Initiation à R

Lise Vaudor

2016-08-29

Contents

1 Generalités

1.1 Pourquoi utiliser le logiciel R?

Le langage R est un **langage de programmation** et un **environnement mathématique** utilisé pour le **traitement de données** et l'**analyse statistique**.

Il permet de réaliser

- des calculs arithmétiques
- des tris et croisements de données
- une très large variété de **graphiques**
- des **programmes** (automatisation de traitements)
- de la **modélisation et simulations numériques**
- une très large variété de **traitements statistiques** (c'est ce pour quoi il est le plus reconnu,)
- etc.

Il peut doc remplir les fonctions d'une calculatrice, d'un tableur, d'un langage de programmation, et d'un logiciel de statistiques...

Il est en outre **libre** et **gratuit**.

En contrepartie de sa polyvalence et de sa flexibilité, R peut être un peu déroutant au premier abord, car il ne s'agit pas d'un logiciel "clic-boutons": on exécute les différentes opérations à travers l'execution de **lignes de commande**.

1.2 Installation de base et premiers pas

Pour installer R, on peut le télécharger par exemple à l'adresse suivante: http://cran.r-project.org/

L'installation prend au plus quelques minutes.

Au lancement de R, une fenêtre (la **console**) apparaît: le symbole ">" indique que R est prêt à executer toute ligne de commande que nous allons lui fournir.

Un exemple de **ligne de commande**:

```
2+2
```

```
[1] 4
```

Taper entrée pour exécuter la commande. R exécute la commande et nous affiche le résultat.

Vous pouvez ainsi exécuter diverses opérations arithmétiques:

```
32.7*59.6 # multiplication
```

```
[1] 1948.92
```

```
53/59  # division
```

```
[1] 0.8983051
```

```
2.33^5 # puissance
```

```
[1] 68.67199
```

```
(45-69)*75 # combinaisons d'operations
```

```
[1] -1800
```

Les indications précédées du symbole # sont des commentaires. Ils sont ignorés par R mais sont très utiles par exemple pour vous rappeler dans quel but vous cherchez à exécuter telle ou telle commande.

Si l'on exécute plusieurs lignes de commandes dans la console, on peut "récupérer" les anciennes lignes de commandes avec la flèche ↑ ou au contraire en récupérer de plus récentes avec ↓.

Exercice 1

Exécutez le calcul ci-après:

```
>(33+52)/((57+11)
```

Que remarquez-vous et pourquoi? Vous pouvez sortir de cette situation grâce à la touche "Echap"...

1.3 Utilisation d'objets

Lorsque vous exécutez une commande, vous pouvez en observer le résultat directement dans la console:

```
32.7*59.6
```

```
[1] 1948.92
```

```
53/59
```

```
[1] 0.8983051
```

Vous pouvez également choisir d'attribuer ce résultat à un **objet**.

```
a=32.7*59.6
b=53/59
```

Vous pouvez de manière tout-à-fait équivalente utiliser les commandes suivantes:

```
a<-32.7*59.6
53/59->b
```

On dit qu'on **assigne une valeur à un nom (ou à une variable)**. On a ainsi créé les objets a et b.

Lorsque vous exécutez les commandes ci-dessus, rien ne s'affiche dans la console. Cela ne signifie pas pour autant que rien ne s'est passé... Vous avez créé les objets a et b, qui font désormais partie de votre environnement de travail... Jetez donc un coup d'oeil à la zone "Workspace" de RStudio pour vous en convaincre.

Pour afficher les objets dans la console, plusieurs possibilités:

```
a
```

```
[1] 1948.92
```

```
b
```

```
[1] 0.8983051
```

```
print(a)
```

```
[1] 1948.92
```

```
print(b)
```

```
[1] 0.8983051
```

Vous pourrez par la suite manipuler les objets de différentes façon... Par exemple, ici on peut les utiliser pour de simples opérations arithmétiques:

```
a+b  # calcul puis affichage
```

```
[1] 1949.818
```

```
c=a+b # calcul et creation d'objet
print(c) # affichage
```

```
[1] 1949.818
```

1.4 Utilisation de scripts et IDE

En pratique, on aura la plupart du temps besoin de garder une trace pérenne des différentes commandes et de leur succession. Or, lorsque l'on écrit directement les lignes de commande dans la console, on peut les sauvegarder dans un "historique" (mais tous les tâtonnements successifs seront aussi sauvegardés, ce qui n'est guère pratique pour s'y retrouver par la suite). On écrit donc les lignes de commandes à sauvegarder dans un **script** (i.e. un fichier texte contenant l'ensemble des liges de commande mises au propre et commentées!), puis on les envoie sur la console pour les **exécuter**.

Il existe, sous R, un éditeur de scripts accessible par le menu Fichier-> Nouveau script ou Fichier->Ouvrir un script. L'éditeur de script est en gros l'équivalent d'un "bloc-notes" dans lequel vous pourrez écrire et sauvegarder vos commandes. Les lignes de commande sélectionnées peuvent être envoyées depuis le script vers la console à l'aide de la la commande Ctrl+R.

Nous allons quant à nous travailler sur un éditeur de script (ou plus précisément un IDE, pour Integrated Development Environement) plus pratique que cet éditeur basique: le logiciel RStudio. Il est lui aussi libre et gratuit et peut être téléchargé à l'adresse suivante: http://www.rstudio.com/ide/.

L'utilité d'un éditeur de script peut vous échapper de prime abord mais au fur et à mesure de votre apprentissage vous apprendrez à apprécier le confort d'utilisation qu'il vous procure pour travailler sous R... Quelques mots d'explication, tout-de-même:

Lorsque l'on travaille avec RStudio, quatre zones apparaissent ("Source" en haut à gauche, "Console" en bas à gauche, " Workspace" en haut à droite, "Plots" en bas à droite).

La zone **Source** constitue l'éditeur de code à proprement parler. C'est dans cette zone que vous allez écrire vos scripts.

Les calculs sont exécutés dans la zone **Console**. On peut envoyer les codes de la zone "Source" vers la zone "Console"

- grâce au bouton **Run** (qui exécute la ou les lignes de commande sélectionnée(s))
- grâce au bouton **Source** (qui exécute l'ensemble des lignes de commande du script).

Dans la zone **Workspace** sera affiché l'ensemble des objets de l'**environnement de travail**. On peut cliquer sur les noms des différents objets pour en connaître les détails.

Dans la zone **Plots** s'afficheront les graphiques produits. C'est aussi là que s'afficheront les fichiers d'aide relatifs aux diverses fonctions que vous utiliserez.

