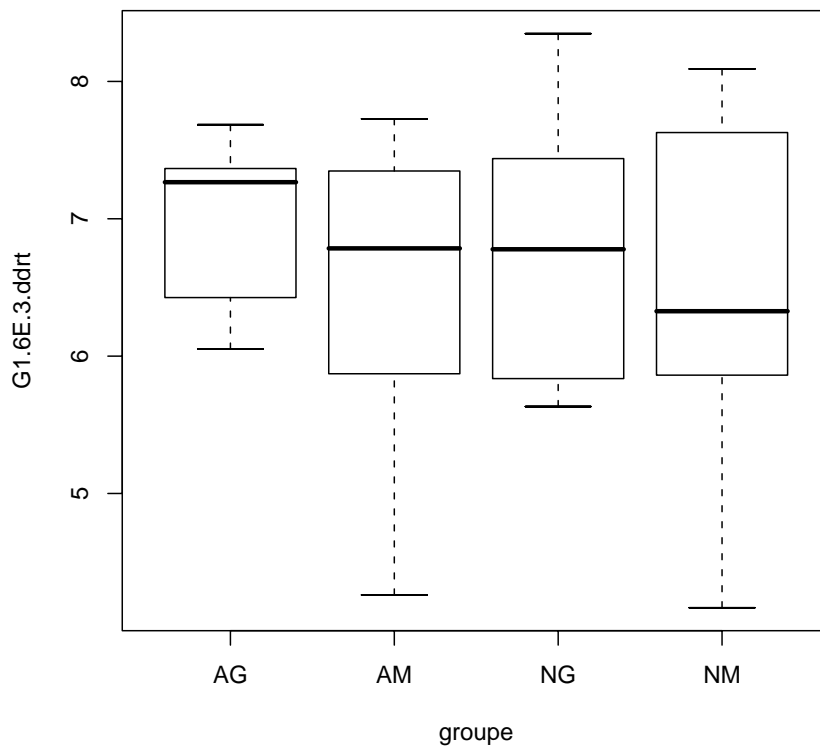
4.2 Boîtes de dispersions

Les analyses portent dans un premier temps sur une variable en particulier, par exemple `G1.6E.3.ddrt`. Pour comparer les répartitions des valeurs de cette variable selon les modalités

de la variable groupe, choisir, dans le menu déroulant, la rubrique Graphes puis la sous-rubrique Boîte de dispersion S'ouvre alors la boîte de dialogue suivante :



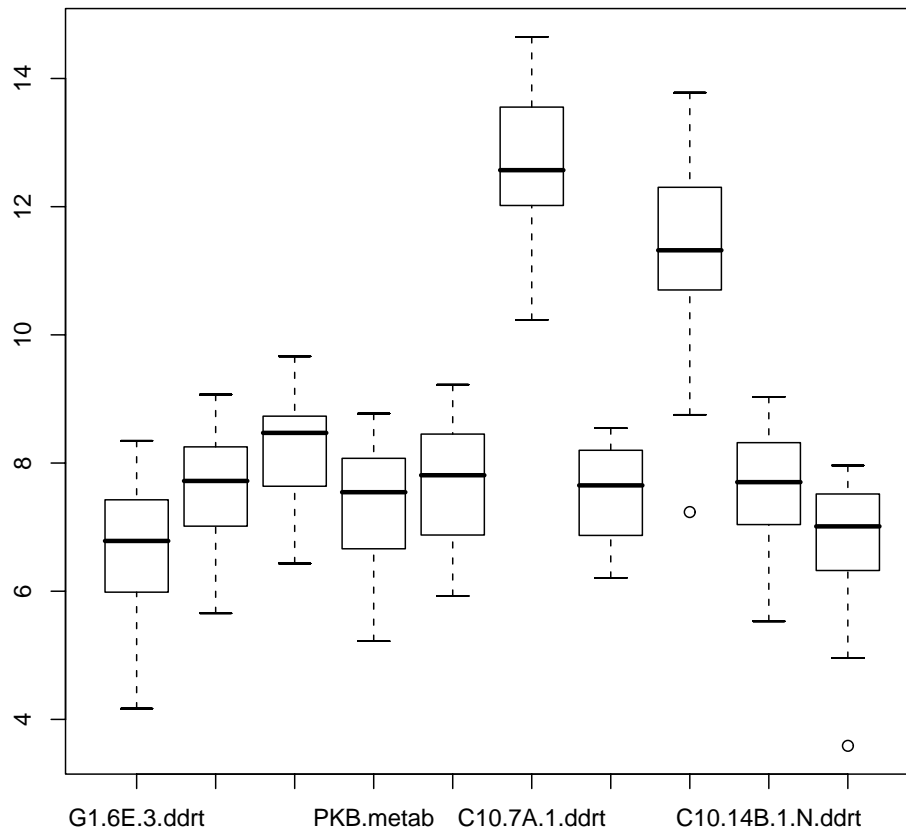
En cliquant sur le bouton Graphe par groupe ..., on peut construire une boîte de dispersion pour chaque modalité d'une variable qualitative. Le graphique suivant résulte de la commande décrite ci-dessus :



