Master 1 MAS - (Re)prise en main de R

Magalie Fromont et Ioana Gavra

Présentation générale

R est un système d'analyse statistique et de traitement des données créé par Ross Ihaka et Robert Gentleman (Université d'Auckland, Nouvelle-Zélande).

C'est à la fois un logiciel et un langage de programmation interprété basé sur la notion de vecteur, proche du langage S développé par R. Becker, J. Chambers et A. Wilks des Bell laboratories, utilisable sur la plupart des systèmes d'exploitation.

R est diffusé selon le principe de licence GNU. Il s'agit donc d'un logiciel libre, signifiant qu'il est téléchargeable gratuitement (sur le site officiel du CRAN - Comprehensive R Archive Network- https://cran.r-project.org), mais aussi que son code source est accessible, réutilisable ou modifiable en vue d'une diffusion publique. Son développement est à présent géré par la *R Development Core Team* dont les principales tâches consistent à proposer des mises à jour régulières du logiciel et de sa documentation.

R est un langage de programmation interprété basé sur la notion de vecteur.

Le socle commun de R, mis à dispositin lors du téléchargement, contient un certain nombre de fonctions mathématiques et statistiques de première nécessité (somme, produit, moyenne, variance...) regroupées dans une bibliothèque ou librairie ou paquet (*package* en anglais) nommée simplement 'base'.

Par son caractère libre, R peut être enrichi individuellement **pour chaque nouvelle session de travail** de nombreuses autres librairies développées par une communauté très active d'utilisateurs et d'utilisatrices au gré des avancées statistiques.

Attention : ces librairies additionnelles doivent être installées d'abord, puis appelées à chaque nouvelle session.

Quelques références :

- Plusieurs documents sont disponibles sur le site du CRAN ;

- [Analyse R] (https://larmarange.github.io/analyse-R) ;
- Introduction à la programmation en R, de V. Goulet ;
- Statistiques avec R, de P. A. Cornillon, A. Guyader, F. Husson, N. Jégou, J. Josse, M. Kloareg, E. Matzner-Løber, L. Rouvière (2013) Presses Universitaires de Rennes, France ;
- Le logiciel R, de P. Lafaye de Micheaux, R. Drouilhet, B. Liquet (2014).Springer-Verlag France ;
- Logiciel R et programmation, de E. Gallic ;
- Introduction à R, de M. Fromont.

L'interface RStudio et les sessions de travail

L'interface RStudio

L'interface RStudio qui sera utilisée dans ce cours est divisée en 4 fenêtres :

- En haut à gauche - fenêtre de script : c'est un éditeur de texte permettant de taper ses commandes et programmes dans des scripts ou des projets pouvant être sauvegardés sur son espace personnel ;
- En bas à gauche - fenêtre de console : c'est la partie logiciel dans laquelle les commandes peuvent être exécutées et où apparaissent les résultats ;
- En haut à droite - fenêtre environnement : y sont recensés les données et résultats ainsi que les fonctions gardées en mémoire ;
- En bas à droite - fenêtre gestion des sorties : constituée de plusieurs onglets, elle permet d'accéder au répertoire courant ou répertoire de travail dans l'onglet **Files**, de voir les sorties graphiques dans l'onglet **Plots**, de voir les packages installés et chargés dans l'onglet **Packages** et d'accéder à l'aide en ligne dans l'onglet **Help**.

Ànoter que ces fenêtres peuvent être agencées différemment.

RStudio peut être téléchargé gratuitement à l'adresse: https://www.rstudio.com/.

Les sessions de travail

Ouvrir RStudio permet d'ouvrir une session de travail R.

Dans la console, le prompt > s'affiche à la fin du message de démarrage, indiquant que R attend une instruction ou commande.

Une instruction est exécutée dans la console en appuyant sur la touche 'Entrée'.

Si l'instruction est complète, le prompt s'affiche à nouveau.

Si l'instruction est incomplète, le prompt est remplacé par + : on peut alors soit compléter l'instruction puis exécuter, soit cliquer sur STOP ou appuyer sur Echap.

Si l'instruction est erronée, un message d'erreur apparaît.

Les flèches au clavier permettent de rappeler des instructions.

