Chapitre 1

Les concepts de base, l'organisation des données

Objectif

Ce chapitre présente les concepts de base du logiciel R (mode calculatrice, opérateur d'affectation, variables, utilisation de fonctions, paramètres) ainsi que les différents types et structures de données que R peut manipuler.

SECTION 1.1

Votre première session

Après avoir lancé le logiciel R en double-cliquant sur son icône sur le Bureau de Windows (ou bien à partir du Menu Démarrer), vous voyez apparaître, à la fin de l'affichage qui se déroule dans la console de R (appelée *R Console*), le **caractère d'invite de commande** > vous invitant à taper votre première instruction en langage R.

```
R version 2.10.0 (2009-10-26)
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.

>
```

Il s'agit du symbole d'incitation à donner une instruction (*prompt symbol*). Tapez par exemple `"R est mon ami"` puis validez en tapant la touche `ENTRÉE` (ou `RETURN`). Vous obtenez alors :

```
> "R est mon ami"
[1] "R est mon ami"
```

Comme vous pouvez le constater, R est bien éduqué et répond gentiment à votre requête. Ce sera généralement le cas, peut-être pour se faire pardonner son manque de convivialité. Nous expliquerons plus tard pourquoi la réponse de R est précédée de `[1]`.

1.1.1 R est une calculatrice

R, comme beaucoup d'autres langages de ce type, remplace aisément les fonctionnalités d'une calculatrice (très sophistiquée !). Il permet aussi, et c'est une grande force, de faire des calculs sur des vecteurs. Voici déjà quelques exemples très simples.

```
> 5*(-3.2)          # Attention, le séparateur décimal doit
                    # être un point (.)
[1] -16
> 5*(-3,2)          # sinon, l'erreur suivante est générée:

Erreur : ',' inattendu(e) dans "5*(-3,"

> 5^2               # Identique à 5**2.
[1] 25
> sin(2*pi/3)
[1] 0.8660254
> sqrt(4)           # Racine carrée de 4.
[1] 2
> log(1)            # Logarithme népérien de 1.
[1] 0
> c(1,2,3,4,5)      # Crée un vecteur contenant les cinq premiers
                    # entiers.
[1] 1 2 3 4 5
> c(1,2,3,4,5)*2    # Calcul des cinq premiers nombres pairs.
[1]  2  4  6  8 10
```

Astuce

Tout code R qui suit le caractère «#» est considéré par R comme un commentaire. En fait, il n'est pas interprété par R.

Vous pouvez maintenant quitter le logiciel R en tapant l'instruction suivante : `q()`.

Il vous est alors proposé de sauver une image de la session. En répondant oui, les commandes tapées précédemment seront de nouveau accessibles lors d'une prochaine réouverture de R, au moyen des flèches «haut» et «bas» du clavier.

1.1.2 Stratégie de travail

- Prenez l'habitude de stocker vos fichiers dans un dossier réservé à cet usage (nommé par exemple `TravauxR`). En outre, nous vous conseillons de taper toutes vos instructions R dans une fenêtre de script appelée *script* ou *R Editor*, accessible depuis le menu «Fichier/Nouveau script». Ouvrez une nouvelle fenêtre de script, cliquez dans le menu «Fenêtres/-Juxtaposées» puis copiez le script suivant :

```
5*(-3.2)
5^2
sin(2*pi/3)
sqrt(4)
c(1,2,3,4,5)
c(1,2,3,4,5)*2
```

Mac

Pour les Mac, le menu est «Fichier/Nouveau Document», et il n'est pas possible de juxtaposer les fenêtres.

À la fin de votre session, vous pourrez sauver ce script, dans le dossier `TravauxR`, sous le nom `monscript.R` par exemple, et le rouvrir lors d'une session ultérieure depuis le menu «Fichier/Ouvrir un script» («Fichier/Ouvrir Document» pour les Mac).

Ensuite, vous pouvez taper successivement les combinaisons de touches `CTRL+A` (`COMMAND+A` pour les Mac) pour sélectionner l'ensemble de ces instructions, puis `CTRL+R` (`COMMAND+ENTER` pour les Mac) pour les coller et les exécuter en une seule étape dans la console de R. Vous pouvez aussi exécuter une seule ligne d'instructions R du script en tapant `CTRL+R` lorsque le curseur clignotant se trouve sur la ligne en question dans la fenêtre de script.

