INITIATION AU LOGICIEL R

Arthur Tenenhaus
arthur.tenenhaus@imed.jussieu.fr

I. Introduction

Lors de la conception de ce document, je me suis fortement inspiré du formidable tutorial de Emmanuel PARADIS «□R pour les débutants□» ; disponible gratuitement à l'adresse suivante□: cran.r-project.org/doc/contrib/rdebuts_fr.pdf. De prime abord, R peut sembler complexe pour une utilisation par un non-spécialiste□Ce n'est pas le cas□!! Ce document devrait vous le prouver.

Le document se décompose 5 parties□

1. La première partie fournit une présentation générale de la structure de R.
2. La seconde partie présente les bases du langage, la gestion des données et de la mémoire ainsi que l'aide en ligne.
3. La troisième partie présente les outils graphiques de R□
   Nous présenterons certaines fonctions graphiques de base ainsi que des fonctions plus complexes de graphes multivariés conditionnées (disponibles dans les packages grid et lattice).
4. La quatrième partie présente des exemples d'analyses statistiques□Nous aborderons la régression linéaire, l'analyse discriminante, l'analyse en composante principale, la régression logistique , le clustering et les support vector machines (SVM)□
5. La cinquième partie fournit les bases de la programmation sous R.

II. Présentation générale du logiciel R

II.1 Quelques informations utiles

R est distribué gratuitement à partir du site du CRAN (Comprehensive R Archive Network)□: http://www.r-project.org/. R est un système d'analyse statistique crée par Ross Ihaka et Robert Gentleman distribué librement sous les termes de la GNU General Public Licence□; son développement et sa distribution sont assurés par plusieurs statisticiens rassemblés dans le R Development Core Team. R propose de nombreuses fonctions pour les analyses statistiques□ Des plus classiques comme l'analyse en composante principale ou la régression au plus modernes comme les Support Vector Machines. R propose également un très grand nombre de fonctions graphiques fournissant un outil très puissant de visualisation et d'exploration des données.
R est un langage interprété et pas compilé c'est-à-dire que les commandes tapées au clavier sont directement exécutées sans qu'il soit nécessaire de construire un programme complet comme cela est souvent le cas pour la plupart des langages informatiques. Il s'ensuit que la syntaxe de R est intuitive.

Les variables, les données, les fonctions, les résultats, etc. sont stockés dans la mémoire vive de l'ordinateur sous forme d'objets qui ont chacun leur nom. Notons que le nom d'un objet doit commencer par une lettre (majuscule ou minuscule) et peut comporter des lettres des chiffres, et des points. L'utilisateur agira alors sur les objets soit par le biais d'opérateurs soit par le biais de fonctions.

Le schéma 1 présente l'interface de R sous Mac□elle ne diffère que très peu de celle de PC.

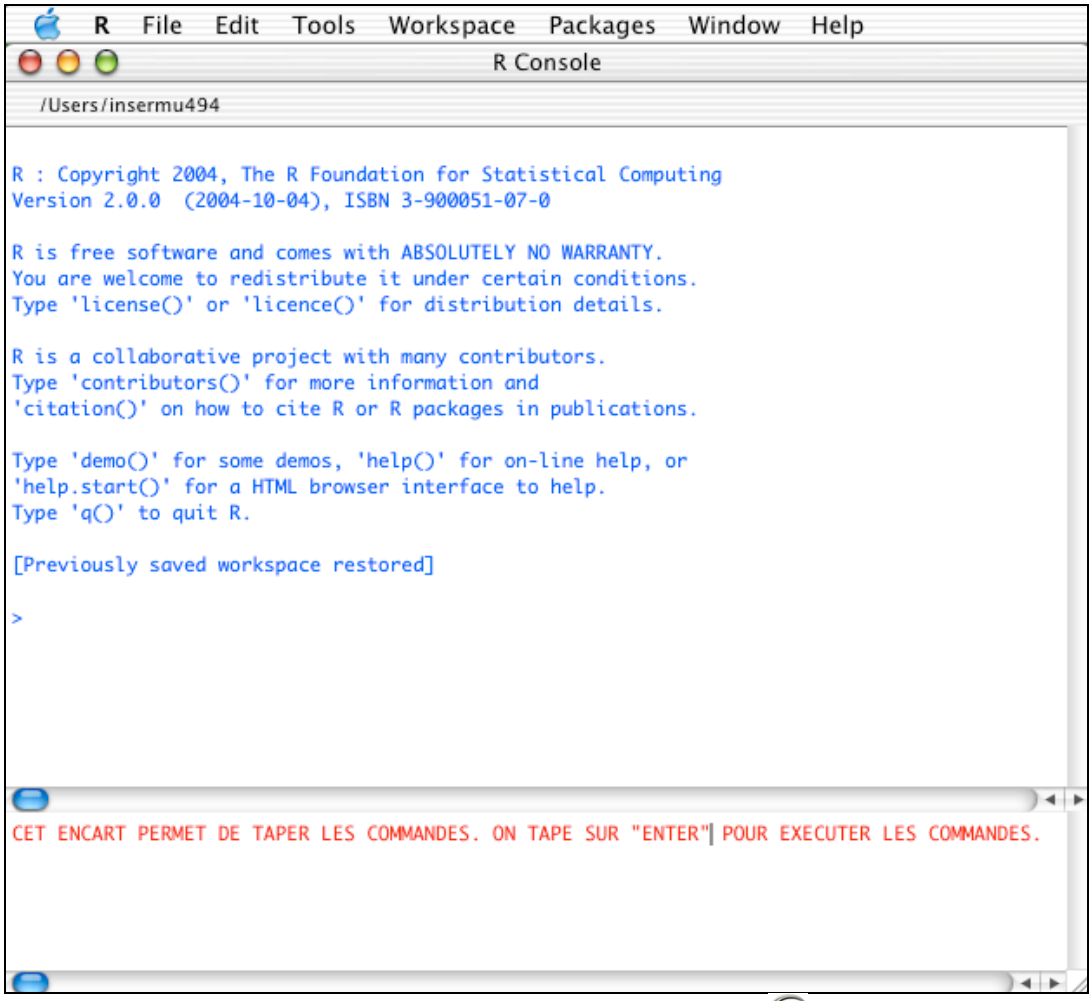


Schéma 1. Capture d'écran de l'interface de R

II.2 Les packages

Toutes les fonctions de R sont stockées dans une bibliothèque. Cette bibliothèque contient des packages de fonctions. Le package nommé «□base□ est le cœur de R et contient les fonctions de base du langage pour la lecture et la manipulation des données, la création de certains types de graphiques et certaines analyses statistiques.
Lors de l'installation de R, un grand nombre de packages sont installés par défaut. On peut accéder à la liste des packages pré-installés grâce à la commande `installed.packages()`. Voici la liste des packages fournit par défaut lors de l'installation (*tab. 1*).

```
installed.packages()
```

	Package	Libpath	Version	Priority	Bundle
KernSmooth	"KernSmooth"	"/Library/Frameworks/R.framework/Resources/library"	"2.22-14"	"recommended"	NA
MASS	"MASS"	"/Library/Frameworks/R.framework/Resources/library"	"7.2-8"	"recommended"	"VR"
Base	"base"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
Boot	"boot"	"/Library/Frameworks/R.framework/Resources/library"	"1.2-19"	"recommended"	NA
Class	"class"	"/Library/Frameworks/R.framework/Resources/library"	"7.2-8"	"recommended"	"VR"
cluster	"cluster"	"/Library/Frameworks/R.framework/Resources/library"	"1.9-6"	"recommended"	NA
datasets	"datasets"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
foreign	"foreign"	"/Library/Frameworks/R.framework/Resources/library"	"0.7"	"recommended"	NA
grDevices	"grDevices"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
graphics	"graphics"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
grid	"grid"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
lattice	"lattice"	"/Library/Frameworks/R.framework/Resources/library"	"0.10-11"	"recommended"	NA

```
methods     "methods"     "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
mgcv        "mgcv"        "/Library/Frameworks/R.framework/Resources/library"  "1.1-5"    "recommended"  NA
nlme        "nlme"        "/Library/Frameworks/R.framework/Resources/library"  "3.1-52"   "recommended"  NA
nnet        "nnet"        "/Library/Frameworks/R.framework/Resources/library"  "7.2-8"    "recommended"  "VR"
rpart       "rpart"       "/Library/Frameworks/R.framework/Resources/library"  "3.1-20"   "recommended"  NA
spatial     "spatial"     "/Library/Frameworks/R.framework/Resources/library"  "7.2-8"    "recommended"  "VR"
splines     "splines"     "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
stats       "stats"       "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
stats4      "stats4"      "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
survival    "survival"    "/Library/Frameworks/R.framework/Resources/library"  "2.13-2"   "recommended"  NA
tcltk       "tcltk"       "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
tools       "tools"       "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
utils       "utils"       "/Library/Frameworks/R.framework/Resources/library"  "2.0.0"    "base"         NA
```

