Feuille de Travaux Dirigés n° 1
Initiation à R

1 Introduction : Qu'est-ce-que R ?

- R est un logiciel permettant de faire des analyses statistiques et de produire des graphiques.
- Nous allons utiliser R comme une boîte à outils pour faire des analyses statistiques.
- Mais R est également un langage de programmation complet. C'est cet aspect qui fait que R est différent des autres logiciels statistiques.
- Les informations sur R sont disponibles sur la homepage du projet :
  `http://www.r-project.org/`
  C'est le premier résultat pour la recherche de la lettre R avec le moteur de recherche google et la meilleure source d'informations sur le logiciel R. Vous pourrez y trouver les différentes distributions du logiciel, de nombreuses bibliothèques de fonctions et des documents d'aide.
- Enfin, R est un clône gratuit du logiciel S-Plus commercialisé par MathSoft et développé par Statistical Sciences autour du langage S (conçu par les laboratoires BELL).

2 Comment se procurer le logiciel R ?

Le logiciel R est gratuit. La page officielle du logiciel est :
`http://www.r-project.org/`
Si vous avez un ordinateur à la maison, vous pouvez le télécharger à l'adresse :
`http://mirrors.toulouse.inra.fr/R/`

3 Remarques d'ordre général sur R :

- Bien sûr il existe une version française du logiciel R.
- R fonctionne avec plusieurs fenêtres sous Windows. En particulier, nous distinguons la fenêtre `R Console`, fenêtre principale où sont réalisées par défaut les entrées de commandes et sorties de résultats en mode texte. À celle-ci peuvent s'ajouter un certain nombre de fenêtres facultatives, telles que les fenêtres graphiques et les fenêtres d'informations (historique des commandes, aide, visualisation de fichier, etc...), toutes appelées par des commandes spécifiques via la console.
- Le menu `File` ou `Fichier` contient les outils nécessaires à la gestion de l'espace de travail, tels que la sélection du répertoire par défaut, le chargement de fichiers sources externes, la sauvegarde et le chargement d'historiques de commandes, etc.

- Le menu `Edit` ou `Edition` contient les habituelles commandes de copier-coller, ainsi que la boîte de dialogue autorisant la personnalisation de l'apparence de l'interface.
- Le menu `Misc` traite de la gestion des objets en mémoire et permet d'arrêter une procédure en cours de traitement.
- Le menu `Packages` automatise la gestion et le suivi des librairies de fonctions, permettant leur installation et leur mise à jour de manière transparente au départ du site `CRAN (Comprehensive R Archive Network)`
  `http://cran.r-project.org/`
  ou de toute autre source locale.
- Enfin, les menus `Windows` ou `Fenêtres` et `Help` ou `Aide` assument des fonctions similaires à celles qu'ils occupent dans les autres applications Windows, à savoir la définition spatiale des fenêtres et l'accès à l'aide en ligne et aux manuels de références de `R`.
- Ce qui est entré par l'utilisateur figure en rouge, et la réponse de `R` est en bleu.
- Les nombres entre crochets au début de chaque ligne donnent l'indice du premier nombre de la ligne.
- Quand deux vecteurs ne sont pas de même longueur, le plus court est **recyclé**.

4 Pour en savoir plus sur `R`

- Pour un public francophone, un point de départ est le manuel d'Emmanuel Paradis, « `R` pour les débutants », 81 pages, qui a la particularité d'exister également en version anglaise « `R` for Beginners ». Les deux documents sont disponibles ici :
  `http://cran.r-project.org/`
  dans la rubrique « Documentation », sous-rubrique « Contributed ».
- Plusieurs milliers de pages d'enseignement en français de statistiques sous `R` sont disponibles ici :
  `http://pbil.univ-lyon1.fr/R/`

5 Objectif de ce T.D.

Ce T.D. a pour objectif de vous montrer les commandes de base de `R` (ouverture, fermeture, sauvergarde, aide,...) et de vous faire manipuler des tableaux de données (saisie sous `R`, importation de fichier de données,...).

