

Introduction à \mathbb{R}

Florence Yerly

Dept. de mathématiques, Université de Fribourg (CH)

SP 2013

Qu'est ce que R ?

- Un logiciel de statistiques libre et gratuit ;
- Un logiciel multi-plateforme (UNIX, Windows MacOS X)
- R permet de faire des calculs statistiques avancés ainsi que des graphiques de haute qualité ;
- Très utilisé dans les domaines académiques, mais aussi dans le domaine privé ;
- Un clone du logiciel commercial S+.

On peut télécharger R ici : <http://cran.r-project.org>

On y trouve aussi des packages supplémentaires, des tutoriels (en plusieurs langues), etc.

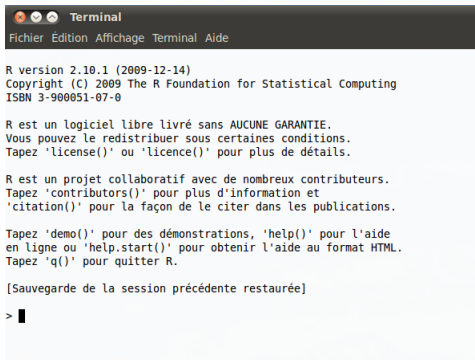
Avantages :

- Logiciel libre, donc tout le monde peut contribuer et modifier les fichiers à sa convenance.
- Rapidité d'insertion des nouveaux outils statistiques

Désavantages :

- L'interface graphique est très pauvre.
- Pas vraiment d'unité entre les différents packages.

Voici à quoi ressemble R :



```
Terminal
Fichier Édition Affichage Terminal Aide

R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

[Sauvegarde de la session précédente restaurée]

> █
```

- > invite l'utilisateur à entrer une commande.
- + indique que la commande précédente n'est pas terminée. On peut stopper un processus avec la combinaison Ctrl+C.
- q() est la commande pour quitter R, on peut ensuite sauver ou non son travail en tapant y/n/c (oui, non, annuler)

- Tous les objets créés en R sont stocké (si demandé) dans un fichier `.RData`.
 - > `ls()` donne la liste des objets existants
 - > `rm(x)` enlève l'objet x
 - > `rm(list=ls())` enlève tous les objets existants
- On navigue dans les anciennes commandes avec les flèches \uparrow et \downarrow du clavier.
- `getwd()` donne le dossier de travail de R, on peut le modifier en utilisant `setwd(path)`.

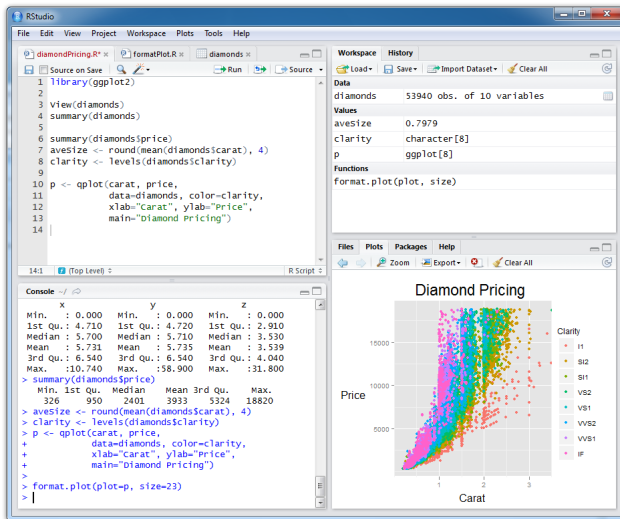
R se résume à une console qui exécute des commandes. Pour garder une trace du travail, il est vivement conseillé d'écrire les commandes dans un script et ensuite de les exécuter dans la console.

Pour créer un script, un simple éditeur de texte suffit (Bloc-Note de Windows, gedit, par ex.) et d'enregistrer le fichier avec l'extension `.r`.

La commande `source(file.r)` permet d'exécuter un script `file.r` dans la console. Les commentaires sont signalés par `#`.

On peut aussi utiliser une interface graphique pour R, par exemple RStudio (multiplateforme) <http://rstudio.org>

RStudio : une très bonne interface graphique.



The screenshot displays the RStudio environment with the following components:

- Source Editor:** Contains R code for loading data, summarizing it, and creating a faceted scatter plot.
- Console:** Shows the execution output, including summary statistics for 'x', 'y', and 'z', and the results of the plotting commands.
- Workspace:** Lists the loaded data object 'diamonds' and the function 'format.plot'.
- Plots Panel:** Displays a scatter plot titled 'Diamond Pricing' with 'Price' on the y-axis and 'Carat' on the x-axis. Points are colored by 'Clarity'.

```
1 library(ggplot2)
2
3 View(diamonds)
4 summary(diamonds)
5
6 summary(diamonds$price)
7 aveSize <- round(mean(diamonds$carat), 4)
8 clarity <- levels(diamonds$clarity)
9
10 p <- qplot(carat, price,
11            data=diamonds, color=clarity,
12            xlab="carat", ylab="Price",
13            main="Diamond Pricing")
14
```