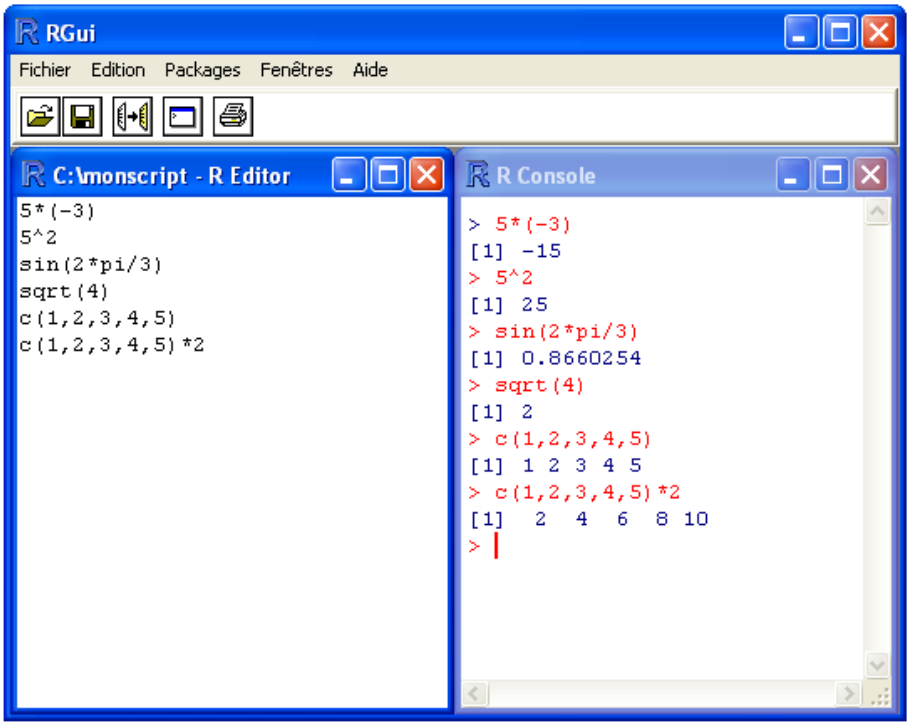

FIG. 1.1: Vue de la fenêtre de script et de la console de commandes.

Vous pouvez également utiliser la fonction `source()` depuis la console de R pour aller lire et exécuter le contenu de votre fichier. Cela évitera de surcharger inutilement votre console, comme nous le verrons dans une pratique ultérieure. Pour cela, il est utile de procéder ainsi :

a) cliquez une fois dans la fenêtre *R Console* ;

b) allez dans le menu «Fichier/Changer le répertoire courant» («Divers/Changer de répertoire de travail» pour les Mac) ;

c) explorez votre système de fichiers pour sélectionner le dossier `TravauxR` ;

d) tapez dans la console : `source("monscript.R")`.

- Prenez le plus possible l'habitude d'utiliser le système d'aide en ligne de R. Cette aide très complète, en anglais, est accessible au moyen de la fonction `help()`. Vous pouvez par exemple taper `help(source)` pour obtenir de l'aide sur la fonction `source()`.

Renvoi

Toutes ces notions seront détaillées dans les chapitres 4 et 7.

Astuce

Pour les utilisateurs travaillant sous l'environnement Microsoft, nous conseillons l'utilisation de l'éditeur de code Tinn-R, disponible à `http://www.sciviews.org/Tinn-R/`. Il offre en effet une meilleure interaction entre le code d'un script et son exécution. Il permet aussi de colorer syntaxiquement le code.

Linux

Notez l'existence des éditeurs JGR et Emacs/ESS sous Linux.

1.1.3 Affichage des résultats et redirection dans des variables

Comme vous l'avez sans doute remarqué, R répond à vos requêtes en affichant le résultat obtenu après évaluation. **Ce résultat est affiché puis perdu**. Dans une première utilisation, cela peut paraître agréable, mais dans une utilisation plus poussée il apparaît plus intéressant de rediriger la sortie R de votre requête en la stockant dans une variable : cette opération s'appelle aussi **affectation du résultat dans une variable**. Une affectation évalue ainsi une expression, mais n'affiche pas le résultat qui est en fait stocké dans un objet. Pour afficher ce résultat, il vous suffira de taper le nom de cet objet, suivi de la touche ENTRÉE.

Pour réaliser cette opération, on utilise la **flèche d'affectation** `<-`. La flèche `<-` s'obtient en tapant le signe inférieur (`<`) suivi du signe moins (`-`).

Pour créer un objet dans R, on utilise donc la syntaxe suivante :
`Nom.objet.a.creer <- instructions`

Par exemple :

```
> x <- 1        # Affectation.
> x             # Affichage.
[1] 1
```

On dit alors que `x` vaut `1`, ou que l'on affecte `1` à `x`, ou encore que l'on met dans `x` la valeur `1`. Notez que l'on peut aussi utiliser l'opération d'affectation dans l'autre sens `->` de la façon suivante :

```
> 2 -> x
> x
[1] 2
```

Attention

On peut aussi utiliser le signe =, mais son utilisation est moins générale et donc déconseillée. En effet, l'égalité en mathématique est une relation symétrique ayant un sens bien précis, très différent de celui de l'affectation. Par ailleurs, il y a des cas où l'utilisation du signe = ne fonctionne pas du tout.