tab. 1. Liste des packages disponibles à l'installation

De nombreux packages, développés par les utilisateurs de R, sont disponibles sur le site internet du CRAN et téléchargeable gratuitement : http://cran.us.r-project.org/src/contrib/PACKAGES.html. Pour installer un nouveau packages à travers le site du CRAN, il suffit d'appeler la fonction `browse.pkgs()` par la fenêtre de commande. Une fenêtre apparaît alors (Cf. figure 1) dans laquelle sont listés tous les packages disponibles. Il suffit de sélectionner le(s) package(s) d'intérêts.

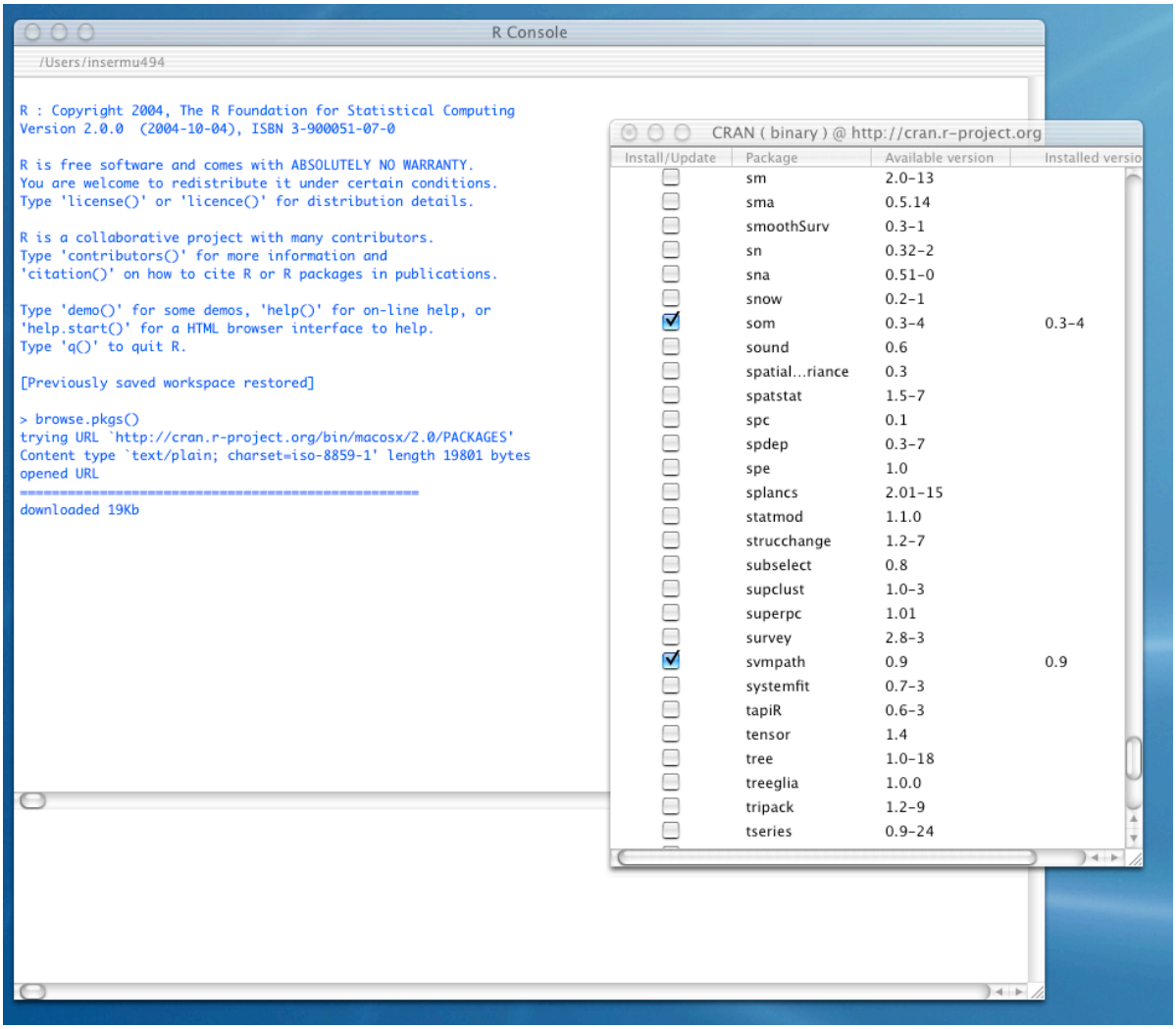


figure 1. Capture d'écran de l'installation de nouveaux packages.
Illustration de l'installation des packages som et svmpath

```
installed.packages()
```

	Package	Libpath	Version	Priority	Bundle
som	"som"	"/Users/insermu494/Library/R/library"	"0.3-4"	NA	NA
svmpath	"svmpath"	"/Users/insermu494/Library/R/library"	"0.9"	NA	NA
KernSmooth	"KernSmooth"	"/Library/Frameworks/R.framework/Resources/library"	"2.22-14"	"recommended"	NA
MASS	"MASS"	"/Library/Frameworks/R.framework/Resources/library"	"7.2-8"	"recommended"	"VR"
Base	"base"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
Boot	"boot"	"/Library/Frameworks/R.framework/Resources/library"	"1.2-19"	"recommended"	NA
Class	"class"	"/Library/Frameworks/R.framework/Resources/library"	"7.2-8"	"recommended"	"VR"
cluster	"cluster"	"/Library/Frameworks/R.framework/Resources/library"	"1.9-6"	"recommended"	NA
datasets	"datasets"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
foreign	"foreign"	"/Library/Frameworks/R.framework/Resources/library"	"0.7"	"recommended"	NA
grDevices	"grDevices"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
graphics	"graphics"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
grid	"grid"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
lattice	"lattice"	"/Library/Frameworks/R.framework/Resources/library"	"0.10-11"	"recommended"	NA
methods	"methods"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
mgcv	"mgcv"	"/Library/Frameworks/R.framework/Resources/library"	"1.1-5"	"recommended"	NA
nlme	"nlme"	"/Library/Frameworks/R.framework/Resources/library"	"3.1-52"	"recommended"	NA
nnet	"nnet"	"/Library/Frameworks/R.framework/Resources/library"	"7.2-8"	"recommended"	"VR"
rpart	"rpart"	"/Library/Frameworks/R.framework/Resources/library"	"3.1-20"	"recommended"	NA
spatial	"spatial"	"/Library/Frameworks/R.framework/Resources/library"	"7.2-8"	"recommended"	"VR"
splines	"splines"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
stats	"stats"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
stats4	"stats4"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
survival	"survival"	"/Library/Frameworks/R.framework/Resources/library"	"2.13-2"	"recommended"	NA
tcltk	"tcltk"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
tools	"tools"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA
utils	"utils"	"/Library/Frameworks/R.framework/Resources/library"	"2.0.0"	"base"	NA

Tab 2. Liste des packages installés

Les packages installés par défaut sur l'ordinateur ne sont pas forcément utilisable directement. Une étape préliminaire de chargement est parfois obligatoire. Pour connaître les packages chargés par défaut, on regarde la colonne Priority du résultat de la commande `installed.packages()`. Les packages associés à une priority = «□ase□ sont directement utilisable. D'autres packages (priority = «□ecommended□ ou priority = «□NA□) doivent être chargé en mémoire par l'intermediaire de la fonction `library()`.
Par exemple, on souhaite utiliser les fonctions du package MASS. La priority étant recommended, on doit charger le package via la commande `library(MASS)`.
Notons que tous les packages téléchargés du CRAN sont «□ibrary-dépendant□□ leur priority étant NA. Par exemple, on souhaite utiliser la fonction `som()` pour constuire des cartes de Kohonen. La fonction `som()` est dans le package som□ il est indispensable de chargées le packages som□

```
library(som);
Warning message:
package 'som' was built under R version 2.0.1
```