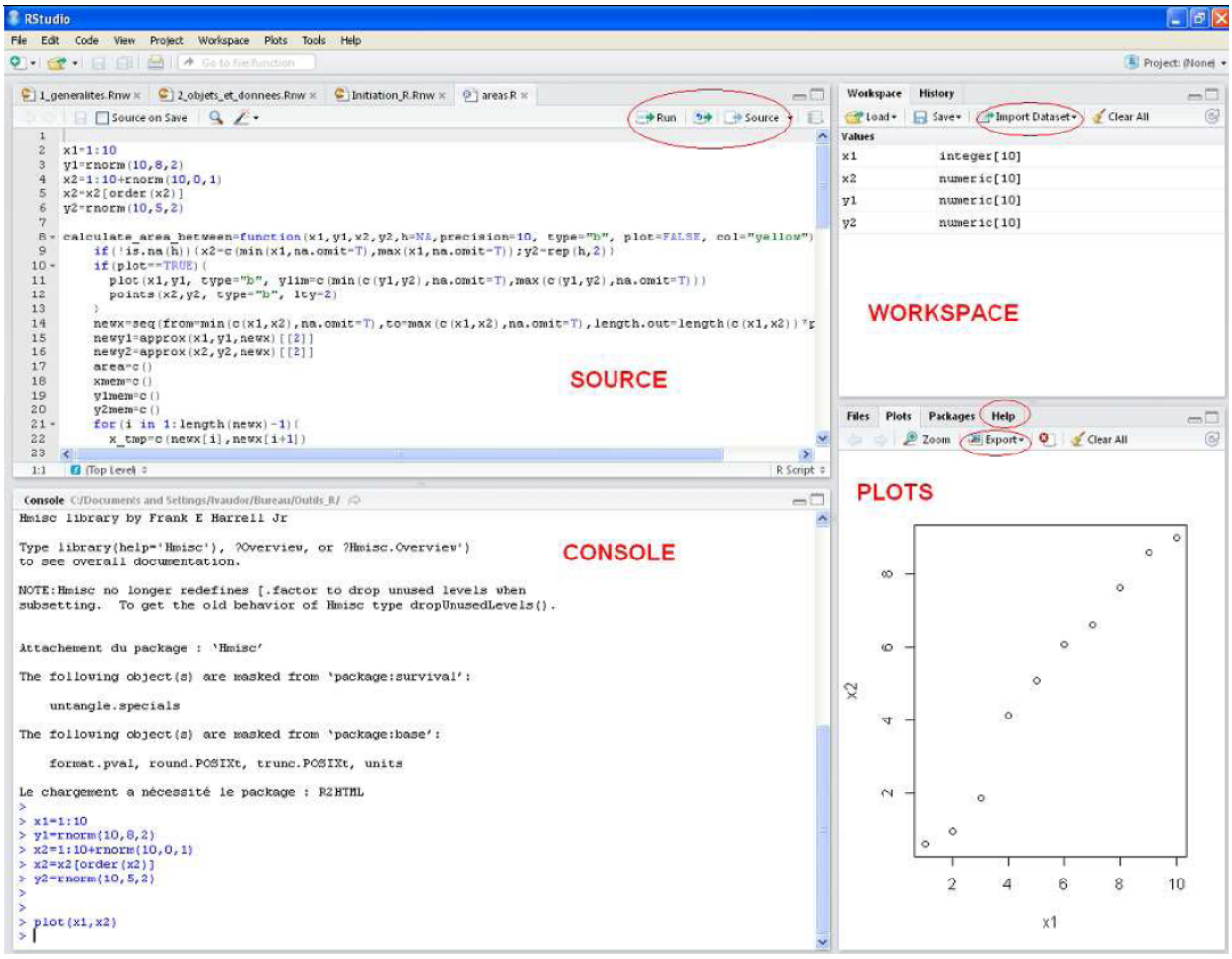


Figure 1: Aperçu de RStudio et de quelques-unes de ses fonctionnalités.

2 Création d'objets

Nous avons vu dans la section précédente comment créer des objets très simples. Il y a en fait une multitude de types d'objets possibles dans R.

2.1 Vecteurs, facteurs, matrices et listes

2.1.1 Vecteurs

On appelle "vecteur" toute séquence de valeurs, par exemple:

vecteur1=

2.3	3.6	1.1	2.4	2.5	10.2	5.1	2.0

ou vecteur2=

"Paris"	"Lyon"	"Marseille"	"Rennes"	"Montpellier"

En R, ces vecteurs s'écrivent:

```
v1 <- c(2.3,3.6,1.1,2.4,2.5,10.2,5.1,2.0)
v2 <- c("Paris","Lyon","Marseille","Rennes","Montpellier")
```

On peut également créer des vecteurs correspondant à:

- des séquences de valeurs régulièrement espacées

```
v3<-seq(from=0,to=10,by=2) # valeurs de 0 à 10 par pas de 2
v4<-seq(from=2,to=13,length.out=20)  # 20 valeurs regulierement espacees de 2 a 13
v5<-0:10  # nombres entiers de 0 a 10
```

- des séquences de valeurs répétées

```
v6<-rep("date1",5)
v7<-rep(v5,3)
```

Les vecteurs peuvent être de "mode" différent selon le type de valeurs qu'ils contiennent (par contre toutes les valeurs d'un vecteur sont supposées être d'un même type). Ils peuvent par exemple être de mode

- **numérique** comme v1,v3, v4, v5 et v7 par exemple,
- **caractère** comme v2 et v6 par exemple,
- **logique**, c'est à dire que les valeurs qu'ils contiennent sont de type vrai/faux (TRUE/FALSE ou T/F)

Remarquez que l'on peut aussi utiliser `c()` pour combiner plusieurs vecteurs:

```
vglobal<-c(v3,v4,v5)
```

Si l'on tente quelque chose comme

```
vessai=c(v6,v7)
```

R ne renvoie pas de message d'erreur, mais fait en sorte que toutes les valeurs de vessai soient d'un même type (des chaînes de caractère ici: voyez les guillemets autour des valeurs de v7).

Exercice 2

Créer en utilisant c(), rep(), seq(), un vecteur dont le nom est "vecteurexo1" et dont les valeurs sont 1,1,1,1,3,4,5,6,7,8.6,8.7,8.8,8.9,9.

Vous remarquerez que le nombre de chiffres significatifs affichés par R est le même pour tous les éléments.

Exercice 3

Tentez l'exécution de la commande ci-dessous:

```
v6_bis=rep(date1,5)
```

R n'exécute pas votre commande et affiche un message d'erreur: pourquoi?

2.1.2 Facteurs

Les facteurs ressemblent généralement à des vecteurs de mode caractère, à la nuance près qu'ils comprennent généralement plusieurs "niveaux", comme par exemple

```
f1=factor(c(rep("date1",5),rep("date2",5)))
f1
```

```
 [1] date1 date1 date1 date1 date1 date2 date2 date2 date2 date2
Levels: date1 date2
```

```
levels(f1)
```

```
[1] "date1" "date2"
```

La nuance entre vecteurs et facteurs est importante pour un certain nombre de fonctions implémentées dans R, il est donc assez souvent nécessaire de convertir les vecteurs en facteurs et inversement (cf page 11)

2.1.3 Matrices

On appelle "matrice" toute "grille" (à 2 dimensions) de valeurs, par exemple :

M1=

1	3	7
2	8	0

ou M2=

0	0	1	0	0
1	1	1	0	0
0	0	1	0	0
0	0	1	1	0

En R, ces matrices s'écrivent:

```
M1 <- matrix(c(1,2,3,8,7,0),nrow=2,ncol=3)
M1 <- matrix(c(1,3,7,2,8,0), nrow=2, ncol=3, byrow=T)
M2 <- matrix(c(0,1,0,0,0,1,0,0,1,1,1,1,0,0,1,0,0,0,0,0),nrow=4)
```

On peut également considérer qu'une matrice résulte de l'accolement de plusieurs lignes ou de plusieurs colonnes. Par exemple on peut écrire M1 des deux manières suivantes:

```
M1<-cbind(c(1,2),c(3,8),c(7,0))
M1<-rbind(c(1,3,7),c(2,8,0))
```

Exercice 4

Tentez d'exécuter la ligne de commande suivante: que se passe-t-il et pourquoi?

```
matrix(c(1,1,1,2,2,2,3,3),nrow=3)
```

Exercice 5

Créer la matrice suivante, qu'on appellera "Mexo1":

0	3	1	5	0
0	3	1	5	0
0	3	1	5	0

2.2 Listes

Une liste est un assemblage d'objets de natures et ou de tailles différentes. Par exemple, la liste l1

```
l1=list(sites=v2, nb=M1)
```

rassemble un vecteur (v2) et une matrice (M1) au sein d'un même objet.

En pratique, de nombreuses fonctions implémentées en R renvoient un objet de type liste (par exemple, un objet "régression linéaire", renvoyé par la fonction lm, comprend entre autres choses un vecteur de coefficients de la régression et un vecteur de résidus de la régression (donc deux vecteurs de tailles différentes).

2.3 Tableaux de données (data.frame) chargés depuis un fichier

2.3.1 Choix du répertoire de travail et lecture du fichier

On peut charger des tableaux de données dans R depuis des fichiers de type différents. Par exemple, des fichiers .txt, ou .csv, etc. Il est en général préférable de travailler avec des fichiers de données sous format .csv, (qui est au .xls ce que le .txt est au .doc, c'est à dire qu'il s'agit du format "brut" standard pour les données tabulaires).

Pour charger correctement ces données, le plus important est de bien spécifier le **type de séparateur de colonnes** (virgule, point-virgule, espace...) utilisé par le fichier, à l'aide de l'argument sep="...".