Astuce

Notez qu'il est possible, en utilisant une paire de parenthèses, d'affecter une valeur à une variable tout en affichant le résultat de l'évaluation :

```
> (x <- 2+3)
[1] 5
```

Enfin, si une commande n'est pas complète à la fin d'une ligne, R affichera un signe d'invite différent, par défaut le signe plus (+), sur la deuxième ligne ainsi que sur les lignes subséquentes. R continuera d'attendre des instructions jusqu'à ce que la commande soit syntaxiquement complète.

```
> 2*8*10+exp(1)
[1] 162.7183
> 2*8*
+ 10+exp(
+ 1)
[1] 162.7183
```

Attention

Voici les **règles pour choisir un nom de variable** dans R : tout nom de variable ne peut être constitué que de caractères alphanumériques ainsi que du point (.) ; les noms de variables sont *case sensitive*, signifiant que R fait la distinction entre minuscules et majuscules ; un nom de variable ne peut pas contenir des espaces ou commencer par un chiffre, sauf s'il est encadré de guillemets "".

Prise en main

L'indice de masse corporelle (IMC) permet de déterminer la corpulence d'une personne. Il se calcule au moyen de la formule suivante :

$$\text{IMC} = \frac{\text{Poids (kg)}}{\text{Taille}^2 \text{ (m)}}.$$

Calculons notre IMC. Pour cela, il suffit de taper dans votre fenêtre de script les lignes suivantes :

```
# Il est possible d'écrire 2 instructions
# sur la même ligne grâce au signe ;
Mon.Poids <- 75 ; Ma.Taille <- 1.90

Mon.IMC <- Mon.Poids/Ma.Taille^2
Mon.IMC
```

Lancez ce script en utilisant la stratégie de travail vue précédemment. Vous pouvez ensuite modifier ce script pour calculer votre propre IMC.

Nous proposons une fonction permettant de visualiser votre type de corpulence. Pour cela, veuillez lancer maintenant les deux instructions suivantes :

```
source("http://www.biostatisticien.eu/springeR/IMC.R",
encoding="utf8")
affiche.IMC(Mon.IMC)
```

Vous apprendrez comment programmer ce genre de résultat au cours des chapitres ultérieurs.

1.1.4 Utilisation de fonctions

Nous avons vu quelques utilisations des fonctions `sin()`, `sqrt()`, `exp()` et `log()`. Le logiciel R contient de nombreuses autres fonctions dans sa version de base, et on peut en ajouter des milliers d'autres (en installant des *packages*, ou même en les réalisant soi-même).

On notera que toute fonction dans R est définie par son **nom** et par la liste de ses **paramètres**. La plupart des fonctions renvoient une **valeur**, qui peut être un nombre, un vecteur, une matrice...

Utiliser une fonction (on dit aussi **appeler** ou **exécuter**) se fait en tapant le nom de celle-ci, suivi, entre une paire de parenthèses, de la liste des paramètres (formels) que l'on veut utiliser. Les paramètres sont séparés par des virgules. Chacun des paramètres peut être suivi du signe `=` et de la valeur que l'on veut donner au paramètre. Cette valeur du paramètre formel sera appelée paramètre effectif, paramètre d'appel ou parfois paramètre d'entrée.

Attention

Prenez garde toutefois au fait que le R utilise le terme anglais *argument* pour désigner ce que nous appelons paramètre. Nous avons choisi de ne pas utiliser l'anglicisme *argument* dans cet ouvrage.

Nous utiliserons donc l'instruction

```
nomfonction(par1=valeur1,par2=valeur2,par3=valeur3)
```

où `par1, par2, ...` sont appelés les paramètres de la fonction tandis que `valeur1` est la valeur que l'on donne au paramètre `par1`, etc. On peut toutefois noter qu'il n'est pas forcément nécessaire d'indiquer les paramètres mais seulement leurs valeurs, pour autant que l'on respecte leur ordre.
Pour toute fonction présente dans R, certains paramètres sont obligatoires et d'autres sont facultatifs (car une valeur par défaut est déjà fournie dans le code de la fonction).

Attention

Ne pas oublier de saisir la paire de parenthèses lors de l'appel d'une fonction. En effet, une erreur courante en première utilisation consiste à les oublier :

```
> factorial
function (x)
gamma(x + 1)
<environment: namespace:base>
> factorial(6)
[1] 720
```

La sortie de la première instruction fournit le code (c'est-à-dire la recette de fabrication) de la fonction considérée tandis que la deuxième instruction l'exécute. Cela est également valable pour les fonctions définies sans paramètre comme le montre l'exemple suivant :

```
> date()
[1] "Fri Oct  8 15:08:53 2010"
> date
```

```
function ()
.Internal(date())
<environment: namespace:base>
```

Bien évidemment, il n'est pas question ici de commenter le code des fonctions précédentes.

Afin de mieux comprendre la façon dont les paramètres peuvent être utilisés, prenons l'exemple de la fonction `log(x,base=exp(1))`. On peut remarquer qu'elle admet deux paramètres : `x` et `base`.

