Initiation au langage et objets de R

Résumé

La vignette d'initiation précise l'environnement de travail (packages, répertoire courant). Elle propose de manipuler les différents types d'objets existants dans R et décrit quelques outils d'importation/exportation de données.

Organisation des tutoriels R.

- *Démarrer rapidement avec R*
- *Initiation à R*
- *Fonctions graphiques de R*
- *Programmation en R*
- *MapReduce pour le statisticien*

Les aspect statistiques sont développés dans les différents scénarios de Wikistat.

1 Environnement

Quelques compléments pour une utilisation facile concernant le chargement de librairies complémentaires et le répertoire de travail.

1.1 Librairies

La liste complète des *packages* ou librairies disponibles gratuitement est consultable sur le site du CRAN. Sous Windows, l'installation d'un package supplémentaire peut se faire via le menu

`Packages>Installer le(s) package(s)`

et en choisissant un site miroir du CRAN. On peut également télécharger l'archive `.zip` correspondant au package et utiliser ensuite

`Packages/Installer ... depuis des fichiers zip`

Sous linux, on peut installer un package R avec la fonction `install.packages()`.

1.2 Répertoire courant

Pour pouvoir récupérer des données, il est utile de connaître le répertoire de travail, c'est-à-dire le répertoire sous lequel les divers résultats seront sauvegardés par défaut. Ce dernier s'obtient à l'aide la commande :

```
getwd() # avec par exemple :
[1] "C :/Program Files/R/R-2.5.1" #(Windows)
[1] "/home/Enseignements/R" #(Linux)
```

Tandis que la commande

`setwd("C:/User/Mes documents/CoursR")` change de répertoire courant. C'est également possible à partir du menu

`Fichier > Changer le répertoire courant...`

2 Structures de données

Sous R, les éléments de base sont des objets : vecteurs, matrices, listes ..., sur lesquels sont appliquées des fonctions qui fournissent des résultats numériques et graphiques. Ces objets se différencient par leur *mode* décrivant leur contenu, et leur *classe* décrivant leur structure. Les objets atomiques sont de mode homogène et les objets récursifs (listes) sont de mode hétérogène.

Les différents modes sont null (objet vide), logical, numeric, complex, character. Les classes d'objets les plus courantes sont : vector, matrix, array, factor, data.frame, list.

On peut avoir des vecteurs, matrices, tableaux, ... de mode null (objet vide), logical (TRUE, FALSE, NA), numeric, complex, character tandis que les listes et les data frame peuvent être composés d'éléments hétérogènes.

Une confusion entre classes d'objets dans l'appel d'une fonction est la source d'erreur la plus fréquente.

2.1 Opérations sur les scalaires

Entrer les commandes en identifiant les différents types de données

```
2+2
exp(10)
```

```
a = log(2)
b <- cos(10) #  "<-" est équivalent à "="
a+b
a
b
2==3
b = 2<3
ls()  # variables de l'environnement de travail
rm(a) # effacer une ou plusieurs variables
a
a="texte"
```

2.2 Type caractère

Manipulation de Chaînes de caractères.

```
c="ABCdef";nchar(c);c
is.character(c)
substr(c,1,3)
# changer la casse
tolower(c)
toupper(c)
# coller
paste("alpha",c,sep="-")
```

2.3 Type booléen et opérateurs logiques

Les variables booléennes prennent les valeurs `TRUE` ou `FALSE` ; les opérateurs de comparaison $<, ><=, >=, !=, ==$ retournent ces valeurs tandis que `&, |, !` sont les opérateurs logiques "et", "ou", "non".

```
a = 3 ; b = 6
a<=b
a!=b
(b-3==a) & (b>=a)
(b  ==a) | (b>=a)
```

2.4 Les vecteurs (`vector`)

Un vecteur regroupe des éléments de même mode. La création d'un vecteur peut se faire par la commande `c(e1,e2,...)`. On peut également générer une séquence avec la commande `seq(a,b,t)` débutant par `a` inférieure ou égale à `b` et de pas `t` ; `rep(x,n)` est un vecteur répétant n fois l'élément `x`.

```
d = c(2,3,5,8,4,6); d
is.vector(d)
c(2,5,"texte")
```

Séquences et répétitions.

```
1:10
seq(from=1,to=20,by=2)
seq(1,20,by=5)
seq(1,20,length=5)
rep(5,times=10)
rep(c(1,2),3)
rep(c(1,2),each=3)
e = rep(1,10)
```

Extraction dans un vecteur par `[]` et valeurs manquantes.

```
d[2];d[2:3];d[-3]
# attention aux indices :
d[-1:2];d[-(1:2)]
# NA (Not Available) signale une donnée manquante
d[3]=NA;d;summary(d)
is.na(d);help(NA)
# fonctions "any" et "all"
any(is.na(d));all(is.na(d))
```

