Pour démarrer en R

3IF, INSA de Lyon, 2017/2018
Irène Gannaz

INSA-Lyon, Département Informatique
version 1.00, 23 novembre 2017 – rédigé avec LaTeX

1 Introduction

L'objectif de ce document est de donner une brève introduction au logiciel `R`. Le logiciel R est un logiciel libre, disponible sur `http://www.r-project.org/`. C'est un logiciel de référence en statistique, qui permet l'utilisation des méthodes statistiques classiques à l'aide de fonctions prédéfinies. Il est également possible de créer ses propres programmes dans un langage de programmation assez simple d'utilisation (proche de Matlab).

Il est possible d'accéder à des techniques spécifiques en statistique et science des données développées par les utilisateurs à l'aide d'un système de paquets mis à disposition sur le site du CRAN (http ://cran.r-project.org/). Chaque utilisateur peut également déposer un paquet.

Le logiciel R fonctionne initialement en ligne de commande, mais des interfaces permettent désormais une utilisation plus conviviale. Nous proposons ici de travailler avec l'interface RStudio, téléchargeable sur `http://www.rstudio.com/`.

RStudio est séparé en 4 fenêtres graphiques :

- la console (en bas à gauche) qui permet d'executer les instructions
- le script (en haut à gauche) pour écrire les commandes que l'on veut exécuter et les sauvegarder dans un fichier. Une instruction du script peut être exécutée à l'aide du raccourci Ctrl+Entrée.
- la fenêtre Files/Plots/Packages/Help (en bas à droite) qui permet entre autres de visualiser les graphiques ou l'aide.
- la fenêtre Workspace/History (en haut à droite) qui permet de voir l'ensemble des objets en mémoire et l'historique des instructions réalisées.

Le répertoire de travail par défaut est celui à partir duquel vous avez lancer l'interface RStudio, mais vous pouvez le modifier (fonction `setwd() ou menu de Rstudio`. Je vous conseille de vous placer dans un répertoire de travail bien défini où vous mettrez les fonctions du TP.

Ci-après, je vous donne quelques instructions de base nécessaires pour le TP. Seul un petit aperçu

des possibilités de R est donné. Nous ne verrons pas en particulier comment lire des jeux de données, comment appliquer les études statistiques basiques, représenter des graphiques complexes...

2 Syntaxe de base

R peut être utilisé pour des opérations élémentaires :

```
cos(pi/3)^2*sqrt(2)
```

On peut stoker le résultat dans une variable à l'aide de la notation <- (la notation = est aussi valable mais je vous conseille de privilégier la flèche).

```
a <- cos(pi/3)^2*sqrt(2)
```

Pour effacer les variables en mémoire dans la session, il faut taper la commande suivante :

```
rm(list=ls())
```

Pour créer des vecteurs, la commande est c, comme concaténation :

```
x <- c(0,2,4)
y <- c(1,7,4)
```

On peut mettre les vecteurs x et y bout-à-bout

```
c(x,y)
```

ou les mettre côte-à-côte dans une matrice

```
cbind(x,y)
```

On peut créer un vecteur constant avec l'instruction rep :

```
z <- rep(4,8)
```

Ici z est un vecteur de taille 8 ne contenant que des 4.

La commande matrix permet de créer une matrice

```
M <- matrix(x,2,4)
```

Il est également possible de faire des tableaux à plus de deux dimensions :

```
T <- array(x,dim=c(2,2,2))
```

Pour accéder aux éléments d'un vecteur ou d'une matrice, on utilse les crochets :

```
x[2]
M[1,4]
M[(M>2)]
M[,4]
```

La dernière instruction donne la quatrième colonne de M.

Enfin, une structure très utile en R est la structure de listes.

```
L <- list(nom='toto',age=21,notes=c(14,7,12),vect=seq(1,4,by=0.5))
```

Les éléments de la liste sont accessible par un $ et on peut accéder aux noms de la liste à l'aide de la commande `names` :

```
L$vect
names(L)
```

Pour *résumer* le contenu de L, on pourra utiliser

```
str(L)
```

Il existe d'autres structures, mais qui ne seront pas abordées dans ce TP.