'history(n)' permet de récupérer l'historique des n dernières instructions (on peut aussi retrouver l'historique complète depuis l'onglet **History**).

Le caractère spécial ';' permet de séparer des instructions sur une même ligne ou d'exécuter une instruction sans en afficher le résultat, ♯ permet de commenter (la ligne qui suit ce caractère n'est pas comprise comme une instruction).

Le répertoire courant ou répertoire de travail est le dossier à partir duquel le logiciel va chercher les fichiers de scripts et de données. Tout ce qui a été fait au cours d'une session de travail peut être enregistré dans ce répertoire courant (données, historique des fonctions…) : cette possibilité est offerte à l'issue de la session, lorsque l'on quitte R.

On peut connaître le répertoire courant soit dans l'onglet **Files**, soit avec la commande `getwd()`, et le changer soit à partir de l'outil **Session** de l'interface, soit avec la commande `setwd()`.

Attention : les répertoires dans lesquels sont enregistrés les scripts et projets sont indépendants du répertoire courant.

Au cours d'une session, un nouveau script peut être créé et enregistré sous le nom de son choix dans un répertoire de son choix (pas nécessairement le répertoire courant), depuis le Menu **File**.

Un ancien script peut être récupéré depuis le Menu **File** ou en cliquant sur l'onglet correspondant dans l'éditeur de texte s'il apparaît.

Exercice 1. Ouvrir RStudio et regarder quel est le répertoire courant. Changer ce répertoire courant pour un répertoire personnel de son choix (un dossier spécifique dédié au cours de logiciel R en Master 1 pourra être créé au préalable) en testant les trois méthodes possibles : utilisation de l'onglet **Files**, exécution commande dans la console, exécution de commande à partir d'un script R que l'on aura pris soin de sauvegarder dans un répertoire de son choix.

Pour clore une session de travail, il suffit de quitter R depuis la console avec la commande `q()` ou en fermant l'application RStudio de façon classique.

On peut alors choisir de sauvegarder ou non l'image de la session (objets créés et historique des commandes) dans des fichiers cachés `.RData` et `.Rhistory` du répertoire courant.

Pour charger dans une nouvelle session les objets contenus dans `.RData` et/ou l'historique contenue dans `.Rhistory` : depuis la console, on utilisera `load(".RData")` et `loadhistory(".Rhistory")` ; depuis l'onglet **Files**, on cliquera sur `.RData` ou `.Rhistory`.

On peut aussi utiliser `save.image ("../travail.RData")` pour sauvegarder les objets de la session dans un fichier de son choix, puis `load ("../travail.RData")` pour charger ces objets dans une nouvelle session.

Attention : les chemins dans R sont écrits à l'aide d'un slash et non d'un anti-slash !

À propos des entrées/sorties

Pour gérer les entrées/sorties au sein des programmes :

- `source("mescommandes.txt")` permet d'exécuter les commandes contenues dans le fichier "mescommandes.txt" ;
- `sink("messorties.txt")` permet de rediriger les sorties vers le fichier "messorties.txt" jusqu'à l'arrêt de la redirection ordonné par `sink()`.

À propos de l'aide

Consulter l'aide de R est **indispensable**. Pour ce faire, on peut soit utiliser l'onglet **Help**, soit exécuter une commande spécifique.

Exercice 2. Exécuter et commenter les commandes suivantes :

```
help.start()

help(package = "base")
library(help = "base")

help(mean)
help.search("mean")
?mean
??mean
help(for)
help("for")
```

On peut également chercher de l'aide en ligne dans :

- le moteur de recherche de son choix ou un moteur de recherche spécifique à R (basé sur google) : http://www.rseek.org/ (sur rseek les requêtes sont à formuler en anglais).
- des cheat sheets (antisèches) : https://www.rstudio.com/resources/cheatsheets/
- des forums ou groupes d'échanges comme `Grrr` sur Slack (francophone) ou `StackOverflow` (anglophone).

À propos des packages

`library()` permet de visualiser tous les packages disponibles.

`library(MASS)` permet d'appeler le package additionnel MASS dans la session de travail.

Les outils de l'onglet **Packages** permettent également de charger des packages installés (en cochantla case du package installé souhaité) ou d'installer de nouveaux packages.