Pour lire le fichier, il suffit normalement de spécifier le nom du fichier, ainsi que la nature du séparateur de colonnes:

```
air=read.table("../../datasets/air.txt",sep=",", header=T)
```

Attention, on doit spécifier le chemin en remplaçant les antislashs \ par des slashs /

Il se peut que le fichier ne se trouve pas dans le **répertoire de travail** courant de R ("**w**orking **d**irectory") de R (i.e. un dossier où, sauf précision de la part de l'utilisateur, R lit et écrit les fichiers). Dans ce cas, vous devrez soit préciser le chemin complet du fichier lors de l'appel de read.table, soit changer de répertoire de travail. La fonction getwd permet d'afficher le répertoire de travail courant, tandis que la fonction setwd permet de le modifier.

```
getwd()
setwd("C:/Documents and Settings/lvaudor/Bureau/tutoRial/Initiation_R_dev/")
air=read.table("../../datasets/air.txt",sep=",", header=T)
```

2.3.2 Options de lecture du fichier

En général, lire un tableau de données sera la première chose que vous voudrez faire en travaillant sous R. Néanmoins, ce n'est pas forcément l'étape la plus facile: en pratique la lecture des tableaux de données est souvent problématique, par exemple si les noms de colonnes contiennent des **caractères spéciaux ou des espaces**, si le **séparateur décimal du fichier est une virgule** et non un point, si le fichier contient des **commentaires**, etc.

Ce genre de problèmes arrive très souvent à ceux qui travaillent ordinairement avec Excel et pour qui le tableau de données est également un document de travail (avec des commentaires, des noms de colonnes à rallonge destinés à être les plus explicites possible, des graphiques, des mises en forme destinées à rendre le document plus lisible, etc.).

N'oubliez que pour R, ce tableau est censé ne comporter que de la donnée **BRUTE**!... Veillez donc à bien "nettoyer" le fichier que vous voulez lire dans R: avoir des **noms de colonnes simples et sans espaces**, enlever les commentaires et mises en forme inutiles, etc.

On peut également spécifier certains des arguments de la fonction read.table pour contrôler la lecture du fichier. Ces options sont détaillées dans le fichier d'aide associé à la fonction: on peut l'ouvrir grâce à l'une ou l'autre des commandes suivantes:

```
help(read.table)
?read.table
```

Exercice 6

Tentez la lecture du fichier “air.txt” sans préciser le séparateur décimal puis sans préciser header=T Que se passe-t-il et pourquoi?

Comme pour les autres objets, on peut alors voir cette table simplement en tapant son nom puis “Entrée” dans la console. On peut également utiliser les fonctionnalités de RStudio pour afficher et modifier la table (zone “Workspace” en haut à droite).

On peut avoir accès à chaque colonne du tableau de données en tapant le nom du tableau suivi de $ et du nom de la colonne, par exemple

```
air$Ozone
```

```
 [1]  41  36  12  18  NA  28  23  19   8  NA   7  16  11  14  18  14  34
[18]   6  30  11   1  11   4  32  NA  NA  NA  23  45 115  37  NA  NA  NA
[35]  NA  NA  NA  29  NA  71  39  NA  NA  23  NA  NA  21  37  20  12  13
[52]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 135  49  32  NA  64  40  77
[69]  97  97  85  NA  10  27  NA
 [ reached getOption("max.print") -- omitted 78 entries ]
```

Par défaut, en R, les éléments vides correspondent à l’indication NA pour “**n**ot **a**vailable”. On peut utiliser l’argument na.strings de read.table pour définir les éléments vides d’une autre manière (par exemple na.strings=“”), cependant il est généralement préférable de laisser les éléments vides en NA car certaines fonctions de R les gèrent automatiquement sous cette forme.

On peut également attacher le tableau de données à l’environnement R pour ne pas avoir à taper “air$...” à chaque fois que l’on veut accéder aux données correspondant à une colonne:

```
 [1]  41  36  12  18  NA  28  23  19   8  NA   7  16  11  14  18  14  34
[18]   6  30  11   1  11   4  32  NA  NA  NA  23  45 115  37  NA  NA  NA
[35]  NA  NA  NA  29  NA  71  39  NA  NA  23  NA  NA  21  37  20  12  13
[52]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 135  49  32  NA  64  40  77
[69]  97  97  85  NA  10  27  NA
 [ reached getOption("max.print") -- omitted 78 entries ]
```

Exercice 7

Charger le tableau de données de “representations.csv”. Utiliser deux méthodes: celle par ligne de commande comme ci-dessus, et celle utilisant les fonctionnalités de RStudio (zone “Workspace”, bouton “Import Dataset”).

Attacher la table.

Nous utiliserons ce tableau de données dans la suite du tutoriel. Les variables renseignées pour environ 200 personnes âgées de 20 à 35 ans sont les suivantes:

- la taille (en cm)
- le poids (en kg)
- le sexe (homme ou femme)
- une note d’auto-évaluation physique (allant de 0 à 10)
- les dépenses mensuelles moyennes en vêtements en euros

NB: ces données sont “fictives”!

```
representations=read.table("../../datasets/representations.csv",sep=";", header=T)
attach(representations)
```

2.4 Conversion d'objets

Pour interroger R quant au type (vecteur, facteur, tableau, matrice, etc.) ou au mode (numérique, caractère, logique, etc.) d'un objet, on utilise les fonctions de type "is.*typeoumode*. Par exemple,

```
is.factor(v6)
```

```
[1] FALSE
```

```
is.numeric(v6)
```

```
[1] FALSE
```

renvoient FALSE.

On peut convertir un objet d'un type ou mode à un autre en utilisant les fonctions de type "as.*typeoumode*. Par exemple,

```
v6=as.factor(v6)
```

convertit le vecteur v6 en facteur.

3 Manipulation d'objets

3.1 Opérateurs

- **Opérateurs numériques**:Ils permettent d'effectuer des opérations arithmétiques simples, comme des additions, des multiplications,etc.
- **Opérateurs de comparaison**:Ils permettent d'effectuer des comparaisons de valeurs numériques, ou de vecteurs
- **Opérateurs logiques**: Ils permettent de vérifier si une proposition est vraie ou non

Notation	Opération	Exemple
`-`	soustraction	`6-4` >>2
`*`	multiplication	`2*5` >>10
`/`	division	`6/2` >>3
`^`	puissance	`2^5` >>32
`<`	plus petit	`3<4` >>TRUE
`>`	plus grand	`7>10` >>FALSE
`<=`	plus petit ou égal	`5<=5` >>TRUE
`>=`	plus grand ou égal	`6>=3` >>TRUE
`==`	égal	`2==3` >>FALSE
`!=`	différent	`5!=5` >>FALSE
`!`	NON logique	`!(5!=5)` >>TRUE
`&`	ET logique	`3<4 & 5<=1` >>FALSE
`|`	OU logique	`3<4 | 5<=1` >>TRUE

Exemples d'utilisation

** Opérateurs numériques:**

Dans le tableau de données "air", la température (Temp) est indiquée en degrés Fahrenheit. On peut les convertir en degrés Celsius de la manière suivante:

```
TempCelsius<-(Temp-32)*5/9
```

Opérateurs de comparaison: Est-ce que la température est supérieure ou égale à 20°C?

```
TempCelsius<=20
Month==7
```

Remarquez la différence entre `Month=7` qui assigne la valeur 7 à une variable Month, et `Month==7` qui compare les valeurs de Month à la valeur 7... `=` est un opérateur d'assignation, tandis que `==` est un opérateur de comparaison...

opérateurs logiques: Est-ce qu'on dispose bien des données sur l'ozone?

```
!(is.na(Ozone))
```

is.na(Ozone) renvoie T pour les éléments d'Ozone qui sont vides. Le point d'exclamation représente une négation, donc !(is.na(Ozone)) renvoie au contraire T pour les éléments d'Ozone qui sont renseignés.

3.2 Système d'indexation

On peut s'intéresser à une partie d'un objet, par exemple un ou plusieurs éléments d'un vecteur ou d'une matrice, une partie d'une table de données, etc.

On a accès au i^{ieme} élément d'un vecteur par la commande

```
v[i]
```

Par exemple:

```
Ozone[3]
```

```
[1] 12
```

```
Ozone[1:10]  # les dix premieres valeurs
```

```
 [1] 41 36 12 18 NA 28 23 19  8 NA
```

Si l'on s'intéresse à l'élément d'une matrice ou d'un tableau de données qui se situe sur la i^{ieme} ligne et sur la j^{ieme} colonne, on y a accès par:

```
M[i,j]
```

```
M1[1,3] # la valeur sur la ligne 1 et la colonne 3
```

```
[1] 7
```

```
M1[,3]  # toutes les valeurs sur la colonne 3
```

```
[1] 7 0
```

```
M1[2,1:2] # les deux premieres valeurs de la ligne 2
```

```
[1] 2 8
```

Pour le i^{ieme} élément d'une liste on utilise des double crochets:

```
liste[[i]]
```

Pour les objets de type tableau de données ou listes, les différentes variables ou éléments peuvent avoir un nom et être appellées par celui-ci. Par exemple:

```
air$Ozone
l1$sites
```

L'utilisation d'opérateurs de comparaison et d'opérateurs logiques permet notamment de rechercher certains éléments des objets ayant certaines caractéristiques, grâce à la fonction `which`.