Toute opération arithmétique sur des vecteurs ou des matrices sera réalisée élément par élément. Par exemple :

```
8*c(8,88,888,8888)+13
```

Pour spécifier que l'on veut faire des opérations matricielles, il faut encadrer les opérteurs par des %.

```
x <- c(2,3,0)
A <- cbind(c(1,8,7),c(0,1,7),c(8,4,0))
A*x
A%*%x
```

Les deux instructions ci-dessus sont très différentes ! Enfin, l'inversion de matrice s'obtient à l'aide de la fonction `solve`.

Si vous souhaitez de l'aide sur une fonction, disons `f`, utilisez l'une des instructions ci-dessous

```
?f
help(f)
```

3 Les graphiques

Seules quelques fonctionnalités très restreintes sont présentées ici.

```
x11()
```

permet d'ouvrir une nouvelle fenêtre graphique.

```
par(mfrow=c(2,3))
```

découpera la fenêtre en 2 lignes et 3 colonnes. Chaque graphique sera ensuite remplit successivement.

La syntaxe d'un graphique est ensuite la suivante

```
plot(x,y,main='Titre principal',xlab='axe des x',ylab='axe des y')
```

On peut ajouter d'autres caractéristiques. Par exemple, `type='l'` donnera une représentation continue de `y` en fonction de `x`.

```
graphics.off()
```

ferme toutes les fenêtres graphiques ouvertes.

4 Programmation

Comme dans la majorité des langages, je vous suggère de structurer votre code en des fichiers avec les fonctions nécessaires à votre code, et un fichier qui appelle ces fonctions. Supposons que que vous ayez mis vos fonctions dans le fichier `utile.R`. Alors, au début du fichier qui appelle ces fonctions, vous devez faire apparaître

```
source('utile.R')
```

Cette instruction permet de charger toutes les fonctions contenues dans `utile.R`.

La syntaxe d'une fonction est la suivante :

```
rosace <- function(a,b,absolue)
{
  # le # sert a mettre des commentaires
  # cette fonction sert a tracer des rosaces

  theta <- seq(0,2*n*pi,0.01)

  if(absolue==TRUE)
  {
    rho <- 1+b*abs(cos(a*theta))
  }else{
    rho <- 1+b*cos(a*theta)
  }

  return(list(angle=theta,rayon=rho))
}
```

On exécute ensuite la fonction avec l'instruction `res <- rosace(9/4,1,FALSE)`. Alors `res` est une liste contenant un vecteur `angle` et un vecteur `rayon`, auxquels on peut accéder avec les commandes `res$angle` et `res$rayon`. Pour représenter la rosace, on pourra ainsi appeler

```
plot(res$rayon*exp(1i*res$angle),type='l')
```

Si on souhaite mettre des valeurs par défaut, on peut remplacer la première ligne par :

```
rosace <- function(a,b,absolue=FALSE)
```

Alors `rosace(9/4,1)` exécutera `rosace(9/4,1,FALSE)`. On peut également changer l'ordre des paramètres, tant qu'on précise leur nom, `rosace(b=1,a=9/4)`.

L'exemple ci-dessus montre un exemple d'utilisation de `if`. L'instruction `while` s'utilise de manière similaire. Nous donnons ici un exemple d'utilisation de `for`.

```
suite <- function(a,n=100)
{
  # une suite constante egale a 1
  x <- 1
  for(i in 1:n)
  {
    x <- (a+1)*x-a
  }

  # on affiche la valeur obtenue
```

```
  cat('valeur de x_n :', x, '\n')
}
# on essaie ? (avec des nombres pas pris au hasard ici !)
suite(1.3)
suite(127.8)
# conclusion ?
```

Dernier point : `R` passe les paramètres comme copie. Ainsi dans le code suivant, la valeur de `a` n'est pas modifiée :

```
aha <- function(a=0){
  a <- a+1
}
a <- 4
aha(a)
```

5 Les paquets

Pour installer un paquet, on utilisera l'instruction `install.packages`. Par exemple pour installer le paquet `MASS` :

```
install.packages('MASS')
```

Cette instruction nécessite bien entendu une connexion internet. `RStudio` vous proposera alors de choisir le serveur à utiliser pour télécharger le package et procédera ensuite à l'installation. Pour spécifier que l'on souhaite charger le package dans la session en cours, on utilise la commande `library` :

```
library('MASS')
```

Toutes les fonctions proposées dans le package peuvent alors être utilisées.