

# TD de statistique : introduction à R

Jean-Baptiste Lamy

11 octobre 2007

## 1 Introduction : pourquoi R ?

R est un logiciel pour l'analyse statistique. C'est un logiciel libre; il est disponible gratuitement et tourne sur différent système (PC Linux, PC Windows, Mac). Vous pouvez le télécharger ici : <http://cran.r-project.org/> si vous voulez l'installer chez vous.

R permet de tracer toute sorte de graphiques (histogramme, camember,...), de calculer différents paramètres (moyenne, écart type,...) et de réaliser les tests statistiques. R "connaît" les formules pour effectuer ces tests statistiques, en revanche c'est à vous de lui dire *quel* test utiliser!

R est un programme en ligne de commande : l'utilisateur (c'est à dire vous!) donne des ordres au programme en tapant des commandes avec clavier. Ces commandes doivent être entrées dans un langage spécifique que nous allons étudier.

## 2 Démarrer R

Pour démarrer R sous Windows ou Mac, double-cliquez sur l'icône correspondante.

Pour démarrer R sous Linux, double-cliquez sur l'icône du terminal (icône avec un petit écran noir), puis dans la fenêtre qui s'affiche tapez "R" et validez.

Le signe ">" en début de ligne est "l'invite de commande" de R : le programme vous demande d'entrer une commande.

La commande suivante (à taper au clavier; le ">" est l'invite de commande et ne doit pas être tapé) permet de quitter R :

```
> q()
```

**Astuce** lorsque l'on utilise R, il est possible de reprendre la ligne de commande taper précédemment en appuyant sur la flèche du haut du clavier. Appuyer plusieurs fois permet de récupérer des lignes plus anciennes.

## 3 Syntaxe

### 3.1 Commentaires

Dans R, tout ce qui suit le caractère # (= dièse) est un commentaire et n'est pas pris en compte par R :

```
> # Ceci est du baratin qui n'est pas pris en compte par R!
```

### 3.2 Variables

R étant prévu pour faire des calculs statistiques, il ne manipule que des tableaux de données. Ces tableaux sont stockés dans des *variables*, ce qui permet de leur donner un nom. Le nom des variables doit commencer par une lettre et peut contenir des lettres, des chiffres, des points et des caractères de soulignement (\_), mais surtout pas d'espace.

L'opérateur = est utilisé pour donner une valeur à une variable; il peut se lire "prend la valeur de" (NB : on trouve aussi l'opérateur <- ("flèche") qui a exactement la même signification).

```
> age = 28
```

Pour afficher la valeur de la variable âge, il suffit de taper le nom de la variable :

```
> age  
[1] 28
```

la variable "age" est ici un tableau avec une seule case, qui contient le chiffre 28. Le 1 entre crochet indique qu'il s'agit de la case n°1. R numérote les cases des tableaux en commençant à 1.

## 4 Types de donnée

R peut manipuler des nombres entiers, des *flottants* (= nombres à virgule), des chaînes de caractère et des *booléens* (valeur vraie ou fausse) :

```
> age           = 28 # Entier
> poids        = 64.5 # Flottant
> nom          = "Jean-Baptiste Lamy" # Chaîne de caractère
> enseignant   = TRUE # Booléen
> etudiant     = FALSE # Booléen
```

Enfin, la valeur spéciale NA (*non available*) est utilisée lorsqu'une donnée est inconnue.

## 5 Tableaux

R définit plusieurs types de tableaux que nous allons voir ; il y a peu de différences entre eux et tous s'utilisent de la même manière.

### 5.1 Vecteurs

Un vecteur est un tableau à une dimension. Toutes les cases du vecteur doivent contenir des données du même type (des entiers, des chaînes de caractère,...). La fonction `c()` permet de créer un vecteur :

```
> ages = c(28, 25, 23, 24, 26, 23, 21, 22, 24, 29, 24, 26, 31, 28, 27, 24, 23, 25, 27, 25,
24, 21, 24, 23, 25, 31, 28, 27, 24, 23)
> ages
[1] 28 25 23 24 26 23 21 22 24 29 24 26 31 28 27 24 23 25 27 25 24 21 24 23 25
[26] 31 28 27 24 23
```

“[1]” indique que ce qui suit est la première valeur du vecteur, et “[26]” que la deuxième ligne commence à la 26ème valeur ; “[1]” et “[26]” ne sont pas des éléments du vecteur.

