Start-R pour utilisateur de statistiques

Résumé

Cette vignette introduit le logiciel libre R, son environnement et décrit les premières commandes nécessaires au démarrage d'une utilisation de méthodes statistiques avec ce logiciel. Les notions d'estimation, de variabilité et de loi d'un estimateur sont illustrées par des simulations. Les autres tutoriels abordent des fonctionnalités plus complexes.

- *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 et d'autres ressources sont disponbibles.

1 Introduction

1.1 Pourquoi R ?

Le logiciel R sous licence GNU est facile à installer à partir de la page du CRAN ou d'un site miroir ; ils contiennent toutes les ressources nécessaires à l'utilisateur de R, débutant ou expérimenté : fichiers d'installation, mises à jour, librairies, FAQ, newsletter, documentation... Il est le logiciel le plus utilisé de la communauté statistique académique et aussi de plus en plus dans les services R&D des entreprises industrielles en concurrence avec les logiciels commerciaux. Son utilisation nécessite un apprentissage à travers des tutoriels comme par exemple ceux listés dans le résumé mais il est facile de démarrer à partir de quelques notions de base sur son utilisation ; c'est l'objectif de ce start-R.

Dans sa structure, R est un langage de programmation d'une syntaxe voisine à celle du langage C et capable de manipuler des objets complexes sous forme de matrice, scalaire, vecteur, liste, facteur et aussi *data frame*. Proposant donc une programmation matricielle, il offre des fonctionnalités analogues à Matlab et dispose également d'une très riche librairie de quasiment toutes les procédures et méthodes statistiques de la littérature. Plus précisément, toutes les recherches récentes sont d'abord développées et diffusées à l'aide de ce logiciel par la communauté scientifique.

1.2 Utilisation

Il existe de nombreuses librairies (cf. `Rcmdr`) d'interface graphique par menu mais celles-ci sont contraignantes, trop limitées dans les choix et options, elles ne peuvent éviter une utilisation par lignes de commandes ; autant s'y mettre tout de suite, c'est le choix fait ici.

Il existe également un environnement de programmation ou IDE : RStudio relativement efficace ; à l'utilisateur de faire ses choix. Nous nous limitons ici aux choix de l'interface graphique par défaut `Rgui` (*R graphical user interface*) et à celui de l'usage de RStudio si ce dernier est installé.

1.3 Ouvrir, utiliser, fermer R

- Sous Windows Cliquer sur l'icone (R ou RStudio) ou lancer le programme à partir du menu `Démarrer`. Lors de la première exécution, il est utile de préciser le répertoire de travail (menu `Fichier` de R). Il est aussi possible de lancer R à partir de la session précédente en cliquant sur le fichier de sauvegarde de suffixe `.RData`.
- Sous Linux ; à partir d'une fenêtre de commande, se placer dans le bon répertoire pour taper la commande `R` ou `rstudio`.
- Dans les deux cas, le logiciel répond avec une invite de commande : > ou ouvre (RStudio) sous la forme d'une fenêtre avec 3 sous-fenêtres. En ouvrir une 4ème avec le menu : `File > New File > Sript R`. Ces 4 sous-fenêtres sont comme sos Matlab :
  - un éditeur de texte pour gérer le script à exécuter, sauvegarder pour constituer l'annexe du rapport,
  - une liste des objets (environnement) créés dans R ou l'historique des commandes,
  - la console d'exécution classique de R,
  - une fenêtre avec menu pour visualiser le répertoire, les graphiques, la liste des *packages* installés pour les charger, mettre à jour, en installer d'autres, l'aide en ligne.

`help(plot)` ou `plot?` lance la fonction `help` qui ouvre l'aide en ligne de, par exemple, la fonction `plot`. `quit()` fait quitter R après une question proposant de sauvegarder l'environnement de travail. Y répondre favorablement crée le fichier `.RData` dans le répertoire courant ainsi qu'un fichier contenant l'historique des commandes.

