TP d'introduction à R : un langage de programmation

Attention : Ceci est un "TP-cours" où vous contruisez votre propre savoir (quelle chance !). Prenez des notes ! Posez des questions ! On ne devient pas R-guru en 3h sans mobilisation du cerveau...

1 Commencer avec R

R est un système d'analyse statistique et graphique créé par Ross Ihaka et Robert Gentleman. Il dérive du dialecte S créé par AT&T Bell Laboratories. R est distribué librement, sous licence GNU GPL.

Pour lancer le logiciel R depuis les machines du réseau, il suffit de taper "R" dans un terminal. Pour quitter la session, il suffit de taper la commande `quit()`.

R est un langage interprété, et non compilé, ce qui signifie que les commandes tapées au clavier sont directement exécutées sans avoir besoin d'écrire un programme complet comme dans les langages comme C, Fortran ou Java.

La syntaxe de R est relativement simple et intuitive. Les variables, données et fonctions du langages sont stockés dans la mémoire de l'ordinateur pendant tout le fonctionnement de la session, sous la forme d'*objets* portant chacun un *nom*. L'utilisateur agit sur ces objets avec des *opérateurs* (arithmétiques, logiques, de comparaison, ...) et des fonctions.

1.1 Un premier programme en R

Exercice 1 *Lancer R dans une console. Taper le code suivant :*

```
print("Bonjour!")
x <- 5
x
```

Tester la commande `quit()`. *Observer la possibilité de conserver les objets définis durant la session.*

R étant un langage interprété, il peut être utile de conserver dans un script à part les commandes que vous voulez utiliser, pour ne pas avoir à retaper toute la séquence et utiliser des fonctions intéressantes.

Exercice 2 *Lancer votre éditeur de texte favori, et tapez dedans la séquence d'instructions précédente. Sauvegardez dans un fichier "TP1.R". Pour charger le script dans l'interpréteur R, on utilise la commande* `source()`. *Rechargez votre script. Que constatez-vous ?*

Par la suite, vous sauvegarderez toutes vos définitions d'objets dans votre script.

1.2 Les variables

L'assignation d'une valeur à une variable se réalise grâce à la flèche : `<-`, dans un sens ou dans l'autre.

Exercice 3 *Observer ce que produisent les commandes suivantes tapées successivement :*

```
n <- 15
5 -> n
x <- 1
X <- 10
n
x
X
n <- 10 + 2
n
n <- 3 + rnorm(1)
(10 + 2)*5
5 < 7
name <- "Jules"; n1 <- 10; n2 <- 100; m<- 0.5
ls()
```

1.3 Créér, lister et effacer des données en mémoire

La fonction `ls()` permet de connaître, à un instant donné, les objets qui sont présents en mémoire.

La fonction `ls.str()` fournit des informations complémentaires sur les objets, et la fonction `ls(pat = "m")` va donner uniquement les objets dont les noms comportent un "m".

Exercice 4

1. *Testez les fonctions* `ls` *et* `ls.str` *avec différents patterns.*
2. *Que fait la sequence d'instructions suivante :*

```
M <- data.frame(n1, n2, m)
ls.str(pat = "M")
```

3. *Effacez tous les objets en mémoire à l'aide de la fonction* `rm`

1.4 L'aide en ligne

La fonction `help()` permet d'obtenir de l'aide sur la façon d'obtenir de l'aide... Pour résumer, il y a deux façons d'avoir une aide sur une fonction donnée, en tapant la commande précédée d'un point d'interrogation, ou en donnant la commande comme paramètre de l'aide. On peut aussi passer du mode aide en ligne au mode aide console en jouant sur le commutateur `htmlhelp={T,F}`.

Exercice 5 *Recherchez de l'aide sur la fonction* `rm`*. Comment détruit-on en une seule commande tous les objets de la session ? Comment détruit-on seulement les objets qui ont un "m" dans leur nom ? Testez l'aide en ligne* `help.start()`*.*

Dans ce qui suit, quand on spécifiera une fonction à utiliser pour résoudre un exercice, prenez l'habitude d'invoquer l'aide sur cette fonction pour comprendre son utilisation.

2 Variables : types, assignation, comparaison

En R, le type de base est le **vecteur**, dont nous reparlerons d'ici quelques sections. Les données dans R sont *typées*, c'est-à-dire que l'on ne sera pas autorisé à mélanger des booléens avec des données de type numériques, ou des chaînes de caractères. Voici les types de base :

- `numeric` qui peut être `integer` (entier) ou `double` (double). Les opérateurs applicables sont : `+`, `-`, `*`, `/`, `^` , `%%` (modulo), `%/%` (division entière)
- **logical** correspond au type booléen, il prend deux valeurs possibles `TRUE` et `FALSE` (en majuscule) que l'on peut abréger avec `T` et `F`. Les opérateurs sont `!` (négation), `&&` (ET logique), `||` (OU logique), `xor()` (OU exclusif).
- `character` désigne les chaînes de caractères. Une constante chaîne de caractère doit être délimitée par des guillemets. Fonctions de manipulation : `paste(..)`, `grep(..)`, etc.