Dans certains cas, il peut aussi être intéressant de comparer les répartitions des valeurs contenues dans chaque ligne ou dans chaque colonne d'un tableau. Par exemple, la commande suivante produit sur un même graphique les boîtes de dispersion des 10 1ères variables du tableau poulets.

```
boxplot(poulets[,1:10])
```

Le graphique obtenu est le suivant :

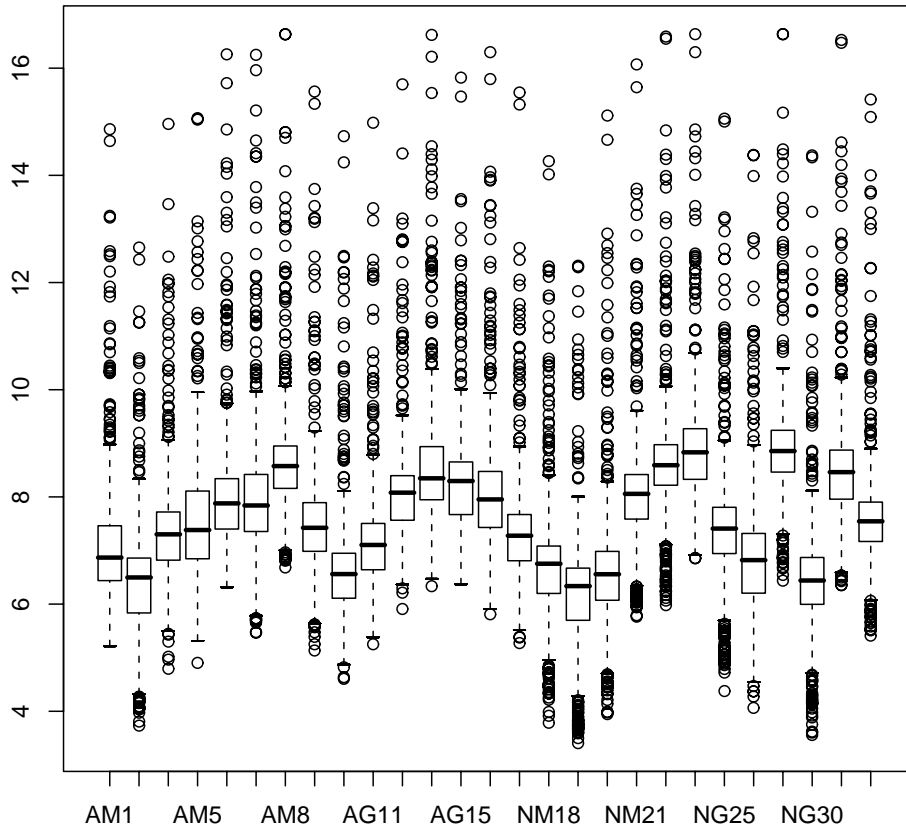


Si l'on veut construire un graphique affichant les boîtes de dispersion sur chacune des lignes du tableau constitué des variables quantitatives de poulets, il est tentant d'appliquer la même méthode que précédemment sur le tableau `poulets[,1:314]` transposé, à savoir sur `t(poulets[,1:314])`. Cependant, cette méthode ne donnera pas le résultat escompté car `t(poulets[,1:314])` n'a pas hérité du format d'un jeu de données de R (une armoire dont les variables sont les tiroirs). Pour que R comprenne que chaque ligne du tableau `poulets[,1:314]` doit être vue comme un tiroir de l'armoire `t(poulets[,1:314])`, il faut donner

au tableau transposé le format d'un jeu de données par la fonction `data.frame` :

```
boxplot(data.frame(t(poulets[,1:314]))) # Boîtes de dispersion sur le tableau
# poulets[,1:314] transposé
```

On obtient alors le graphique suivant :



4.3 Un exemple d'opération sur des données : le centrage

Dans un premier temps, on souhaite calculer les valeurs centrées de la variable `G1.6E.3.ddrt` du tableau `poulets`. La commande utilise la fonction `mean` qui calcule la moyenne d'une collection de valeurs :