6 Pour commencer avec `R`

a) **Démarrer `R` :**
Vous lancez le logiciel `R` en cliquant sur l'icône `R`. Le symbole `>` signifie que `R` est prêt à travailler. Il ne faut pas taper ce symbole au clavier car il est déjà présent en début de ligne sur la `R Console`. C'est à la suite de ce symbole `>`

que vous pourrez taper les commandes R. Une fois la commande tapée, vous devez toujours la valider par la touche Entrée.

b) **Quitter R :**
Pour quitter R, vous utilisez la commande
`>q()`
La question `Save workspace image? [y/n/c]` est posée : R propose de sauvegarder le travail effectué. Trois réponses possibles : `y` (pour yes), `n` (pour no) ou `c` (pour cancel, annuler). En tapant `c`, la procédure de fin de session sous R est annulée. Si vous tapez `y`, cela permet que les commandes tapées pendant la session soient conservées en mémoire et soient donc « rappelables » (mais vous ne pouvez pas les imprimer).

c) **Sauvegarder sous R :**
Si vous quittez R en choisissant la sauvegarde de l'espace de travail, deux fichiers sont créés :

(i) le fichier `.Rdata` contient des informations sur les variables utilisées ,

(ii) le fichier `.Rhistory` contient l'ensemble des commandes utilisées.

d) **Travailler avec R :**
Par exemple, tapez la commande suivante et validez :
`>2 + 5`
Le résultat s'affiche sous la forme :
`[1] 7`
Le chiffre 1 entre crochets indique l'indice du premier élément de la ligne [1] , le second chiffre est le résultat de l'opération demandée.

e) **Consulter l'aide de R**
Pour toutes les commandes, vous pouvez consulter une fiche de documentation en tapant, par exemple pour la commande `read.table` :
`>?read.table`
Faire défiler le texte avec la touche « Entrée » ou « Flèche vers la bas ». Une fois arrivé à « END », taper `q`. Grâce à cette aide, il suffit de retenir le nom de la commande, mais pas toute la syntaxe.

Vous pouvez rappeler les commandes déjà éxécutées (pendant cette séance) en utilisant la touche « Flèche vers le haut ».

7 Rentrer des données sous R

Différentes commandes sont disponibles pour saisir des données sous R.

1. Par exemple dans le résultat suivant l'indice de l'élément 123 est 1 et celui de 142 est 20

```
 [1]  123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141
[20]  142 143 144 145
```

7.1 Affectation

Un objet peut être créé avec l'opérateur « assigner » ou « affecter » qui s'écrit `<-` :

```
> n<-15
> N<-12
```

Pour vérifier le contenu d'un objet, taper son nom, par exemple pour `n` :

```
> n
[1] 15
```

Remarques :

- `R` différencie les lettres minuscules et les lettres majuscules.
- Quand on assigne un nom à un objet, l'affichage de cet objet n'est plus automatique, il faut le demander en tapant simplement le nom donné à l'objet.
- En fait, on peut remarquer que le signe `=` marche également pour faire des affectations. Essayer :

  ```
  > a=3
  > a
  3
  ```

Choisisser donc la manière que vous voulez pour affecter !

7.2 Suite

Premier exemple. Vous souhaitez créer la suite d'entiers de 1 à 12 :

Première façon

```
> suite <- 1:12
> suite
[1] 1  2  3  4  5  6  7  8  9  10  11  12
```

Deuxième façon

La fonction `seq` crée une suite (séquence) de nombres et possèdent trois arguments : `from, to` et `by`.

```
> seq(from=1,to=12,by=1)
[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

Remarque : On peut aussi écrire simplement :

```
> seq(1,12,1)
[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

Second exemple. Vous souhaitez créer un vecteur formé par les éléments d'une suite arithmétique de premier terme `20`, de dernier terme `40` et de raison `5`, on utilise encore la fonction `seq` :

```
>seq(from=20,to=40,by=5)
[1] 20  25  30  35  40
```

7.3 Combinaison ou vecteur

Il est possible de saisir une série de valeurs numériques, caractères ou logiques.

Premier exemple.

```
> serie1<-c(1.2,36,5.33,-26.5)
```

`serie1` est un vecteur numérique. Comment le savoir ?
Taper la commance `class` ou `mode` sur le nom de votre vecteur.
Nous reviendrons sur `mode` au paragraphe suivant.

```
> serie1
[1] 1.20  36.00  5.33  -26.50
```

Que remarquez-vous ?