Labels et opérations.

```
f = c(a=12,b=26,c=32,d=41);f
names(f);f["a"]
names(f)=c("a1","a2","a3","a4")
f>30;f[f>30]  # noter les différences
which(f>30)
```

```
f[2] = 22;f+100;f+d  # un problème ?
cos(f);length(f);sum(f)
t(f) # transposition
e=rep(2,4); 2*e; 2+e
e+f ; e*f # opérations terme à terme
t(f)%*%e # produit scalaire
a<-c(3,-1,5,2,-7,3,9)
abs(a);sort(a);order(a)
```

2.5 Facteurs

Un facteur est un vecteur avec une liste prédéfinie de valeurs, les niveaux (*levels*). Cela correspond typiquement à une variable qualitative nominale.

```
vect=c("a","b","c","b","b","a")
vect
vect.f=as.factor(vect)
vect.f
as.integer(vect.f)
as.character(vect.f)
```

2.6 Les matrices (`matrix`)

Comme les vecteurs, les matrices sont de mode quelconque mais ne contiennent que des éléments de même nature. Pour créer une matrice, on utilise la commande `matrix(vec,nrow=n,ncol=p)` où `vec` est le vecteur contenant les éléments de la matrice de taille n par p, qui seront rangés en colonne sauf si l'option `byrow=T` est utilisée.

```
A = matrix(1:15,ncol=5);A
B = matrix(1:15,nc=5,byrow=T)
B2=B;B2[1,1]="toto";B2
cbind(A,B);rbind(A,B) # concaténations
A[1,3];A[,2];B[2,] # composants
A[1:3,2:4]
```

Opérations sur les matrices ;

```
g = seq(0,1,length=20)
C = matrix(g,nrow=4)
dim(C)
C[C[,1]>0.1,]  # *
# ranunif : tirage aléatoire uniforme
D = matrix(runif(16),ncol=4)
D>0.5
D[D[,1]>0.5,2]   # **
A+B;A*B  #  opérations terme à terme
cos(A); cos(A[1:2,1:2])
# inversion
solve(A);solve(A[1:2,1:2])
# produit matriciel
t(A) %*% B
A[1:2,1:2] %*% B[1:2,1:3]
t(B); diag(A)
apply(A,2,sum)   # ***
apply(D,1,max)
# Eléments propres
s=eigen(A[1:2,2:3])
 s$values
 s$vectors
```

Questions

1. Que font `rbind` et `cbind` ?
2. Décortiquer la ligne marquée * et décrire ce qu'elle fait.
3. Même chose avec **.
4. Que renvoie la ligne *** ? Noter l'importance de cette fonction pour éviter des boucles.

Réponses

1. `rbind` et `cbind` collent deux vecteurs ou matrices respectivement en ligne ou en colonne.
2. `C[C[,1]>0.1,]` peut se décomposer ainsi :

   `C[,1]` extrait la première colonne de la matrice `C`

   `C[,1]>0.1` renvoie un vecteur logique de longueur le nombre de

lignes de C contenant `TRUE` si la valeur est supérieure à 0.1 et `FALSE` sinon.
`C[C[,1]>0.1,]` extrait de la matrice C les lignes où les éléments sur la première colonne sont supérieurs à 0.1 et toutes les colonnes (rien après la virgule).

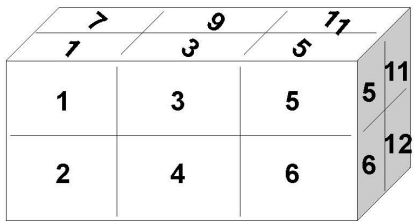
3. la ligne ** extrait de la colonne 2 de la matrice D, les lignes où l'élément sur la première colonne est supérieur à 0.5.
4. la ligne *** renvoie un vecteur de longueur 5 (le nombre de colonnes de A) dont chaque élément est la somme des éléments d'une colonne de A.

2.7 Les tableaux (`array`)

Les tableaux sont des matrices de dimensions supérieures à 2. On peut les générer à partir de la commande `array(vec,c(n,p,q,...))` où `vec` est le vecteur contenant les éléments du tableau et l'argument `c(n,p,q,...)` désigne les dimensions du tableau : n lignes, p colonnes, q matrices, ...

```
array(c(1:8, rep(1,8),seq(0,1,len=8)), dim = c(2,4,3))
E = .Last.value
E[, , 1]
dim(E);length(E)
nrow(E);ncol(E)
E+10
H=array(1:12,c(2,3,2))
apply(H,1,mean)
apply(H,2,mean)
apply(H,3,mean)
```

Une représentation de l'`array` H :



Questions

1. Expliquer les résulats des 3 appels à la fonction `apply()`.
2. Créer un `array` à 4 dimensions et calculer la somme des éléments dans toutes les dimensions.