Le paramètre `x` est obligatoire, c'est le nombre dont on veut calculer le logarithme. Le paramètre `base` est un paramètre optionnel puisqu'il est suivi du signe `=` et de la valeur par défaut `exp(1)`.

Astuce

Un paramètre obligatoire est un paramètre qui n'est pas suivi du signe `=`. Un paramètre est optionnel s'il est suivi du signe `=`.

Ainsi, dans l'appel suivant, le calcul effectué sera celui du logarithme népérien du nombre `1` puisque la valeur de base n'est pas renseignée :

```
> log(1)
[1] 0
```

Remarque

Certaines fonctions ne possèdent aucun paramètre obligatoire, comme la fonction `matrix` que l'on verra plus tard.

Un dernier point important à noter est que l'**on peut appeler une fonction en jouant sur les paramètres de plusieurs façons différentes**. Cela est un atout important de R en termes de simplicité d'utilisation, et il convient de bien comprendre ce principe de fonctionnement. Ainsi, pour calculer le logarithme népérien de 3, on peut utiliser n'importe laquelle des expressions suivantes.

```
log(3)
log(x=3)
log(x=3,base=exp(1))
log(x=3,exp(1))
log(3,base=exp(1))
log(3,exp(1))
log(base=exp(1),3)
log(base=exp(1),x=3)
```

Attention

Il faut prendre garde au fait que l'appel suivant

```
log(exp(1),3)
```

revient à calculer le logarithme de `exp(1)` en base `3`.

Pour terminer, il est bon de noter que nous avons pu voir, sur la page précédente, le code de la fonction `factorial()` :

```
> factorial
function (x)
gamma(x + 1)
<environment: namespace:base>
```

Cette fonction a été définie par les développeurs de R au moyen des instructions suivantes :

```
> factorial <- function(x) gamma(x+1)
```

Il est donc très facile en R de programmer soi-même une nouvelle fonction en utilisant la fonction `function()`. Par exemple, voici comment programmer la fonction des deux variables n et p qui calcule le coefficient du binôme de Newton $\binom{n}{p} = \frac{n!}{p!(n-p)!}$:

```
> binome <- function(n,p) factorial(n)/(factorial(p)*
+                         factorial(n-p))
```

Il est ensuite possible d'utiliser votre nouvelle fonction comme n'importe quelle autre fonction R :

```
> binome(4,3)
[1] 4
```

Nous verrons de façon beaucoup plus détaillée comment créer des fonctions plus élaborées dans le chapitre 6.

SECTION 1.2

Les données dans R

Comme la plupart des langages informatiques, R dispose des types de données classiques. Selon la forme des données saisies, R sait d'ailleurs reconnaître automatiquement le type de ces données. Une des grandes forces de R réside aussi dans la possibilité d'organiser les données de façon structurée. Cela sera très utile lors de l'utilisation de nombreuses procédures statistiques qui seront détaillées ultérieurement.

1.2.1 Nature (ou type, ou mode) des données

Les fonctions `mode()` et `typeof()`, identiques à de rares subtilités près non détaillées ici, permettent de gérer le type des données.

Remarque

La fonction `class()` est plus générale puisqu'elle permet de gérer à la fois le type et la structure des données. Elle sera présentée plus loin. Pour des raisons pédagogiques, nous utilisons ici la commande `typeof()`.

Énumérons maintenant les divers types de données (aussi appelés modes).

1.2.1.1 Type numérique (`numeric`)

Il y a deux types numériques : les entiers (`integer`) et les réels (`real` ou `double`).

Lorsque vous saisissez :

```
> a <- 1
> b <- 3.4
> c <- as.integer(a)
> typeof(c)
[1] "integer"
```

les variables `a` et `b` sont du type `"double"` et la variable `c` a la même valeur que `a` excepté qu'elle a été forcée à être du type `"integer"`. L'intérêt est que son stockage prend moins de place en mémoire. Les instructions commençant par `as.` sont très courantes en R pour convertir une donnée en un type différent. Nous verrons dans la section suivante comment vérifier que le type d'un objet est numérique.

1.2.1.2 † Type complexe (`complex`)

On fabrique un nombre complexe à l'aide de la lettre `i`. On utilise les fonctions `Re()` pour la partie réelle, `Im()` pour la partie imaginaire, `Mod()` pour le module et `Arg()` pour l'argument.