Deuxième exemple.

```
> serie2<-c("bleu","vert","marron")
```

`serie2` est un vecteur de chaînes de caractères

```
> serie2
[1] "bleu" "vert" "marron"
```

Remarque : Si un vecteur est composé de caractères et de nombres, le vecteur sera un vecteur de chaînes de caractères. Quand les composantes du vecteur sont des chaînes de caractères, il est obligatoire de les déclarer entre guillemets, sinon `R` ne reconnaît pas les composantes du vecteur : `> serie2<-c(bleu,vert,marron)`

```
Error : Object "bleu" not found
```

Troisième exemple.

```
> serie3<-c(T,T,F,F,T)
```

serie3 est un vecteur logique

```
> serie3
[1] TRUE  TRUE  FALSE  FALSE  TRUE
```

Quatrième exemple.

Lors d'une étude statistique, il peut arriver que certaines données ne soient pas disponibles : on dit que **la donnée est manquante**. Pour saisir une donnée manquante, on utilise le symbole `NA` (Not Available) que l'objet soit numérique, caractère ou logique :

```
> serie4<-c(1.2,36,NA,-26.5)
```

la 3ième valeur est laissée en valeur manquante

```
> serie4
[1] 1.20  36.00  NA  -26.50
```

7.4 Mode et longueur

Les objets sont caractérisés par deux attributs : **le mode et la longueur**. **Le mode** est le type des éléments d'un objet. Comme nous venons de le voir, un objet peut être numérique, caractère ou logique. **La longueur** est le nombre d'éléments de l'objet.

Par exemple, si vous saisissez une série d'observations obtenues sur un échantillon sous la forme d'un vecteur, la longueur de ce vecteur correspondra à la taille de l'échantillon. Pour connaître le mode et la longueur d'un objet, on utilise les fonctions `mode` et `length` :

```
> mode(serie1)
[1] "numeric"
> mode(serie2)
[1] "character"
> mode(serie3)
[1] "logical"
> length(serie1)
[1] 4
>length(serie2)
[1] 3
> length(serie3)
[1] 5
```

7.5 Saisie « au clavier » d'un jeu de données

En utilisant la fonction SCAN, la saisie d'une série de données peut paraître moins fastidieuse.

```
> jeu1<-scan()
```

R vous redonne la main et vous pouvez taper les valeurs du jeu de données :

```
1:1.2
2:36
3:5.33
4:-26.5
5:
```

Le premier retour-chariot après une chaîne vide met fin à la saisie

```
> jeu1
[1] 1.20  36.00  5.33  -26.50
```

7.6 Éléments d'un vecteur

Il est possible de demander l'affichage d'un (ou de plusieurs) élément(s) d'un vecteur en spécifiant entre crochets, en plus du nom du vecteur, l'indice de l'élément du vecteur. Par exemple, pour afficher le troisième élément :

```
> serie1[3]
[1] 5.33
> serie1[3:4]
[1] 5.33   -26.50
```

8 Manipuler des vecteurs

Plusieurs opérations sont possibles sur les vecteurs : concaténation, extraction, calculs, répétition, légende et tri.

8.1 Concaténer deux vecteurs

Il est possible de concaténer deux vecteurs (formés de variables de même type) pour en former un nouveau :

```
>x <- c(2.3,3.5,6,14,12)
>y <- c(3.2,5,0.7,1,3.5)
>z <- c(x,y)
>z
[1] 2.3  3.5  6.0  14.0  12.0  3.2  5.0  0.7  1.0  3.5
```

8.2 Extraire des données d'un vecteur

Il est possible d'extraire des données à partir d'un vecteur selon trois façons :