Par exemple, on peut obtenir toutes les valeurs de TempCelsius supérieures ou égales à 20:

```
condition=TempCelsius>20
indices=which(condition)
indices
TempCelsius[indices]
```

Notez bien la différence entre les **indices** et les **valeurs** des vecteurs: `which` renvoie des **indices**, c'est à dire des numéros d'éléments. Les valeurs correspondant à ces indices sont celles renvoyées par TempCelsius[indices].

On peut obtenir les numéros des lignes pour lesquelles Ozone et Solar.R sont tous deux renseignés par la commande suivante:

On peut alors s'intéresser seulement à la partie du tableau de données pour laquelle on dispose de ces renseignements:

```
air[lignes_renseignees,]
```

Exercice 8

Afficher la partie du tableau de données "air" correspondant aux 4 premières colonnes, et aux mois de juillet et août.

On peut récupérer les indices d'un vecteur dans l'ordre (croissant ou décroissant) de ce vecteur. Par exemple, les instructions suivantes renvoient 5, 1, 4, 3, 2, i.e. l'élément le plus petit de x est en position 5, le deuxième plus petit est en position 1, etc.,

```
x <- c(3, 10, 5, 4, 1)
order(x, decreasing=F)
```

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

Cela permet notamment de “ranger” une table de données selon l’ordre défini par une ou plusieurs des variables du tableau. Par exemple, on peut remplacer la table “air”, par la même table ordonnée par date (en priorité, par mois, et pour chaque mois, par jour):

```
air=air[order(Month,Day, decreasing=F),]
print(air[1:10,])
```

```
   Ozone Solar.R Wind Temp Month Day
1     41     190  7.4   67     5   1
2     36     118  8.0   72     5   2
3     12     149 12.6   74     5   3
4     18     313 11.5   62     5   4
5     NA      NA 14.3   56     5   5
6     28      NA 14.9   66     5   6
7     23     299  8.6   65     5   7
8     19      99 13.8   59     5   8
9      8      19 20.1   61     5   9
10    NA     194  8.6   69     5  10
```

3.3 Fonctions

Nous avons d’ores et déjà utilisé un certain nombre de fonctions, comme c(), seq(), rep(), array(), matrix(), getwd(), is.na(), etc. D’autres fonctions très utiles sont décrites dans l’aide-mémoire.

Elles ont toutes un point commun: elles s’écrivent avec des parenthèses, dans lesquelles l’utilisateur précise la valeur des arguments si besoin est (par exemple pour getwd(), les parenthèses restent vides).

Les arguments peuvent être obligatoires (la fonction ne peut pas fonctionner si ces arguments ne sont pas fournis par l’utilisateur) ou au contraire optionnels. Par exemple, dans

```
read.table(file="air.txt",sep=",", header=T)
```

l’argument `file` est obligatoire, et l’argument `header` est optionnel.

Toutes les fonctions que nous avons utilisées jusqu’à présent sont définies sur le package de base de R.

On peut être amené cependant à rechercher une fonction plus particulière, non existante sur le package de base, mais fournie par un autre package. Par exemple, on peut être amené à utiliser le package “ade4” pour réaliser des analyses factorielles.

Pour être en mesure d’utiliser les fonctions d’un package, il faut alors

- Installer le package (les fonctions du package sont enregistrées sur le disque dur), par exemple par la commande (il faut être connecté à internet pour pouvoir procéder comme suit):

```
install.packages("ade4")
```

- Charger le package (les fonctions du package sont mises en mémoire dans l’environnement R)

```
library(ade4)
```

L’utilisateur peut créer lui-même ses fonctions, par exemple s’il souhaite répéter plusieurs fois une même série d’opérations.

Une fonction s’écrit de la manière suivante:

```
mafonction <-function(argument1,argument2){
  ...
  ...
  resultat <- ...
  return(resultat)
}
```

Par exemple, la fonction

```
Tconversion  <-function(x){
    reponse=(x-32)*5/9
    return(reponse)
}
```

convertit les températures en degrés Fahrenheit, en températures en degrés Celsius. On peut par exemple la tester de cette manière:

```
Tconversion(451)
```

4 Statistiques descriptives

Voici maintenant quelques pistes pour réaliser quelques statistiques très simples sur les données. Il s'agit de décrire les variables une à une (on parle de **statistiques univariées**), soit par le calcul de **métriques** (par exemple calcul de moyenne) soit en passant par des **représentations graphiques**. Ce cours constitue donc une courte (et simple) introduction à l'utilisation des méthodes statistiques avec le logiciel R.

Plusieurs métriques simples permettent de décrire les variables considérées indépendamment les unes des autres (on parle de métrique ou de statistique univariée).

4.1 Moyenne et médiane

On peut décrire la distribution d'une variable grâce à la moyenne ou la médiane.

```
mean(taille)
```

```
[1] 170.7638
```

```
median(taille)
```

```
[1] 171
```

Dans le cas où la variable d'intérêt comporte des "NA", il est nécessaire de préciser na.rm=T pour calculer ces métriques:

```
mean(Ozone)
```

```
[1] NA
```

```
mean(Ozone, na.rm=T)
```

```
[1] 42.12931
```

La moyenne d'un échantillon $x = (x_1, x_2, ..., x_n)$ est égale à

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

La médiane d'un échantillon $x = (x_1, x_2, ..., x_n)$ est égale à la valeur x_i telle qu'il y ait la moitié des observations au-dessus de x_i, et la moitié en dessous.

La distinction entre moyenne et médiane peut être importante pour certains types de distributions. Dans le cas de la variable "taille", il y a très peu de différence entre moyenne et médiane. Cela est lié au fait que la distribution de "taille" est symétrique autour de la moyenne: il y aura environ autant d'individus au dessus de la moyenne que d'individus en dessous de la moyenne.

Pour une distribution asymétrique (comme celle de la variable "depenses" par exemple, la différence n'est pas anodine. Exécutez les commandes suivantes pour vous en convaincre:

```
mean(depenses)
```

```
[1] 73.46734
```

```
median(depenses)
```

```
[1] 62
```

Dans ce cas, la médiane est nettement plus faible que la moyenne. Cela est lié au fait que certains individus ont des dépenses largement plus fortes que la moyenne, alors qu'il n'y en a pas qui ont des dépenses largement plus faibles (de fait, une dépense ne peut pas être négative, en ce sens, la variable ne **peut pas** prendre une valeur beaucoup plus faible que sa moyenne).

4.2 Variance et écart-type

On peut décrire la variabilité des données (variance, écart-type):

```
var(taille)
```

```
[1] 128.8581
```

```
sd(taille)
```

```
[1] 11.35157
```

La variance et l'écart type d'un échantillon $x = (x_1, x_2, ..., x_n)$ sont égales à

$$var(x) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 sd(x) = \sqrt{var(x)}$$

$\sum (x_i - \bar{x})^2$ est la somme des écarts au carré entre les observations et la moyenne. Autrement dit, la variance est, à peu de choses près,la moyenne des [[écarts à la moyenne] au carré].

Ces mesures de variabilité sont moins évidentes à interpréter, dans l'absolu, que des métriques comme la moyenne ou la médiane. Elles sont faciles à interpréter soit en relatif (telle variable a une plus forte variance qu'une autre), soit dans l'absolu, dans le cas où la variable d'intérêt suit une loi normale (ou gaussienne). Dans ce cas, l'écart-type donne une idée de l'"étalement" de la fameuse "cloche" propre aux variables dont la distribution est normale.