La fonction `c()` permet aussi de *concaténer* (= mettre bout à bout) des vecteurs :

```
> hbA1c_groupe_temoin       = c(75.0, 69.2, 75.4, 87.3)
> hbA1c_groupe_intervention = c(70.5, 64.2, 76.4, 81.6)
> hbA1c = c(hbA1c_groupe_temoin, hbA1c_groupe_intervention)
> hbA1c
[1] 75.0 69.2 75.4 87.3 70.5 64.2 76.4 81.6
```

Il est possible d'accéder à un élément du vecteur avec des crochets. Par exemple pour accéder au second élément du vecteur `poids` :

```
> hbA1c[2]
[1] 69.2
```

Il est aussi possible d'accéder à l'ensemble des poids répondant à une condition, par exemple l'ensemble des poids supérieurs à 70.0 :

```
> hbA1c[hbA1c > 70.0]
[1] 75.0 75.4 87.3 70.5 76.4 81.6
```

Enfin, la fonction `length()` permet de récupérer le nombre d'éléments d'un tableau :

```
> length(hbA1c)
[1] 8
> length(hbA1c[hbA1c > 70.0])
[1] 6
```

Il est possible de créer un vecteur contenant une suite de nombres entiers avec la fonction `seq` :

```
> seq(1, 10)
[1] 1 2 3 4 5 6 7 8 9 10
> seq(1, 10, by = 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
[16] 8.5 9.0 9.5 10.0
```

## 5.2 Matrices

Une matrice est un tableau à deux dimensions, c'est à dire avec des lignes et des colonnes. Comme pour les vecteurs, toutes les cases d'une matrice doivent contenir des données du même type.

Une matrice est créée à partir d'un vecteur contenant les valeurs, et d'un nombre de ligne (nr, pour Number of Row) et de colonne (nc, pour Number of Column) :

```
> ma_matrice = matrix(c(1.5, 2.1, 3.2, 1.6, 1.4, 1.5),nr=3, nc=2)
> ma_matrice
      [,1] [,2]
[1,]  1.5  1.6
[2,]  2.1  1.4
[3,]  3.2  1.5
```

Les éléments de la matrice peuvent être accédés en donnant entre crochets le numéro de la ligne puis celui de la colonne :

```
> ma_matrice[1, 1]
[1] 1.5
```

Il est aussi possible de récupérer une ligne ou une colonne entière, en ommettant le numéro correspondant :

```
> ma_matrice[1,]
[1] 1.5 1.6
```

## 5.3 Listes

Une liste est un tableau à une dimension, qui peut contenir des données de différents types (contrairement au vecteur).

```
> ma_liste = list("JB", 28)
```

## 5.4 Tableaux de donnée (*data frame* en anglais)

Un tableau de donnée est un tableau où chaque colonne correspond à un attribut différents (âge, taille, poids par exemple) et chaque ligne à un individu différent. Voici un exemple :

nom	taille	poids
Jean-Baptiste Lamy	1.70	64.0
Bertrand Lamy	1.80	63.0
...	...	...

Il est possible de créer des tableaux de données dans R, cependant il est beaucoup plus facile de les charger à partir d'un fichier. On utilise pour cela la fonction `read.table()`.

Voici le fichier `taille_poids.csv` :

```
nom,taille,poids
Jean-Baptiste Lamy,1.70,64.0
Bertand Lamy,1.80,63.0
M. X,1.67,70.5
M. Y,1.69,95.0
M. Z,1.75,NA
```

Ce fichier peut être chargé ainsi dans R :

```
tableau = read.table("taille_poids.csv", sep=",", header=TRUE)
```

`sep=","` indique que dans le fichiers les différentes colonnes sont séparées par des virgules, et `header=TRUE` indique que la première ligne du fichier contient les noms des colonnes. Ces valeurs correspondent aux fichiers CSV (*Comma-Separated Value file* : fichier dont les valeurs sont séparées par des virgules) qui peuvent être généré facilement, par exemple en exportant depuis un tableur comme OpenOffice ou Excel.

```
> tableau
      nom taille poids
1 Jean-Baptiste Lamy  1.70  64.0
2      Bertand Lamy  1.80  63.0
3           M. X    1.67  70.5
4           M. Y    1.69  95.0
5           M. Z    1.75   NA
```

Les noms des colonnes sont disponibles via la fonction `names()` :

```
> names(tableau)
[1] "nom" "taille" "poids"
```