1. Utiliser un vecteur pour préciser le numéro d'ordre des composantes à extraire. Ainsi pour extraire les 2ème et 5ème composantes du vecteur `x` :
   ```
   > x[c(2,5)]
   [1] 3.5 12.0
   ```
2. L'utilisation du signe tiret permet de supprimer des composantes, par exemple pour supprimer les 2ème et 3ème composantes du vecteur `x` :
   ```
   > x[-c(2,3)]
   [1] 2.3 14.0 12.0
   ```
3. Utiliser un vecteur formé de valeurs logiques. Par exemple, pour obtenir un vecteur ne contenant que les composantes supérieures à 4, vous pouvez utiliser la commande :
   ```
   > x[x>4]
   [1] 6 14 12
   ```

Si vous disposez de deux vecteurs ayant le même nombre de composantes, vous pouvez demander à afficher les valeurs de l'un pour lesquelles les valeurs de l'autre sont supérieures (ou inférieures) à une certaine valeur. Par exemple, les vecteurs `x` et `y` sont composés de `5` valeurs. Vous pouvez demander d'extraire de `y` les valeurs de `y` pour lesquels `x` est supérieur à `4` en utilisant la ligne de commande suivante :

```
> y[x>4]
[1] 0.7 1.0 3.5
```

8.3 Faire des calculs sur les composantes d'un vecteur

`R` peut faire des calculs sur l'ensemble des composantes d'un vecteur :

```
> 20+x*5
[1] 31.5  37.5  50.0  90.0  80.0
> (x+y)/2
[1] 2.75  4.25  6.50  12.00  12.75
```

8.4 Remplacer des données dans un vecteur

Il est possible de remplacer certaines composantes d'un vecteur par de nouvelles valeurs.
Considérons d'une suite de valeurs numériques :

```
> x <- 1:10
```

Premier exemple :
Si vous voulez remplacer la 3ème valeur de `x` par 35, vous utiliserez alors la ligne de commande suivante :

```
> x[3] <- 35
```

puis vous demanderez à `R` d'afficher le résultat

```
> x
[1] 1  2  35  4  5  6  7  8  9  10
```

Deuxième exemple :
Si vous voulez remplacer la valeur `1` par la valeur `25`, vous utiliserez alors la ligne de commande suivante :

```
> x[x==1] <- 25
```

puis vous demanderez à `R` d'afficher le résultat

```
> x
[1] 25  2  35  4  5  6  7  8  9  10
```

Troisième exemple :
Si vous voulez remplacer toutes les valeurs supérieures ou égales à 5 par 20, vous utiliserez alors la ligne de commande suivante :

```
> x[x>=5] <- 20
```

puis vous demanderez à `R` d'afficher le résultat

```
> x
[1] 20  2  20 4 20 20 20 20 20 20
```

8.5 Répéter les données d'un vecteur

La fonction `rep` a deux arguments `x` et `times` et crée un vecteur où `x` est répété `times` fois.

Premier exemple :
Vous créez une variable `donnees` par :

```
> donnees <- c(1,2,3)
```

Si vous voulez qu'un nouveau vecteur contienne deux fois le vecteur donnees, alors vous écrirez :

```
> rep(x=donnees,times=2)
```

Deuxième exemple :
Vous pouvez également demander qu'un vecteur contienne 50 fois la valeur 1 :

```
> rep(1,50)
```

Troisième exemple :
ou 4 fois la chaîne de caractères « chien » :

```
> rep("chien",4)
```

8.6 Nommer les composantes d'un vecteur

Il est possible de donner un nom à chaque composante d'un vecteur.
Exemple
Le vecteur `notes.Jean` contient les notes obtenues par Jean en Anglais, Informatique et Biologie.

Première façon :
Vous pouvez utiliser la commande :
```
> notes.Jean <- c(Anglais=12,Informatique=19.5,Biologie=14)
```
Afficher le vecteur `notes.Jean`. Vous obtenez le résultat suivant :

```
Anglais  Informatique  Biologie
12.0     19.5          14.0
```

Seconde façon :
Une autre façon de nommer les composantes d'un vecteur est de définir un vecteur formé de chaînes de caractères, puis utiliser la fonction `names` :
```
> matiere <- c("Anglais","Informatique","Biologie")
> matiere
> note <- c(12,19.5,14)
> note
[1] 12.0  19.5  14.0
> names(note) <- matiere
> note
Anglais  Informatique  Biologie
12.0     19.5            14.0
```