Premiers pas dans R avec quelques calculs simples

Exercice 3. Exécuter dans un script et/ou dans la console les commandes suivantes :

```
2+2
# blabla
blabla
2+2 # ceci est une addition
```

R respecte les priorités opératoires :

```
1+3/5*5
(1+3/5)*5
```

R connaît quelques fonctions numériques et constantes mathématiques de base :

```
pi
exp(2)
log(10)
factorial(4)
log(0)
sin(5*pi)
```

Commenter les deux derniers résultats : que s'attendait-on à obtenir ?

Exécuter les 4 lignes de code suivantes et commenter.

```
0.1+0.2==0.3
0.6+0.1+0.3==0.1+0.7+0.2
0.6+0.1+0.3==0.2+0.7+0.1
all.equal(0.6+0.1+0.3,0.2+0.7+0.1)
```

Pour visualiser la valeur exacte d'un vecteur numérique dans R, on puet utiliser :

```r
sprintf("%.54f",0.1)
sprintf("%.54f",0.2)
sprintf("%.54f",0.3)
sprintf("%.54f",0.1+0.2)
sprintf("%.54f",0.6+0.1+0.3)
sprintf("%.54f",0.1+0.7+0.2)
sprintf("%.54f",0.6+0.1+0.3)
sprintf("%.54f",0.2+0.7+0.1)
```

Objets de R

Création, modification, suppression

Afin de faciliter l'utilisation répétée d'une même quantité, on peut *affecter* un nom à une valeur ou au résultat d'une opération à l'aide des symboles flèches `<-` ou `->` ou tout simplement égal `=`.

Si aucun objet portant ce nom n'existe, l'affectation le crée. Sinon, elle modifie la valeur de l'objet.

Exercice 4. Exécuter et commenter les commandes suivantes :

```r
x <- 3*5 + sin(5*pi/6)
print(x)
x
y <- x
y
(x <- exp(3)+2)
x
x*3
(z = log(x-2))
x
y->x
```

Remarque sur les noms d'objets : Pour les noms d'objets, on peut utiliser toutes les lettres minuscules a–z et majuscules A–Z (qui se distinguent dans R), les chiffres 0–9, le point « . » et le caractère « _ ». En fonction de la configuration de l'ordinateur on peut utiliser aussi des lettres accentuées, mais il est fortement déconseillé de le faire.

R stocke en mémoire tous les objets de la session. On peut visualiser les objets gardés en mémoire avec la fonction `objects()` ou avec la fonction `ls()`.

Autres commandes pour visualiser des objets :

- `ls(pat="a")` affiche la liste des objets contenant le caractère “a” ;
- `ls(pat="$\;\hat{}\;$a")` afficher la liste des objets commençant par “a” ;
- `ls.str()` affiche la liste des objets en mémoire avec des détails sur ces objets.

Pour supprimer un objet on peut utiliser la fonction `rm()` :

```
z
rm(z)
z
rm(x,y)
```

Pour supprimer tous les objets de la session on peut utiliser `rm(list=ls())`.

Classes et attributs

R comprend plusieurs classes d'objets dont les vecteurs, les facteurs, les arrays, les matrices, les data frames, les listes, les data tables, les tibbles, les ts…

Pour connaître la classe d'un objet `x`, on utilise la commande `class(x)`. On peut savoir si un objet x est d'une certaine classe à l'aide des commandes `is.vector(x)`, `is.factor(x)`, `is.matrix(x)`, `is.data.frame(x)`… (la réponse est `TRUE` ou `FALSE`).

Chaque objet est caractérisé par son nom, son contenu, mais aussi par des attributs, dont le mode (nul : `NULL` ; numérique : `1, 2.333, pi, Inf, -Inf, 2.1e23`,… ; caractère : “blabla” ; complexe : `2+0i, 2i`; logique : `TRUE, FALSE`) et la longueur. Pour connaître ces deux attributs, on utilise les commandes `mode(x)` ou `typeof(x)`, et `length(x)`. Pour connaître les autres attributs : `attributes(x)`.