III. Les bases du langage, l'aide en ligne et la gestion des données et de la mémoire

III.1 Création d'objets basiques

On affecte directement une valeur à un objet. L'objet n'a pas besoin d'être déclaré.

```
n = 8
n
[1] 8
```

Le symbole [1] indique que l'affichage commence au premier élément de n. On aurait également pu affecter une valeur à l'objet *n* de la manière suivante□

```
n <- 8
n
[1] 8
```

Dans la suite du document, on utilisera le symbole «□□, plus intuitif.

```
x = 1
x
[1] 1

x = 10
x
[1] 10

y = 10 + 2
y
[1] 12

z = 3 + rnorm(1)
z
[1] 2.768172

w = 8; name = "Arthur"; dicton = "Vive le logiciel R";
w ; name ; dicton
[1] 8
[1] "Arthur"
[1] "Vive le logiciel R"
```

Quelques remarques□

1. Notons l'usage du□□;□□ pour séparer des commandes distinctes sur la même ligne.
2. Si on affecte une valeur à un objet existant, sa valeur précédente est effacée.
3. La fonction `rnorm(1)` génère un nombre aléatoire normale de moyenne zéro et de variance 1.

III.2 Gestion de l'espace mémoire

III.2.1 Lister les objets en mémoire : `ls()`

```
x = 10□; X = 5
w = 8
name = "Arthur"
dicton = "Vive le logiciel R"
z = 3+rnorm(1)
y = 10 + 2
```

La fonction `ls()` permet d'afficher la liste des objets en mémoire (seuls le nom des objets est affiché)□

```
ls()
[1] "X"       "dicton"     "name" "w"    "x"    "y"    "z"
```

Notons que R différentie les majuscules des minuscules. Si on souhaite lister uniquement les objets contenant un caractère donné dans leur nom, on utilisera l'option pat (abréviation de pattern)□

```
ls(pat = "n")
[1] "dicton" "name"
```
Pour restreindre la liste aux objets qui commence par un caractère donné on utilisera l'option «□□ :

```
ls(pat = "^n")
[1] "name"
```

La fonction `ls.str()` affiche une liste détaillée des objets en mémoire

```
> ls.str()
X□: num 10
dicton :  chr "Vive le logiciel R"
name :  chr "Arthur"
w :  num 8
x :  num 10
y :  num 12
z :  num 3.77
```

III.2.2 Effacer les objets de la mémoire : rm()

La fonction `rm()` permet d'effacer des objets de la mémoire□ces options sont proches de celles de `ls()`

```
ls()
[1] "dicton" "name" "w"    "x"    "y"    "z"
```

Pour effacer l'objet nommé *x* de la mémoire, on tape la commande suivante□

```
rm(x);ls()
[1] "dicton" "name" "w"    "y"    "z"
```

Pour effacer tous les objets dont le nom commence par «□□□

```
> rm(list = ls(pat = "^n")); ls()
[1] "dicton" "w"    "z"    "y"
```

Pour effacer tous les objets de la mémoire□

```
> rm(list = ls()); ls()
character(0)
```

III.3 L'aide en ligne : ? ou help

L'aide en ligne est extrêmement utile dans l'utilisation courante de R. Nous allons illustrer l'utilisation de l'aide de R par un exemple.
On souhaite faire une régression, deux cas de figure se présentent.

1. On ne connaît pas le nom de la fonction et, a fortiori, on ne sait pas l'utiliser

```
help.search("regression")
```

Cette commande fournit une liste de fonctions reliées d'une manière ou d'une autre au mot «.egression». Voici une copie d'écran du résultat obtenu.

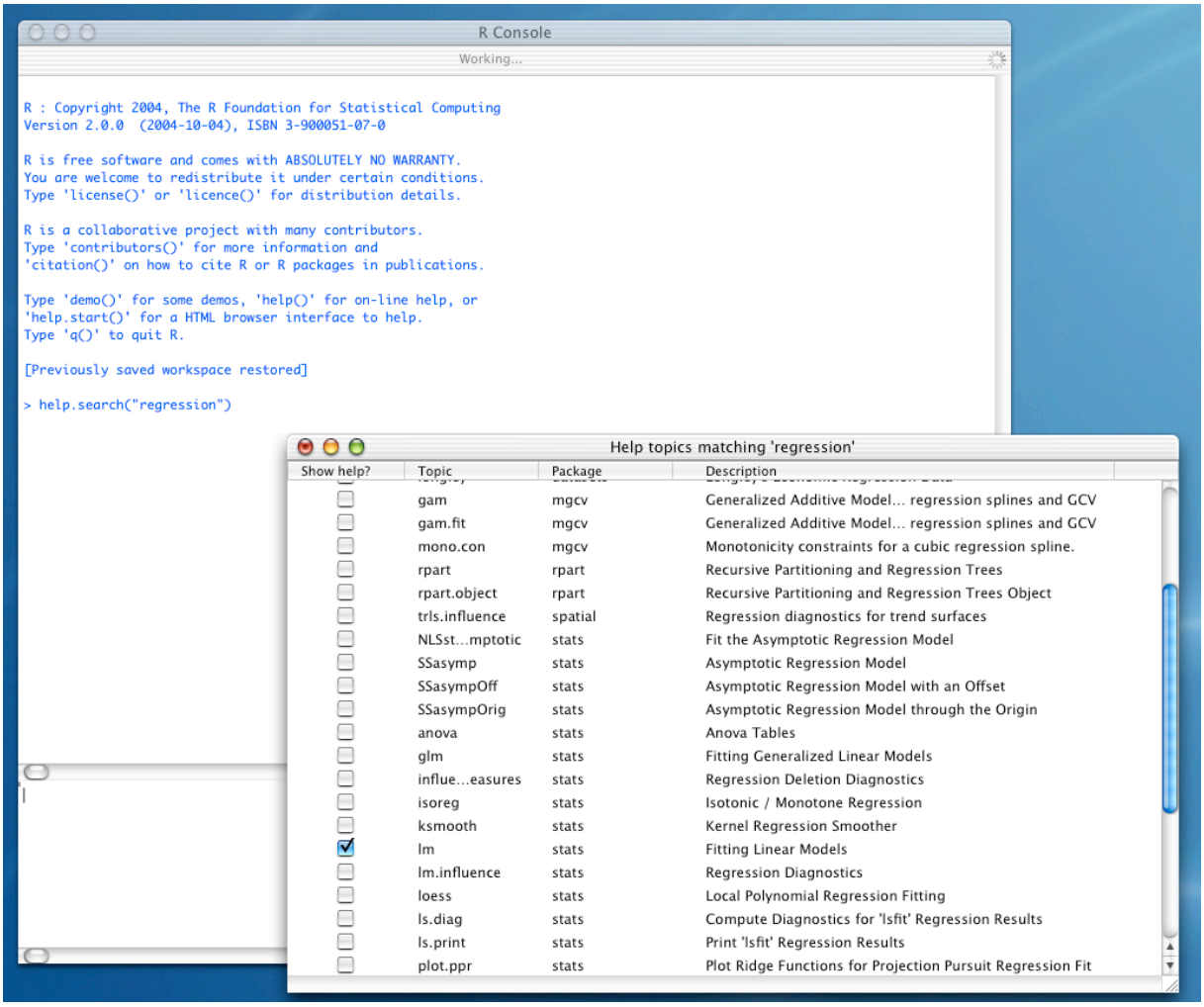


Figure 2. Utilisation de l'aide en ligne

On découvre alors grâce au bref descriptif associé à chaque réponse que la fonction `lm()` permet de construire un modèle linéaire. «.itting Linear Models», en cliquant sur la fonction associé. On obtient un descriptif détaillé comme l'illustre la capture d'écran suivante.