Remarque : Pour supprimer les noms :
```
> names(note) <- NULL
```

8.7 Trier les composantes d'un vecteur

Vous pouvez trier les composantes d'un vecteur par ordre croissant en utilisant la fonction sort.
Retour à l'exemple précédent :
Pour trier les notes dans l'ordre croissant :
```
> sort(note)
```
ou dans l'ordre décroissant :
```
> rev(sort(note))
```

8.8 Opérateurs logiques

L'extraction ou le remplacement de valeurs dans un vecteur passe par l'utilisation de vecteur logiques. Reprenons l'un des exemples précédents.
```
> suite <- 1:12
> suite
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12
```

Nous souhaitons connaître les éléments du vecteur `suite` strictement supérieurs à 6 :

```
> suite > 6
[1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
TRUE
```

Le logiciel nous renvoie un vecteur logique. La valeur de chacune de ses coordonnées nous indique si l'élément qui est à la même position dans le vecteur `suite` est strictement supérieur à 6 ou non. Ainsi `TRUE` signifie strictement supérieur à 6 et `FALSE` signifie inférieur ou égal à 6.

Il est possible d'associer des vecteurs logiques `logic1` et `logic2` en utilisant les connecteurs logiques suivant :

- `!logic1` est la négation de `logic1`, également appelé opérateur NON, c'est-à-dire le vecteur dont les éléments sont `TRUE` lorsque ceux de `logic1` sont `FALSE` et dont les éléments sont `FALSE` lorsque ceux de `logic1` sont `TRUE`.
- `logic1 & logic2` est la conjonction, également appelé opérateur ET, des deux vecteurs logiques `logic1` et `logic2`.
- `logic1 && logic2` est identique à `logic1[1] & logic2[1]` et seul le premier élément de chacun des deux vecteurs est utilisé.
- `logic1 | logic2` est la disjonction inclusive, également appelé opérateur OU inclusif, des deux vecteurs logiques `logic1` et `logic2`.
- `logic1 || logic2` est identique à `logic1[1] | logic2[1]` et seul le premier élément de chacun des deux vecteurs est utilisé.
- `xor(logic1, logic2)` est la disjonction exclusive, également appelé opérateur OU exclusif, des deux vecteurs logiques `logic1` et `logic2`.

Voici quelques exemples d'utilisation d'opérateurs logiques.

```
> !(suite >6)
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
FALSE
> suite <= 6 & suite >= 6
[1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
FALSE
> suite <= 6 && suite >= 6
[1] FALSE
> suite <= 6 | suite >= 6
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> suite <= 6 || suite >= 6
[1] TRUE
```

Enfin voici pour mémoire les tables de vérité des opérateurs logiques. Celles-ci explicitent également le comportement des opérateurs en cas de valeurs manquantes.

```
> x <- c(NA, FALSE, TRUE)
> names(x) <- as.character(x)
> !x
 <NA> FALSE  TRUE
   NA  TRUE FALSE
> outer(x, x, "&")
       <NA> FALSE  TRUE
<NA>     NA FALSE    NA
FALSE FALSE FALSE FALSE
TRUE     NA FALSE  TRUE
> outer(x, x, "|")
      <NA> FALSE TRUE
<NA>    NA    NA TRUE
FALSE   NA FALSE TRUE
TRUE  TRUE  TRUE TRUE
> outer(x, x, "xor")
      <NA> FALSE  TRUE
<NA>    NA    NA    NA
FALSE   NA FALSE  TRUE
TRUE    NA  TRUE FALSE
```

9 Lire des données dans un fichier

Quand les données sont plus volumineuses, il n'est pas très conseillé d'utiliser `R` comme outil de saisie. Dans ce cas, vous pouvez utiliser un éditeur de texte ou un tableur quelconque pour saisir vos données (excel par exemple) et le transférer ensuite sous `R`.
Il est nécesaire d'indiquer au logiciel `R` l'endroit où sont stockés les fichiers de données. Ceci peut être fait soit à chaque chargement de fichier soit pour la durée de chaque utilisation du logiciel.
Pour connaître le répertoire de travail actuellement utilisé par `R`, qui est par défaut le répertoire où le logiciel est installé, il suffit de taper l' instruction suivante :

```
> getwd()
```

Pour changer le répertoire de travail par défaut, pour la durée de la session `R`, pour, par exemple, le répertoire "C:\Data", il suffit de taper :

```
> setwd("C:\\Data")
```

ou de manière équivalente

```
> setwd("C:/Data")
```

Pour des raisons liées à la syntaxe de `R`, plus précisement la syntaxe des systèmes Unix, la barre oblique inversée "\" a été remplacée soit par une barre oblique "/" soit par deux barres obliques inversées "\\".