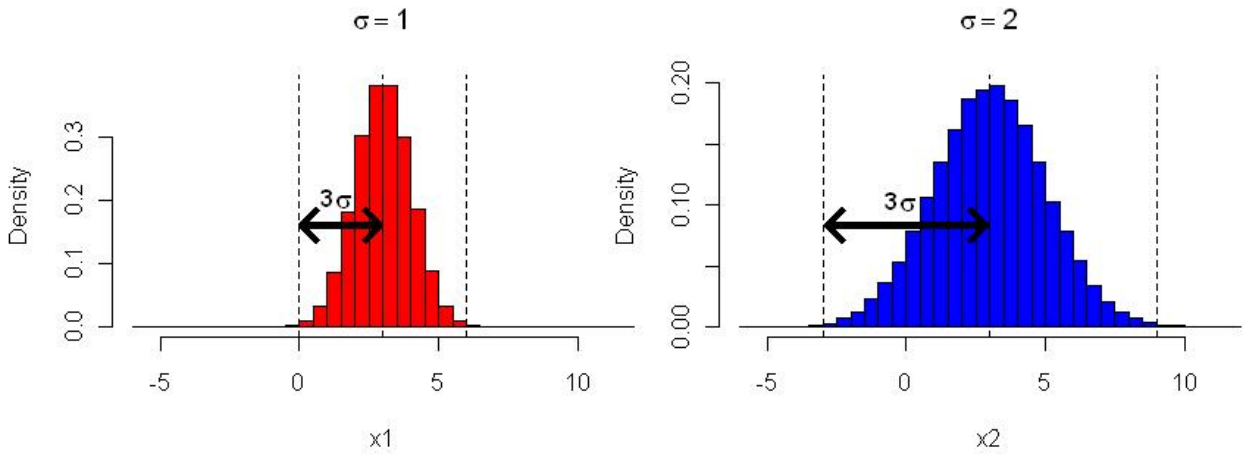


Figure 2: Effet du paramètre σ (écart type de la loi normale) sur l'étalement de la distribution.

4.3 Description de groupes ou de sous-échantillons

On peut être intéressé par les métriques d'une variable non seulement "dans l'absolu", mais aussi en fonction de groupes donnés.

```
mean(taille[which(sexe=="homme")])
```

```
[1] 176.3495
```

```
mean(taille[which(sexe=="femme")])
```

```
[1] 164.7708
```

Dans ce cas, la fonction tapply peut être particulièrement utile, en simplifiant le code ci-dessus (qui est d'autant plus long qu'il y a de nombreux groupes). Elle signifie "Applique la fonction f (ici 'mean') à la variable v1 (ici 'taille'), en fonction des groupes définis par la variable v2 (ici 'sexe').

```
tapply(taille, sexe, "mean")
```

```
   femme    homme
164.7708 176.3495
```

De la même manière, on peut calculer très efficacement d'autres métriques par groupe:

```
tapply(taille, sexe, "median")
```

```
femme homme
  165   176
```

```
tapply(taille, sexe, "sd")
```

```
    femme     homme
10.195437  9.381572
```

```
tapply(taille, sexe, "var")
```

```
   femme    homme
103.9469  88.0139
```

Exercice 9

Essayez de calculer la moyenne d'Ozone par mois. Quel résultat obtenez-vous? Comment pouvez-vous régler le problème?

4.4 Quantiles, minimum, maximum

Définition: le quantile d'ordre q d'une variable X correspond à la valeur λ telle que q% des données sont inférieures à q.

$$pr(X \leq \lambda) = q \tag{1}$$

Considérons par exemple la variable taille:

```
quantile(taille, c(0.05,0.95))
```

```
 5% 95%
151 191
```

Les quantiles d'ordre 5% (ou 0.05) et 95% (ou 0.95) de la variable taille sont 151 cm et 191 cm. Cela signifie que seulement 5% des individus font moins de 151 cm, et 95% des individus font moins de 191 cm (i.e. seulement 5% des individus font plus de 191 cm).

Les quantiles d'ordre 25%, 50%, et 75% sont aussi appelés premier quartile, deuxième quartile (ou médiane), et troisième quartile.

Par ailleurs, les minima et maxima des variables peuvent être affichés comme suit:

```
min(depenses)
```

```
[1] 12
```

```r
max(depenses)
```

```
[1] 256
```

Exercice 10

Décrire la distribution de la variable “depenses”, en général, puis en fonction du sexe.

5 Graphiques

5.1 Graphiques de base, nuages de points

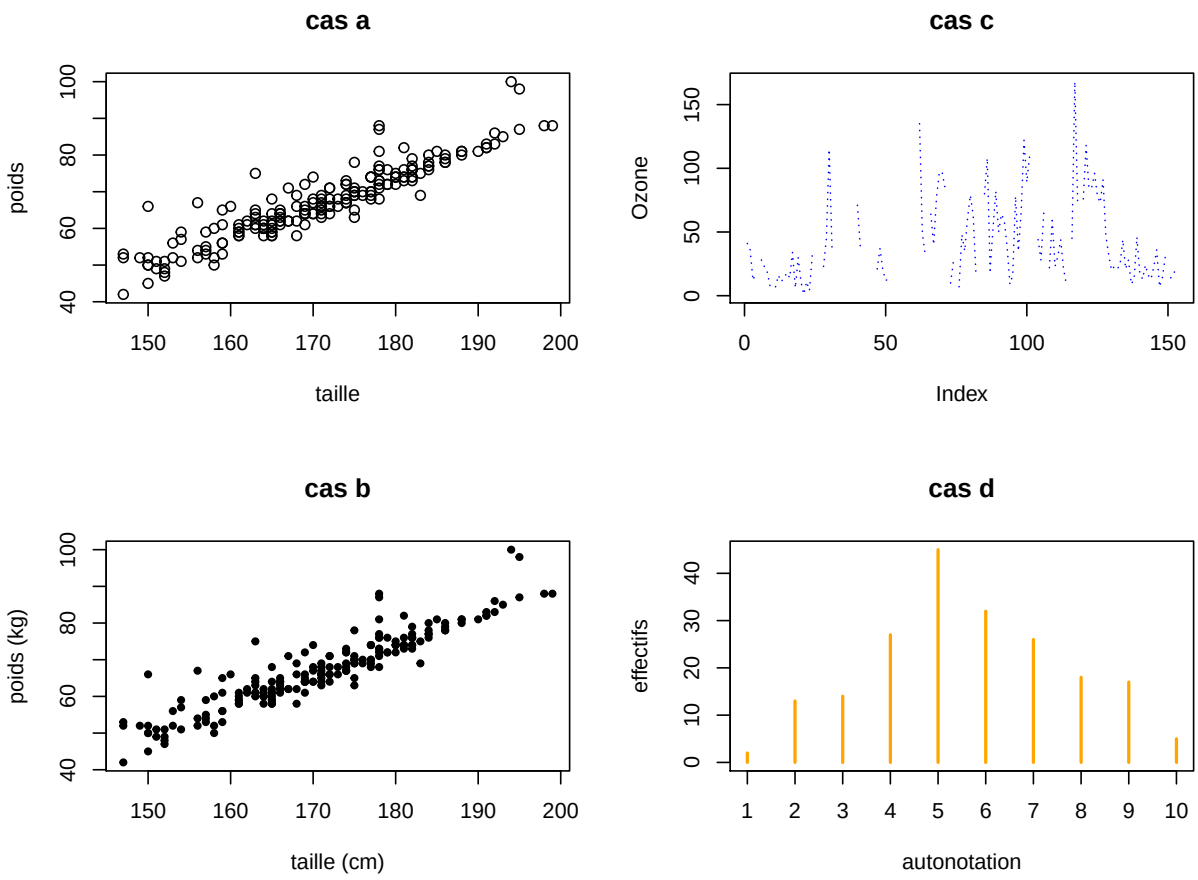
On peut tracer un nuage de points à l'aide de la fonction `plot`:

Cette fonction a de nombreux arguments, permettant entre autres de modifier

- le type de graphique (type) (par exemple, est-ce que les points doivent être reliés les uns aux autres par une ligne?)
- la couleur (col),
- le symbole (pch),
- le titre principal (main),
- les étiquettes des axes (xlab, ylab)
- la largeur des lignes (lwd)
- le type de lignes (lty)
- la taille des symboles (cex)

Dans sa version la plus simple, on lui fournit simplement les variables x et y -cas a-. On peut aussi (par exemple) lui préciser un symbole différent pour les points, et des étiquettes d'axes -cas b-. On peut tracer une ligne (type=“l”) pointillée (lty=3) -cas c-, ou encore réaliser un diagramme en bâtons (type=“h”) et lui préciser la couleur des bâtons (col=“orange”) -cas d-.

```r
layout(matrix(1:4, nrow=2))
plot(taille, poids, main="cas a")
plot(taille, poids, pch=20, xlab="taille (cm)", ylab="poids (kg)", main="cas b")
plot(Ozone, type="l", col="blue", lty=3, main="cas c")
effectifs=table(autonotation)
plot(effectifs, col="orange", type="h", lwd=2, main="cas d")
```