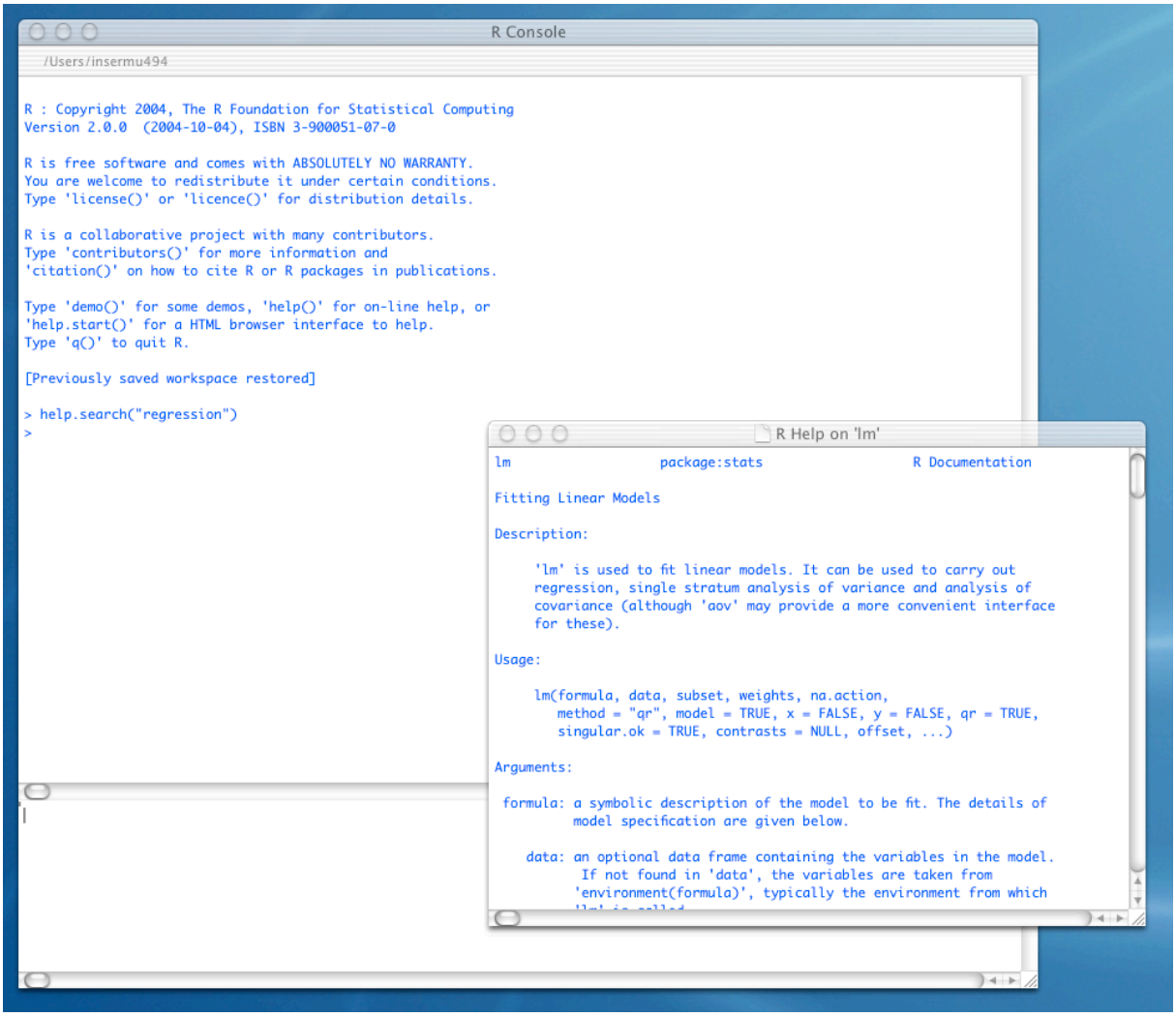


Figure 3. Utilisation de l'aide en ligne

Une manière plus directe d'accéder à la fiche détaillée nécessite la connaissance du nom de la fonction. C'est le deuxième cas de figure.

2. on connaît le nom de la fonction mais on ne sait pas comment l'utiliser

   `?lm` ou `help(lm)`

Apparaît alors la fiche détaillé contenant les informations suivantes :

a. **Nom du package** d'où provient la fonction
b. **Descritpion**⊡brève description
c **Usage**⊡
    1. Pour une fonction⊡donne le nom de la fonction avec tous ses arguments et éventuelles valeurs par défaut
    2. Pour un opérateur⊡donne l'usage typique
b. **Arguments**⊡Pour une fonction détail chacun des arguments.
c. **Details**⊡Descritpion détaillée
d. **Value**⊡Le type de résultat retourné par la fonction ou l'opérateur
e. **See also**⊡Autres rubriques d'aide proche ou similaires à celle documentée
f. **Examples**⊡des exemples d'utilisation de la fonction

Notons qu'il est très utile de regarder la partie «□Examples□» de l'aide en ligne. Elle illustre de manière concrète l'utilisation de la fonction.
D'autres paragraphes peuvent être disponible dans l'aide tels que **Note**, **References** ou **Author(s)**

Remarque 1□On peut ouvrir l'aide au format html grâce à la fonction `help.start()`

Remarque 2□La fonction `apropos()` trouve toutes les fonctions qui contiennent dans leur nom la chaîne de caractère passée en argument. **ATTENTION**□Seules les packages chargés en mémoire sont scannées.

```
> apropos(mean)
[1] "kmeans"         "weighted.mean"     "mean"              "mean.Date"
[5] "mean.POSIXct"   "mean.POSIXlt"      "mean.data.frame"   "mean.default"
[9] "mean.difftime"
```

REMARQUE IMPORTANTE□Si on utilise la fonction `help.search()`, R recherche la fonction d'intérêt sur la liste de tous les packages disponibles sur l'ordinateur. Si on utilise□la fonction ? ou `help()`, R ne recherche les fonctions que dans les packages chargés par défaut ou via `library()`.

III.4 Les données avec R

Nous avons vu que les variables, les données, les fonctions, les résultats d'analyses sont stockés dans des objets. Nous allons présentés la structure de ces objets.
Les objets dans R ont tous deux attributs□Le mode et la longueur. Le mode est le type des éléments d'un objet□il en existe 4 principaux□numérique, caractère, complexe, logique. La longueur est le nombre d'élément de l'objet. Pour connaître le mode est la longueur d'un objet, on utilise respectivement les fonctions `mode()` et `length()`. Le tableau suivant donne un aperçu des objets dans lesquels les données peuvent êtres stockés.
Ajoutons que les valeurs manquantes sont représentées par *NA* (**N**ot **A**valaible) quelquesoit l'objet ou le mode.

Objet	Modes	Plusieurs modes possibles dans le même objet ?
vecteur	num, car, comp, log	Non
facteur	num, car,	Non
array	num, car, comp, log	Non
matrice	num, car, comp, log	Non
data.frame	num, car, comp, log	Oui
Ts	num, car, comp, log	Oui
liste	num, car, comp, log, fonction, expression	Oui

Tab 3. Liste des objets dans lesquels peuvent être stockés des éléments.
num = Numérique, car = Caractère, comp = complexe, log = logique

III.4.1 Générer des données et les stockés dans les différents objets

III.4.1.1 Les Vecteurs⊡`vector()`

La fonction `vector()` a deux arguments le mode des éléments qui compose le vecteur et la longueur du vecteur.

```
a = vector("numeric", 5)
b = vector("character", 5)
c = vector("logical", 5)
d = vector("complex", 5)
```

Équivalent à

```
a = numeric(5)
b = character(5)
c = logical(5)
d = complex(5)
```

```
a;b;c;d

[1] 0 0 0 0 0
[1] "" "" "" "" ""
[1] FALSE FALSE FALSE FALSE FALSE
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
```

On peut également construire un vecteur à l'aide de la fonction `c()`

```
vecteur = c(5, 7.2, 3.6, 4.9); vecteur
[1] 5.0 7.2 3.6 4.9
```

III.4.1.2. Les Facteurs⊡Factor()

La fonction `factor()` crée des variables qualitatives nominales.

```
factor(1:3)
[1] 1 2 3
Levels: 1 2 3

factor(1:3, levels = 1:5)
[1] 1 2 3
Levels: 1 2 3 4 5

factor(1:3, levels = 1:5, labels = c("A", "B", "C", "D", "E"))
[1] A B C
Levels: A B C D E
```

L'option levels spécifie le nombre de niveau possible pour la variable qualitative (par défaut les valeurs uniques du vecteur). L'option labels définit le nom des niveaux.