```
> poulets$G1.6E.3.ddrt - mean(poulets$G1.6E.3.ddrt) # Soustrait à toutes les valeurs de la
# variable G1.6E.3.ddrt sa moyenne.
[1] -0.375243769 -2.411818869 -1.227913143 0.523226246 0.111226231 0.826118303
[7] 1.054716304 0.093894423 -0.621842000 -0.587709069 0.644192329 1.012216805
```

```
[13] 0.741370505 0.593150576 -0.346262415 -0.869849867 -2.505060676 -0.753662801
[19] 0.539531617 1.369700586 1.417654792 0.204893946 -0.836812823 1.674964575
[25] -1.040263205 0.764981170 0.004600228
```

Lorsque l'on souhaite appliquer cette opération à l'ensemble des lignes d'un tableau, on commence par calculer les moyennes par ligne. La fonction `apply` est dédiée à cette automatisation d'un calcul. Les arguments de cette fonction sont :

- X, le tableau de données
- MARGIN, qui prend la valeur 1 si l'opération est effectuée sur chaque ligne et 2 si elle est effectuée sur chaque colonne
- FUN, la fonction correspondant à l'opération souhaitée.

Par exemple, la commande suivante permet le calcul des moyennes de chaque ligne du tableau `poulets[,1:314]` :

```
moyennes = apply(X=poulets[,1:314],MARGIN=1,FUN=mean,na.rm=TRUE)
# Calcule la moyenne de chaque ligne du tableau poulets[,1:314].
```

Notons qu'un argument supplémentaire, `na.rm`, s'est glissé dans la commande ci-dessus ; c'est en fait un argument de la fonction `mean` permettant de définir la façon dont sont gérées les données manquantes. En l'occurrence, cet argument prend la valeur `TRUE`, ce qui revient à ignorer les valeurs manquantes.