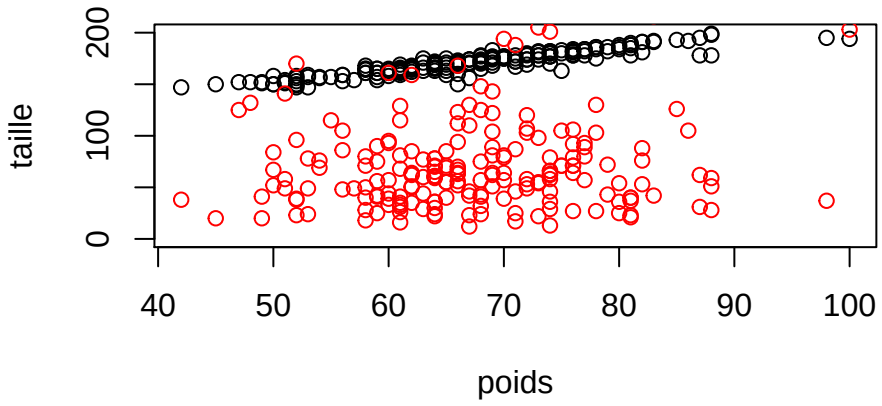
5.2 Superposer plusieurs graphes

On peut superposer plusieurs graphiques, par exemple en appelant d'abord une fonction graphique "de base" puis en appelant une des fonctions suivantes:

- `points()`pour superposer un graphique de type "plot"
- `abline()`pour rajouter une ligne, de pente "a" et d'ordonnée à l'origine "b"
- `text()` pour rajouter du texte
- `lines()` pour rajouter une ligne verticale, horizontale, ou entre des points spécifiés

Pour superposer plusieurs nuages de points (pour de telles superpositions il est souvent utile de définir "l'emprise" en x et y du graphique, avec les arguments xlim et ylim):

```
plot(poids,taille, ylim=c(0,200))
points(poids, depenses, col="red")
```



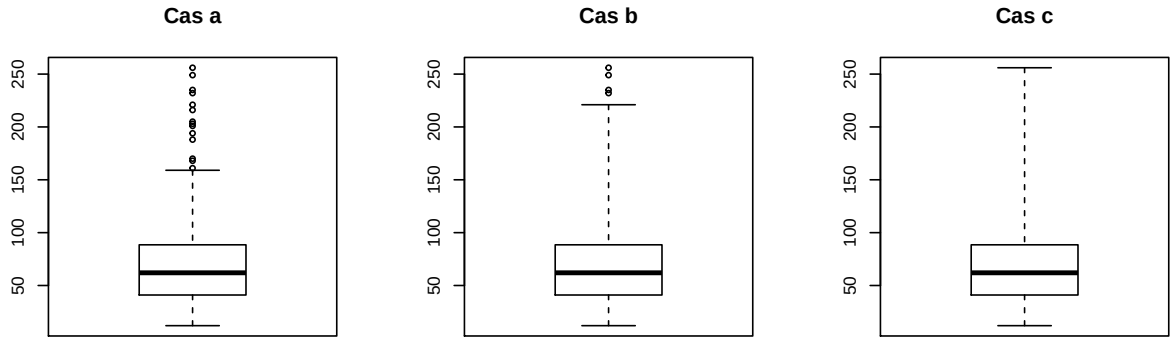
5.3 Boîtes à moustaches

Les boîtes à moustaches (ou boxplots, en anglais) sont un moyen extrêmement classique de représenter les distributions. La "boîte" est délimitée par les 1er et 3ème quartiles. Une barre horizontale marque la médiane. Par défaut, dans R, les moustaches s'étendent:

- soit jusqu'aux valeurs les plus extrêmes si celles-ci sont situées à une distance de moins d'1.5 fois la longueur de la boîte
- soit jusqu'à 1.5 fois la longueur de la boîte, les valeurs situées au-delà étant alors décrites comme "outliers"

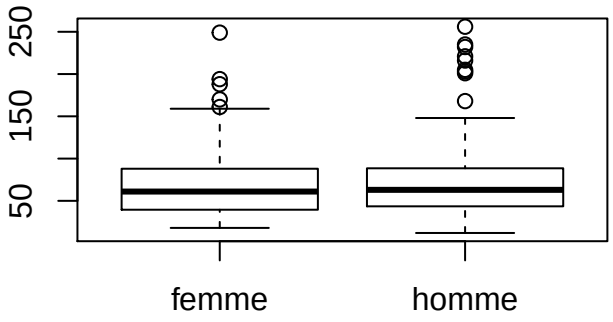
On peut modifier l'argument "range" de la fonction boxplot, pour que les moustaches s'étendent jusqu'à une distance de τ fois la longueur de la boîte ("range= τ", τ étant précisé par l'utilisateur), ou pour que les moustaches s'étendent jusqu'aux valeurs minimales et maximales observées("range= 0"). Dans tous les cas, les extrêmités des moustaches correspondent à des valeurs observées.

```
layout(matrix(1:3,nrow=1))
boxplot(depenses, main="Cas a")
boxplot(depenses, range=3, main="Cas b")
boxplot(depenses, range=0, main="Cas c")
```



Le plus souvent, les boîtes à moustaches sont utilisées pour représenter simultanément les distributions des observations pour plusieurs groupes, par exemple:

```
boxplot(depenses~sexe)
```

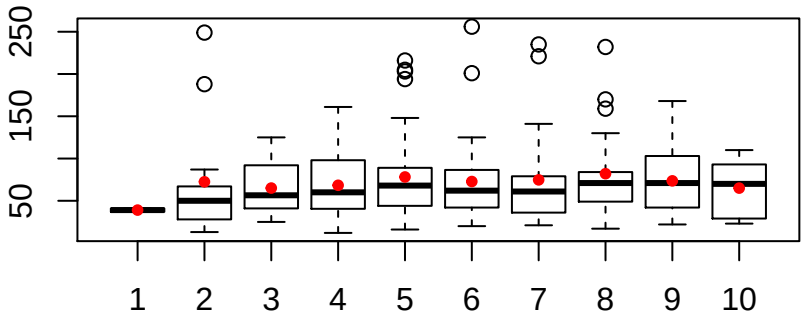


Exercice 11

Tracer les boîtes à moustaches représentant la distribution des dépenses en fonction de la variable autonotation.

Superposer la dépense moyenne par groupe (pensez à utiliser la fonction tapply).

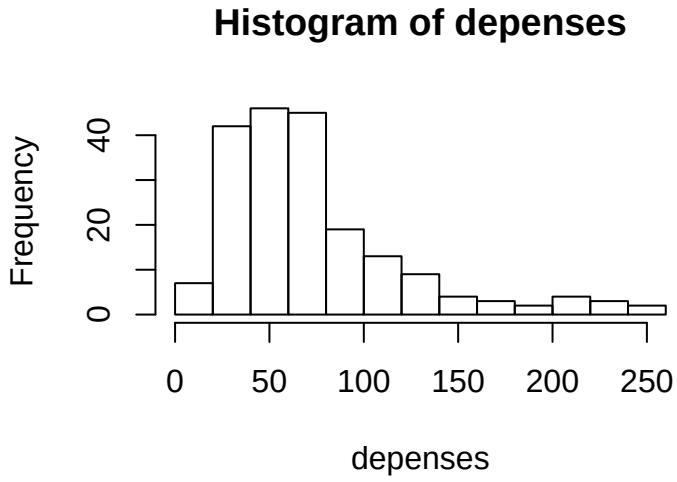
```
boxplot(depenses~autonotation)
moyennes=tapply(depenses,autonotation,"mean")
points(moyennes,col="red",pch=20)
```



5.4 Histogrammes

Les histogrammes offrent un mode de représentation des distributions plus précis que les boîtes à moustaches.