Remarque : pour connaître la classe d'un objet i.e. le type associé à un objet, on utilise la fonction `class(nom_objet)`.

Les opérateurs de comparaison classiques `==`, `!=`, `>`, `>=`, etc., sont disponibles en R et renvoient une valeur de type `logical`.

Exercice 6 *Choisir trois valeurs numériques pour des variables a, b, et c. Calculer le discriminant delta correspondant à l'équation* $ax^2 + bx + c = 0$ *(rappel ? !* $\Delta = b^2 - 4ac$*. Afficher la valeur de delta, puis de delta* > 0*, delta* $= 0$*, delta* < 0*.*

3 Structures de contrôle et boucles en tout genre

L'exercice précédent vous a sans doute donné envie de proposer une réponse un peu plus élaborée à la question cruciale, "est-ce que l'équation $ax^2 + bx + c = 0$ admet des solutions dans $\mathbb{R}$?". Pour cela, il faut commencer à établir des conditions, ce qui se fait de façon fort classique avec l'instruction `if ... else ...`.

Exercice 7 *Lancer la commande* `help("if")` *et remarquez que l'aide sur les conditionnelles présente aussi la syntaxe des boucles. Notez celle-ci pour plus tard.*

Exercice 8 *Reprendre l'exercice sur la résolution de l'équation du second degré, en intégrant une conditionnelle pour tenir compte des trois cas, en affichant "Deux solutions réelles distinctes" quand delta* > 0*, "une solution réelle double" si delta* $= 0$ *et "pas de solution réelle, mais deux solutions complexes" sinon. Testez en changeant les valeurs de a, b, c pour couvrir tous les cas.*

Exercice 9 *Calculez la somme des 100 premiers nombres entiers à l'aide d'une boucle* `for`*, puis d'une boucle* `while`*. Nous verrons un peu plus tard comment faire cela en une seule commande !*

4 Fonctions

Vous avez certainement éprouvé de la frustration et un sentiment d'inutilité lors des tests de l'exercice 8. Changer de valeurs nécessite de recopier les lignes de commande contenant la conditionnelle. Pour outrepasser cette difficulté, on peut tout simplement définir un bloc ou une fonction. Un bloc permet de définir une succession d'instructions, que l'on peut nommer. Par exemple :

```
bloc <- {
if(delta == 0){print("Une racine double")}
else if (delta > 0) {print("Deux racines réelles")}
else {print("Deux racines complexes")}
}
```

Il aurait alors été suffisant de relancer uniquement l'affectation des variables a, b, c, le calcul de *delta* et l'évaluation de `bloc` lors des tests de l'exercice 8.

Mais pour aller plus loin, on peut créer une fonction, qui prendra en paramètre trois variables numériques, et affichera le message adéquat. Une fonction se déclare avec le mot clé `function`. Une fonction est un bloc d'expressions, forcément nommée, visible comme un objet en mémoire avec `ls()`, qui peut être supprimée avec `rm()`, comme tout autre objet. Elle prend des paramètres en entrée (non typés, le typage se fera dynamiquement lors de l'interprétation), et renvoie une valeur en sortie (même si on n'écrit pas spécifiquement d'instruction pour cela).

Exercice 10

1. *Déclarez la fonction suivante :*

```
petit <- function (a, b){
d <- ifelse(a < b, a, 0)
return(d)
}
```

2. *Observez la syntaxe du* `ifelse` *qui est encore une autre façon d'exprimer une conditionnelle (R est plein de surprises !)*
3. *Observez à l'aide de* `ls.str(pat="petit")` *quel est le type de cet objet.*
4. *Lancez cette fonction sur plusieurs paramètres, éventuellement de types saugrenus.*
5. *Ajoutez une instruction après le* `return` *et relancez la fonction. Que constatez-vous ?*
6. *Enfin, enlevez la ligne contenant le* `return`*, et relancez la fonction. Que constatez-vous ? Essayez maintenant* `class(petit(4,5))`*.*

Exercice 11 *Écrire une bonne fois pour toute une belle fonction qui résoud les équations du second degré !*

On peut également spécifier des paramètres par défaut, qui seront donc optionnels lors de l'appel. Attention, cependant, cela nécessite de faire des appels en explicitant les variables dont on précise la valeur. Par exemple, un tel appel pour la fonction `petit` serait : petit(a=2,b=3).