Console Output:

```
Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
Median : 5.700   Median : 5.710   Median : 3.530
Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  326   950   2401   3933   5324  18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+           data=diamonds, color=clarity,
+           xlab="carat", ylab="Price",
+           main="Diamond Pricing")
>
> format.plot(plot=p, size=23)
>
```

Workspace:

Data	Value
diamonds	53940 obs. of 10 variables
aveSize	0.7979
clarity	character[8]
p	ggplot[8]

Plots Panel:

Diamond Pricing

Price

Carat

Clarity

- I1
- SI2
- SI1
- VS2
- VS1
- VVS2
- VVS1
- IF

- > `help.start(.)` lance l'aide HTML en ligne de R.
- > `help(help)` documentation de la fonction `help`.
- > `?help` idem
- > `help.search("test")` cherche dans tous les packages installés les fonctions qui correspondent à `test`.
- > `example(mean)` exécute les exemples liés à la fonction `mean`.

- R ignore les espaces et tout ce qui se trouve après #.
- R fait la différence entre les majuscules et les minuscules.
- On peut utiliser R comme une calculatrice

```
> 3+4
```

```
[1] 7
```

Le résultat n'est pas mis en mémoire.

- `> x= 3+4` ou `> x<- 3+4`

n'affiche rien, mais le résultat est mis en mémoire, on y accède ainsi :

```
> x
```

```
[1] 7
```

- > `x=c(1,3.2,4,6,10.2,-11)` crée un vecteur avec une collection de nombres.
- > `y=1:10` vecteur contenant les entiers de 1 à 10.
- > `z=10:1` idem mais à l'envers.
- > `x[2]` renvoie la deuxième valeur de `x`.
- > `y[-3]` renvoie le vecteur `y` sans la troisième valeur.
- > `x[1:4]` renvoie les 4 premières valeurs de `x`.

Autres fonctions importantes pour créer des vecteurs :

```
seq(from=0,to=1,by=0.25)
```

```
rep(1:3,1:3)
```

Opérations (terme par terme)	+	-	*	/	^
Multiplication matricielle AB			A %*% B		
Division entière de x par y			x %/% y		
x Modulo y			x%%y		

Quelques fonctions :

abs, exp, log, log10, sqrt, sin, cos, tan, max, min,
sum, sort, var, mean, gamma, sign, length, sinh, ...

3 manières de procéder :

- en combinant des vecteurs lignes, `rbind`
> `x.mat=rbind(1:5, 6:10, 1)`
- en combinant des vecteurs colonnes, `cbind`
> `y.mat=cbind(1:5, 6:10, 1)`
- en transformant un vecteur en matrice, `matrix`
> `z.mat=matrix(1:12, ncol=4)`

`x.mat[1, 2]` retourne l'élément de la première ligne, deuxième colonne de la matrice

`x.mat[1,]` retourne la première ligne de la matrice

`t(.)` transpose la matrice

`solve(.)` calcule l'inverse de la matrice

`nrow(.)` retourne le nombre de lignes
`ncol(.)` retourne le nombre de colonnes
`dim(.)` retourne (n, m) où n est le nombre de lignes
 et m le nombre de colonnes
`length(.)` retourne le nombre d'éléments

Autres fonctions : `diag`, `eigen`, `chol`, `qr`, ...

Il est possible de créer des vecteurs avec des chaînes de caractères :

```
> x=c("Lundi", "Mardi", "Mercredi")
```

Une liste permet de stocker plusieurs objets de différents types :

```
> mylist=list(Day=x, Location="Fribourg", Temp=c(3, 6, 2))
```

```
> mylist
```

```
$Day
```

```
[1] "Lundi" "Mardi" "Mercredi"
```

```
$Location
```

```
[1] "Fribourg"
```

```
$Temp
```

```
[1] 3 6 2
```

Accès aux données :

```
mylist$Day    ou    mylist[[1]]
```

```
mylist$Temp[2]  ou  mylist[[3]][2]
```

C'est certainement l'objet R le plus utilisé ! Il permet de stocker des données de différents types mais de même grandeur.

```
> classeAB=data.frame(Nom=c("Alain", "Zoé", "Anne"),  
                      Note=c(3.5, 5.5, 4))
```

```
> names(classeAB)
```

```
[1] "Nom" "Note"
```

donne le nom des colonnes

```
> classeAB$Note
```

```
[1] 3.5 5.5 4
```

accès à une colonne

Valeurs logiques

<code>==</code>	égal à
<code>!=</code>	n'est pas égal à
<code>></code> (<code><</code>)	plus grand que (plus petit que)
<code>>=</code> (<code><=</code>)	plus grand ou égal à (plus petit ou égal à)
<code>&</code>	et logique
<code> </code>	ou logique
<code>!</code>	négation

```
> x=c(1, 3.2, 4, 6, 10.2, -11)
```

```
> x > 3
```

```
[1] FALSE TRUE TRUE TRUE TRUE FALSE
```

R permet l'indexation par valeur logique.

```
> x[x>3 & x<=6]
```

```
[1] 3.2 4 6
```


Avec R, on peut importer des données enregistrées dans des fichiers .txt, .dat, .csv :

- > `file.dat=read.table("file.dat",header=TRUE)`
Importe les données du fichier file.dat qui contient aussi les entêtes des colonnes.
- > `file.csv=read.csv("file.csv",header=TRUE,sep="\t")`
- > `file.txt=read.table("file.txt",header=TRUE,sep=" ")`

Pour exporter un data frame : `write.table(x,file,...)` ou `write.csv(x,file,...)`

Packages

Les packages sont des modules supplémentaires qui peuvent contenir des fonctions et des bases de données.