III.4.1.3. Les Matrices⊡`matrix()`

Une matrice est un vecteur qui possède un argument supplémentaire qui définit les dimensions de la matrice

```
matrix(0, 5, 7)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    0    0    0    0    0    0
[2,]    0    0    0    0    0    0    0
[3,]    0    0    0    0    0    0    0
```

```
[4,]    0    0    0    0    0    0    0
[5,]    0    0    0    0    0    0    0

x = 1:20
> x
 [1]   1    2    3    4    5    6    7    8    9   10   11   12
[13]  13   14   15   16   17   18   19   20

mat1 = matrix(x, 4, 5)
> mat1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20

mat2 = matrix(x, 4, 5, byrow = TRUE)
> mat2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
```

Les options de la fonction `matrix()` sont utiles. Par exemple, on peut donner des noms aux colonnes ou aux lignes de la matrice grâce aux fonctions `rownames()`, `colnames()` ou `dimnames()`.
La fonction `paste()` est utile dans une situation où on souhaite nommé les lignes et / ou les colonnes d'une matrice. En effet, la fonction `paste()` permet de concaténer des objets.

```
nom_var = paste("V", 1:5, sep = "")
> nom_var
[1] "V1" "V2" "V3" "V4" "V5"

nom_ind = paste("I", 1:4, sep = "")
> nom_ind
[1] "I1" "I2" "I3" "I4"
```

Dans cette situation est utilisé, implicitement, la notion de recyclage. On recycle le plus petit des objets de manière à ce qu'il soit de la même longueur que le plus grand. La concaténation devient alors immédiate.

```
colnames(mat2) = nom_var
rownames(mat2) = nom_ind
```
Équivalent à `dimnames(mat2) = list(nom_ind, nom_var)`

```
mat2
> mat2
    V1 V2 V3 V4 V5
I1   1  2  3  4  5
I2   6  7  8  9 10
I3  11 12 13 14 15
I4  16 17 18 19 20
```

Remarque⊡Tous les éléments d'une matrice doivent être de même mode.

III.4.1.3.1 Opérations sur les matrices et les vecteurs

R offre des facilités pour le calcul et la manipulation de matrices. Le paragraphe suivant illustre des situations souvent rencontrées.

```
mat1 = matrix(1 : 4, nrow = 2,  ncol = 2)
mat1
     [,1] [,2]
[1,]    1    3
[2,]    2    4

mat2= matrix(5 : 8, nr = 2, nc = 2 )
mat2
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

La fonction `rbind()` juxtapose des matrices en conservant les lignes

```
rbind(mat1, mat2)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    7
[4,]    6    8
```

La fonction `cbind()` juxtapose des matrices en conservant les colonnes

```
cbind(mat1, mat2)
      [,1] [, 2] [, 3] [, 4]
[1, ]    1     3     5     7
[1, ]    2     4     6     8
```

L'opérateur pour le produit terme à terme de matrices est «□□ tandis que l'opérateur pour le produit de deux matrices est «□%*%□. Les opérateurs pour l'addition et la soustraction de matrices terme à terme sont respectivement «□+□ et «□ ».

```
rbind(mat1, mat2) %*% cbind(mat1, mat2)
      [, 1] [, 2] [, 3] [, 4]
[1, ]     7    15    23    31
[2, ]    10    22    34    46
[3, ]    19    43    67    91
[4, ]    22    50    78   106

cbind(mat1, mat2)  %*% rbind(mat1, mat2)

      [, 1] [, 2]
[1, ]    74   106
[2, ]    88   128
```

La transposition d'une matrice se fait avec la fonction `t()`.

```
t(rbind(mat1, mat2) %*% cbind(mat1, mat2))
      [, 1] [, 2] [, 3] [, 4]
[1, ]     7    10    19    22
[2, ]    15    22    43    50
[3, ]    23    34    67    78
[4, ]    31    46    91   106
```

La fonction `diag()` sert à extraire et - ou modifier la diagonale d'une matrice. Elle permet également de construire des matrices diagonales.

```
diag(mat1)
[1] 1 4

diag(rbind(mat1, mat2) %*% cbind(mat1, mat2))
[1]   7  22  67 106

diag(4)
      [, 1] [, 2] [, 3] [, 4]
[1, ]     1     0     0     0
[2, ]     0     1     0     0
[3, ]     0     0     1     0
[4, ]     0     0     0     1

v = c(1, 2, 3, 4)
diag(v)
      [, 1] [, 2] [, 3] [, 4]
[1, ]     1     0     0     0
[2, ]     0     2     0     0
[3, ]     0     0     3     0
[4, ]     0     0     0     4
```

R propose également des fonctions spéciales pour le calcul matriciel. Citons `solve()` pour l'inversion de matrice, `qr()` pour la décomposition, `eigen()` pour le calcul des valeurs propres et `svd()` pour la décomposition en valeurs singulières.

```
a = solve(diag(v))□; a
      [, 1] [, 2]   [, 3] [, 4]
[1, ]     1     0       0     0
[2, ]     0   0.5       0     0
[3, ]     0     0 0.33333     0
[4, ]     0     0       0  0.25

eigen(diag(v))
$values
[1] 4 3 2 1

$vectors

      [, 1] [, 2] [, 3] [, 4]
[1, ]     0     0     0     1
[2, ]     0     0     1     0
[3, ]     0     1     0     0
[4, ]     1     0     0     0
```

III.4.1.4. Les data.frame□`data.frame()`

```
a = c(1, 2, 3)□; a□; mode(a);
[1] 1 2 3
[1] "numeric"
b = c("a", "b", "c")□; b□; mode(b)
[1] "a" "b" "c"
[1] "character"

df = data.frame(a, b)
> df
  a b
1 1 a
2 2 b
3 3 c
```

Une autre manière de construire une data.frame est présentée ci-dessous□

```
df2 = data.frame(a = 1:6, b = letters[1:6]);
> df2
  a b
1 1 a
2 2 b
3 3 c
4 4 d
5 5 e
6 6 f
```

Le premier élément de la data.frame contient un vecteur numérique tandis que le second contient un vecteur de mode character. Les éléments d'une data.frame ne doivent donc pas nécessairement avoir le même mode. En revanche, tous les éléments de la data.frame doivent être de la même longueur. Dans le cas contraire, l'élément le plus court est «□recyclé□ un nombre entier de fois. L'exemple ci-dessous illustre cette contrainte.

```
a = c(1, 2, 3, 4, 5, 6)
b = c("a", "b", "c")
df = data.frame(a, b)
>df
  a b
1 1 a
2 2 b
3 3 c
4 4 a
5 5 b
6 6 c

a = c(1, 2, 3, 4, 5)
b = c("a", "b", "c")
df = data.frame(a, b)
Error in data.frame(a, b) : arguments imply differing number of rows: 5, 3
```

Ajoutons que les éléments de mode caractères sont considérés comme des facteurs.

III.4.1.5. Les listes : `list()`

Une liste se crée de la même manière qu'une data.frame. Tout comme les data.frame, les éléments qui compose la list ne sont pas nécessairement du même mode. En revanche, les éléments d'une list ne sont pas nécessairement de la même longueur : Pas de recyclage.