Exercice 12 *Écrire une fonction* `ma_somme` *qui prend en entrée un paramètre* `deb` *qui prendra une valeur par défaut de 1 et un paramètre* `fin` *qui prend comme valeur par défaut 100, et qui calcule la somme des entiers entre* `deb` *et* `fin`*. Testez différents types d'appels, en spécifiant ou non les variables...*

On peut jouer de façon très fine avec les fonctions en R. Par exemple, tout objet peut être paramètre d'une fonction, y compris une autre fonction. Par exemple :

```
#met au carré une variable
CARRE <- function(x){
return(x^2)
}
#met au cube
CUBE <- function(x){
return(x^3)
}
#somme de 1 à n
#avec une FONCTION générique
somme <- function(FONCTION,n){
s <- 0
for (i in 1:n){
s <- s + FONCTION(i)
}
return(s)
}
#appel avec carré
print(somme(CARRE,10)) # résultat = 385
#appel avec cube
print(somme(CUBE,10)) # résultat = 3025
```

Exercice 13 *Définir une fonction "encore plus générique" en vous inspirant de la précédente. Il s'agira de remplacer l'opérateur* `+` *intervenant dans la boucle par une autre fonction (binaire, c'est-à-dire qu'on supposera qu'elle prend deux paramètres) passée en paramètre. De même, il faudra pouvoir passer en paramètre la valeur initiale de* `s`*. Testez avec le produit, pour une valeur initiale 1 et la fonction* `CARRE`*.*

5 Structures de données vecteurs, matrices et dataframe

Nous allons maintenant nous intéresser (longuement, mais c'est important !) à la façon dont R organise les données. Au début du TP, il est précisé que le type de base est le vecteur. Pour bien s'en convaincre, il suffit d'interpréter une variable de type quelconque :

```
> x = 5
> x
[1] 5
```

Le `[1]` devant la valeur de x signifie en réalité qu'on a affaire à une ligne de vecteur, qui commence à l'indice 1.

5.1 Les vecteurs sous toutes les coutures

La function qui permet de créer des vecteurs est la fonction `c()`. Par exemple :

```
monvec = c(1,5,8,23,4,90,12)
```

Exercice 14 *Observez la séquence d'instructions suivantes, en notant pour chacune d'entre elle ce qu'elle produit. N'oubliez pas que vous pouvez vous servir de l'aide de R !*

```
monvec = c(1,5,8,23,4,90,12)
monvec
class(monvec)
length(monvec)
monvec[4]
masequence = 1:10
masequence
monvec + masequence
masequence = 1:length(monvec)
monvec + masequence
monvec < masequence
sum(monvec)
prod(monvec)
monvec * monvec
```

```
grosvec = 1:200
grosvec
rep(6,3)
```

Comme vous pouvez le constater, les vecteurs sont très souples d'utilisation ! C'est pourquoi il est bien important de documenter à l'aide de commentaires l'utilisation que l'on en fait.

Exercice 15 *Écrire une seule instruction permettant de résoudre l'exercice 9.*

On peut sélectionner certains élements dans les vecteurs, en indiquant entre crochets soit l'ensemble des indices sélectionnés, soit une condition qui doit être vérifiée par les élements sélectionnés. Dans ce dernier cas, on parle de "filtre".

Exercice 16 *Observer le comportement des instructions suivantes, puis écrire une fonction qui prend en entrée un vecteur numérique et renvoie en sortie les éléments pairs de ce vecteur.*

```
vect = c(27,32,56,78,11,45)
vect[c(2,4)]
vect[1:3]
vect[vect>40]
```

Exercice 17 *Testez les fonctions* `min()`, `max()`, `sort()`, `rank()`, `order()`. *Si x est un vecteur numérique, que fait* `x[order(x)]` *?*

Pour faciliter l'utilisation des vecteurs en donnant une sémantique à leurs éléments, on peut nommer les éléments d'un vecteur à l'aide de la fonction `names`.

Exercice 18 *Explorez l'aide de la fonction* `names` *et créez un vecteur contenant les valeurs 1 à 7, avec les noms respectifs des jours de la semaine. Il est alors possible d'accéder directement au numéro du jour de la semaine en sélectionnant le nom entre crochet dans le vecteur. Expérimentez.*

Les fonctions usuelles permettant de calculer des observations statistiques sur les vecteurs sont très utiles en R. Par exemple, on dispose des fonctions `mean` et `sd` (pour standard deviation) qui calculent respectivement la moyenne et l'écart-type des vecteurs qu'elles prennent en paramètre.