Réponses

1. Dans le premier cas, on calcule la moyenne de tous les éléments ligne par ligne. Les éléments de la ligne 1 sont tous les élements de la tranche supérieure horizontale du parallélépipède H (1,3,5,7,9,11) ; de la tranche inférieure pour la ligne 2 (2,4,6,8,10,12). Dans le deuxième cas, le calcul est effectué colonne par colonne, le vecteur résultat est donc de longueur 3 ; il contient la moyenne des éléments des 3 tranches verticales (gauche - [1,2,7,8], centre - [3,4,9,10] et droite - [5,6,11,12]). Dans le troisième cas, le calcul de moyenne est fait sur les 2 tranches verticales "avant" (1,2,3,4,5,6) et "arrière" (7,8,9,10,11,12).
2. `H2=array(1 :24,c(2,3,2,2))` crée un `array` à 4 dimensions équivalent dans cet exemple à 2 tableaux H tels que représentés ci-dessus.
   - `apply(H,1,sum)` [1] 144 156
   - `apply(H,2,sum)` [1] 84 100 116
   - `apply(H,3,sum)` [1] 114 186
   - `apply(H,4,sum)` [1] 78 222

2.8 Les listes (`list`)

Une liste est une collection ordonnée d'objets qui peuvent être de classes différentes. Les listes sont en particulier utilisées par certaines fonctions (cf. tutoriel "Programmation") pour renvoyer des résultats complexes sous forme d'un seul objet. On utilise la fonction `list(nom1=el1,nom2=el2,...)` pour générer une liste. On peut accéder à chaque élément de la liste à l'aide de son index entre double crochets `[[...]]`, ou par son nom précédé du signe `$`.

```
x = list("toto",1:8);x
x[[1]];x[[1]]+1;x[[2]]+10 # *
y = list(matrice=D,vecteur=f,texte="toto",scalaire=8)
names(y);y[[1]]
y$matrice;y$vec
y[c("texte","scal")] # **
```

```
y[c("texte","scalaire")]
length(y)
length(y$vecteur)
cos(y$scalaire)+y[[2]][1]
summary(y)
```

Questions

1. Quel est le problème avec la ligne * ?
2. Et avec ** ?

Réponses

1. C'est la 2ème commande de la ligne qui renvoie une erreur. On cherche à ajouter 1 à un élément qui n'est pas numérique.
2. Aucun composant de la liste ne s'appelle `scal` ("lettre à lettre").

2.9 Tableau de données (`data.frame`)

Un *data frame* est analogue à une matrice dont les colonnes peuvent être hétérogènes. Un tableau de données est un ensemble de vecteurs rangés colonne par colonne, chaque colonne correspondant à une variable, chaque ligne à un individu. En particulier, lors d'études statistiques, les données à étudier sont souvent représentées par un *data frame* sous R. Pour créer un tableau de données, on peut regrouper des variables de même longueur à l'aide de la commande `data.frame(nom1=var1,nom2=var2,...)`. On peut aussi transformer une matrice en un tableau de données en utilisant la commande `as.data.frame(mat)`.

```
taille = runif(12,150,180)
masse = runif(12,50,90)
sexe = rep(c("M","F","F","M"),3)
H = data.frame(taille,masse,sexe)
H;summary(H)
# analogies entre data.frame, list et matrix
H[1,];H$taille;H$sexe
is.data.frame(H)
is.matrix(H)
MH = as.matrix(H)
summary(MH)
```

```
as.list(H)
rm(taille);taille # (1)
H$taille
attach(H);taille  # (2)
search()          # (3)
detach();taille   # (4)
```

Questions

1. Tester la fonction `summary` sur d'autres types d'objets.
2. Quel est l'effet de la conversion "forcée" du data.frame en matrice opérée par la fonction `as.matrix()` ?
3. Commenter l'enchaînement des lignes 1 à 4. Quel est l'effet de la fonction `attach` ? de la fonction `search` ? de la fonction `detach` ?
4. Extraire la masse des individus dont la taille est supérieure à 160.
5. Extraire la masse et le sexe de ces mêmes individus.
6. Extraire la taille des individus de sexe masculin dont la masse est inférieure à 70. C'est possible en une seule ligne (voir l'opérateur &, `help("&")`).