<code>library()</code>	indique les packages installés sur la machine
<code>install.packages("ISwR")</code>	installe le package ISwR sur la machine ou voir dans les menus
<code>library(ISwR)</code>	charge le package, le rend disponible
<code>require(ISwR)</code>	idem
<code>help(package=ISwR)</code>	donne des informations sur le package ISwR
<code>data(package="ISwR")</code>	répertorie les bases de données de ISwR
<code>data(tlc)</code>	charge (rend disponible) la base de données tlc
<code>summary(tlc)</code>	résume les données contenues dans tlc

Variables de distribution standard

R possède de nombreuses fonctions de distribution déjà implémentées.

Code	Distribution	paramètre
binom	Loi binomiale	n, p
geom	Loi géométrique	p
pois	Loi de Poisson	λ
unif	Loi uniforme	min, max
exp	Loi exponentielle	rate
norm	Loi normale	mean, sd
chisq	Loi du χ^2	degré de liberté
t	Loi de Student	degré de liberté

Il suffit de mettre une des 4 lettres suivantes devant le nom de code de la distribution pour obtenir :

- d la densité, par ex. `dnorm`
- p la probabilité, par ex. `pnorm`
- q le quantile, par ex. `qnorm`
- r une variable aléatoire, par ex. `rnorm`

Exemples :

- | | |
|---------------------------------|---|
| > <code>dexp(1,2)</code> | donne la valeur en 1 de la densité pour une v.a. exponentielle de paramètre 2 |
| > <code>pbinom(2,10,0.1)</code> | probabilité que la binomiale de paramètres 10 et 0.1 soit égale à 2 |
| > <code>qnorm(0.05)</code> | donne le t pour lequel $P(X < t) = 0.05$ pour une v.a. normale |
| > <code>rpois(10,2)</code> | donne 10 réalisations d'une v.a. de Poisson de paramètre 2 |

R possède de nombreux outils graphiques de haute qualité (2D et 3D).

<code>windows()</code>	pour ouvrir une fenêtre graphique (Windows)
<code>quartz()</code>	pour ouvrir une fenêtre graphique (MacOS X)
<code>x11()</code>	pour ouvrir une fenêtre graphique (UNIX)
<code>par(...)</code>	spécifie les options graphiques de la fenêtre
<code>par(mfrow=c(2,2))</code>	créé une fenêtre graphique avec 4 sous-graphiques arrangés sur 2 lignes et 2 colonnes
<code>par(pty="s")</code>	le graphique sera carré
<code>help(par)</code>	pour plus d'informations ...

Quelques exemples :

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

dessine x en fonction de y (points)

```
plot(x,y,type="l")
```

avec l'option "ligne"

```
plot(x,y,col="red")
```

avec l'option couleur rouge

```
barplot,hist
```

dessine un "barplot", un histogramme

```
boxplot
```

dessine un "boxplot"

```
pie
```

dessine un camembert

```
pairs
```

dessine tous les couples de coordonnées à partir d'un jeu de données.

...

Ce que l'on peut ajouter à un graphique existant ...

<code>abline</code>	ajoute une droite d'ordonnée à l'origine et de pente donnés
<code>axis</code>	ajoute des axes
<code>box</code>	ajoute une boîte autour du graphique
<code>legend</code>	ajoute une légende
<code>lines, points</code>	ajoute des lignes, des points
<code>title</code>	ajoute un titre
<code>mtext, text</code>	ajoute du texte dans la marge ou dans le graphique
<code>...</code>	

Plusieurs commandes R peuvent être groupées :

```
{expression1 ; expression2 ; ...}
```

↪ la valeur du groupe est la valeur de la dernière expression.

Boucle "For" :

```
for (index in values) {expressions}
```

Boucle "While" :

```
while (condition) {expressions}
```

Evaluation conditionnelle "If" :

```
if (condition) {expression1} else {expression2}
```

Exemples :

```
for (i in 1:10) print(i)
```

```
while(i < 11) {print(i) ; i=i+1}
```

```
if (x>0) y=x*log(x) else y=0
```



```
name=function(arg1,arg2,...){expressions}
```

Exemple:

```
> fun1=function(x,y){ (x+y)^2}
```

```
> fun1(1,1)
```

```
[1] 4
```

```
> fun1(1:4,5)
```

```
[1] 36 49 64 81
```