Les valeurs manquantes

Elles sont notées `NA` (not available) ou `NaN` (not a number). On peut savoir si un objet est composé de valeurs manquantes à l'aide de `is.na(x)` ou `is.nan(x)`. Ces commandes renvoient un objet de mode logique de même longueur que `x`.

`na.fail(x)` retourne un message d'erreur si l'objet `x` contient au moins une valeur manquante.

Exercice 5. Commenter les résultats des commandes suivantes :

```
log(-2)
 NA+1+10
 a <- NA+2
```

```r
a
a == NA
is.na(a)
typeof(a)
```

Type `numeric` : `integer` et `double`

Exercice 6. Exécuter et commenter les commandes suivantes :

```r
a <- 2
mode(a)
typeof(a)
b <- as.integer(a)
mode(b)
typeof(b)
c <- pi
as.integer(c)
is.integer(a)
is.numeric(c)
```

Type `complex`

On peut aussi générer et travailler avec des nombres complexes sur R, soit à l'aide du duo partie réelle/partie imaginaire soit module/argument. L'unité imaginaire est notée `i`.

Exercice 7. Exécuter et commenter les commandes suivantes :

```r
a <- complex(1,1,-1)
1-i
1-1i
mode(a)
typeof(a)
Re(a)
Im(a)
as.integer(2.5+3i)
```

Quel est l'argument de `a` ? Définir `b=1+i` en utilisant son module et son argument.

```
complex(1, argument=pi/4, modulus=sqrt(2))
Mod(a)
Arg(a)
-pi/4
```

Type `character`

Les éléments de type `character` sont placés entre guillemets (simples ou doubles)

Exercice 8. Exécuter et commenter les commandes suivantes :

```
a <- "Logiciel R"
mode(a)
b <- 'Logiciel R'
b
```

Type `NULL`

Cela représente un type et un objet en soi : l'objet vide. Il est de longueur 0.

Exercice 9. Exécuter et commenter les commandes suivantes :

```
a <- NULL
a
mode(a)
b <- null
b <- "null"
mode(b)
NULL <- 4
```

Type `logical`

Très utiles en programmation, les quantités de type logique prennent 2 valeurs : `TRUE` ou `FALSE` qu'on peut abréger respectivement en `T` ou `F`.

Exercice 10. Exécuter et commenter les commandes suivantes :

```
a <- TRUE
mode(a)
is.numeric(a)
```

```r
b <- F
b
FALSE+TRUE+TRUE
mode(FALSE+TRUE)
```

Attention à l'utilisation des abréviations :

```r
F=TRUE
F*T
rm(F)
F*T
```

Exercice 11. À quels opérateurs correspondent les commandes suivantes ?

```r
!a
(TRUE)&(FALSE)
(TRUE)|(FALSE)
(TRUE)|(TRUE)
xor(TRUE,FALSE)
xor(TRUE,TRUE)
```

Conversions

Exercice 12. À l'aide des fonctions `as.numeric()`, `as.character()`, `as.logical()`, compléter le tableau de conversion suivant :

De	En	Description du résultat
logical	numeric	
logical	character	
numeric	character	
numeric	logical	
character	logical	
character	numeric	

Exercice 13. Exécuter et commenter les commandes suivantes :

```r
as.logical(5.1)
as.character(T)
as.numeric("2.5")
```

Quelques classes d'objets importantes

Les vecteurs

Dans R, un vecteur est une suite de données de même type. C'est la structure privilégiée du logiciel R. La grande majorité des fonctions sont optimisées pour l'utilisation avec des vecteurs.

Création de vecteurs

Exemples de vecteurs numériques :

```r
x <- c(1,2,3)
class(x)
is.vector(x)
y <- 1:3
class(y)
typeof(x)
typeof(y)
```

Afficher l'espace mémoire occupé par `x` et `y` à l'aide de la fonction `object.size()`.

```r
object.size(x)
object.size(y)
# Le type `double` utilise plus de mémoire que le type `integer` (voir _Le logiciel R_, de
```

Remarque : en terme d'affichage, `x` et `y` sont les mêmes, mais les deux vecteurs sont stockés en mémoire de façon différente.

On peut créer facilement des séquences régulières également à l'aide de fonctions comme `seq`, `sequence` et `rep` ou de l'opérateur `:`.