1.4 Les tutoriels

Les tutoriels sont organisés de la façon suivante :

- une succession de commandes à saisir est donnée. Chaque commande est mise en évidence par une graphie particulière `ls()`.
- *Attention*, lorsque R identifie une commande incomplète (parenthèse ouverte et pas fermée par exemple), le symbole "+" apparaît automatiquement sur la ligne suivante.
- Deux commandes indépendantes peuvent être tapées sur la même ligne séparées par un " ;". Elles seront traitées séquentiellement.
- Quelques remarques ou commentaires viennent parfois mettre en évidence certains points particuliers de syntaxe ou autre.
- Des questions permettent de tester l'acquisition des fonctions étudiées.
- Une réponse possible à chaque question est fournie à la suite.

2 Lignes de commande

2.1 Avec Rgui

R fonctionne comme une calculette. Vous pouvez taper les lignes ci-dessous après l'invite de commande (>). Il est recommandé d'ouvrir en plus la fenêtre d'un éditeur pour mémoriser les commandes et copier / coller ces lignes dans la fenêtre R avant de les archiver dans un fichier pour constituer l'annexe d'un rapport.

2.2 Avec RStudio

Cette fenêtre d'édition est incluse comme indiqué précédemment par l'ouverture d'un nouveau fichier de script R.

Le menu : `Session > Set Working Directory` permet de sélectionner le répertoire par défaut contenant par exemple le fichier de données à lire et où seront sauvegardés script, environnement et historique des commandes d'une session.

2.2.1 Premières commandes

Entrer les lignes ci-dessous dans la fenêtre de l'éditeur de texte. Dans Rgui, les copier / coller dans la console pour les exécuter. Dans RStudio, positionner le curseur sur une ligne et cliquer sur le bouton `Run] pour l'exécuter ou sélectionner préalablement plusieurs commandes.`

```
# Ceci est un commentaire
2+2
sqrt(2)
a = exp(2)   # création d'une variable scalaire
b = a + pi
b            # affichage de la valeur
# liste des variables
ls()
```

3 Types de variables

`La principale difficulté dans l'utilisation de R est de bien identifier les types d'objets manipulés.`

```
# Vecteur
x = 1:10  # définition d'une séquence
x
y = 2*x + 3
y[5] ; y[1:3] ; y[-3]  # composants d'un vecteur

# Matrice
A = matrix(1:15,ncol=5); A
B = matrix(1:15,nc=5,byrow=TRUE) ; B
A[1,3] ; A[,2] ; A[2,] ; A[1:3,1:3] # composants

# Liste
x=list(mat=A, texte="testliste",vec=y)
```

```
x[[2]] ; x$vec # composants

# Base de donnée ou data frame
# Tableau contenant des vecteurs de types
# éventuellement différents
taille = c(147, 132, 156, 167, 156, 140)
poids = c( 50, 46, 47, 62, 58, 45)
sexe = c("M","F","F","M","M","F")
H = data.frame(taille,poids,sexe)
H
summary(H)
plot(H$poids,H$taille)
```

D'autres fonctionnalités de R sont vues dans les différents scénarios proposées sur le site Wikistat.

4 Variables et nombres aléatoires

Tout en complétant la connaissance de R, cette section propose d'illustrer, par des simulations, les propriétés des estimateurs élémentaires (moyenne, écart-type, histogramme).

