TP 1 : Initiation

5 septembre 2017

1 Introduction

Le logiciel R est un logiciel d'étude statistique, il s'agit d'un logiciel libre, vous pouvez le télécharger et l'installer sur une machine personnelle. Le logiciel est disponible sur la plateforme du CRAN (*Comprehensive R Archive Network*), un réseau mondial de sites qui stockent les différentes versions, les mises à jour, les bibliothèques et la documentation.

1.1 Lignes de commande

Lancer le logiciel R.
La fenêtre de commandes du logiciel R permet de soumettre des instructions les unes à la suite des autres. Une instruction (ou commande) est une ligne de code précédée du prompteur :

```
> Une ligne de commandes
```

Une ligne de commandes peut définir des objets `R`, demander des informations sur un objet, définir une fonction, utiliser une fonction R prédéfinie dans le logiciel etc. Le symbole '`<-`' ou '`=`' est utilisé pour définir un objet (voir plus bas). Il est possible de soumettre plusieurs commandes séparées par le symbole '`;`' sur une même ligne. Une même commande peut s'écrire sur plusieurs lignes, auquel cas R matérialise le début de la 2ème ligne d'instructions par le symbole '`+`'.

Il est possible d'insérer des commentaires dans le programme en les faisant précéder du caractère # :

```
# ceci est un commentaire
```

Le logiciel R possède un système d'aide efficace. Par exemple, pour afficher la page d'aide de la fonction `var`, on utilise la commande suivante :

```
> help(var)
```

On peut aussi utiliser la commande `?` suivi du nom de la fonction.

1.2 Espace de travail

Créer un répertoire STATS (que vous placerez où vous voulez) et mettre STATS comme répertoire de travail.

Pour cela, vous pouvez chercher la commande adéquate dans le menu, ou utiliser la commande `setwd('chemin du répertoire')`. La commande `getwd()` permet de connaître le répertoire dans lequel on travaille.

Faire un nouveau document, que vous enregistrerez sous le nom TP1. Celui-ci sera créé dans votre dossier STATS sous le nom TP1.R.

A ce stade, vous avez deux fenêtres. La première fenêtre est la console. Vous pouvez y taper directement vos commandes. Lorsque vous quitterez R (menu déroulant ou commande `q()`), R posera la question : `Save workspace image? [y/n/c]`. Si vous répondez `y`, R sauvegarde tous les objets créés au cours de la session. Si vous répondez `n`, ces objets sont perdus. Pour continuer la session, il faut répondre par la lettre `c`.

Généralement, on ne sauvegarde pas la session. On va plutôt enregistrer les commandes que l'on juge importante dans un script, dans notre cas "TP1.R". Lorsque l'on rouvrira le logiciel R, la commande `source("TP1.R")` permettra d'exécuter toutes les commandes préalablement enregistrées.

La deuxième fenêtre est donc votre fichier TP1. Ici vous pourrez écrire les commandes que vous jugerez importantes. Pour exécuter une commande à partir du script, placez le curseur sur la ligne et lancer la commande Executer dans le menu Edition. Pour exécuter plusieurs lignes, il faudra les sélectionner.

1.3 Objets

R permet de manipuler des données organisées en structures de différentes formes (vecteurs, tableaux, listes etc). Toutes ces structures sont composées d'éléments de base, ces derniers pouvant être de différents types (numériques, logique etc.). Pour accéder au type d'un élément de base, on peut utiliser la fonction `typeof`. Pour connaître la classe de la structure d'un objet R, on peut utiliser la fonction `class`.

Pour créer un objet, on utilisera le signe `<-`. Pour voir tous les objets qui ont été créées durant la session, utilisez la commande `ls()`. Pour supprimer un objet, on tape `rm(objet)`.

```
> a <- 1
> a
> ls()
> rm(a)
> a
> ls()
```

1.4 Types d'objets

Type numérique. Le logiciel distingue deux types numériques : le type `integer` pour les entiers et le type `double` pour les réels. Il est possible de changer le type d'un élément numérique avec les fonction `as.integer` et `as.double`.

```
> a <- 1
> typeof(a)
> is.integer(a)
> b <- as.integer(a)
> is.integer(b)
```

R représente correctement les valeurs infinies avec `Inf` et `-Inf` et les formes indéterminées sont représentées avec `NaN`.

Type complexe. Le logiciel R permet de manipuler les nombres complexes avec le type `complex`, grâce à la lettre `i` et aux fonctions `Re` et `Im`.