Comme pour les matrices, il est possible d'accéder aux cases, lignes et colonnes d'un tableau. Les noms des colonnes peuvent être utilisés à la place de leurs index :

```
> tableau[1, 2] # Première ligne, deuxième colonne : taille du premier individu
[1] 1.7
> tableau[1, "taille"] # Première ligne, colonne taille : pareil
[1] 1.7
> tableau[1,] # Première ligne
      nom taille poids      IMC
1 Jean-Baptiste Lamy  1.7   64 22.14533
> tableau[,"taille"] # Colonne taille
[1] 1.70 1.80 1.67 1.69 1.75
```

Il est aussi possible d'indexer avec une condition : par exemple, pour obtenir un tableau avec seulement les individus dont la taille est supérieure à 1m70 (ne pas oublier la virgule, qui sert à indiquer que l'on veut récupérer des lignes!) :

```
> tableau[tableau["taille"] > 1.7,]
      nom taille poids      IMC
2  Bertand Lamy  1.80   63 19.44444
5      M. Z   1.75   NA      NA
```

La fonction `attach()` permet de définir le tableau "par défaut" sur lequel porte les analyses suivantes :

```
> attach(tableau)
```

Ensuite, il est possible d'accéder au colonne du tableau directement par leur nom :

```
> taille
[1] 1.70 1.80 1.67 1.69 1.75
```

Enfin, la fonction `write.table()` permet d'enregistrer un tableau de donnée (`row.names = FALSE` permet de désactiver les numéros de colonne) :

```
write.table(tableau, "taille_poids_2.csv", sep = ",", row.names = FALSE)
```

Ce fichier pourra ensuite être rechargé avec `read.table()` comme ci-dessus.

## 6 Opérateur et calcul

### 6.1 Opérations

R permet de réaliser la plupart des opérations courantes à l'aide des opérateurs suivants : + (addition), - (soustraction), \* (multiplication), / (division), ^ (puissance).

```
> 2 * 3 + 1
[1] 7
```

Les opérations sont réalisées sur chaque élément des tableaux. Par exemple, pour calculer l'Indice de Masse Corporelle (IMC) sur chaque individu du tableau de donnée chargé précédemment, en appliquant la formule  $IMC = \frac{poids}{taille^2}$  :

```
> imc = poids / (taille ^ 2)
> imc
[1] 22.14533 19.44444 25.27878 33.26214 NA
```

R a calculé l'IMC pour chacun des 5 individus! Notez que la donnée manquante (NA) se "propage", ce qui est logique : si le poids de M. Z est manquant, il n'est pas possible de calculer son IMC, qui est donc manquant lui aussi.

Il est possible d'ajouter une quatrième colonne avec l'IMC à notre tableau de la manière suivante :

```
> tableau["IMC"] = imc
> tableau
      nom taille poids      IMC
1 Jean-Baptiste Lamy  1.70  64.0 22.14533
2      Bertand Lamy  1.80  63.0 19.44444
3      M. X   1.67  70.5 25.27878
4      M. Y   1.69  95.0 33.26214
5      M. Z   1.75   NA      NA
```

## 6.2 Tests

Les tests se font avec les opérateurs `<`, `>`, `<=` (inférieur ou égal), `>=` (supérieur ou égal), `==` (égal), `!=` (différent de). Ils retournent une (ou plusieurs) valeur(s) booléenne(s).

```
> 1 < 3
[1] TRUE
> imc > 25
[1] FALSE FALSE TRUE TRUE NA
```

Il est possible de combiner plusieurs tests avec des `&` (et) ou des `|` (ou).

## 7 Fonctions

R définit un grand nombre de fonctions; nous en avons déjà vu quelques unes. La fonction `help()` permet d'obtenir de l'aide sur une fonction :

```
help(mean)
```

Voici une liste des principales fonctions :

**summary(tableau)** affiche un résumé du tableau

**length(vecteur)** retourne le nombre de case d'un vecteur

**ncol(tableau)** retourne le nombre de colonne d'un tableau

**nrow(tableau)** retourne le nombre de ligne d'un tableau

**q()** quitte R

**min(vecteur)** retourne la plus petite valeur d'un vecteur

**max(vecteur)** retourne la plus grande valeur d'un vecteur

**sort(vecteur)** retourne une copie d'un vecteur après l'avoir trié

**mean(vecteur)** calcule la moyenne

**median(vecteur)** calcule la médiane

**var(vecteur)** calcule la variance

**sd(vecteur)** calcule la déviation standard

**sqrt(nombre)** retourne la racine carré d'un nombre

**sum(vecteur)** retourne la somme de toutes les valeurs du vecteur

**round(flottant)** arrondit un nombre

**rank(vecteur)** retourne un vecteur avec les rangs (c'est à dire avec 1 pour la plus petite valeur, 2 pour la seconde, etc)