4.1 Estimation

Générer n valeurs aléatoires d'une variable Y selon une loi normale de moyenne 80 et d'écart-type 5. Décrire sommairement cette série de valeurs. Associer les quantités calculées avec leur traduction en anglais *mean, median, standard error, standard déviation, standard error mean*.

```
n=10
Y=rnorm(n,80,5)  # génération
mean(Y)      # moyenne
sd(Y)        # écart-type
sd(Y)/sqrt(length(Y)) # écart-type de la moyenne
summary(Y)  # quartiles et moyenne
boxplot(Y)  # diagramme boîte
# histogramme de la densité
hist(Y, probability=T, col="blue")
# estimation par la méthode du noyau
lines(density(Y), col="red", lwd=2)
# tracer la loi théorique
x=1:100
curve(dnorm(x,mean=80,sd=5),add=TRUE,
   col="green",lwd=2)
```

4.2 Loi des grands nombres

Une moyenne et un écart-type sont la réalisation d'une variable aléatoire appelée "estimateur"; ce sont des estimations. Refaire les calculs et graphiques en posant $n = 10, n = 1000, n = 10000$; comparer les résultats obtenus, notamment les estimations des indicateurs par rapport aux valeurs théoriques. Etudier leur comportement en fonction de la taille n de l'échantillon.

```
n=100
# matrice de nombres aléatoires de
# 10 colonnes et n lignes
Y=matrix(rnorm(n*10,80,5),n,10)
# moyenne de chaque colonne
apply(Y,2,mean)
mean(apply(Y,2,mean)) # moyenne des moyennes
# écart-type de chaque colonne
apply(Y,2,sd)
mean(apply(Y,2,sd))    # moyenne des écarts-types
```

Faire varier $n = 10, 100, 1000$ et comparer les résultats obtenus.

4.3 Théorème de la limite centrale

La simulation proposée illustre le résultat fondamental du théorème de la limite centrale : une somme de variables aléatoires indépendantes et de même loi converge vers une variable aléatoire de loi gaussienne. Le programme ci-dessous exécute les opérations suivantes :

- initialisation par des 0 d'un vecteur de taille $n = 1000$,
- chaque valeur de ce vecteur est une variable aléatoire X obtenue par la somme de N variables suivant une loi uniforme sur l'intervalle $[0, 1]$,
- estimation de la densité de X
- comparaison avec la loi théorique limite qui est la loi gaussienne de moyenne $N/2$ et de variance $N/12$.

```
n=1000
N=12
X=rep(0,n)
# n itérations
for (i in 1 : n) X[i]=sum(runif(N))
# histogramme
hist(X, col="blue", probability=T)
# estimation par méthode su noyau
lines(density(X), col="red", lwd=2)
x=X
sigma2=N/12
curve(dnorm(x,mean=N/2,sd=sqrt(sigma2)),
   add=T, col="green", lwd=2)
```

Faire varier $N = 4, 8, 12, 20$. Remarquer que la convergence est très rapide. Ceci "justifie" la pratique qui revient à considérer que la loi d'un estimateur est gaussienne lorsque n est "suffisamment" grand avec $n > 30$.

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).

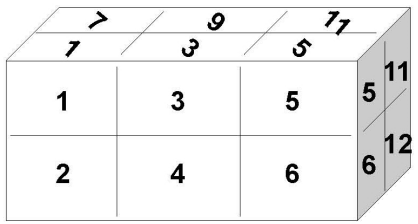
3. la ligne ** extrait de la colonne 2 de la matrice D, les lignes où l'élément sur la première colonne est supérieur à 0.5.
4. la ligne *** renvoie un vecteur de longueur 5 (le nombre de colonnes de A) dont chaque élément est la somme des éléments d'une colonne de A.

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.

Fonctions graphiques de R

Résumé

Un rapide aperçu des très riches fonctions graphiques de R pour gérer fenêtres et fichiers, tracer des camemberts, diagrammes en colonnes, mosaïc plot, histogrammes, diagrammes boîtes, images, contours, vues en 3D, ajouter des composantes graphiques.

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 Introduction

Plus un problème et les données afférentes sont complexes, plus est important la pertinence des graphes qui permettront d'en appréhender la structure et visualiser l'information utile. Aussi, quelque soit le logiciel utilisé, commercial comme SAS ou libre comme R, les versions basées sur une interface *amicale* : SAS/Insight ou Rcmdr, sont évidemment séduisantes, car faciles à utiliser par des menus, mais drastiquement limitées par les choix des options nécessairement très contraints parmi des possibilités considérables.