Type booléen. Le type `logical`, résultat d'une condition logique, peut prendre les valeurs `TRUE` (ou `T`) et `FALSE` (ou `F`).

```
> c <- (1 > 2)
> c
> typeof(c)
```

Type chaîne de caractères. Un élément entre guillemets est de type chaîne de caractères : type `character`.

Données manquantes. En statistique, certaines données sont parfois manquantes. L'instruction `NA` (*non available*) indique que la donnée correspondante n'est pas disponible.

1.5 Vecteurs

Les vecteurs sont la forme de structure la plus simple, elle représente une suite d'éléments de même type. L'opérateur `c()` permet de concaténer des valeurs. La longueur d'un vecteur s'obtient avec la fonction `length`.

```
> x <- c(2, 6, 5, 8, 1, 3, 9)
> char <- c("a","c","d")
> x

[1] 2 6 5 8 1 3 9
```

Exercice 1

Définir un vecteur x1 égal à (10,11,12). Concaténer les vecteurs x et x1 grâce à la commande `c()`.

On peut créer des vecteurs particuliers grâce à certaines commandes. L'entrée 3 :5 donne le vecteur (3,4,5). Voir comment fonctionnent les commandes `rep` et `seq`. On peut parler du type d'un vecteur car tous les éléments qui le constituent sont de même nature.

```
> typeof(char)

[1] "character"
```

On accède aux éléments d'un vecteur avec les crochets []. *Quels sont les résultats des commandes suivantes ?*

```
> x[2]
> x[c(2,4)]
> x[-1]
```

1.6 Matrices

On définit une matrice à l'aide de la fonction `matrix`. Les options `ncol` et `nrow` permettent de définir les dimensions de la matrice.

```
> M <- matrix(0, nrow = 2, ncol = 3)
> M

     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

```
> M <- matrix(rep(1:2, 3), nrow = 2, ncol = 3)
> M

     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

Il existe une option `byrow=TRUE` qui permet de rentrer les valeurs par lignes et non par colonnes.

```
> M <- matrix(rep(1: 2, 3), nrow = 2, ncol = 3, byrow = TRUE)
> M

     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    2    1    2
```

Pour connaître les dimensions d'une matrice, on utilise la fonction `dim` (la fonction `length` retourne le nombre d'éléments de la matrice) :

```
> dim(M)

[1] 2 3

> length(M)

[1] 6
```

On a la possibilité d'attribuer des noms aux lignes et aux colonnes d'une matrice via l'option `dimnames` (ces noms doivent être de type *character*).

On peut aussi rajouter des lignes (fonction `rbind`) ou des colonnes (fonction `cbind`) à une matrice existante (voire concaténer des matrices). Voici un exemple :

```
> cbind(M, c(7, 7))

     [,1] [,2] [,3] [,4]
[1,]    1    2    1    7
[2,]    2    1    2    7

> rbind(M, c(8, 8, 8))

     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    2    1    2
[3,]    8    8    8

> cbind(M, M)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    1    1    2    1
[2,]    2    1    2    2    1    2
```

Exercice 2 :

1. *Définissez la matrice*

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

2. *Définissez la matrice B en concaténant sous la matrice A la ligne* $(1, 1, 1)$.
3. *Calculez la dimension de B.*

On accède aussi aux éléments d'une matrice avec les crochets []. *Quels sont les résultats des commandes suivantes ?*

```
> M[1,3]
> M[,1]
> M[2,]
> M[2,c(1,3)]
```

1.7 Listes

Une liste est une collection d'objets non nécessairement de même type. On peut créer une liste via la fonction `list` en définissant les éléments un à un :

```
> L1 <- list(x,M,"blabla")
> class(L1)

[1] "list"

> L1

[[1]]
 [1]  2.0  6.0  5.0  8.0  1.0  3.0  9.0

[[2]]
     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    2    1    2

[[3]]
[1] "blabla"
```

On peut aussi accéder aux éléments de la liste en utilisant les crochets : *que donne la commande suivante ?*

```
> L1[[2]]
```

Il est souvent plus agréable de donner un nom aux différents objets que contient la liste :

```
> L2 <- list(VecteurDeL2=x, ChaineDeL2 = "blablaba")
> L2

$VecteurDeL2
 [1]  2.0  6.0  5.0  8.0  1.0  3.0  9.0

$ChaineDeL2
[1] "blablaba"
```

Il est alors aussi possible d'accéder aux éléments de la liste en utilisant ces noms prédéfinis : *que donne la commande suivante ?*

```
> L2$VecteurDeL2
```

Exercice 3 *Définissez le vecteur A = (1,4,5), la matrice Mat de taille 3×4 composée uniquement de 1, et la liste l1 composée des deux éléments précédents. Donner des noms aux éléments de l1.*

1.8 Tableaux

Un tableau (dataframe) est une liste de vecteurs de même longueur. Le tableau est l'objet de référence en statistique car il permet de renseigner les valeurs prises par des variables (disposées en colonnes) sur une collection d'individus (disposés en lignes).