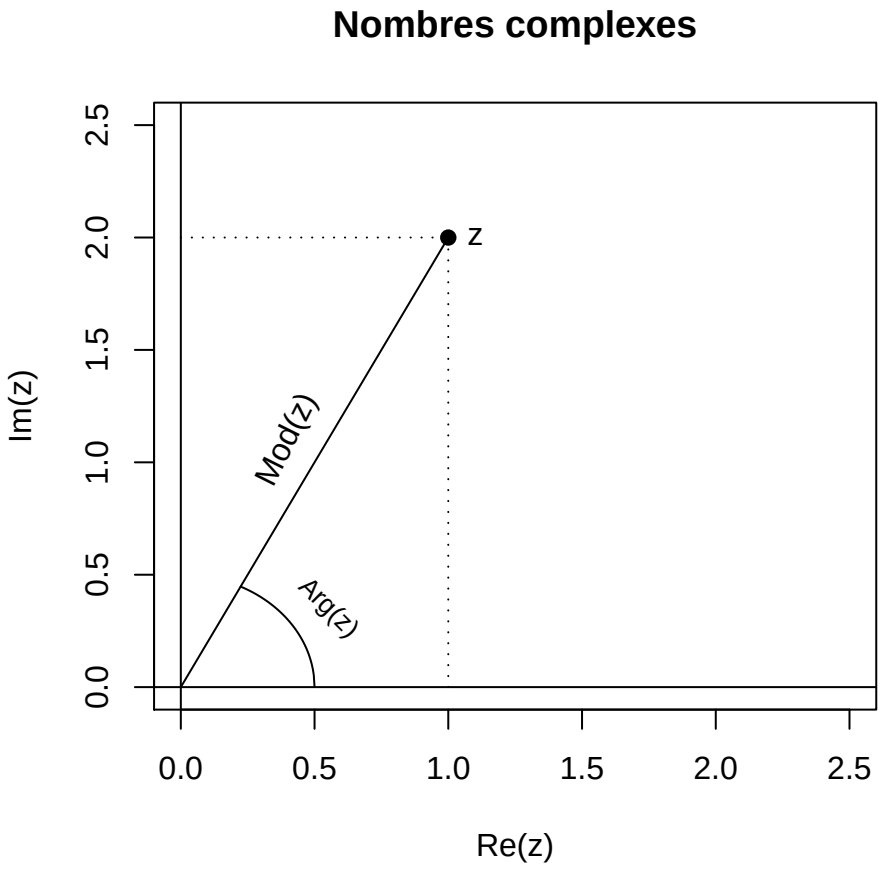

FIG. 1.2: Caractéristiques d'un nombre complexe.

Voici quelques exemples :

```
> 1i
[1] 0+1i
> z <- 1+2i
> typeof(z)
[1] "complex"
> is.complex(z) # Permet de savoir si un objet est du type
                # complex.
[1] TRUE
> Re(z)
[1] 1
> Im(z)
```

```
[1] 2
> Mod(z)
[1] 2.236068
> Arg(z)
[1] 1.107149
```

1.2.1.3 Type booléen ou logique (`logical`)

Le type `logical`, résultat d'une condition logique, peut prendre les valeurs `TRUE` ou `FALSE`. Voici quelques instructions pour créer des valeurs logiques :

```
> b>a
[1] TRUE
> a==b
[1] FALSE
> is.numeric(a)
[1] TRUE
> is.integer(a)
[1] FALSE
> x <- TRUE
> is.logical(x)
[1] TRUE
```

Attention

`TRUE` et `FALSE` peuvent être saisis de manière plus succincte en tapant respectivement `T` et `F`.

Lorsque cela se révèle nécessaire, ce type de données est naturellement converti en type numérique sans qu'il y ait à le spécifier : `TRUE` vaut `1` et `FALSE` vaut `0`. Cela est illustré par l'exemple suivant :

```
> TRUE + T + FALSE*F + T*FALSE + F
[1] 2
```

1.2.1.4 Données manquantes (`NA`)

Une donnée manquante ou non définie est indiquée par l'instruction `NA` (pour *non available* = non disponible), et plusieurs fonctions existent pour gérer ce type de données. En fait, `R` considère ce type de données comme étant une valeur logique constante. Il ne s'agit donc pas d'un type de données à proprement parler. Voici quelques exemples faisant intervenir l'instruction `NA` :

```
> x <- c(3,NA,6)
> is.na(x)
[1] FALSE  TRUE FALSE
> mean(x)               # Tentative de calcul de la moyenne de x.
[1] NA
```

```
> mean(x,na.rm=TRUE)    # Le paramètre na.rm signifie enlever
                        # les NA (NA.remove).
[1] 4.5
```

Cette notion est très importante lors de la lecture de fichiers de données statistiques et sera développée dans le chapitre 3.

Attention

Ne pas confondre `NA` avec le mot réservé `NaN` signifiant *not a number* :

```
> 0/0
[1] NaN
```

Notez également que l'instruction suivante ne renvoie pas `NaN` mais l'infini, représenté en R par le mot réservé `Inf`.

```
> 3/0
[1] Inf
```

1.2.1.5 Type chaînes de caractères (`character`)

Toute information mise entre guillemets (simple ' ou double ") correspond à une chaîne de caractères :

```
> a <- "R est mon ami"
> mode(a)
[1] "character"
> is.character(a)
[1] TRUE
```

Les conversions en chaîne de caractères depuis un autre type sont possibles. Les conversions réciproques s'appliquent dès lors que le contenu entre les guillemets peut être correctement interprété par R. Notez enfin que certaines conversions se font de manière automatique. Voici quelques exemples :

```
> as.character(2.3)          # Conversion vers une chaîne de
                             # caractères.
[1] "2.3"
> b <- "2.3"
> as.numeric(b)              # Conversion depuis une chaîne de
                             # caractères.
[1] 2.3
> as.integer("3.4")          # Conversion depuis une chaîne de
                             # caractères.
```

```
[1] 3
> c(2,"3")                    # Conversion automatique.
[1] "2" "3"
> as.integer("3.quatre")      # Conversion impossible.
[1] NA
```

Remarque

Nous verrons au chapitre 3 les différences existant entre les double et simple guillemets.