`demo(graphics)` donnera un aperçu de la palette des possibilités tandis qu'une visite à l'un de ces sites : gallerie 1, gallerie 2, gallerie 3, ouvre de vastes perspectives sur les possibilités offertes.

C'est la principale raison, qui motive de cheminer par l'apprentissage du lange de commande R plutôt que par celui d'une interface graphique aussi amicale soit-elle. Bien entendu ce tutoriel ne présente que quelques facettes des graphiques les plus utilisés tandis que beaucoup de ressources et sites, comme ceux ci-dessus, rendent accessibles des exemples complexes et ciblés sur des besoins particuliers.

Attention cependant aux excès d'esthétisme, un *beau* graphique n'est pas nécessairement un graphique *pertinent* ; éviter par exemple les abus d'effets de perspective sans signification statistique.

2 Commandes de base

2.1 Aide en ligne

Comme pour tout utilisation avancée d'une fonction de R, le recours à l'aide en ligne est incontournable. Apprendre à y naviguer et comprendre sa logique font partie de l'apprentissage.

`help(plot)` offre un premier accès alors que

`help(par)` embrouille vite les cartes.

2.2 Gestion de la fenêtre graphique

Toute commande graphique ouvre une fenêtre adaptée mais

`x11()` ouvre une autre fenêtre pour éviter d'écraser le graphe précédent,

`dev.off()` ferme correctement la dernière fenêtre,

`split.screen(c(1,2))` partage la fenêtre en 2 de même que l'option `mfrow` de la commande `par`.

2.3 Exportation de fichiers graphiques

Voici quelques possibilités pour intégrer un graphique dans un traitement de texte ou générer un fichier contenant le graphique avant intégration ou archivage.

- Sous Windows, clic droit dans la fenêtre pour copier le graphique dans le presse papier avant de le coller dans un texte,
- L'obtention du format standard `jpeg` nécessite d'enchaîner les commandes :

```
jpeg("fichier.jpeg") # ou bmp(), png(), pdf()
# le graphique n'apparaît pas à l'écran :
plot(1:100); text(20,80,"abcdef")
dev.off() # fermeture indispensable du fichier
```

qui est alors créé dans le répertoire courant.
- Utiliser le menu `Fichier`, rubrique `Sauver sous` de la fenêtre graphique.

3 Description élémentaire

3.1 Variables qualitatives

Fonctions `pie, barplot, mosaicplot`.

```
vec=c(12,10,7,13,26,16,4,12)
pie(vec);pie(vec,clockwise=T)
names(vec)=LETTERS[1:8]
# de très nombreuses options sont disponibles
pie(vec)
barplot(vec)
# pour découper la fenêtre en 1 ligne 2 colonnes
par(mfrow=c(1,2))
pie(vec)
barplot(vec)
par(mfrow=c(1,1))
mosaicplot(Titanic, color = T)
colors()
```

Questions

1. Quelle est la différence entre `par(mfrow=c(2,2))` et `par(mfcol=c(2,2))` ?
2. Consulter l'aide en ligne pour tester certains arguments optionnels des fonctions `pie()` et `barplot()`.
3. Commenter les commandes suivantes (extraites de l'aide en ligne de `pie()`):

```
n=200
pie(rep(1,n), labels="", col=rainbow(n), border=NA)
```

Réponses

1. Les 2 commandes découpent la fenêtre graphique en 4 cellules. La différence entre les 2 commandes est illustrée par le tableau ci-dessous (`par(mfrow=c(2,2))` - `par(mfcol=c(2,2))`) :