Télécharger les fichiers au format texte (.txt) `table1.txt`, `table2.txt`, `table3.txt`, `table4.txt`, les fichiers au format csv (.csv) `table5.csv` et `table6.csv` et les

fichiers au format excel (.xls) `table7.xls` et `table8.xls` sur votre ordinateur dans un répertoire de l'ordinateur puis faites devenir ce répertoire le répertoire de travail par défaut de `R`.

Les données suivantes ont été saisies dans le fichier `table1.txt` :

```
53.5 160
74.4 172
52.6 151
88.6 163
49.2 169
```

```
> read.table("table1.txt")
```

`R` affiche le tableau de données en numérotant les lignes et les colonnes, les lignes correspondant aux individus et les colonnes aux variables. `R` affiche un message d'avertissement concernant le nom des variables.

Vous pouvez également conserver la table comme un objet pour pouvoir la réutiliser directement :

```
> tab<-read.table("table1.txt")
```

et demander l'affichage de cet objet :

```
> tab
```

ou seulement d'une colonne de cet objet :

```
> tab$V1
```

ou seulement de l'élément de la première ligne et de la première colonne :

```
> tab[1,c(1)]
```

ou

```
> tab[1,1]
```

ou les éléments des deux premières lignes et de la première colonne :

```
> tab[1:2,1]
```

ou les éléments des deux premières lignes et des deux premières colonnes :

```
> tab[1:2,1:2]
```

Pour travailler ensuite sur les variables de la table, vous pouvez leur attribuer un nom (plus simple que la syntaxe utilisée) :

```
> V1<-tab$V1
> V2<-tab$V2
```

Si vous avez spécifié le nom des variables dans la première ligne de votre fichier de données (comme dans le fichier table2.txt), vous devez l'indiquer par l'option `header=TRUE` ou `header=T` :

```
> read.table("table2.txt",header=T)
```

Par défaut, `R` lit la première ligne comme une ligne de données et nomment les colonnes sous la forme V1, V2, ...(comme pour `table1.txt`).

Par défaut, on utilise un point `(.)` pour les décimales. Mais si les décimales sont notées par une virgule dans votre fichier de données (comme dans table3.txt), il faut le spécifier par :

```
> read.table("table3.txt",dec=",")
```

Par défaut, on utilise un espace pour séparer les valeurs appartenant à différentes colonnes. Mais si les colonnes sont séparées par un point virgule dans votre fichier de données (comme dans table4.txt), il faut le spécifier par :

```
> read.table("table4.txt",sep=";")
```

Pour ouvrir un fichier de données sans avoir à indiquer son emplacement en utilisant une boîte de dialogue conviviale :

```
> read.table(file.choose())
```

Enfin la plupart des formats de fichiers sont connus par `R`.

Commençons avec les fichiers .csv. Il en existe deux types : anglo-saxon avec un "." comme séparateur décimal et une "," comme séparateur de colonne et français avec une "," comme séparateur décimal et un ";" comme séparateur de colonne. Le premier se lit avec l'instruction

```
> read.csv(file.choose())
```

et le second avec l'instruction

```
> read.csv2(file.choose())
```

L'instruction suivante permet de lire le fichier de données au format excel "table7.xls" :

```
> library(xlsReadWrite)
> (data <- read.xls(file.choose(), colNames = TRUE, sheet = 1,
type = "data.frame", from = 1, rowNames = NA))
```

L'option `colNames = TRUE` indique que la première ligne du jeu de données contient les noms des colonnes, la commande `sheet = 1, from = 1` signifie qu'il faut importer des données de la première feuille du classeur excel et que celles-ci débutent à la première ligne de la feuille. La première colonne ne contient pas les noms des lignes donc l'option `rowNames = NA` est sélectionnée ; pour lire le fichier "table8.xls" il faut changer cette option en `rowNames = T` puisque la première colonne contient les noms des lignes. Plus précisement, lorsque l'option `rowNames = NA` est sélectionnée, la première colonne du jeu de données sera utilisée comme nom de lignes à deux conditions :

a) Les colonnes du jeu de données ont été nommées.

b) Le nom de la première ligne est la chaîne de caractère vide `""`.