1.2.1.6 † Données brutes (`raw`)

R offre la possibilité de travailler directement avec des octets (affichés sous forme hexadécimale). Cela peut parfois être utile lors de la lecture de certains fichiers au format binaire. Nous en verrons des exemples dans le chapitre 5.

```
> x <- as.raw(c(9,10,15,16))
> x
[1] 09 0a 0f 10
> mode(x)
[1] "raw"
```

Récapitulatif

Tab. 1.1: Les différents types de données en R.

Type des données	Type sous R	Présentation
réel (entier ou non)	`numeric`	`3.27`
complexe	`complex`	`3+2i`
logique (vrai/faux)	`logical`	`TRUE` ou `FALSE`
manquant	`logical`	`NA`
texte (chaîne)	`character`	`"texte"`
binaires	`raw`	`1c`

1.2.2 Structure des données

R offre la possibilité d'organiser les différents types de données définies précédemment. La fonction `class()` permettra d'accéder aux différents types de structure que nous allons présenter.

1.2.2.1 Les vecteurs (`vector`)

Cette structure de données est la plus simple. Elle représente **une suite de données de même type**. La fonction permettant de créer ce type de structure (c'est-à-dire les vecteurs) est la fonction `c()` (pour collection ou concaténation). D'autres fonctions comme `seq()` ou bien les deux points `:` permettent aussi de créer des vecteurs. Notez que lors de la création d'un vecteur, il est possible de mélanger des données de plusieurs types différents. R se charge alors d'opérer une conversion implicite vers le type de donnée le plus général comme vous pouvez le constater dans les exemples ci-dessous.

```
> c(3,1,7)
[1] 3 1 7
> c(3,TRUE,7)
[1] 3 1 7
> c(3,T,"7")
[1] "3"    "TRUE" "7"
> seq(from=0,to=1,by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(from=0,to=20,length=5)
[1]  0  5 10 15 20
> vec <- 2:36
> vec
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[20] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

Attention

Les indications `[1]` et `[26]` fournissent le rang de l'élément qui les suit dans le vecteur `vec`.

```
> is.vector(vec)
[1] TRUE
> x <- 1:3
> x
[1] 1 2 3
> y <- c(1,2,3)
> y
[1] 1 2 3
> class(x)
[1] "integer"
> class(y)
[1] "numeric"
```

Expert

Notez que les instructions `c()` et `:` fournissent le même affichage, mais `x` et `y` sont ici stockées en interne de façon différente. Le type `integer` utilise moins de mémoire que le type `numeric`.

1.2.2.2 Les matrices (`matrix`) et les tableaux (`arrays`)

Ces deux notions généralisent la notion de vecteur puisqu'elles représentent des suites à double indice pour les matrices et à multiples indices pour les tableaux (*array*). Ici aussi **les éléments doivent avoir le même type**.

L'instruction suivante

```
> X <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> X
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

permet de créer (et stocker dans la variable `X`) une matrice comportant quatre lignes (*row* signifie ligne) et trois colonnes remplies par lignes successives (`byrow =TRUE`) avec les éléments du vecteur `1:12` (c'est-à-dire les douze premiers entiers).

De la même manière, il est possible de créer une matrice remplie par colonnes successives (`byrow=FALSE`).

```
> Y <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)
> Y
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> class(Y)
[1] "matrix"
```

La fonction `array()` permet de créer des matrices multidimensionnelles à plus de deux dimensions comme cela est illustré sur la figure suivante (pour un *array* ayant trois dimensions).

```
> X <- array(1:12,dim=c(2,2,3))
> X
, , 1
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
, , 3
     [,1] [,2]
[1,]    9   11
[2,]   10   12
> class(X)
[1] "array"
```

Attention

Il est possible de créer des tableaux à plus de trois dimensions au moyen du paramètre `dim` qui peut en effet avoir une longueur supérieure à 3.