On peut créer un tableau de données en utilisant la fonction `data.frame`. Par exemple :

```
> var1<-c("a","b","a","fi","jk")
> var2<-c(5,8,9,1,3)
> tab<-data.frame(var1,var2)
> class(tab)

[1] "data.frame"
```

```
> is.list(tab)

[1] TRUE

> is.data.frame(tab)

[1] TRUE

> tab

  var1 var2
1    a    5
2    b    8
3    a    9
4   fi    1
5   jk    3
```

Un tableau conserve le nom des vecteurs, dans notre cas *var1* et *var2*. Il est possible de donner des noms aux lignes avec l'option `row.names` qui doit fournir un vecteur de type caractère et de longueur égale au nombre de lignes du tableau de données. Par exemple :

```
> var1<-c("a","b","a","fi","jk")
> var2<-c(0.25,0.35,0.15,0.25,0.3)
> tab<-data.frame(tabA = var1,tabB = var2,row.names=
          c("Paul","Pierre","Alain","Olivier","Bernard"))
> tab

        tabA tabB
Paul       a 0.25
Pierre     b 0.35
Alain      a 0.15
Olivier   fi 0.25
Bernard   jk 0.30
```

Un tableau étant par définition une liste particulière, il est possible d'accèder aux éléments d'un tableau avec les doubles crochets [[]], ou avec le $ en utilisant les noms de colonnes. La commande `attach` permet d'utiliser les noms des variables du tableau sans rappeler le tableau auxquelles elles appartiennent. On utilise la commande `detach` pour annuler le attach.

```
> tabB

Erreur : objet 'tabB' introuvable

> attach(tab)
> tabB

[1] 0.25 0.35 0.15 0.25 0.30
```

Importer les données d'un fichier externe dans un tableau R

La fonction `read.table` permet de lire et d'importer sous R des données provenant d'un fichier texte (ASCII). Cette instruction s'utilise comme suit :

```
> MesDonnees <- read.table(file= chemin vers le fichier, header = TRUE,
                                    sep = "\t", row.names = )
```

L'option `header =` indique si le fichier contient le nom des variables sur la première ligne. Le type de séparateur est indiqué par l'option `sep =` , par défaut il s'agit de la tabulation. L'option `dec ="."` indique le séparateur décimal (virgule ou point). L'option `row.names =` permet de renseigner le nom des lignes à l'aide d'un vecteur de chaînes de caractères. Il est aussi possible d'indiquer un numéro de colonne si ces noms sont présents dans le fichier.

Exercice 4

Téléchargez le fichier cathedral.ods et importez les données sous la forme d'un tableau que vous nommerez Cat. Ce fichier décrit les dimensions de cathédrales anglaises. La première colonne indique le nom de la cathédrale, la seconde renseigne s'il s'agit d'une cathédrale gothique ou romane, les deux dernières colonnes donnent les dimensions de celle-ci. Vous utiliserez comme nom de ligne le nom de la cathédrale.

1.9 Facteurs

La structure `factor` permet de manipuler les variables catégorielles plus facilement qu'avec des vecteurs de caractères, notamment grâce à l'utilisation des fonctions `factor` et `levels`.

```
> x <- factor(c("bleu","vert","rouge","bleu","vert"))
> x

[1] bleu  vert  rouge bleu  vert
Levels: bleu rouge vert

> levels(x)

[1] "bleu"  "rouge" "vert"

> class(x)

[1] "factor"
```

Un tableau peut contenir des objets de la classe facteur. Par exemple, dans le tableau tab défini précédemment, la variable tabA est reconnue automatiquement comme un facteur :

```
> class(tab$tabA)

[1] "factor"
```

Notez qu'un objet facteur est de type entier, même si celui-ci a été construit à partir d'un vecteur de chaines de caractères :

```
> typeof(var1)

[1] "character"

> typeof(tab$tabA)

[1] "integer"

> typeof(Cat$style)

[1] "integer"
```