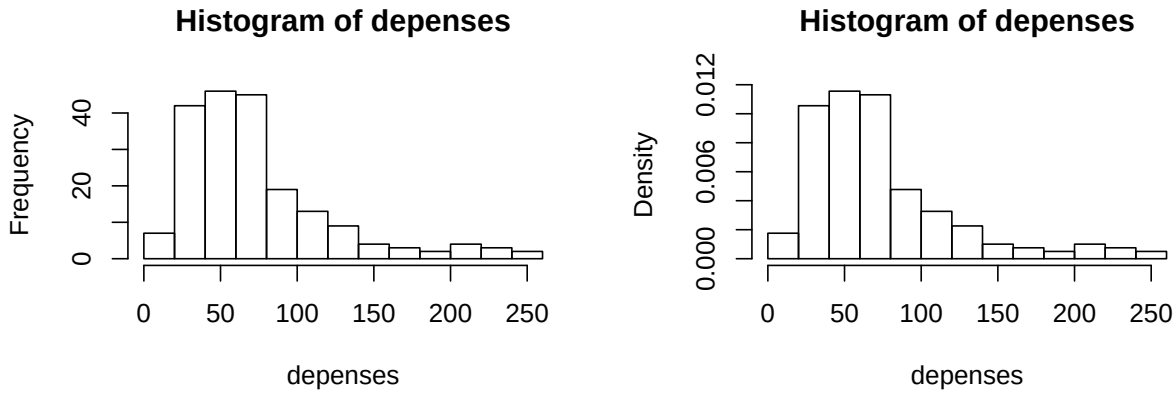
```
hist(depenses)
```



La fonction `hist` a plusieurs options importantes.

Par défaut, elle affiche, en axe des ordonnées, les effectifs (i.e. la fréquence d'une classe). Ainsi, on peut lire (par exemple) que 7 personnes ont des dépenses vestimentaires comprises environ entre 0 et 20 euros par mois. On peut modifier cela grâce à l'option "freq=F": dans ce cas l'axe des ordonnées permet de visualiser la **densité de probabilité**, c'est à dire que l'aire représentée pour une classe est égale à la proportion des observations qu'elle représente. Ainsi, la valeur de densité affichée pour la classe $[40, 60]$ est d'environ 0.012: cela signifie que cette classe représente environ $(60 - 40) * 0.012 = 24\%$ des observations.

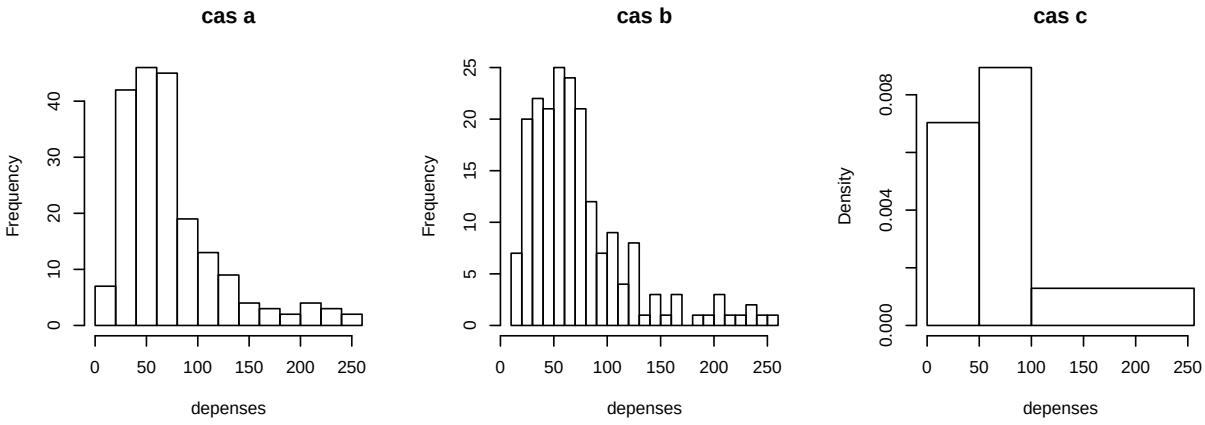
```r
layout(matrix(1:2,nrow=1))
hist(depenses)
hist(depenses, freq=F)
```



Remarque: Supposons que l'on préfère représenter les proportions d'individus dans chaque classe par la hauteur (et non l'aire) des barres (NB: ce qu'on réalise alors n'est pas un histogramme au sens strict du terme). On peut réaliser ce graphique de la manière suivante:

La fonction "breaks" permet de détailler les classes (soit le nombre de classes, soit les emplacements des limites de classes)

```r
layout(matrix(1:3,nrow=1))
hist(depenses, main="cas a")
hist(depenses, breaks=30, main="cas b")
hist(depenses, breaks=c(0,50,100,max(depenses)), main="cas c")
```



6 Aide-mémoire

Type d'action	Fonction	Objet renvoyé ou résultat de la fonction
Général	`help(...)`	ouvre le fichier d'aide de la fonction spécifiée

Type d'action	Fonction	Objet renvoyé ou résultat de la fonction
Général	`getwd()`	indique le nom du répertoire de travail courant
Général	`setwd(...)`	change le répertoire de travail
Général	`ls()`	liste d'objets contenus par l'environnement courant
Général	`rm(...)`	efface un objet de l'environnement
Général	`install.packages(...)`	installe le package spécifié
Général	`require(...)`	charge le package spécifié
Lecture et écriture de données	`read.table(...)`	charge un tableau de données depuis un fichier
Lecture et écriture de données	`write.table(...)`	écrit un tableau de données dans un fichier
Lecture et écriture de données	`attach(...)`	attache un tableau à l'environnement
Lecture et écriture de données	`detach(...)`	détache un tableau de l'environnement
Visualisation des données	`print(...)`	affiche l'objet dans la console
Visualisation des données	`edit(...)`	ouvre le tableau spécifié dans un éditeur
Visualisation des données	`colnames(...)`	affiche les noms de colonnes
Visualisation des données	`rownames(...)`	affiche les noms de lignes
Création d'objets	`c(...)`	crée un vecteur Création d'objets
Création d'objets	`seq(...)`	crée une séquence de valeurs régulièrement espacées
Création d'objets	`rep(...)`	crée une séquence de valeurs répétées
Création d'objets	`cbind(c1,c2)`	accole des colonnes c1 et c2
Création d'objets	`rbind(r1,r2)`	accole des lignes r1 et r2
Création d'objets	`data.frame(var1,var2)`	accole plusieurs variables ou tableaux
Indexation	`which(condition)`	indices des éléments vérifiant la condition
Indexation	`sort(x)`	vecteur x ordonné
Indexation	`order(x)`	indices du vecteur ordonné
Description	`length(x)`	longueur de x
Description	`nrow(M)`	nombre de lignes de M
Description	`ncol(M)`	nombre de colonnes de M
Description	`dim(M)`	dimension de M
Description	`mean(x)`	moyenne de x
Description	`var(x)`	variance de x
Description	`sd(x)`	écart-type de x
Description	`cor(x,y)`	coefficient de corrélation entre x et y
Description	`median(x)`	médiane de x
Description	`min(x)`	minimum de x
Description	`max(x)`	maximum de x
Description	`summary(x)`	"résume" la distrib. de x
Description	`tapply(x,y,"f")`	applique "f" à "x" pour chaque groupe de "y"
Graphiques de base	`plot(...)`	trace un nuage de points
Graphiques de base	`boxplot(...)`	trace des boîtes à moustaches
Graphiques de base	`hist(...)`	trace des histogrammes
Graphiques de base	`barplot(...)`	trace un graphique en barres
Graphiques de base	`contour(...)`	trace des courbes de niveau
Ajouts sur graphiques	`points(...)`	ajoute un nuage de points
Ajouts sur graphiques	`axis(...)`	change les valeurs affichées sur les axes
Ajouts sur graphiques	`text(...)`	ajoute du texte
Contrôle du dispositif graphique	`x11()`	ouverture d'un nouveau dispositif graphique R
Contrôle du dispositif graphique	`bmp(...)`	ouverture d'un fichier graphique bmp
Contrôle du dispositif graphique	`jpg(...)`	ouverture d'un fichier graphique jpg
Contrôle du dispositif graphique	`tiff(...)`	ouverture d'un fichier graphique tiff
Contrôle du dispositif graphique	`pdf(...)`	ouverture d'un fichier graphique pdf
Contrôle du dispositif graphique	`dev.off()`	fermeture des dispositifs graphiques
Contrôle du dispositif graphique	`layout(...)`	découpage de la fenêtre graphique

