Le logiciel R

1 Présentation

R est un logiciel proposant une large gamme de méthodes statistiques et des facilités graphiques importantes. C'est un **logiciel open-source** et grâce à son langage de programmation associé, il permet :

- les manipulations de données
- les calculs (en algèbre linéaire par exemple)
- la programmation
- les analyses statistiques (les résultats de ces analyses seront par défaut affichés à l'écran mais peuvent aussi être stockés dans des objets)
- la construction de nombreux graphiques (ceux-ci s'affichent dans une nouvelle fenêtre et peuvent s'exporter dans de nombreux formats, jpg, pdf, ps...)

Un grand nombre de *packages* (ensembles de programmes dédiés à des problèmes spécifiques) sont disponibles. Nous en reparlerons.
Le site internet `http://www.r-project.org/` est très utile pour toute information sur le logiciel (son téléchargement, une aide, accés aux *packages* ...)

De nombreuses documentations sont disponibles en ligne. On recommande particulièrement le document **R pour les débutants** de Emmanuel Paradis. Il est disponible toujours à la même adresse `http://www.r-project.org/` et dans l'onglet *Manuals*. Ensuite en cliquant sur *contributed documentation* et en allant en bas de page, vous trouverez des documents disponibles en français.

Les menus sont très peu développés et vous l'aurez compris, on travaillera principalement avec des **lignes de commandes** ! Ces commandes sont à taper directement dans la **console**.
Remarque : Il existe d'autres interfaces, notamment Rcmdr. Cependant pour pleinement les exploiter il importe de maitriser **R** *en lignes de commande au préalable.*

Les lignes de commandes peuvent être rappelées avec les fléches « haut » et « bas ».

Vous allez créer des **objets** (variables, fonctions...). Vous pourrez sauvegarder votre environnement de travail avec tous ces objets (voir plus loin).

Le nom d'un objet commencera toujours par une lettre et peut comporter outre des lettres, des chiffres et des points. Attention **R** distingue les majuscules des minuscules.

2 R comme une calculatrice !

Tapez les instructions suivantes (en terminant par "Entrée") :

```
> 2+6
```

Remarque :
le signe > en début de ligne est généré par **R** et signale que **R** attend que vous lui donniez une instruction ...

```
> 3/2
> (2-6)/2
> 2-6/2
> 1+(3*2)
> 1+3*2
> 1/0
> 1/Inf
```

Que constatez-vous ?

```
> 3^2
> 3^(1/2)
> sqrt(3)
> log(exp(1))
> 1/log(1)
> exp(-2)
```

Que fait la fonction max et comment s'utilise-t-elle ? Taper :

```
> ? max
```

Garder des valeurs en mémoire : l'affectation.
R manipule des *objets* (exemples : vecteurs, matrices, listes, ...) auxquels on assigne des valeurs ou des expressions. Une assignation se fait à l'aide de "$<-$" ou de "$=$", comme par exemple $x < -3$ ou $x = 3$ pour assigner la valeur 3 à l'objet x. Taper

```
> x<-2
> x
> 2->y
> y
> z=2
> z
> x+x
> z=x+x
```

ou encore, taper :

```
> a<-2
> b=3
```

puis affecter à c la valeur $(\exp(b)\sqrt{\log(a)})^{-\frac{b}{a}}$

3 Les objets

Les éléments de base du langage R sont des **objets** qui peuvent être des données (vecteurs, matrices, ...), des fonctions, des graphiques,...

Les objets R se différrencient par leur **classe**. Les principales classes d'objets sont
vector, *matrix*, *array*, *factor*, *data.frame*, *list*.
Leur contenu peuvent être de **nature (mode)**
null (objet vide), *logical*, *numeric*, *complex* ou *character*.

Seuls les *data.frame* et *list* peuvent être de nature hétérogène.