```
a = c(1, 2, 3, 4, 5)
b = c("a", "b", "c")
liste1 = list(a, b)
> liste1
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "a" "b" "c"
```

On peut nommer les éléments d'une liste de la manière suivante :

```
names(liste1) = c("L1", "L2")  ;liste1

$L1
[1] 1 2 3 4 5

$L2
[1] "a" "b" "c"

liste2 = list(L1 = a, L2 = b)
liste2
$L1
[1] 1 2 3 4 5

$L2
[1] "a" "b" "c"
```

III.4.2 Conversion de mode

Dans de nombreuses situations pratiques, il est utile de convertir le mode d'un objet en un autre. Une telle conversion sera possible grâce à une fonction de la forme : as.mode (as.numeric, as.logical, as.character, …).

Conversion en	fonction	Règles
numérique	as.numeric	FALSE 0 TRUE 1 "1", "2", ... 1, 2, ... "A", ... NA
logique	as.logical	0 FALSE autres nombres TRUE "FALSE" FALSE "TRUE" TRUE autres caractères NA
caractère	as.character	1, 2, ... "1", "2", ... FALSE "FALSE" TRUE "TRUE"

Tab 4. Conversion de mode

Considérons quelques exemples simples :

Premier cas⊡On souhaite convertir des objets de mode logical en mode numeric

```
logique = c(FALSE, FALSE, TRUE, TRUE, FALSE, TRUE)
conversion_numerique =  as.numeric(logique)
conversion_numerique
[1] 0 0 1 1 0 1
```

deuxième cas⊡On souhaite convertir des objets de mode character en mode numeric

```
caractere = c("1", "2", "3", "A", "/", "T", "%", "-")
conversion_numerique =  as.numeric(caractere)
Warning message:
NAs introduced by coercion
conversion_numerique
[1]  1 2 3 NA NA NA NA NA
```

Troisième cas⊡On souhaite convertir des objets de mode numeric en mode logical

```
numerique = 0:5
conversion_logique1 = as.logical(numerique)
conversion_logique1
[1] FALSE TRUE TRUE TRUE TRUE TRUE
```

Quatrième cas⊡On souhaite convertir des objets de mode character en mode logical

```
caractere = c("FALSE", "TRUE", "F", "T", "false", "t", "A", "(")
conversion_logique2 = as.logical(caractere)
conversion_logique2
[1] FALSE TRUE FALSE TRUE FALSE NA NA NA
```

Cinquième cas⊡On souhaite convertir des objets de mode numeric en mode character

```
numerique = 1:8
conversion_caractere1 = as.character(numerique)
conversion_caractere1
[1] "1" "2" "3" "4" "5" "6" "7" "8"
```

Sixième cas⊡On souhaite convertir des objets de mode logical en mode character

```
logique = c(TRUE, FALSE)
conversion_caractere2 = as.character(logique)
conversion_caractere2
[1] "TRUE"  "FALSE"
```

III.4.3 Conversion d'objet

Dans de nombreuses situations pratiques, il est utile de convertir un objet en un autre. Une telle conversion sera possible grâce à une fonction de la forme⊡ as.objet (as.matrix, as.data.frame, as.list, as.factor, …).

On souhaite convertir une matrice en une data.frame.

```
> a = matrix(1:25, nrow = 5, ncol = 5);a;
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

b = as.data.frame(a);b;

  V1 V2 V3 V4 V5
1  1  6 11 16 21
2  2  7 12 17 22
3  3  8 13 18 23
4  4  9 14 19 24
5  5 10 15 20 25
```

Conversion d'un facteur en numerique

```
facteur = factor(c(1, 5, 10));facteur
[1] 1 5 10
Levels: 1 5 10

facteur_numerique1 = as.numeric(facteur);facteur_numerique1;
[1] 1 2 3
```

Pour conserver un facteur (en numérique) en conservant les niveaux tels qu'ils sont spécifiés, on convertira d'abord en caractère puis en numérique.

```
facteur_caractere = as.character(facteur);facteur_caractere
[1] "1" "5" "10"

facteur_numerique = as.numeric(facteur_caractere);facteur_numerique
[1] 1 5 10
```

III.4.4 Accéder aux valeurs d'un objet

III.4.4.1 Accéder aux valeurs d'un objet par le système d'indexation

L'indexation est un moyen efficace et flexible d'accéder de façon sélective aux éléments d'un objet.

III.4.4.1.1 Les vecteurs et les matrices

```
x = 1:20 ; x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Pour accéder au cinquième élément du vecteur x, il suffit de taper la commande suivante

```
x[5]
[1] 5

x = c(1.1, 5.3, 9, 4.2, 3.6, 7.2, 8.4, 1.6, 8.8, 3.5);x
[1] 1.1 5.3 9.0 4.2 3.6 7.2 8.4 1.6 8.8 3.5
x < 5
[1]  TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE FALSE TRUE
```

```
y = x[x < 5];y
[1] 1.1 4.2 3.6 1.6 3.5
z = x[c(2, 6)];z
[1] 5.3 7.2
```

Pour accéder aux éléments supérieurs à 10 du vecteur x, il suffit de taper la commande suivante

```
x = 1:20
x[x > 10]
[1] 11 12 13 14 15 16 17 18 19 20
```

Pour remplacer les éléments de x supérieurs à 10 par la valeur 20, il suffit de taper la comande suivante

```
x[x > 10] = 20 ; x
[1] 1 2 3 4 5 6 7 8 9 10 20 20 20 20 20 20 20 20 20 20
```

Pour remplacer les éléments égales à 20 par la valeur 0, il suffit de taper la commande suivante

```
x[x==20] = 0; x
[1] 1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0
```

Notons l'usage du «=» qui n'est pas une affectation mais un test d'égalité.

```
a = matrix(1:25, 5, 5);
a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

À l'instar des vecteurs, on peut accéder aux valeurs des matrices par indexation.

```
a[3, 3] = 2; a
a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8    2   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

On peut accéder aux valeurs de la $j^{ième}$ colonne de la matrice X par la commande suivante $X[\ , j]$. De la même manière, on accède aux valeurs de la $i^{ième}$ ligne de la matice X par la commande suivante $X[i\ , \ ]$

On souhaite accéder aux valeurs de la troisième colonne de la matrice a

```
a[ , 3]
[1] 11 12 2 14 15
```

On peut également affecter des valeurs à des lignes ou colonnes de matrice.

```
a[ , 3] = 3; a
a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6    3   16   21
[2,]    2    7    3   17   22
[3,]    3    8    3   18   23
[4,]    4    9    3   19   24
[5,]    5   10    3   20   25

[3, ] = 3; a
a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6    3   16   21
[2,]    2    7    3   17   22
[3,]    3    3    3    3    3
[4,]    4    9    3   19   24
[5,]    5   10    3   20   25

a[3, ] = 11:15 ; a
a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    3   13    3    3
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

Le système d'indexation sert aussi à supprimer des lignes ou colonnes⊡On peut supprimer la $j^{ième}$ colonne de la matrice X par la commande suivante $X[\ ,\ -j]$. De la même manière, on peut supprimer la $i^{ième}$ ligne de la matice X par la commande suivante $X[-i\ ,\ ]$

On souhaite supprimer la troisième ligne de la matrice *a*, il suffit de taper la commande suivante

```
a[-3, ];
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    4    9   14   19   24
[4,]    5   10   15   20   25
```

Une utilisation pratique de cette indexation est, par exemple, la possibilité de sélectionner les éléments pairs d'une variable entière

```
x = 1:10
x%%2
[1] 1 0 1 0 1 0 1 0 1 0
x%%2 == 0
[1] FALSE TRUE FALSE  TRUE FALSE TRUE FALSE TRUE FALSE TRUE
xpair = x[x%%2 == 0];xpair
[1] 2 4 6 8 10
```

Ce système d'indexation utilise donc des valeurs logiques retournées par des opérateurs de comparaison. Ces valeurs logiques peuvent être calculées au préalable, elles seront éventuellement recyclées.

```
temp = c(FALSE, TRUE)
x[temp]
[1] 2 4 6 8 10
```

III.4.4.1.2 Les data.frames

L'indexation peut-être également utilisée avec les data.frames, mais avec la difficulté que les différentes colonnes du data.frame peuvent-être de mode différents.