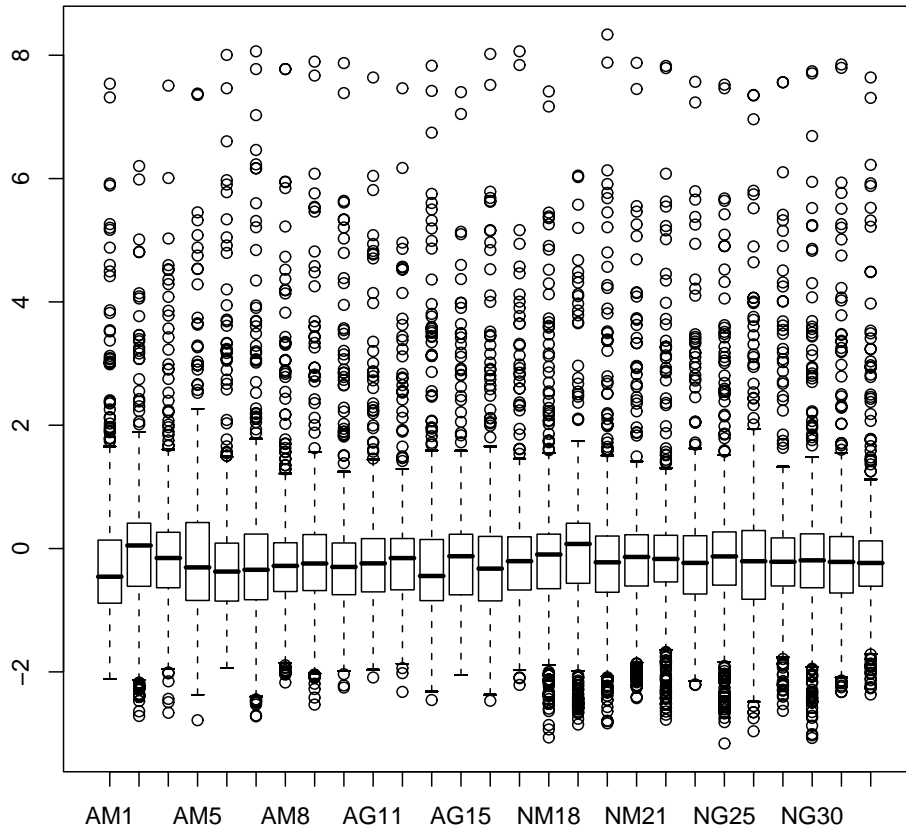
Pour soustraire aux valeurs d'une ligne leur valeur moyenne, on dispose de la fonction `sweep` qui, comme son nom l'indique, balaie le jeu de données. Les arguments de cette fonction sont les suivants :

- x, le tableau de données
- MARGIN, qui prend la valeur 1 si l'opération est effectuée sur chaque ligne et 2 si elle est effectuée sur chaque colonne
- STATS, la statistique qui intervient dans l'opération

Ainsi, le centrage des lignes du tableau `poulets[,1:314]` est obtenu par la commande suivante :

```
poulets[,1:314] = sweep(x=poulets[,1:314],MARGIN=1,STATS=moyennes)
# Soustrait à toutes les lignes de poulets[,1:314] leur moyenne.
```

La construction de boîtes de dispersion par ligne, selon la recette proposée dans la section 4.2, montre bien l'impact du centrage précédent :



5 Automatisation d'un traitement

L'identification des variables du tableau de données poulets pour lesquelles les valeurs moyennes diffèrent significativement d'un groupe à un autre sert ici d'illustration à l'automatisation d'un calcul. Il s'agit en effet d'appliquer une même règle de décision à chaque variable du tableau. Cette automatisation passe par deux étapes : d'une part la définition de la fonction réalisant l'opération souhaitée sur une seule variable et d'autre part l'application systématique de cette fonction à toutes les variables.

5.1 Principes généraux de la création d'une fonction

De manière générale, on définit une fonction dans R à l'aide de la commande `function` :

```
NomFonction = function(a1,a2,...) # Nom et arguments de la fonction
{                                # Début de la fonction
...                              # Corps de la fonction (suite d'opérations)
return(sortie)                  # sortie est le résultat de la fonction
}                                # Fin de la fonction
```

Avant de créer une fonction, il est nécessaire de connaître ses *entrées* et ses *sorties* :

- *entrées* ; les arguments de la fonction, soit les éléments nécessaires à l'opération que l'on souhaite réaliser.
- *sorties* ; spécifiées par la fonction `return` dans la dernière ligne de la fonction.