Exemple :

```
> x<-2
> is.vector(x)  # test si x est un vecteur
> mode(x)   #donne la nature  d'un objet x
> length(x)  # quelle est la longueur de x?
> y="a"
> mode(y)
> z=T # ou z=TRUE
> mode(z)
```

Mais on peut aussi avoir besoin de forcer nous-même la nature d'un objet :

Exemple :

```
> x=2
> mode(x)
> as.character(x)  # Considere x comme un caractere
[1] "2"
> y<-as.character(x)
> y
> mode(y)
```

A) LES VECTEURS

1) Création d'un vecteur "à la main" : $y <- c(,,,)$

Les vecteurs sont des séries de valeurs **de même type**. Par exemple une série de nombres ou une série de chaine de caractères.

```
> y<-c(6,9,10)
> y
> is.vector(y)
> is.numeric(y)
> a<-c("A","B","C")
> is.character(a)
> z=c(T,T,F,F,F)
> is.numeric(z)
> is.character(z)
> is.logical(z)
```

Vous l'aurez remarqué : les chaines de caractères doivent être entourées de guillemets, les valeurs logiques sont codées TRUE ou FALSE abrégées T et F. Enfin les données manquantes sont codées par la chaine de caractère NA.

Toujours plus loin dans la construction de vecteurs ...

```
> d=1:6
> x1<-rep(1,4) # rep : replicate
> x2<-rep(1:4,2)
> x3<-rep(1:4,each=2) # que constatez-vous ?
> d<-c(7,9,13)
> d1<-rep(d,3)
> d2<-rep(d,1:3)
> d3<-rep(1:2,c(10,15))
> length(d3) # longueur d'un vecteur
> d3>2  # logical
> length(d3>2)
> y1<-seq(1,10,0.5)
> y2<-seq(1,2,length=20) # seq : sequence : par rapport à y1
> y<-paste("X",1:10,sep="")
```

2) Opération sur les vecteurs

- **Opérations sur les vecteurs numériques**

Essayer d'utiliser +,-,*,/,sqrt(), log(), exp(), abs(), t()

Ou encore :

```
> z<-3*x1+y1
> z<-x1%*%t(y1)  # multiplier x par transpos\'{e}e de y1
```

Finalement, c'est quoi z ?

- **Opérations logiques sur les vecteurs**

Avec par exemple

```
> x<-1:6
> y<-c(1,4,2,5,4,3)
```

vous pouvez tester $x < y$, $x == y$, $x! = y$, $(x <= 3)\&(y > 3)$.

- **Autres fonctions utilisables sur les vecteurs :**

min(x), max(x), length(x), sum(x), prod(x), sort(x), mean(x), cumsum(x), cumprod(x), summary(x) ...

```
> x<-c(2,4,6,5,3,1)
> rev(x)
> sort(x)
> sort(x,decreasing=TRUE)
> rev(sort(x))

> summary(x)
```

Pour en savoir plus sur une fonction, ne pas hésiter à utiliser *help()*.

3) Extraction d'éléments

```
> y<-c(1,4,2,5,4,3)
> y[2]
> y[length(y)]
> y[2:4]
> which(y>=4)
> y[which(y>=4)]
> y[y>=4]
> y[2]<-0
> y
> is.numeric(y)
> y[2]<-"essai"
> is.numeric(y)
> y
> is.character(y)
```

Ou encore essayer :

```
> y<-c(6,9,10)
> a<-c("A","B","C")
```

Que donne $y[1]$? $y[c(1,3)]$? $y[y > 8]$? $y[-1]$? $a[c(T,T,F)]$?

B) LES FACTEURS

Ce sont des vecteurs pour des variables qualitatives. Ces variables ont donc différentes modalités. On parle de **niveau du facteur**. Elles peuvent être ordonnées ou non.

Si une variable n'est pas sous la forme d'un facteur, elle peut être transformée en facteur :

```
> x<-c("homme","femme","femme","homme","homme")
> is.factor(x)
> is.character(x)
> x<-as.factor(x)
> x
> is.factor(x)
> is.character(x)
> levels(x)
> levels(x)<-c("F","H") #Renomme les facteurs
> x
> str(x) #quelle est la structure de x?
> as.numeric(x)
```

L'ordre des facteurs est important lorsqu'on définit des modèles, en particulier des régressions logistiques.

```
> y<-relevel(x,"H")
> y
> str(y)
```

Essayez encore :

```
> x=factor(rep(c("B1","T1","B2","T2"),each=2))
> x
> class(x)
> table(x)
```