1 - 1	2 - 3
3 - 2	4 - 4

   Avec `par(mfrow=c(2,2))`, les 4 graphiques à venir sont intégrés ligne par ligne ; avec `par(mfcol=c(2,2))`, ils le sont colonne par colonne.
2. Par exemple, pour modifier les couleurs du camembert :

```
pie(vect,col=rgb(0,0,seq(0,1,l=8)))
pie(vect,col=rainbow(8))
...
```

3. Commentaire de la ligne de commande :
   - `n=200` # La valeur 200 est affectée à `n`.
   - `rep(1,n)` # Crée le vecteur de taille `n` ne contenant que des 1.
   - `pie(rep(1,n)` # Produit un camembert pour `n` variables d'effectifs tous égaux à 1. Les secteurs sont donc de même angle.
   - `rainbow(n)` # Renvoie un vecteur de longueur `n` contenant un dégradé des couleurs de l'arc-en-ciel.
   - `border=NA` # Évite de tracer les bords des secteurs.
   - `labels=""` # Aucune étiquette n'identifie les secteurs.

3.2 Variables quantitatives

Commandes `boxplot, hist, plot, pairs`.

```
x=rnorm(50)
boxplot(x)
hist(x)
stripchart(x)
```

Question

1. Représenter dans la même fenêtre graphique, le "stripchart", le diagramme-boîte horizontal et l'histogramme correspondant l'un sous l'autre.

Réponse

```
par(mfrow=c(3,1))
stripchart(x)
boxplot(x,horizontal=T)
hist(x)
```

Croisement de variables.

```
data(iris)
pairs(iris[,1:4])
par(mfrow=c(2,2))
plot(iris[,1],iris[,2],xlab="Sepal Length",
   ylab="Sepal Width",main="Iris data",
   col="red",type="l")
points(iris[,1],iris[,2],col="green",pch=21)
boxplot(iris[,1:4])
hist(iris[,1],xlab="Sepal Length",
    main="Histogramme")
```

Ajouter des éléments graphiques.

```
x=seq(-10,10,l=50)
plot(x,sin(x))
plot(x,sin(x),type="l")
abline(v=0,col="blue",lwd=5,lty=3)
abline(h=sin(0.7),col=3)
text(-5,-0.5,"texte",font=3)
```

Gestion des libellés.

```
par(mfrow=c(1,2))
plot(x,sin(x),type="l",col=1,main="sinus")
plot(x,cos(x),type="b",col=3,xlab="Abscisses")
```

3.3 Vers la 3D

Commandes `image, persp, contour`.

```
M=matrix(1:100,nc=10)
# images, nappes et contours
image(M)
x = seq(-10, 10, length= 30);y=x
f = function(x,y){r=sqrt(x^2+y^2); 10 * sin(r)/r}
z = outer(x, y, f)
z[is.na(z)] = 1
persp(x,y,z)
persp(x, y, z, theta=30, phi=30,expand = 0.5,
   col="lightblue")
image(x,y,z)
contour(x,y,z)
filled.contour(x,y,z)
image(x,y,z)
contour(x,y,z,add=T)
```

Pour mieux comprendre la fonction `outer()`, tester :

```
x=y=1:5
z=outer(x,y,"+");z
```

Programmation en langage R

Résumé

Un aperçu de la syntaxe du langage S mis en œuvre dans R : fonctions, instructions de contrôle et d'itérations, fonction `apply`.
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 introduction

R est la version *GNU* du langage S conçu initialement aux Bell labs par John Chambers à partir de 1975 dans une syntaxe très proche du langage C. En septembre 2013, l'index TIOBE le classe en 18ème position loin derrière le C (1er) ou Java (2ème) mais devant MATLAB (19) ou SAS (21).

2 Structure de contrôle

Il est important d'intégrer que R, comme Matlab, est un langage interprété donc lent, voire très lent, losqu'il s'agit d'exécuter des boucles. Celles-ci doivent être éviter dès qu'une syntaxe, impliquant des calculs matriciels ou les commandes de type `apply`, peut se substituer.