Par exemple :

```
MaFonction = function(x,y) { # MaFonction a deux arguments: x et y
z=x+sqrt(y)                 # z est un objet défini localement dans MaFonction
return(1/z) }               # MaFonction calcule 1/(x + sqrt(y))
```

On utilise désormais cette nouvelle fonction comme n'importe quelle autre fonction de R :

```
> MaFonction(x=1,y=2)
[1] 0.4142136
```

ou encore :

```
> MaFonction(1,2)
[1] 0.4142136
```

Il est également possible de fixer des valeurs par défaut aux arguments d'une fonction. Par exemple, si l'on souhaite que, par défaut, $y=2$ dans la fonction ci-dessus :

```
MaFonction = function(x,y=2) { # Par défaut, y=2
z=x+sqrt(y)
return(1/z) }
```

Désormais, lors de l'appel de la fonction, si l'argument y n'est pas spécifié, alors sa valeur par défaut lui est automatiquement affectée :

```
> MaFonction(1)
[1] 0.4142136
```

5.2 Un exemple d'analyse statistique : le test de comparaison de deux moyennes

Dans le cas présent, la fonction que l'on souhaite créer réalise un test de Student comparant les moyennes de deux séries de valeurs. Commençons par un test sur la série des 27 valeurs de la 1ère colonne du tableau poulets,

```
> variable=poulets$G1.6E.3.ddrt # variable est la variable nommée G1.6E.3.ddrt
```

le facteur groupe définissant les deux groupe à comparer :

```
> groupe=poulets$genotype # groupe est la variable nommée genotype
```

```
> groupe
[1] M M M M M M M G G G G G G M M M M M M M G G G G G
Levels: G M
```

La commande `t.test(variable~groupe)` réalise alors un test de comparaisons des moyennes de `variable` selon les modalités de `groupe` :

```
> z=t.test(variable~groupe,var.equal=TRUE) # z contient les résultats du test
```

`var.equal` est un argument logique de la fonction `t.test` permettant de postuler l'égalité des variances intra-groupes. Si `var.equal=TRUE` (variances égales), un test de Student est réalisé (en revanche, si `var.equal=FALSE`, c'est un test de Welch qui est réalisé).

Ainsi défini, `z` contient plusieurs informations (moyennes intra-groupes, probabilité critique, intervalle de confiance, ...). Il faut voir cet objet comme une armoire dont les tiroirs contiennent chacun une information. Les noms de ces tiroirs sont accessibles par la commande `names` :

```
> names(z)
[1] "statistic" "parameter" "p.value" "conf.int" "estimate"
[6] "null.value" "alternative" "method" "data.name"
```

Lorsque l'on souhaite voir le contenu d'un tiroir, on utilise la clé `$`, suivie du nom du tiroir :

```
> z$p.value
[1] 0.2000234
```

5.3 Automatisation d'une analyse statistique

La fonction à créer opère sur deux arguments, la variable dont on étudie les différences de moyennes intra-groupes, et le facteur définissant les groupes :

```
MonStudent = fonction(variable,groupe) {  
  z=t.test(variable~groupe,var.equal=TRUE) # z contient tous les résultats du test  
  z=z$p.value # z contient la probabilité critique  
  return(z<=0.05) } # MonStudent est logique :  
# TRUE si la différence de moyennes est significative, FALSE sinon
```

L'application de cette nouvelle fonction à l'ensemble des variables quantitatives du tableau poulets est maintenant possible à l'aide de la fonction apply :

```
> dg = apply(poulets[,1:314],MARGIN=2,FUN=MonStudent,groupe=poulets$genotype)  
# Applique MonStudent à chaque colonne du tableau poulets[,1:314].
```

Pour connaître la liste des variables pour lesquelles la différence de moyennes est significative, on sélectionne dans les noms des variables quantitatives de poulets ceux tels que dg=TRUE :

```
> noms=names(poulets[,1:314]) # noms collecte les noms des variables quantitatives.  
> noms[dg] # Affiche les noms tels que dg=TRUE.
```

```
[1] "G5.5B.1.ddrt" "PKCL2.metab" "NMT2.metab" "PI3K.metab" "G7.4B.3.ddrt"  
[6] "A2.5A.1.ddrt" "A2.5B.1.ddrt" "C5.11B.1.ddrt" "G1.12B.1.ddrt"
```