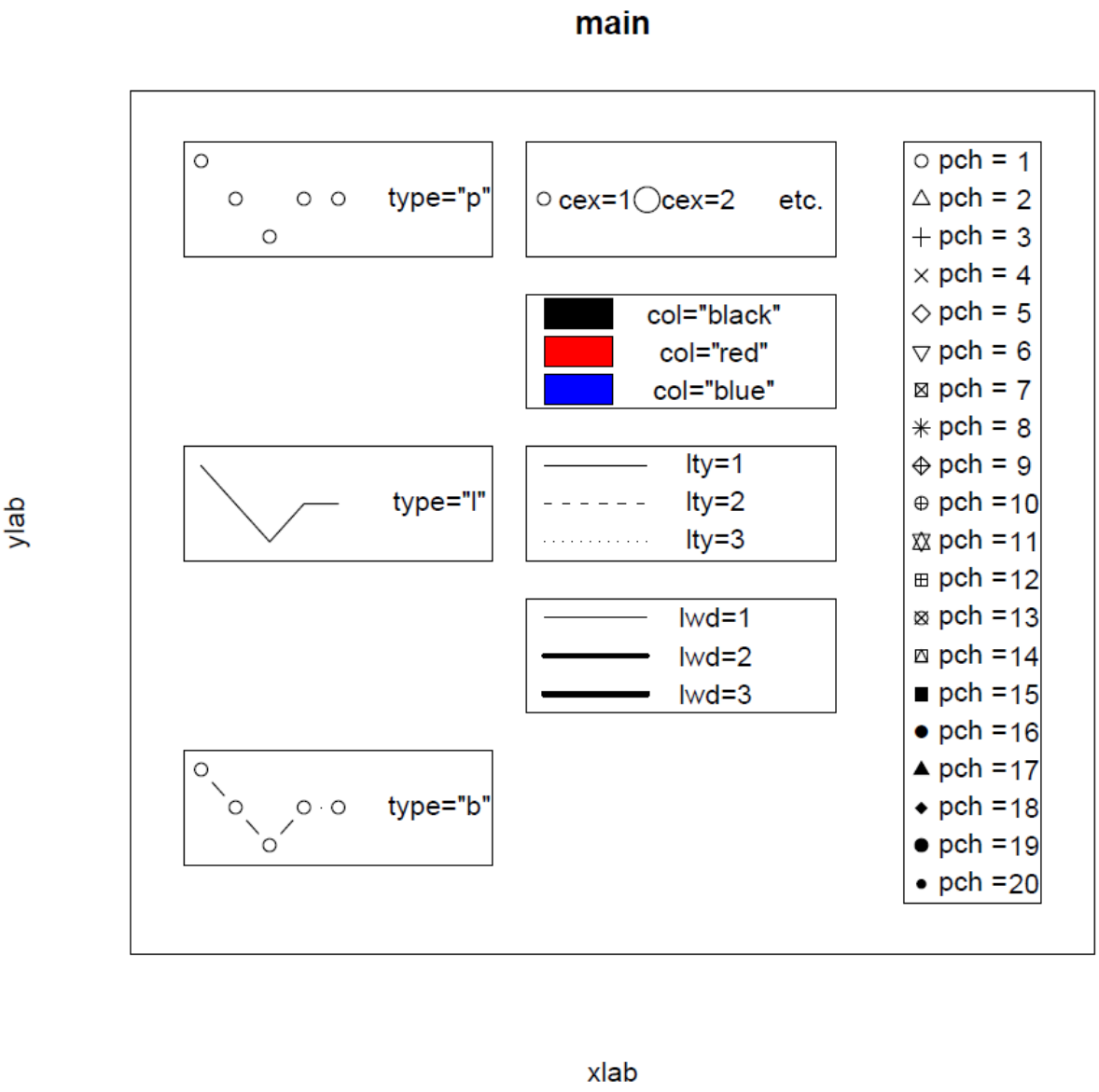


Figure 3: Quelques paramètres de la fonction plot

6.1 Besoin d'aide?

Quelle démarche suivre si une erreur s'est glissée dans votre script?

En premier lieu, il s'agit d'identifier le plus précisément possible la nature et l'emplacement du bug. Si un message d'erreur s'affiche, cela sera relativement simple. Si au contraire aucun message n'apparaît, mais que le résultat obtenu est manifestement erronné, cela peut être plus complexe.

Pour débugger le programme, il faut l'exécuter ligne par ligne, et regarder régulièrement les objets créés ou modifiés pour vérifier qu'ils sont corrects (cela suppose de "rentrer" dans les boucles ou les fonctions que vous avez créées, s'il y en a dans le script). En effet, un message d'erreur peut apparaître à l'exécution de la ligne 15 alors que l'erreur était à la ligne 10. De manière générale, il vaut mieux tester ses scripts régulièrement, au fur et à mesure de leur construction.

Si c'est une fonction de base de R ou une fonction issue d'un package qui ne "marche pas":

Les objets que vous fournissez à la fonction en guise d'arguments sont-ils conformes aux types d'objets attendus (par exemple, l'argument x doit-il être un facteur ou un vecteur)? Pour appeler une fonction avec les arguments corrects, vous pouvez vous inspirer des exemples fournis dans le fichier d'aide associé à la fonction (ouvert via `help(...)`).

Si la solution reste introuvable, les forums/cours/tutoriels dédiés à R sont très nombreux sur internet, et il y a de fortes chances, si vous rencontrez un problème, que quelqu'un d'autre l'ait rencontré avant vous et ait créé un post à ce sujet sur un forum. Par exemple, le forum du Groupe des Utilisateurs du logiciel R (en français) ou les archives R-help regroupent un très grand nombre de questions-réponses sur R.

http://forums.cirad.fr/logiciel-R/

https://stat.ethz.ch/mailman/listinfo/r-help

Pour le trouver, il vaut mieux googler les termes de votre choix + "R-CRAN", plutôt que "R" tout court. Vous pouvez également tester des moteurs de recherche centrés sur R comme celui-ci: http://rseek.org/.

Si malgré vos recherches, vous ne parvenez pas à comprendre l'origine de votre problème, vous pouvez créer vous-même un post sur un forum (en fournissant aux contributeurs le code incriminé dans un "exemple minimal" qui leur permettra de reproduire chez eux l'erreur que vous observez sur votre propre machine). Attention tout de même de ne pas poster avant d'avoir bien cherché par soi-même... Si la solution du problème est trop évidente les contributeurs du forum auront tendance à répondre de mauvaise grâce, voire à ne pas répondre du tout!

6.2 Besoin d'en savoir plus?

J'ai conçu ce tutoriel de manière à ce qu'il soit relativement court et ne décourage personne... Mais il est très loin d'être exhaustif, et si vous décidez de vous plonger dans R vous serez peut-être intéressés par des tutoriels plus complets ou des exemples d'utilisation plus précis.

Il existe de très nombreux ouvrages sur R (dont certains en français, comme "Statistiques avec R" de Cornillon et al.). Vous pourrez en trouver une liste ici (sous la rubrique Documentation/Books): http://cran.r-project.org/doc

Il existe par ailleurs une très large documentation gratuite sur le sujet. Le tutoriel "de base" que je recommande généralement est "R pour les débutants" d'Emmanuel Paradis:

http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

Il est en français, fait environ 80 pages, et il est très clair et didactique.

Parmi les classiques, il y a aussi "An introduction to R" de Venables and Smith, qui fait une centaine de pages, en anglais:

http://cran.r-project.org/doc/manuals/R-intro.pdf

Il existe une multitude d'autres tutoriels disponibles sur internet, qu'ils soient généraux comme les deux tutoriels cités ci-dessus, ou en lien avec un type d'analyse particulier (notamment statistique).

On peut par exemple trouver, dans les archives du site Semin-R, un certain nombre de présentations, centrées sur différents thèmes (par exemple, le chargement de données, l'obtention d'aide, le tracé de graphiques "complexes", etc.): http://rug.mnhn.fr/semin-r/

En ce qui concerne les analyses statistiques, le site du Pôle de Bioinformatique Lyonnais (PBIL) fournit sur son site de très nombreux tutoriels faisant le lien entre l'analyse statistique considérées (statistiques descriptives, tests, et surtout analyses multivariées) et le moyen de l'implémenter en R:

http://pbil.univ-lyon1.fr/R/enseignement.html

Il existe en outre des outils d'apprentissage en ligne comme ceux proposés par Datacamp (https://www.datacamp.com/). Certains cours sont payants mais moyennant la création d'un compte vous pouvez avoir accès à un certain nombre de cours gratuits (notamment ceux pour les débutants). Les cours sont truffés de vidéos et d'exercices interactifs vraiment très bien faits!

Enfin, (last but not least!) vous pourrez retrouver les supports de ce cours, ainsi que des tutoriels tutoriels, conseils et mini-tutos sur des sujets divers et variés (quoique toujours en lien avec R et l'analyse de données) sur mon blog, R-atique:

http://perso.ens-lyon.fr/lise.vaudor