Exercice 14. Exécuter et commenter les commandes suivantes :

```r
z <- -3:4
t <- -(3:4)
```

```
pi:7
rep(0,10)
rep(c(3,7),4)
rep(c(3,7),c(7,3))
seq(0,1,length.out=100)
seq(0,1,by=0.01)
seq(0,1,length=1000)
sequence(5)
```

Attention à l'ordre des opérations pour l'opérateur : !

```
2*4:6; 3:5+1 # on peut remarquer que l'opérateur ":" est prioritaire
n <- 10
1:n-1
1:(n-1)
```

Exemples de vecteurs de caractères :

```
noms <- c("Pierre","Paul","Jacques")
mode(noms)
rep('A',3)
paste('x',1:3,sep='_')
```

Exemples de vecteurs logiques :

```
test <- z>0
class(test)
v <- c(T,F,T)
class(v)
```

Les vecteurs peuvent aussi être définis de façon interactive :

```
(new_vect=scan(n=4))
```

Exercice 15. Créer les vecteurs suivants : * le vecteur des entiers de 5 à 23, * le vecteur de 6 à 20 allant de 2 en 2 * le vecteur $(0, 0.01, 0.02, 0.03, \ldots, 0.99, 1)$ (de deux manières différentes) * le vecteur : (1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5) * le vecteur : (1 1 1 2 2 2 3 3 3 4 4 4 5 5 5) * le vecteur : (1 1 2 2 2 3 3 3 3) * le vecteur : (Individu1 Individu2 … Individu100) * le vecteur : ("A" "B" "B" "C" "C" "C" … "Z" "Z" "Z") (terminant par 26 fois la lettre Z).

On peut générer des vecteurs contenant des données aléatoires à l'aide de la commande `rfun(n,...)`, où `fun` indique la loi de probabilité, `n` le nombre de données générées, `...`

les paramètres de la loi de probabilité. Exemples : `rnorm(n,0,1)`, `rexp(n,1)`, `rpois(n,2)`, `rchisq(n,100)`, `rt(n,100)`, `rf(n,100,50)`, `rbinom(n,100,0.5)`, `runif(n,0,1)`}…\

Remarque : On peut remplacer dans ces fonctions la lettre `r` par `d`, `p` ou `q` pour obtenir la densité de probabilité, la fonction de répartition ou la valeur des quantiles.

Exercice 16. Aller dans l'aide de ces fonctions. Sur quoi faut-il être particulièrement vigilant.e lorsqu'on utilise ces fonctions ?

Opérations sur les vecteurs numériques

Les opérateurs arithmétiques classiques sont donnés par + (addition), - (soustraction), * (multiplication), / (division), ^(puissance), %% (congruence), %/% (division entière).

Les opérateurs de comparaison sont donnés par < (inférieur à), > (supérieur à),<= (inférieur ou égal à), >= (supérieur ou égal à), == (égal à), ! = (différent de).

Ces opérateurs agissent sur les vecteurs élément par élément. Pour effectuer une comparaison globale de deux objets, on peut utiliser `identical`.

Pour réaliser du calcul vectoriel "classique", on utilisera l'opérateur `\%*\%` par exemple.

Opérations sur les vecteurs logiques

Les opérateurs logiques sont donnés par `!x` (non logique), `x$\&$y` (et logique opérant sur tous les éléments de x et y), `x$\&\&$y` (et logique opérant sur le premier élément de x et y), `x$|$y` (ou logique opérant sur tous les éléments de x et y), `x$\|$y` (ou logique opérant sur le premier élément), `xor(x,y)` (ou exclusif).

On code l'inégalité 0 <x< 1 par `x>0 $\&$ x<1`.

On peut utiliser les opérateurs arithmétiques sur des vecteurs logiques, les {FALSE} étant transformés en 0, les {TRUE} en 1.

Sélection/extraction dans un vecteur

Pour accéder à un ou plusieurs éléments d'un vecteur, on se sert des crochets (`[...]`). Il existe principalement deux manières de sélectionner certains éléments d'un vecteur (ou d'en supprimer) : soit en précisant l'indice des éléments à sélectionner ou à supprimer, soit à l'aide d'un vecteur logique.