C) LES MATRICES

1) Création de matrices

- **par "collage" de vecteurs lignes ou colonnes**

```
> A<-1:4
> B<- seq(5,8)
> C<- 9:12
> M1 <-cbind(A,B,C) # collage des colonnes
> M2 <-rbind(A,B,C) # collage des lignes
> dim(M1)
> dim(M2)
> M3=cbind(M1,rep(3,4))
```

- **avec l'ordre "matrix"**

```
> Z1 <-matrix(1:12,nrow=3,byrow=T)
> str(Z1)
> x<-c(2,4,6,5,3,1)
> M2 <-matrix(x,nrow=2)
> M3 <-matrix(x,nrow=2,byrow=T)
> dim(M2)

> colnames(Z1)<-c("col1","col2","col3","col4")
> rownames(Z1)<-paste("Z",1:3,sep="")
> Z1
> dimnames(Z1)

> Z2<-matrix(1:12,ncol=3)
> summary(Z2)
```

- **à partir d'un vecteur existant**

```
> x<-1:12
> dim(x)<- c(3,4)
```

C'est quoi x ?

2) Extraction d'éléments

```
> Z2[2,3]
> Z2[2,]
> Z2[,3]
> Z2[-2,]
```

3) Opérations courantes sur les matrices

```
> diag(1:5)
> diag(4)
> M <-matrix(1:12,nrow=3,byrow=T)
> dim(M)
> MM<-M[c(1:2),c(1:2)]
> diag(MM)
> MMtrans <-t(MM)
> MMinv<-solve(MM)
> MMinv%*%MM  #identite ?
```

Les opérations $+, -, *, /,$ $log()$ s'effectuent élément par élément :

```
> Z1=matrix(1:12,ncol=3)
> 3*Z1
> Z2=matrix(1:12,ncol=3,byrow=TRUE)
> Z1*Z2
```

Mais ça ne correspond pas au produit matriciel !
Essayez alors

```
> Z1%*%Z2
```

Opération avec la fonction *apply*

```
> apply(Z1,1,sum)
> apply(Z1,2,sum)
> apply(Z1,2,sum)/4
> apply(Z1,2,mean)
```

Pour finir, que fait :

```
> scale(Z1)
```

Exercice 1. Retrouver les propriétés vues dans le cours d'analyse de données sur l'application du calcul matriciel au calcul statistique

Exercice 2.

1. Créer deux vecteurs : *poids*, *taille* de composantes respectives $(60, 72, 57, 90, 95, 72)$ et $(1.75, 1.80, 1.65, 1.5, 1.74, 1.91)$.
2. Calculer $bmi = poids/taille^2$
3. Créer un vecteur ayant seulement les composantes > 20 de bmi. Combien a-t-il d'éléments ?
4. Créer une matrice de type individus $\times$ variables avec les variables *poids, taille* et *bmi*.

D) LES LISTES

1) Création d'une liste

```
> jeu=c(1,7,3,6,5)
> noms=c("a","b","c")
> class(jeu)
> class(noms)
> test=list(jeu, noms)
> test
```

et en donnant des noms aux différentes composantes de la liste :

```
> test=list(Jeux=jeu, Id=noms)
> test
> summary(test)
> str(test)
```

2) Extraire des composantes et des éléments dans les composantes

```
> test[1] # liste
> test[[1]] # vecteur
> test$Jeux
> test$Jeux[2]
> test$Id[3]
```

3) Appliquer une fonction à chaque composante

```
> lapply(test,max)
```

et défaire une liste :

```
> unlist(test)
```

E) LES DATA FRAME

1) Création d'un data frame

Un *data frame* est une table de vecteurs de même longueur mais pouvant chacun avoir son type. Les colonnes sont hétérogènes : certaines peuvent être des chaînes de caractères quand d'autres peuvent être numériques. La création se fait à l'aide de la commande *data.frame* :

```
> bloc=c("B1","B1","B1","B2","B2","B2")
> trt=c("T1","T2","T3","T1","T2","T3")
> yield=c(124,213,345,412,348,286)
> donnees=data.frame(bloc,trt,yield)
> donnees
```