Nous calculons le BMI de chacun des individus du jeu de données que nous stockons dans la colonne `BMI` de l'objet `data` :

```
> data$BMI <- data$Masse/(data$Taille/100)^2
```

Si nous souhaitons mettre à jour le jeu de données data dans la feuille correspondante du classeur excel "table7.xls" :

```
> write.xls(data, file.choose(), colNames = TRUE, sheet = 1, from = 1,
rowNames = NA )
```

Si au contraire nous souhaitons créer un nouveau fichier .xls, "table9.xls", dans le

répertoire de travail courant :

```
> write.xls(data, "table9.xls", colNames = TRUE, sheet = 1, from = 1,
rowNames = NA )
```

L'option `rownames` de la fonction write.xls peut prendre deux valeurs `NA`, qui est sa valeur par défaut, ou `TRUE`. Lorsque nous utilisons `rownames = T`, le nom des lignes du jeu de données est systématiquement inclus en tant que première colonne du tableau exporté au format Excel. Si nous utilisons l'option `rownames = NA` alors les noms des lignes ne seront ajoutés au jeu de données que si le nom de la première ligne est différent de `"1"` qui est sa valeur par défaut.

Lecture des fichiers excel, excel 2007, access et access 2007 via ODBC :

```
> library(RODBC)
> connection <- odbcConnectExcel() # excel
> # connection <- odbcConnectAccess() # access
> # connection <- odbcConnectExcel2007() # excel 2007
> # connection <- odbcConnectAccess2007() # access 2007
> data <- sqlFetch(connection,"Sheet1")
> close(connection)
> odbcCloseAll()
> data
```

Dans les instructions précédentes, `"Données"` est le nom de la feuille excel du classeur dont il faut importer les données.

10 Fichiers scripts

Il est souvent plus pratique de composer le code `R` dans une fenêtre spécifique du logiciel : la fenêtre de script.
Les entrées "Nouveau script" ou "Ouvrir un script" permettent de créer un nouveau script de commandes `R` ou d'accéder à un ancien script sauvegardé lors d'une session précédente d'utilisation du logiciel.
Pour exécuter des instructions à partir de la fenêtre de script il suffit de procéder par copier-coller ou de se servir de raccourci clavier "ctrl+R".
Pour sauvegarder un script, il suffit, lorsque la fenêtre de script est active, de sélectionner l'entrée "Sauver" du menu "Fichier".

11 Un glossaire qui peut servir de résumé

Fonction	Description
`q()`	Quitte le logiciel
`?commande`	Demande la fiche de documentation de la **commande**
`s <- valeur`	Initialise la variable `s` avec la **valeur**. **Exemple** : `n<-5`
`n : m`	Crée une suite de nombres entiers de `n` à `m`.
`seq(n, m, i)`	Crée une suite de nombres de `n` à `m` en incrémentant par `i` **Exemple** : `seq(2,4,0.5)` produit la suite 2, 2.5, 3, 3.5, 4
`c(s1, s2, ..., sk)`	Crée une suite en collant les $s_1, s_2, ..., s_k$ dans l'ordre. **Exemple** : `c(2:5,7,seq(8,9,0.5))` produit `2,3,4,5,7,8,8.5,9`
`rep(s, n)`	Crée une suite contenant `n` fois `n` . **Exemple** : `rep(c(1,2,3),2)` donne `1,2,3,1,2,3`
`scan()`	Saisir "au clavier" un jeu de données numériques