Exemples :

```r
x <- c(1,2.5,3.9,pi,8)
x[2]
x[c(1,2,3)]
x[1:3]
x[c(2,2,1,3)]
x[c(1:3,2,1)]
x[-1]
x[-c(1,2)]
x[-(1:2)]

x>3
x[x>3]
```

Exercice 17.

1. Créer le vecteur `v`: `0 1 2 3 4 5 NA 0 1 2 3 4 5 NA 0 1 2 3 4 5 NA`.
2. Remplacer les valeurs `NA` par -1.
3. Sélectionner les valeurs strictmnt supérieures à 3.
4. Sélectionner les valeurs positives et inférieures ou égales à 4.
5. Sélectionner les valeurs qui sont soit < 2 soit ≥ 4
6. Enlever les éléments nuls de `v`.
7. Remplacer les éléments négatifs par leur opposé.
8. Rajouter les valeurs `6,7,8,9` à la fin du vecteur `v`.
9. Enlever les 3 premiers éléments de `v`.
10. Afficher la position du maximum de `v`.
11. Modifier le vecteur `v` pour que sa valeur minimale soit augmentée de 1. Afficher le minimum avant et après pour vérifier.

Remarque : nous reviendrons sur cette partie lorsque nous introduirons le package `dplyr`.

Quelques fonctions pour les vecteurs numériques :

- `sum(x)` calcule la somme des éléments de `x`.
- `prod(x)` calcule le produit des éléments de `x`.
- `diff(x)` calcule la différence entre les éléments consécutifs de x.
- `max(x)` donne le maximum des éléments de `x`.
- `min(x)` donne le minimum des éléments de `x`.
- `which.max(x)` donne l'indice du maximum des éléments de `x`
- `which.min(x)` donne l'indice du minimum des éléments de `x`.
- `range(x)` donne le minimum et le maximum des éléments de `x`.
- `length(x)` donne le nombre d'éléments dans `x`.

- `mean(x)` calcule la moyenne des éléments de `x`.
- `median(x)` calcule la médiane des éléments de `x`.
- `var(x), cov(x)` calcule la variance empirique des éléments de `x`.
- `var(x,y), cov(x,y)` calcule la covariance entre `x` et `y`.
- `corr(x,y)` calcule la corrélation linéaire entre `x` et `y`.
- `sd(x)` calcule l'écart type empirique de `x`.
- `quantile(x)` calcule le quantile empirique de `x`.
- `round(x,n)` arrondit les éléments de `x` à `n` chiffres après la virgule.
- `rev(x)` inverse l'ordre des éléments de `x`.
- `sort(x)` ordonne les éléments de `x`.
- `rev(sort(x))` ordonne les éléments de `x` dans l'ordre descendant.
- `rank(x)` donne les rangs des éléments de `x`.
- `order(x)` donne les coordonnées du plus petit élément de `x`, du deuxième plus petit élément de `x`, etc.
- `pmin(x,y), pmax(x,y)}` donne un vecteur dont le ième élément est le minimum ou le maximum entre `x[i]` et `y[i]`. `cumsum(x), cumprod(x)` donne un vecteur dont le ième élément est la somme ou le produit des éléments `x[1]` à `x[i]`.
- `cummin(x), cummax(x)` donne un vecteur dont le ième élément est le minimum ou le maximum des éléments `x[1]` à `x[i]`.
- `match(x,y)` donne un vecteur de même longueur que `x` contenant les éléments de `x` qui sont dans `y` (`NA` sinon).
- `which(x==2)` donne les indices des éléments de `x` égaux à 2.
- `choose(n,k)` calcule les combinaisons de paramètres `n` et `k`.
- `unique(x)` supprime les éléments dupliqués de `x`.
- `table(x)` crée un tableau des effectifs des différentes valeurs de `x`.
- `sample(x,k)` ré-échantillonne aléatoirement et sans remise k éléments dans `x`.
- `sample(x,k,REPLACE=TRUE)` ré-échantillonne aléatoirement et avec remise `k` éléments dans `x`.

Quelques fonctions pour les vecteurs logiques :

- `all(x)` teste si tous les éléments de `x` sont `TRUE`.
- `any(x)` teste si au moins un des éléments de `x` est `TRUE`.