Réponses

1. Fonction `summary`

```
        vec=c(2,5,3,6,5,4,1,8);summary(vec)
mat=matrix(1:20,nc=4,nr=5)
summary(mat);summary(c(mat))
...
```

2. La conversion en matrice implique que tous les éléments sont désormais du mode `character`. La fonction `summary()` ne calcule plus des indicateurs numériques pour les colonnes `taille` et `masse`.
3. Enchaînement des lignes
   (1) Supprime l'objet `taille` dans l'espace de travail courant ; l'objet `taille` n'est plus reconnu.
   (2) Attache le data.frame `H` ; les composants de `H` deviennent accessibles directement.
   (3) `search()` permet de lister les environnements liés à l'espace de travail courant.

(4) Détache l'environnement en position 2 dans la liste de `search()`.

4. `H[H$taille>160,2]`
5. `H[H$taille>160,2:3]`
6. `H[H$masse<70 & H$sexe=="M",1]`

3 Entrée / Sortie

3.1 Importation d'un jeu de données

`Tab1 = read.table("Tableau.dat")` lit le fichier nommé `Tableau.dat` pour créer le *data frame* `Tab1` en supposant que le fichier est bien dans le répertoire courant. Sinon il faut préciser le chemin.

`help(read.table)` fournit la liste des nombreuses options de cette fonction très utile. Les fonctions `read.csv` et `read.csv2` en sont des cas particuliers, c'est-à-dire avec des options spécifiques adaptées aux fichiers lus / écrits par des tableurs en format `.csv`.

Utiliser un éditeur de texte pour créer les quatre fichiers ci-dessous en respectant scrupuleusement la ponctuation.

5,2.5,3.8 8,3.2,3.4 12,4.6,5	5 2.5 3.8 8 3.2 3.4 12 4.6 5	X1 ;X2 ;X3 5 ;2.5 ;3.8 8 ;3.2 ;3.4 12 ;4.6 ;5	5 ;2,5 ;3,8 8 ;3,2 ;3,4 12 ;4,6 ;5
fic1.csv	fic2.txt	fic3.txt	fic4.txt

Comparer les modes de lecture de fichiers en fonction du type, des séparateurs et de la présence de la première ligne de noms des variables.

```
fic1=read.table("fic1.csv",sep=",")
fic1
fic1b=read.csv("fic1.csv")
fic1b
fic1b=read.csv("fic1.csv",header=FALSE)
fic1b
```

Questions

1. Importer les fichiers `fic2.txt`, `fic3.txt` et `fic4.txt`.

Réponses

1. Importation des fichiers :

```
fic2 = read.table("fic2.txt")
fic3 = read.table("fic3.txt",header=T,sep=";")
fic4 = read.table("fic4.txt",sep=";",dec=",")
```

3.2 Exportation d'objets R

Ecriture dans un fichier et traces des exécutions.

```
A=seq(1,10,l=50)
write.table(A,"A.txt")
sink("A2.txt")  # début
A;summary(A)
sink()  # arrêt et fermeture du fichier
summary(A)
```

3.3 Liens avec un tableur

L'échange de fichiers avec un tableur se fait simplement à travers le format `.csv` comme précédemment en faisant attention aux versions américaines dans lesquelles les séparateurs de deux valeurs sont des "," et les marques décimales des "." tandis que dans les version françaises, les séparateurs de valeurs sont des " ;" et les marques décimales des ",".

Passer d'un tableur à R

- Dans Excel (ou *open office*) :
  1. Ouvrir le fichier de données,
  2. Enregistrer le fichier au format `csv` (Comma Separated Value) en le spécifiant dans la liste déroulante (Type de fichier).
  3. Quitter Excel.
- Dans R : L'importation du fichier de données s'effectue avec la fonction `read.table()`. L'aide en ligne (`help(read.table)`) fournit tous les détails sur l'utilisation de cette fonction. On peut en général assurer l'importation avec les 4 arguments : `file, header, sep, dec` :

`file` nom du fichier qui contient les données avec le chemin pour y accéder,
`header` (TRUE ou FALSE) : permet de préciser si la première ligne contient le titre des colonnes
`sep` permet de préciser le caractère qui délimite les colonnes,
`dec` permet de préciser le séparateur décimal

- N.B. Le chemin pour accéder au fichier est indiqué par des "/" et non par des "\".

Retour dans Excel de données traitées avec R

La fonction `write.table` permet d'envoyer un jeu de données R dans un fichier texte. Exemple : On souhaite exporter le jeu de données `resul.dat` dans un fichier afin de poursuivre son analyse avec Excel. Pour cela, on précise par exemple que le séparateur décimal est la virgule et le séparateur le point-virgule.

```
write.table(resul.dat,"resul.csv",sep=";",dec=",")
```

Il existe des packages R permettant d'assurer plus directement ces transferts mais la procédure décrite ici est en général recommandée. Pour plus de détails, se référer au document **R Data Import/Export** disponible sur le CRAN.