```
a = c(1, 2, 3)
b = c("a", "b", "c")
df = data.frame(a, b)
```

On souhaite accéder au troisième élement de la deuxième colonne de df

```
df[2][3, 1]
[1] c
Levels: a b c
```

Remarquons une nouvelle fois que, par défaut, le mode character dans une data.frame est considéré comme un mode factor.

On souhaite assigner la valeur 1 à la deuxième colonne de df

```
df[2][, 1] = 1
df
  a b
1 1 1
2 2 1
3 3 1
```

Considérons la data.frame suivante

```
df2 = data.frame(a = 1:6, b = matrix(1:24, 6, 4))□; df2
  a b.1 b.2 b.3 b.4
1 1   1   7  13  19
2 2   2   8  14  20
3 3   9  15  21   4
4 4   4  10  16  22
5 5   5  11  17  23
6 6   6  12  18  24
```

En tapant la commande `df2[2]` on n'obtient non pas la matrice *b* mais la deuxième colonne *b.1* de la data.frame

```
df2[2]
  b.1
1   1
2   2
3   3
```

```
4   4
5   5
6   6
```

Pour obtenir la matrice *b* il suffit de taper la commande suivante.

```
df2[2:5]
  b.1 b.2 b.3 b.4
1   1   7  13  19
2   2   8  14  20
3   9  15  21   4
4   4  10  16  22
5   5  11  17  23
6   6  12  18  24
```

Si on souhaite accéder à la troisième ligne de la matrice *b*, il suffit de taper la commande suivante

```
df2[2:5][3, ]
  b.1 b.2 b.3 b.4
3   9  15  21   4
```

On peut par ailleurs affecter des valeurs à la data.frame de la manière suivante

```
df2[2][3, 1] = 999;df2
  a b.1 b.2 b.3 b.4
1 1   1   7  13  19
2 2   2   8  14  20
3 3 999  15  21   4
4 4   4  10  16  22
5 5   5  11  17  23
6 6   6  12  18  24

df2[2:5][3, ] = 999;df2
  a b.1 b.2 b.3 b.4
1 1   1   7  13  19
2 2   2   8  14  20
3 3 999 999 999 999
4 4   4  10  16  22
5 5   5  11  17  23
6 6   6  12  18  24
```

III.4.4.1.3 Les listes

Pour les listes, l'accès aux différents éléments de la liste se fait avec des crochets doubles. Par exemple, pour accéder aux troisième éléments de la liste nommé *L*, il suffira de taper la commande *L[[3]]*. Le résultat pourra ensuite être indexé de la même façon que pour les vecteurs, les matrices. Considérons l'exemple illustratif suivant

```
liste = list(a = 1:6, b = matrix(1:24, 6, 4));liste
liste[[1]]
[1] 1 2 3 4 5 6

liste[[2]]
     [,1] [,2] [,3] [,4]
```

```
[1,]    1    7   13   19
[2,]    2    8   14   20
[3,]    3    9   15   21
[4,]    4   10   16   22
[5,]    5   11   17   23
[5,]    6   12   18   24

liste[[1]][c(TRUE, FALSE)]
[1] 1 3 5

liste[[2]][ ,3]
[1] 13 14 15 16 17 18
```

III.4.4.2 Accéder aux valeurs d'un objet par le nom

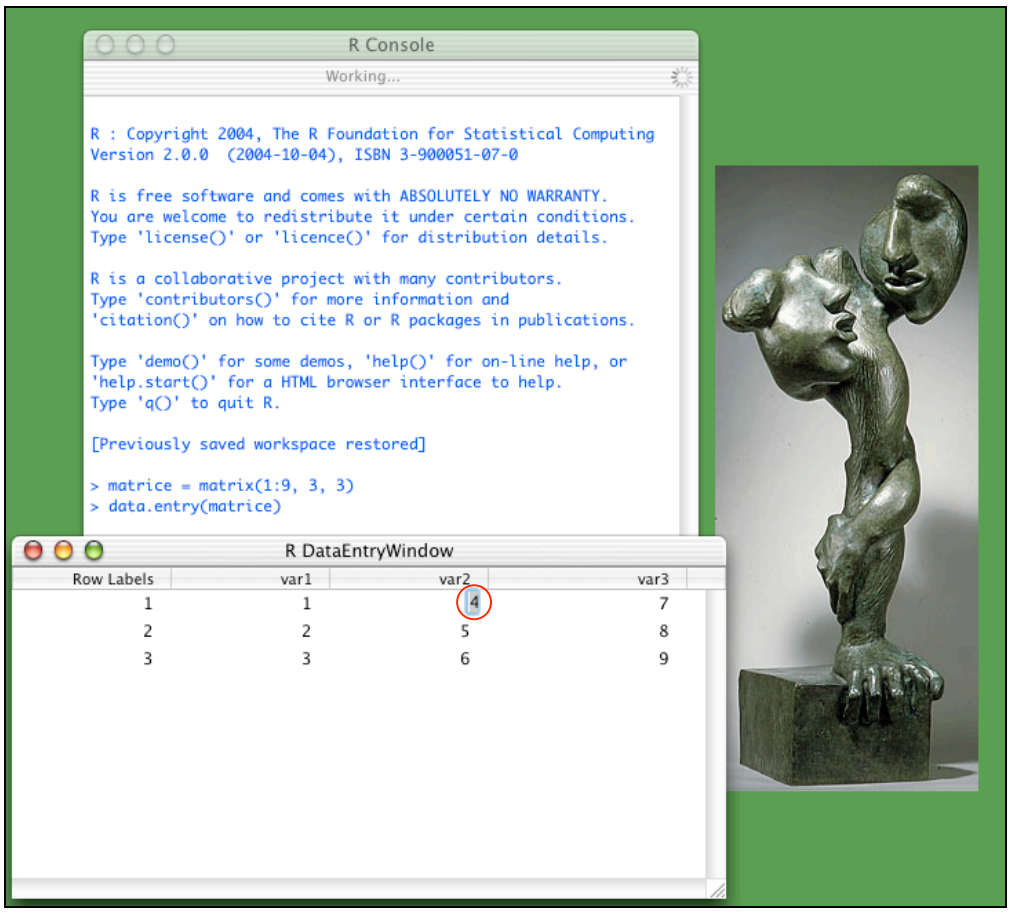
On ne peut accéder aux valeurs d'éléments par le nom que dans le cadre de data.frame ou de liste. L'appel se fait par le symbole $.

```
liste = list(a = 1:6, b = matrix(1:24, 6, 4));liste

liste$a
[1] 1 2 3 4 5 6

liste$b
     [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    2    8   14   20
[3,]    3    9   15   21
[4,]    4   10   16   22
[5,]    5   11   17   23
[5,]    6   12   18   24

liste$b[1, ]
[1] 1 7 13 19

data_frame = data.frame(a = 1:6, b = letters[1:3])

data_frame$a
[1] 1 2 3 4 5 6

data_frame$b
[1] a b c a b c
Levels: a b c
```

III.4.4.3 Accéder aux valeurs par l'éditeur de donnée

Il est possible d'utiliser un éditeur graphique de style tableur pour éditer un objet contenant des données grâce à la fonction `data.entry()`.

```
matrice = matrix(1:9, 3, 3)
data.entry(matrice)
```



Si on tape la commande `matrice` dans la fenêtre de commande, on obtient le résultat suivant

```
matrice
```

```
     var1  Var2  var3
[1,]    1     0     7
[2,]    2     5     8
[3,]    3     6     9
```

III.4.5 Lire les données d'un fichier

R peut lire les données stockées dans des fichiers texte (ASCII) grâce, entre autres, aux fonctions suivantes : `read.table()`, `scan()`. R lit également les fichiers aux formats Excel, SAS, SPSS, mais les fonctions qui le permettent ne sont pas dans le package base.
Dans ce document, on se limitera à la lecture des fichiers au format ASCII. Pour plus d'informations concernant la lecture des autres formats, je vous renvoie au document proposé par le « R development core team » disponible gratuitement sur internet à l'adresse suivante : http://www.r-project.org.

La fonction read.table crée une data.frame où chaque variables sera nommé, par défaut, V1, V2, …. On peut accéder aux valeurs de ces variables soit par le nom ($) soit par l'indexation. Du fait de la grande variabilité des jeux de données, la fonction `read.table()` comporte de nombreuses options dont voici les valeurs par défaut (c'est à dire celles utilisées par R si elles ne sont stipulées par l'utilisateur) ainsi que les détails dans le tableau 5:

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".",
           row.names, col.names, as.is = FALSE, na.strings = "NA",
           colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
           fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#")
```