Pour changer éventuellement les noms de colonnes :

```
> donnees=data.frame(B=bloc,T=trt,Y=yield)
```

```
> class(donnees)
> names(donnees)
> dim(donnees)
```

Pour un aperçu sur la composition et la structure du data frame :

```
> summary(donnees)
> str(donnees)
```

Dans le cas où toutes les colonnes sont numériques, transformer une matrice en data frame :

```
> M=matrix(1:12,4,3,byrow=FALSE)
> colnames(M)=paste("col",1:3,sep="")
> rownames(M)=paste("Z",1:4,sep="")
> is.matrix(M)
> is.data.frame(M)
> Z=as.data.frame(M)
> is.matrix(Z)
> is.data.frame(Z)
> is.list(Z)
```

Un data frame est une liste particulière.

On peut ajouter une colonne de type facteur au data frame :

```
> Z$F=factor(c("a","b","a","c"))
> str(Z)
```

2) Extraction d'éléments

```
> Z$col2
> donnees$T
> donnees$T[2]
> Z[[2]]
> Z[,c(2,3)]
> donnees[,c("B","T")]
> Z[Z$F=="a"]
> Z[Z$F=="a",] # quelle différence ?
```

2) Et autre ...

Application d'une fonction

```
> sapply(donnees,is.factor)
```

ou encore

```
> col1
> attach(Z)
> col1
> detach(Z)
```

4 Importer, exporter des données

R peut lire n'importe quel fichier texte. La commande de base est *read.table*. Elle va créer un *data.frame*.

Exemple : pour lire le fichier *loisirs.txt*

```
CSP      Circuit Mer Montagne Campagne Ville
Agriculteurs 7 56 23 11 3
Artisans 14 46 15 16 10
Cadres         13 45 11 20 12
```

dont la première ligne contient le nom des variables et le stocker dans *l'objet* vacances :

```
> vacances=read.table("loisirs.txt",header=T)
> vacances
> vacances$Circuit
```

Pour lire un fichier *Excel*, il faut tenir compte que l'on a des colonnes *(donc des tabulations)* et que les nombres décimaux s'écrivent par exemple $3,24$ alors que R ne reconnaît que 3.24. Le mieux est de sauvegarder votre fichier Excel en utilisant l'extension *.csv*, qui le plus souvent sépare les colonnes en utilisant le caractère " ;".
On a alors 2 types de fichiers :
- les fichiers texte (".txt") où le séparateur est un espace
- les fichiers csv (comma separated values) où le séparateur est ","' ou ";"'

A titre d'exemple, le fichier *Essai1* comporte 3 colonnes (Surface ; Prix au m^2 ; Prix Total). Après avoir consulté ce fichier avec un éditeur de texte, utiliser la commande suivante pour le charger avec R :

```
> appart = read.table("Essai1.csv",sep=";",dec=",",header=T)
```

Attention : la commande précédente suppose que le fichier *Essai1.csv* est dans le répertoire courant ; dans le cas contraire, il faut donner le chemin pour accéder au fichier, comme par exemple :

```
read.table('C:\\Documents and Settings\\Mes_data\\Essai1.csv',sep=";",dec=",",header=T)
```

ou changer de répertoire de travail : utiliser la commande *Changer le répertoire courant* de l'onglet *Fichier* de **R** pour se placer dans le bon répertoire.

```
ou taper :
> setwd('C:\\Documents and Settings\\Mes_data')
```

et **R** travaillera dans ce répertoire directement.

Pour finir, l'option *colClasses* permet de spécifier la classe des données de chaque colonne.

Pour sauvegarder des données dans un fichier texte, la fonction de base est *write.table*.
Ajouter une colonne au data frame *appart* et sauvegarder ce nouveau data frame :

```
write.table("Essai2.csv",sep=";")
```

les arguments sont les mêmes que pour la lecture.