2 Opérateurs

Opérateurs arithmétiques :
`+, -, *, /, ^ ...`
Étudiez les commandes suivantes :

```
> x <- -10:10
> y <- abs(x)
> x^2
> x+y
> x*y
```

Opérateurs de comparaison :
`==, <, >, <=, >=, != ...`
Étudiez les commandes suivantes :

```
> x<- -10:10
> x==0
> x>0
> x[x>0]
> x[x>=0]
```

Exercice 5 *(D'après l'ouvrage Le logiciel R cité plus haut, p.9).*
Récupérer le fichier `Intima_Media.ods` .

1. *Importer les données dans un dataframe que vous nommerez Intima.*
2. *Rajouter une colonne IMC contenant l'indice de masse corporel de chaque individu dans Intima. On rappelle que*
$$IMC = \frac{Poids\ (kg)}{Taille(m)^2}.$$
3. *Récupérer la mesure de l'intima (variable mesure) pour les personnes ayant un IMC>30.*
4. *Extraire les femmes sportives.*
5. *Extraire les non-obèses chez les personnes âgées de 50 ans ou plus (obèse=IMC>30).*

3 Fonctions R

3.1 Fonctions R prédéfinies

Il existe un nombre très important de fonctions pour manipuler des données. Outre les fonctions mathématiques de base du type `log`, `exp`, `cos`, `abs`, `sqrt` (racine carrée), `floor` (partie entière) ... en voici quelques-unes assez courantes :

- `sum(x)`, `prod(x)` : somme, produit des éléments de `x`,
- `min(x)`, `max(x)` : minimum, maximum des éléments de `x`,
- `which.min(x)`, `which.max(x)` : indice du min, max des éléments de `x`,
- `sort(x)` : trie les éléments de `x` dans l'ordre croissant,

Exercice 6

Soit la matrice

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Que vaut sum(A) ? Calculer la somme de la première colonne de A. Calculez le maximum de la deuxième ligne de A.

Exercice 7

1. *Que fait la fonction* `choose` *(utilisez l'aide) ?*
2. *On considère*

   ```
   > x<-choose(6,0:6)
   ```

   Calculez la somme des élements de x, le maximum, et l'indice du maximum.
3. *Ordonner x dans l'ordre croissant.*

3.2 Écrire ses propres fonctions R

Il est possible d'écrire ses propres fonctions R. La structure générale d'une fonction est la suivante :

```
Mafonction <-  function(arg1,arg2, ...)
{    suite de commandes
     sortie = ...
     return(sortie)
}
```

Notez que plusieurs objets peuvent être donnés en argument alors qu'un seul objet est renvoyé en sortie. Ceci ne pose pas de problème car il est toujours possible de disposer tous les éléments à renvoyer dans une même liste.

Exercice 8

Ecrire une fonction R qui prend en argument deux réels x et y ainsi qu'une matrice N, et qui renvoie :

- *la quantité $z = x + y$,*
- *l'entier nomb de coefficients de N qui sont dans l'intervalle $[x, y]$,*

— *La matrice Ntilde dont les coefficients (i, j) sont égaux à x si i et j sont pairs et à $N(i, j)$ sinon.*

Exercice 9

Écrire une fonction ma.variance qui calcule l'estimateur de la variance défini par :

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \qquad \text{où } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

pour un vecteur $x = (x_1, \ldots, x_n)$ donné en argument. Comparez avec la fonction prédéfinie ***var****.*

Boucles et tests. Le langage R autorise à manipuler des tests et les boucles for, while.

• Boucle `test` : les instructions sont effectuées si la condition est vérifiée.

```
if (condition)  {instructions} else {instructions}
```

• Boucle `for` : les instructions sont effectuées pour tous les i du vecteur.

```
for (i in vecteur) {instructions}
```

```
>  x <- rep(c(1,2),5)
> for (i in 1:4)
>   { if (x[i] == 1)
>       x[i] <- 5
> }
> x

 [1] 5 2 5 2 1 2 1 2 1 2
```

• Boucle `while` : les instructions sont effectuées tant que la condition est réalisée.

```
while (condition)  {instructions}
```

```
> a <- 2
> while (a < 100)
>   {a <- a^2 }
> a

[1] 256
```