Exercice 19 *Calculez la moyenne et l'écart-type des valeurs paires du vecteur* `vect` *de l'exercice 16.*

5.2 Les matrices, vecteurs à deux dimensions

Pour construire une matrice, on utilise la fonction `matrix`, qui permet de créer la matrice à partir d'un vecteur. Par exemple :

```
v <- c(1.2,2.3,4.1,2.5,1.4,2.7)
m <- matrix(v,nrow=2,ncol=3)
attributes(m)
print(m)
```

Par défaut, les matrices sont organisées par colonne, mais on peut les organiser par ligne, en utilisant l'argument `byrow=TRUE`.

Exercice 20 *Créez une matrice m contenant les entiers de 1 à 9, organisée en lignes de 3 colonnes. L'argument* `nrow` *sera alors optionnel.*

Exercice 21 *Encore un peu d'observation... Que font les instructions suivantes :*

```
m = matrix(c(1,5,78,2,3,8,34,67,10,3,45,6),byrow=T,ncol=3)
m
nrow(m)
ncol(m)
2*m
m[1,3]
m[4]
m[1,]
m[,1]
m[,2:3]
cbind(m, matrix(1:12,byrow=TRUE,ncol=3))
rbind(m, matrix(1:6,byrow=TRUE,ncol=3))
```

On peut attribuer des noms aux lignes et aux colonnes avec `rownames()` et `colnames()`. Cela peut-être très intéressant quand on récupère des données depuis un fichier csv.

La fonction `apply()` permet d'appliquer une fonction à chaque ligne ou chaque colonne d'une matrice. Ses arguments sont :

1. le nom de la matrice
2. un nombre pour dire si la fonction doit s'appliquer aux lignes (1), aux colonnes (2) ou aux deux (c(1,2))
3. le nom de la fonction à appliquer

Par exemple, pour calculer la somme de chaque ligne ou de chaque colonne d'un tableau :

```
# On crée d'abord une matrice avec 2 lignes et 3 colonnes
data<-matrix(c(1,2,3,4,5,6), nrow=2)
# On donne un nom aux lignes et aux colonnes
colnames(data)=c("C1","C2","C3")
rownames(data)=c("L1","L2")
# On utilise la fonction apply() pour faire la somme de chaque ligne
apply(data, 1, sum)
# Pour faire la somme de chaque colonne, on remplace 1 par 2
apply(data, 2, sum)
```

Cette fonction est très puissante !

Exercice 22 *Générez un vecteur de taille 100 au hasard avec la fonction* `rnorm` *(l'aide est votre amie !). Puis créez une matrice de 10 lignes et 10 colonnes à partir de ce vecteur. Enfin, calculez la moyenne par ligne, et par colonne, de cette matrice.*

5.3 Les dataframes, structure de stockage par excellence

Les vecteurs et les matrices ont vocation à conserver des données homogènes, c'est-à-dire de même type. Mais les vraies données utilisées en statistiques sont hétérogènes. Il va donc falloir en disposer sous forme de listes de vecteurs, que l'on va appeler des dataframes. Les dataframes vont se manipuler de façon très semblable aux matrices, avec la possibilité de faire un accès direct aux colonnes à l'aide de l'opérateur `$`.

Nous allons jouer avec un dataframe qui est chargé de base dans l'interpréteur R. Il s'agit du dataframe `mtcars`.

Exercice 23 *Tapez* `mtcars` *dans l'interpréteur. Que constatez-vous ? Testez les commandes suivantes :*

```
ls(mtcars)
mtcars$mpg
colnames(mtcars)
rownames(mtcars)
head(mtcars,5)
subset(mtcars,mtcars$vs==0)
summary(mtcars)
```

Exercice 24 *Calculez la moyenne des colonnes* `mpg`*,* `disp`*, et* `hp`*, pour les modèles qui ont 8 cylindres.*

Pour conclure ce TP en beauté, nous allons apprendre comment communiquer avec le monde extérieur depuis l'interpréteur R. En effet, les données sont rarement stockées directement dans l'interpréteur, mais le plus souvent dans des fichiers tabulés (format csv, ods, xls, etc.).

Pour charger un fichier tabulé, on utilise la fonction `read.table()`, et pour exporter un dataframe dans un fichier tabulé, la fonction `write.table()`.

Exercice 25 *Récupérez le fichier tabulé "mais.txt" sur l'espace Moodle du cours. Chargez ce fichier avec la commande :*

```
mais = read.table("mais.txt", sep="\t",header=TRUE)
```

Quelles sont les variables représentées dans ce tableau de données ? Lesquelles sont quantitatives, et lesquelles sont qualitatives (indice, la fonction `summary` *va vous aider) ?*

Nous continuerons à jouer avec ce jeu de données dans un prochain TP... Quittez la session en sauvegardant votre espace de travail, vous gagnerez du temps !