2.1 Structures conditionnelles

`if(condition){instructions}` est la syntaxe permettant de calculer les instructions uniquement si la condition est vraie.

`if(condition){ A }else{ B }` calcule les instructions A si la condition est vraie et les instructions B sinon. Dans l'exemple suivant, les deux commandes sont équivalentes :

```
if (x>0) y=x*log(x) else y=0
y=ifelse(x>0,x*log(x),0)
```

2.2 Structures itératives

Ces commandes définissent des boucles pour exécuter plusieurs fois une instruction ou un bloc d'instructions. Les trois types de boucle sont :

```
for var in seq) {commandes}
```

```
while (condition) {commandes}
```

```
repeat {commandes; if (condition) break }
```

Dans une boucle `for`, le nombre d'itérations est fixe alors qu'il peut être infini pour les boucles `while` et `repeat` ! La condition est évaluée avant toute exécution dans `while` alors que `repeat` exécute au moins une fois les commandes.

```
for (i in 1:10) print(i)
y=z=0;
for (i in 1:10) {
   x=runif(1)
   if (x>0.5) y=y+1
   else z=z+1 }
y;z
for (i in c(2,4,5,8)) print(i)
x = rnorm(100)
y = ifelse(x>0, 1, -1) # condition
y;i=0
while (i<10){
   print(i)
   i=i+1}
```

Questions

1. Que pensez-vous de :
   `for (i in 1:length(b)) a[i]=cos(b[i])`
2. Obtenir l'équivalent de y et z dans la deuxième boucle `for` sans boucle.

3. Dans l'enchaînement de commandes ci-dessous, supprimer d'abord la boucle `for` sur `j` puis les 2 boucles.

```
M=matrix(1:20,nr=5,nc=4)
res=rep(0,5)
for (i in 1:5){
   tmp=0
   for (j in 1:4) {tmp = tmp + M[i,j]}
res[i]=tmp}
```

Réponses

1. Cette boucle est inutile. Il suffit de saisir `a=cos(b)`. L'élément de base de R est la matrice dont le vecteur est un cas particulier.
2. Une solution consiste à sommer les éléments `TRUE` d'un vecteur logique `x=runif(10);y=sum(x>0.5);z=10-y`
3. Suppression de boucles
   - Boucle `for` sur `j` : `for (i in 1:5) res[i]=sum(M[i,])`
   - Les 2 boucles : `res=apply(M,1,sum)`

3 Fonctions

3.1 Principes

Il est possible sous R de construire ses propres fonctions. Il est conseillé d'écrire sa fonction dans un fichier `nomfonction.R`.

`source("nomfonction.R")` a pour effet de charger la fonction dans l'environnempent de travail. Il est aussi possible de définir directement la fonction par la syntaxe suivante :

```
nomfonction=function(arg1[=exp1],arg2[=exp2],...)
    {
    bloc d'instructions
    sortie = ...
    return(sortie)
    }
```

Les accolades signalent le début et la fin du code source de la fonction, les crochets indiquent le caractère facultatif des valeurs par défaut des arguments. L'objet sortie contient le ou les résultats retournés par la fonction, on peut en particulier utiliser une liste pour retourner plusieurs résultats.

3.2 Exemples

Création d'une fonction élémentaire.

```
MaFonction=function(x){x+2}
ls()
MaFonction
MaFonction(3)
x = MaFonction(4);x
```

Gestion des paramètres avec une valeur par défaut.

```
Fonction2=function(a,b=7){a+b}
Fonction2(2,b=3)
Fonction2(5)
```

Résultats multiples dans un objet de type liste.

```
Calcule=function(r){
    p=2*pi*r;s=pi*r*r;
    list(rayon=r,perimetre=p,
    surface=s)}
resultat=Calcule(3)
resultat$ray
2*pi*resultat$r==resultat$perim
resultat$rsurface
```

Questions

1. le recours à un objet de type `list` est-il indispensable pour la fonction `Calcule()` ?