1.2.2.3 Les listes (`list`)

La structure du langage R la plus souple et à la fois la plus riche est celle de la liste. Contrairement aux structures précédentes, les listes permettent de **regrouper dans une même structure des données de types différents** sans pour autant les altérer. De façon générale, chaque élément d'une liste peut ainsi être un vecteur, une matrice, un *array* ou même une liste. Voici un premier exemple :

```
> A <- list(TRUE,-1:3,matrix(1:4,nrow=2),c(1+2i,3),
+            "Une chaîne de caractères")
> A
[[1]]
[1] TRUE
[[2]]
[1] -1  0  1  2  3
[[3]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[[4]]
[1] 1+2i 3+0i
[[5]]
[1] "Une chaîne de caractères"
> class(A)
[1] "list"
```

Dans une telle structure hétérogène en types de données, la manière d'ordonner les éléments est très souvent complètement arbitraire. C'est pourquoi il

est possible de les nommer de façon explicite, ce qui rend la sortie plus conviviale. En voici un exemple :

```
> B <- list(une.matrice=matrix(1:4,nrow=2),
+           des.complexes=c(1+2i,3))
> B
$une.matrice
     [,1] [,2]
[1,]    1    3
[2,]    2    4
$des.complexes
[1] 1+2i 3+0i
> liste1 <- list(complexe=1+1i,logique=FALSE)
> liste2 <- list(chaine="J'apprends R",vecteur=1:2)
> C <- list("Ma première liste"=liste1,Ma.seconde.liste=liste2)
> C
$`Ma première liste`
$`Ma première liste`$complexe
[1] 1+1i
$`Ma première liste`$logique
[1] FALSE
$Ma.seconde.liste
$Ma.seconde.liste$chaine
[1] "J'apprends R"
$Ma.seconde.liste$vecteur
[1] 1 2
```

Renvoi

Cette façon de procéder (en nommant les éléments) permet de faciliter l'extraction d'éléments d'une liste (voir chapitre 3).

1.2.2.4 Le tableau individus×variables (`data.frame`)

Le tableau individus×variables est la structure par excellence en statistique. Cette notion est exprimée dans R par le *data.frame*. Conceptuellement, c'est une matrice dont les lignes correspondent aux individus et les colonnes aux variables (ou caractères) mesurées sur ces derniers. **Chaque colonne représente une variable particulière dont tous les éléments sont du même type**. Les colonnes de la matrice-données peuvent être nommées. Voici un exemple de création d'un *data.frame* :

```
> IMC <- data.frame(Sexe=c("H","F","H","F","H","F"),
+     Taille=c(1.83,1.76,1.82,1.60,1.90,1.66),
+     Poids=c(67,58,66,48,75,55),
+     row.names=c("Rémy","Lol","Pierre","Domi","Ben","Cécile"))
> IMC
```

```
       Sexe Taille Poids
Rémy      H   1.83    67
Lol       F   1.76    58
Pierre    H   1.82    66
Domi      F   1.60    48
Ben       H   1.90    75
Cécile    F   1.66    55
> is.data.frame(IMC)
[1] TRUE
> class(IMC)
[1] "data.frame"
```

Expert

Les *data.frame* peuvent être vus comme des listes de vecteurs de même longueur. Cela est d'autant plus vrai que c'est de cette façon que R structure dans son fonctionnement interne un *data.frame*.

```
> is.list(IMC)
[1] TRUE
```

1.2.2.5 Les facteurs (`factor`) et les variables ordinales (`ordered`)

R permet d'organiser les chaînes de caractères de façon plus astucieuse au moyen de la fonction `factor()` :

```
> x <- factor(c("bleu","vert","bleu","rouge",
+                      "bleu","vert","vert"))
> x
[1] bleu  vert  bleu  rouge bleu  vert  vert
Levels: bleu rouge vert
> levels(x)
[1] "bleu"  "rouge" "vert"
> class(x)
[1] "factor"
```

Il est bien évidemment possible de mettre des facteurs dans un *data.frame*. R indique les différents niveaux (*levels*) du facteur. La fonction `factor()` est donc celle à utiliser pour stocker des variables qualitatives. Pour les variables ordinales, il est plutôt conseillé d'utiliser la fonction `ordered()` :

```
> z <- ordered(c("Petit","Grand","Moyen","Grand","Moyen",
+       "Petit","Petit"),levels=c("Petit","Moyen","Grand"))
> class(z)
[1] "ordered" "factor"
```

Le paramètre `levels` de la fonction `ordered()` permet de spécifier l'ordre des modalités de la variable.

Renvoi

Des exemples d'utilisation de ces deux fonctions sont présentés aux chapitres 3 et 13.

Expert

R permet d'organiser un vecteur de chaînes de caractères de façon plus efficace en prenant en compte les éléments qui se répètent. Cette approche permet d'obtenir une meilleure gestion de la mémoire. En effet, chacun des éléments du facteur ou de la variable ordinale est en fait codé sous la forme d'un entier.

Récapitulatif

TAB. 1.2: Les différentes structures de données en R.

Structure des données	Instruction R	Description
vecteur	`c()`	Suite d'éléments de même nature.
matrice	`matrix()`	Tableau à deux dimensions dont les éléments sont de même nature.
tableau multidimensionnel	`array()`	Plus général que la matrice ; tableau à plusieurs dimensions.
liste	`list()`	Suite de structures R de nature différente et quelconque.
tableau individus×variables	`data.frame()`	Tableau à deux dimensions dont les lignes sont des individus et les colonnes des variables (numériques ou facteurs). Les colonnes peuvent être de nature différente, mais doivent avoir la même longueur. Les éléments à l'intérieur d'une même colonne sont tous de la même nature.
facteur	`factor()`, `ordered()`	Vecteur de chaînes de caractères associé à une table des modalités.