Pour charger et sauvegarder des fichiers au format R :
save(data,file="donformatR.rda")
load("donformatR.rda")

```
> load("Titanic.rda")
> is.data.frame(Titanic)
> str(Titanic)
> summary(Titanic)
```

5 Les graphiques

On distingue dans R :

- des commandes graphiques dites « haut-niveau » qui vont créer un graphe ou la base d'un graphe : plot, pairs, hist, pie, ...
- des commandes graphiques dites « bas-niveau » qui se rajoutent à un graphe existant : points, lines, abline, legend, locator,...

Les paramètres graphiques (taille des caractères, couleur, axes, ...) peuvent être entrés en option des commandes de haut niveau ou être gérés globalement par la fonction *par()* (68 paramètres différents - *?par*). Dans ces paramètres, il y a notamment : titre (*main*), couleurs (*col*), taille des points (*cex*), forme des points (*pch*), relier des points par des lignes (*type*), limites des axes (*xlim*, *ylim*) ... et bien d'autres choses encore !

```
> x<-seq(0,10,length=50)
> y<-sqrt(x)
> plot(x,y)
> plot(x,y,type="l")
> plot(x,y,type="s")
## essayez aussi :
> par(mfrow=c(2,2))
> plot(x,y)
> plot(x,y,type="l")
> plot(x,y,type="s")
> plot(x,y,pch="s")
> abline(v=4,col=2,lty=2)
```

1. **Histogramme** : fonction *hist()*
   Si x désigne un vecteur d'observations, hixt(x) permet de tracer l'histogramme associé à x.

   ```
   > x<- rnorm(1000,5,2) #simuler 1000 r\'{e}alisations d'une loi normale
   > hist(x)
   ```

   Remarque :
   Arguments de *hist* :
   freq =T si les hauteurs des rectangles représentent des effectifs, =F pour des fréquences ;
   breaks = entier : nbre de classes demandé ;
   labels =T si "étiquetage" des rectangles ;
   main = 'chaine caractères' pour mettre un titre à l'histogramme ;
   xlab = 'chaine caractères' pour mettre une légende à l'axe des x ;
   ylab = 'chaine caractères' pour mettre une légende à l'axe des y ;
   col = vecteur chaine caractères couleurs des rectangles ;
   etc ...

   ```
   > hist(iris$Petal.Length)
   ```

2. **Diagramme en barres** : fonction *barplot()* La fonction *barplot()* permet de tracer des diagrammes en barres (séparées ou juxtaposées).

   faire un diagramme en barres séparées (pour variable nominale) :

   ```
   > barplot(c(10,15),names=c("H","F"))
   ```

   faire un diagramme en barres juxtaposées (pour variable ordinale) :

   ```
   > barplot(c(10,15),space=0,names=c("Petit","Grand"))
   ```

3. **Diagramme en bâtons** : on utilise la fonction *plot()* conjuguée à la commande *table*. La fonction *table()* permet de construire le tableau de distribution. On peut ensuite tracer le diagramme en bâtons sur la base de ces effectifs ou après transformation en fréquence.

   ```
   > x=c(rep(1,15),rep(2,5),rep(3,10),rep(4,2))
   > x
   > table(x)
   > plot(table(x),type="h")
   ```

4. **Graphique à 2 variables** : fonction *plot()*.

   ```
   > data(iris)
   > plot(iris$Sepal.Length,iris$Petal.Length)
   > plot(iris$Sepal.Length,iris$Petal.Length,main="Iris",xlab="Sepale",ylab="Petale",pch=22,bg="yellow"
   ```

   Un autre exemple :

   ```
   > x=runif(100)
   > bruit=rnorm(100,sd=0.5)
   > y=1+2*x+bruit
   > plot(x,y,cex=1.5) # représentation graphique du nuage de points
   > lines(x,1+2*x,lwd=2,col="blue") # ajout de la droite de régression
   > segments(x,y,x,1+2*x) # ajout des segments
   > text(0.3,3.5,"La régression linéaire",col="blue",cex=2) # ajout d'un texte en (0.3,3.5)
   ```

   Un dernier exemple : que fait ?

   ```
   > plot(iris$Petal.Length~iris$Species)
   ```

   Quelques comandes utiles :
   - ouvrir et supprimer une fenêtre graphique : *x11()* et *dev.off()*
   - positionner ou connaître les coordonnées d'un point : *locator()*