`s[I]`	Crée une suite composée des éléments de la suite `s` indexées par `I`. Ici `I` peut être de la forme :
	`I` est entier. **Exemple** : `s[3]` renvoie le 3ème élément de `s`
	`I` est une suite. **Exemple** : `s[3:5]` renvoie les 3e, 4e et 5e éléments de `s`
	`I` est une condition. **Exemple** : `s[t>3]` renvoie les éléments de la suite `s` correspondants aux éléments de la suite `t` qui sont supérieurs à 3

`s[-I]`	Crée une suite composée des éléments de la suite `s` qui sont complémentaires à ceux indexés par `I`.
`mode(s)`	Affiche le mode (numérique, caractère,...) de la variable `s`
`length(s)`	Affiche le nombre d'éléments contenus dans la variable `s`
`names(s)`	Renvoie les noms des éléments de `s`. Si `s` est une table, renvoie les noms des colonnes. **Exemple** : `names(s)<-nom` renomme les éléments de `s` en utilisant les valeurs de la suite `nom`.
`sort(s)`	Trier les composantes d'un vecteur par ordre croissant
`rev(sort(s))`	Trier les composantes d'un vecteur par ordre décroissant
`file.choose`	`file.choose()` - sélectionne un fichier stocké sur l'ordinateur
`read.table`	`read.table("file")` - lit le fichier de données `file` ne contenant pas les noms des variables en première ligne. `read.table("file",header=T)` - lit le fichier de données `file` contenant les noms des variables en première ligne. **Exemple** : `t <- read.table$("table.txt",header=T)`
`read.csv`	`read.csv("file")` - lit un fichier de données `file` au format CSV
`read.xls`	`read.xls("file")` - lit un fichier de données `file` au format excel
`write.xls`	`write.xls("file")` - écrit un fichier de données `file` au format excel
`tab$col`	Renvoie la suite composée des éléments de la colonne `col` de la table `tab`. **Exemple** : `e<-amis$email` initialise la variable `e` avec les valeurs de la colonne `email` de la table `amis`.

Attention pour utiliser les fonctions `read.xls` et `write.xls`, il faut au préalable charger le package `xlsReadWrite`.

12 Exercices

Exercice 1

1. Créer le vecteur `x=(101;102;...;112)`.
2. Créer un vecteur de longueur 12 formé de 4 fois la suite de nombres (4;6;3).
3. Créer un vecteur composé de huit 4, de sept 6 et de cinq 3.

Exercice 2

1. Saisir la variable `poids` contenant les `15` valeurs suivantes :
   `28; 27.5; 27; 28; 30.5; 30; 31; 29.5; 30; 31; 31; 31.5; 32; 30; 30.5.`
2. Saisir la variable `poids1` contenant les `5` valeurs suivantes :
   `40; 39; 41; 37.5; 43.`
3. Sans refaire de saisie, créer la variable `nouveau.poids` contenant `20` valeurs (les `5` valeurs de `poids1` répétées `2` fois et les `10` dernières valeurs de `poids`).
4. Enregistrer, dans votre répertoire de travail, la variable `nouveau.poids` dans une feuille nommée "Nouveau Poids" du classeur excel "Poids.xls".

Exercice 3

1. Créer le vecteur `nom` contenant les noms de 10 personnes.
2. Créer le vecteur `age` contenant l'âge des 10 personnes précédentes (entre 20 et 60 ans). Les noms des personnes seront utilisés comme légende pour le vecteur `age`.
3. Créer le vecteur `poids` contenant le poids des 10 personnes (entre 50 et 100 kg) en utilisant à nouveau le nom des personnes comme légende pour ce vecteur.
4. Même chose pour le vecteur `taille` contenant la taille des 10 personnes.
5. Créer le vecteur `poids.lourds` contenant le poids des personnes de plus de 80 kg.
6. Créer le vecteur `taille.poids.lourds` contenant la taille des personnes de plus de 80 kg.
7. Créer le vecteur `taille.vieux.poids.lourds` contenant la taille des personnes de plus de 80 kg et âgées de plus de 30 ans. Pour répondre à cette question, vous pourrez utiliser le connecteur logique ET dont la syntaxe est donnée ci-dessus ou dans l'aide sur opérateurs logiques accessible en tapant l'instruction `?Logic`.