Exercice 10

1. *Ecrire la fonction* ***compte*** *qui prend deux arguments :* ***sequence*** *(un vecteur de caractères) et* ***lettre*** *(un caractère), et qui retourne le nombre d'occurrences de* ***lettre*** *dans* ***sequence****. Appliquer cette fonction pour trouver le nombre d'occurrences de la lettre "a" dans la séquence biologique :*

   ```
   sequence<-c("a","a","t","g","a","g","c","t","a","g","c","t","g")
   ```

2. *Uiliser une boucle for pour obtenir la composition en (a,c,g,t) de la séquence biologique.*

4 Fonctions graphiques élémentaires

L'une des fonctions principales pour représenter graphiquement des données est la fonction `plot()`. Par exemple, pour tracer la densité de la loi normale standard $\mathcal{N}(0,1)$, commençons par créer un vecteur `x` contenant les points où sera évaluée la densité :

```
> x <- seq(-5,5,by=0.1)
```

La fonction `dnorm()` évalue la fonction de densité d'une loi normale standard :

```
> y <- dnorm(x)
```

Pour afficher les points avec des cordonnées `x` et `y`, on utilise la fonction `plot()` :

```
> plot(x,y)
```

Par défaut, `plot()` affiche un nuage de points. Les nombreuses options de la fonction `plot()` permettent de modifier le graphique. On obtient, par exemple, une courbe avec l'option `type='l'` pour relier les points par une ligne :

```
> plot(x,y, type='l')
```

D'autres options du graphique :

- `xlab` et `ylab` pour modifier le nom des axes
- `main` pour rajouter un titre
- `pch` pour jouer sur la forme des points : `pch=0` pour des carreaux, `pch=1` pour des cercles (par défaut), `pch=2` pour des triangles etc.
- `lty` pour jouer sur le type de ligne : par défaut `lty=1` donne des lignes continues, mais on peut obtenir des tirets (`lty=2`), des pointillés (`lty=3`)...
- `lwd` pour jouer sur l'épaisseur des lignes
- `col` spécifie la couleur (p.ex. `col='green'` ou `col=3` pour des points/lignes verts)
- `xlim` et `ylim` pour fixer la limite inférieure et supérieure des axes (p.ex. `xlim=c(-3,5)`)

Tout appel de `plot()` efface le graphique précédent. Pour ajouter un nuage de point ou une courbe à un graphique existant :

- `points(x,y,...)` : ajoute un nuage de points
- `lines(x,y,...)` : ajoute une nouvelle courbe
- `abline(a,b,...)` : ajoute une droite avec ordonnée à l'origine `a` et pente `b`

Les 3 points `...` symbolisent ici des options graphiques qui peuvent être spécifiées.

Exercice 11
Ajouter la densité d'une loi normale $\mathcal{N}(2,0.8)$ au graphique de la densité $\mathcal{N}(0,1)$. Représenter la densité de $\mathcal{N}(2,0.8)$ par une courbe pointillée rouge. Utiliser les commandes `xlim` et `ylim` pour obtenir une représentation convenable des deux densités. Consulter l'aide de la fonction `legend()` pour créer une légende pour ce graphique.

Pour tracer plusieurs graphiques dans une même fenêtre, on utilise `par(mfrow=c(n,m))`, où `n` est le nombre de graphiques par ligne, et `m` le nombre de graphiques par colonne.
Exemple : Pour tracer 6 graphiques en une seule fenêtre, réparties sur deux lignes :

```
> par(mfrow=c(2,3))  # pour diviser la fen\^etre en 6 parties
> plot(x1,y1)        # tracer le 1e graphique
> plot(x2,y2)        # tracer le 2e graphique
...
```

Exercice 12
Tracer les deux densités de l'exercice précédent côte à côte dans une même fenêtre. Ajouter un titre à chaque graphique et veiller à ce que l'échelle des axes soit la même pour les deux graphiques.

Exercice 13
Proposer des représentations graphiques adéquates pour les données suivantes :

1. *la hauteur et la longueur des cathédrales (fichier cathedral.ods)*
2. *la hauteur, la longueur et le type de cathédrales (fichier cathedral.ods)*
3. *les données de temperature de 15 villes françaises (fichiers temperatures.ods et Villes.ods)*

Pour créer une nouvelle fenêtre graphique vide, taper la commande `figure()` sous Window, `X11()` sous Linux et `quartz()` sous Mac. Pour sauvegarder une figure dans un fichier pdf on pourra utiliser les commandes `pdf`() et `dev.off()` :

```
> pdf(file="graphique.pdf")
> plot(x,y)
> dev.off()
```