2. Écrire une fonction qui calcule le périmètre et la surface d'un rectangle à partir des longueurs l1 et l2 des deux côtés. La fonction renvoie également la longueur et la largeur du rectangle.
3. Écrire une fonction qui calcule les n premiers termes de la suite de Fibonacci ($u_1 = 0, u_2 = 1, \forall n > 2, u_n = u_{n-1} + u_{n-2}$)
4. Utiliser cette fonction pour calculer le rapport entre 2 termes consécutifs. Représenter ce rapport en fonction du nombre de termes pour $n = 20$. Que constatez-vous ? Avez-vous lu *Da Vinci Code* ?
5. Écrire une fonction qui supprime les lignes d'un data.frame ou d'une matrice présentant au moins une valeur manquante.

Réponses

1. Les 3 éléments à renvoyer étant de type numérique, un vecteur peut suffire.
2. Fonction `rectangle()` (la fonction `rect()` existe déjà) :

```
rectangle=function(l1,l2){
    p=(l1+l2)*2
    s=l1*l2
    list(largeur=min(l1,l2),longueur=max(l1,l2),
    perimetre=p,surface=s)}
```

3. Utilisation de la fonction : rectangle(4,6) ;res=rectangle(8,7) pour calculer les n premiers termes de la suite de Fibonacci :

```
fibo=function(n){
    res=rep(0,n);res[1]=0;res[2]=1
    for (i in 3:n) res[i]=res[i-1]+res[i-2]
    res}
# Calcul du rapport de 2 termes consécutifs
        res=fibo(20)
        ratio=res[2:20]/res[1:19]
        plot(1:19,ratio,type="b")
```

Le rapport tend vers le nombre d'or $\frac{1+\sqrt{5}}{2} \approx 1.618034$.

4. Une façon, parmi beaucoup d'autres, de répondre à la question consiste à créer une fonction `ligne.NA` qui repère s'il y a au moins une valeur manquante dans un vecteur. Cette fonction filtre les lignes en question.

```
ligne.NA=function(vec){any(is.na(vec))}
filtre.NA=function(mat){
    tmp = apply(mat,1,ligne.NA)
    mat[!tmp,]}
# Application sur une matrice de test
matrice.test = matrix(1:40,nc=5)
matrice.test[2,5]=NA;matrice.test[4,2]=NA
matrice.test[7,1]=NA;matrice.test[7,5]=NA
filtre.NA(matrice.test)
```

4 Commandes de type `apply`

Comme déjà expliqué, il est vivement recommandé d'éviter les boucles très chronophages. La fonction `apply` et ses variantes sur des vecteurs, matrices ou listes permettent d'appliquer une même fonction `FUN` sur toutes les lignes (MARGIN=1) ou les colonnes (MARGIN=2) d'une matrice MAT :

```
apply(MAT , MARGIN, FUN)
```

Les fonctions `lapply` et `sapply` calculent la même fonction sur tous les éléments d'un vecteur ou d'une liste.

`lapply(X,FUN, ARG.COMMUN)` permet d'appliquer la fonction `FUN` à tous les éléments du vecteur ou de la liste X. Les valeurs de X sont affectées au premier argument de la fonction FUN. Si la fonction FUN a plusieurs paramètres d'entrée, ils sont spécifiés dans `ARG.COMMUN`. Cette fonction retourne le résultat sous la forme de listes. La fonction `sapply` est similaire à `lapply` mais le résultat est retourné si possible sous forme de vecteurs.

`tapply(X,GRP,FUN,...)` applique une fonction FUN sur les sous-groupes d'un vecteur X définis par une variable de type factor GRP.

Exemples :

```
data(iris)
apply(iris[,1:4],2,sum)
lapply(iris[,1:4],sum)
```

```
sapply(iris[,1:4],sum)
tapply(iris[,1],iris[,5],sum)
```