Termes à retenir

`<-, ->` : flèches d'affectation dans une variable
`mode(), typeof()` : récupérer la nature d'un objet
`is.numeric()` : déterminer si un objet est de nature numérique
`TRUE, FALSE, is.logical()` : Vrai, Faux, déterminer si un objet est de nature booléenne
`is.character()` : déterminer si un objet est une chaîne de caractères
`NA, is.na()` : Valeur manquante, déterminer s'il existe des valeurs manquantes
`class()` : déterminer la structure d'un objet
`c()` : créer une suite d'éléments de même nature
`matrix(), array()` : créer une matrice, un tableau multidimensionnel
`list()` : créer une liste, collection de structures différentes
`data.frame()` : créer un tableau individus×caractères
`factor()` : créer un facteur

Exercices

1.1- Que renvoie cette instruction : `1:3^2` ?

1.2- Que renvoie cette instruction : `(1:5)*2` ?

1.3- Que renvoient ces instructions : `var<-3` ? `Var*2` ?

1.4- Que renvoient ces instructions : `x<-2` ? `2x<-2*x` ?

1.5- Que renvoient ces instructions : `racine.de.quatre <- sqrt(4)` ? `racine.de.quatre` ?

1.6- Que renvoient ces instructions : `x<-1` ? `x< -1` ?

1.7- Que renvoie cette instruction : `Un chiffre pair <- 16` ?

1.8- Que renvoie cette instruction : `"Un chiffre pair" <- 16` ?

1.9- Que renvoie cette instruction : `"2x" <- 14` ?

1.10- Que renvoie cette instruction : `Un chiffre pair` ?

1.11- Complétez cette sortie de R, où deux symboles ont été enlevés :

```
> 2
+
[1] 6
```

1.12- Que renvoie cette instruction : `TRUE + T +FALSE*F + T*FALSE +F` ?

1.13- Quels sont les cinq types de données sous R ?

1.14- Donnez l'instruction R permettant d'obtenir l'affichage suivant :

```
> X
     [,1] [,2]  [,3]
```

```
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

1.15- Quelles sont les structures (classes) de données que R met à votre disposition ?

Fiche de TP

Étude sur l'indice de masse corporelle

Un échantillon de dossiers d'enfants a été saisi. Ce sont des enfants vus lors d'une visite en 1re section de maternelle en 1996-1997 dans des écoles de Bordeaux (Gironde, France). L'échantillon présenté ici est constitué de dix enfants âgés de 3 ou 4 ans.

Les données disponibles pour chaque enfant sont :
- le sexe : F pour fille et G pour garçons ;
- le fait que leur école soit située en ZEP (zone d'éducation prioritaire, c'est-à-dire réunissant plusieurs indices de précarité sociale) : O pour oui et N pour non ;
- l'âge en années et en mois à la date de la visite (deux variables, une pour le nombre d'années et une pour le nombre de mois) ;
- le poids en kg arrondi à 100 g près ;
- la taille en cm arrondie à 0,5 cm près.

Prénom	Érika	Célia	Éric	Ève	Paul	Jean	Adan	Louis	Jules	Léo
Sexe	F	F	G	F	G	G	G	G	G	G
ZEP	O	O	O	O	N	O	N	O	O	O
Poids	16	14	13.5	15.4	16.5	16	17	14.8	17	16.7
An	3	3	3	4	3	4	3	3	4	3
Mois	5	10	5	0	8	0	11	9	1	3
Taille	100.0	97.0	95.5	101.0	100.0	98.5	103.0	98.0	101.5	100.0

En statistique, il est très important de connaître le type des variables étudiées : qualitatives, ordinales ou quantitatives. R permet de spécifier explicitement ce type au moyen des fonctions de structure que nous avons vues dans ce chapitre.

Voilà quelques manipulations à effectuer avec R. Pensez à bien utiliser la stratégie de travail introduite en début de chapitre.

1.1- Choisissez la fonction R appropriée pour enregistrer les données de chacune des variables précédentes dans des vecteurs que vous nommerez `Individus`, `Poids`, `Taille` et `Sexe`.

1.2- Calculez la moyenne des variables pour lesquelles cela est possible.

1.3- Calculez l'IMC des individus et regroupez les valeurs obtenues dans un vecteur nommé `IMC` (faites attention aux unités).

1.4- Regroupez ces variables dans la structure R qui vous paraît la plus adaptée.

1.5- Utilisez l'aide en ligne de R afin d'obtenir des informations sur la fonction `plot()`.

1.6- Tracez le nuage de points du `Poids` en fonction de la `Taille`. Pensez à fournir un titre à votre graphique et à annoter vos axes.