**quantile(vecteur)** affichage par quantile

Nous avons vu que les valeurs NA se propagent. Cela est parfois gênant, par exemple lorsque l'on calcule une moyenne :

```
> mean(poids)
[1] NA
```

Dans ce cas, il faut demander à R de ne pas tenir compte des valeurs NA pour ce calcul :

```
> mean(poids, na.rm = TRUE)
[1] 73.125
```

## 8 Exercice

Nous souhaitons étudier l'efficacité de trois herbicides sur trois plantes : blé, chiendent et liseron. Pour cela, des cultures de ces plantes ont été mises en présence de l'un des trois herbicides, ou d'aucun d'entre eux. Le nombre de plants vivants dans la culture a été compté avant l'expérience, et 10 jours après. Chaque combinaison plante - herbicide a fait l'objet de 20 expérimentations, plus un témoin sans herbicide (soit 240 expérimentation en tout).

Le tableau de donnée est disponible dans le fichier `herbicide.csv`.

1. Charger le fichier `herbicide.csv` et afficher les données.

**Réponse :**

```
> t = read.table("herbicide.csv", sep=";", header=TRUE)
> attach(t)
> t
```

2. Calculer la moyenne du nombre de plants initial, sa variance, son écart type, ainsi que les valeurs minimum et maximum.

**Réponse :**

```
> mean(nb_plantes)
[1] 100.4958
> mean(nb_plants)
[1] 100.4958
> var(nb_plants)
[1] 115.4142
> sd(nb_plants)
[1] 10.7431
> min(nb_plants)
[1] 80
> max(nb_plants)
[1] 119
```

3. Calculer le pourcentage de plantes ayant survécues, pour chaque expérimentation, et l'ajouter dans une nouvelle colonne du tableau (colonne n°6).

**Réponse :**

```
> t["survivants"] = nb_plants_survivants / nb_plants
```

4. Combien d'expérimentations ont donné lieu à moins de 5% de plants survivants ?

**Réponse :**

```
> nrow(t[t["survivants"] < 0.05,])
[1] 23
```

Attention on utilise `nrow()` ici et pas `length()`, car il s'agit d'un tableau et pas d'un vecteur !

5. Le témoin correspond à l'absence d'herbicide. Extraire les lignes du tableau qui correspondent au témoin et les mettre dans une nouvelle variable que l'on appellera "témoin".

**Réponse :**

```
> temoin = t[t["herbicide"] == "aucun",]
```

6. Calculer la moyenne et l'écart type du pourcentage de plants ayant survécus, sur le témoin (Astuce : pensez à "attacher" le tableau témoin avec `attach()`!).

**Réponse :**

```
> attach(temoin)
> mean(survivants)
[1] 0.937564
> sd(survivant)
[1] 35.49346
```

7. De la même manière, calculer la moyenne et l'écart type du pourcentage de plants ayant survécus pour chacun des trois herbicides. Quel herbicide vous semble le plus efficace globalement ?

**Réponse :**

```
> attach(t[t["herbicide"] == "herbicide1",])
> mean(survivants)
[1] 0.6166904
> sd(survivant)
[1] 35.49346
> attach(t[t["herbicide"] == "herbicide2",])
> mean(survivants)
[1] 0.5596954
> sd(survivant)
[1] 35.49346
> attach(t[t["herbicide"] == "herbicide3",])
> mean(survivants)
[1] 0.1117945
> sd(survivant)
[1] 35.49346
```

=> L'herbicide 3 semble le plus efficace.

8. Quelle plante est celle qui a le mieux résisté aux herbicides, dans l'ensemble ?

**Réponse :**

```
> attach(t[t["plante"] == "blé",])
> mean(survivants)
[1] 0.6577918
> attach(t[t["plante"] == "liseron",])
> mean(survivants)
[1] 0.5073544
> attach(t[t["plante"] == "chiendent",])
> mean(survivants)
[1] 0.504162
```

=> le blé a le mieux résisté.

Lors du prochain TD, nous poursuivrons l'étude de nos herbicides de manière graphique avec R. Il est conseillé de ramené le TD n°1 lors du TD n°2!