file	Le nom du fichier (entre "" ou une variable de mode caractère), éventuellement avec son chemin d'accès (le symbole \ est interdit et doit être remplacé par /, même sous Windows), ou un accès distant à un fichier de type URL (http://...)
header	Une valeur logique (FALSE ou TRUE) indiquant si le fichier contient les noms des variables sur la première ligne.
sep	Le séparateur de champ dans le fichier, par exemple sep = "\t" si dans le cas d'une tabulation
quote	Les caractères utilisés pour citer les variables de mode character
dec	Le caractère utilisé pour les décimales
row.names	Un vecteur contenant les noms des lignes qui peut être un vecteur de mode character, ou le numéro (ou le nom) d'une variable du fichier (par défaut : 1, 2, 3, . . .)
col.names	Un vecteur contenant les noms des variables (par défaut : V1, V2, V3, . . .)
as.is	Contrôle la conversion des variables caractères en facteur (si FALSE) ou les conserve en caractères (TRUE) ; as.is peut être un vecteur logique ou un vecteur numérique précisant les variables conservées en caractère
na.strings	Indique la valeur des données manquantes (sera converti en NA)
colClasses	Un vecteur de caractères donnant les classes à attribuer aux colonnes
nrows	Le nombre maximum de lignes à lire (les valeurs négatives sont ignorées)
skip	Le nombre de lignes à sauter avant de commencer la lecture des données
check.names	Si TRUE, vérifie que les noms des variables sont valides pour R
fill	Si TRUE et que les lignes n'ont pas tous le même nombre de variables, des "blancs" sont ajoutés
strip.white	(Conditionnel à sep) si TRUE, efface les espaces (= blancs) avant

	et après les variables de mode caractère
blank.lines.skip	Si TRUE, ignore les lignes “blanches”
comment.char	Un caractère qui définit des commentaires dans le fichier de données, les lignes commençant par ce caractère sont ignorées (pour désactiver cet argument, utiliser comment.char = "")

Tab 5. Option de la fonction read.table

Nous présentons ci-dessous quatre variantes de `read.table()`. Ces fonctions sont utiles car elles ont des valeurs par défaut différentes des valeurs par défaut de `read.table()`.

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".", fill = TRUE, ...)

read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",", fill = TRUE, ...)

read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".", fill = TRUE, ...)

read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=",", fill = TRUE, ...)
```

La fonction `scan()` est plus flexible que `read.table()` car on peut lui spécifier le type d’objet à créer.

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
     quote = if (sep=="\n") "" else "'\"", dec = ".", skip = 0, nlines = 0,
     na.strings = "NA", flush = FALSE, fill = FALSE, strip.white = FALSE,
     quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE, comment.char = "")
```

file	Le nom du fichier (entre ""), éventuellement avec son chemin d’accès (le symbole \ est interdit et doit être remplacé par /, même sous Windows), ou un accès distant à un fichier de type URL (http://...) ; si file="", les données sont entrées au clavier (l’entrée étant terminée par une ligne blanche)
what	Indique le(s) mode(s) des données lues (numérique par défaut)
nmax	Le nombre de données `a lire, ou, si what est une liste, le nombre de lignes lues (par défaut, scan lit jusqu’à la fin du fichier)
n	Le nombre de données à lire (par défaut, pas de limite)
sep	Le séparateur de champ dans le fichier
quote	Les caractères utilisés pour citer les variables de mode character
dec	Le caractère utilisé pour les décimales
skip	Le nombre de lignes à sauter avant de commencer la lecture des données
nlines	Le nombre de lignes à lire
na.string	Indique la valeur des données manquantes (sera converti en NA)
flush	Si TRUE, scan va à la ligne suivante une fois que le nombre de colonnes est atteint (permet d’ajouter des commentaires dans le fichier de données)
fill	Si TRUE et que les lignes n’ont pas tous le même nombre de variables, des “blancs” sont ajoutés
strip.white	(Conditionnel à sep) si TRUE, efface les espaces (= blancs) avant et après les variables de mode character
quiet	Si FALSE, scan affiche une ligne indiquant les champs lus
blank.lines.skip	Si TRUE, ignore les lignes “blanches”
multi.line	Si what est une liste, précise si les variables du même individu sont sur une seule ligne dans le fichier (FALSE)
comment.char	Un caractère qui définit des commentaires dans le fichier de données, les lignes commençant par ce caractère sont ignorées

Considérons quelques cas typiques d’utilisation des fonctions `read.table()` et `scan()`

Premier cas⊡Le séparateur de champ dans le fichier est l'espace, la première ligne du fichier ne comporte pas le nom des variables, le caractère utilisé pour les décimales est le point.

```
data = read.table("file") ou data = read.table("/…/…/nom_fichier")
```

Deuxième cas⊡Le séparateur de champ dans le fichier est la tabulation, la première ligne du fichier comporte le nom des variables, le caractère utilisé pour les décimales est la virgule.

```
data = read.table("file", header = TRUE, sep = "\t", dec = ",")
```

Dans tous les cas, la fonction `read.table()` retourne une data.frame.

Troisième cas⊡Le séparateur de champ dans le fichier est la tabulation, la première ligne du fchier comporte le nom des variables, le caractère utilisé pour les décimales est le point, les lignes n'ont pas toutes la même longueur (présence de données manquantes)

```
data = read.table("file", header = TRUE, sep = "\t", dec = ".", fill = TRUE)
```

Quatrième cas⊡*utilisation de la fonction scan()*⊡Le séparateur de champ dans le fichier est l'espace, la première ligne du fichier ne comporte pas le nom des variables, le caractère utilisé pour les décimales est le point. On souhaite stocker les données dans une liste

Le jeu de données (iris.txt) est composé de 6 variables numériques. Iris est stocké à l'adresse suivante `"/Documents/donnees/"`.

```
data = scan("/Documents/donnees/iris.txt", what = list(0, 0, 0, 0, 0, 0))
```

Le jeu de données (iris1.txt) est composé de 5 variables numériques et d'une variable nominale.

```
data = scan("/Documents/donnees/iris1.txt", what = list(0, 0, 0, 0, 0, " "))
```

III.4.6 Enregistrer des données

La fonction `write.table()` écrit dans un fichier les objet de type vecteur, matrice ou data.frame. Les arguments et options de cette fonction sont⊡

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n",
            na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
            qmethod = c("escape", "double"))
```

x	Le nom de l'objet à écrire
file	Le nom du fichier (par défaut l'objet est affiché `a l'écran)
append	Si TRUE ajoute les données sans effacer celles éventuellement existantes dans le fichier
quote	Une variable logique ou un vecteur numérique : si TRUE les variables de mode caractère et les facteurs sont écrits entre "", sinon le vecteur indique les numéros des variables à écrire entre "" (dans les deux cas les noms des variables sont écrits entre "" mais pas si quote = FALSE)
sep	Le séparateur de champ dans le fichier
eol	Le caractère imprimé à la fin de chaque ligne ("\n" correspond à un retour-charriot)
na	Indique le caractère utilisé pour les données manquantes
dec	Le caractère utilisé pour les décimales
row.names	Une variable logique indiquant si les noms des lignes doivent être écrits dans le fichier

col.names	Idem pour les noms des colonnes
qmethod	Spécifie, si quote=TRUE, comment sont traitées les guillemets doubles " incluses dans les variables de mode caractère : si "escape" (ou "e", le défaut) chaque " est remplacée par \", si "d" chaque " est remplacée par ""

La fonction `write()` (moins flexible mais plus facile d'utilisation) permet également de sauver des objets de type vecteur, matrice ou data.frame.

```
write(x, file = "data", ncolumns = if(is.character(x)) 1 else 5, append = FALSE)
```

x	Le nom de l'objet à écrire
file	Le nom du fichier (par défaut l'objet est affiché `a l'écran)
ncolumns	Définit le nombre de colonnes du fichier
append	Si TRUE ajoute les données sans effacer celles éventuellement existantes dans le fichier

Enfin pour enregistrer des objets de n'importe quels types, on utilisera la fonction `save()`.

```
save(..., list = character(0), file = stop("'file' must be specified"), ascii =
FALSE)
```

list	Un vecteur de caractère contenant le nom des objts à sauver
file	Chemin ou nom du fichier où doivent être sauver les données.
ascii	Si 'TRUE', une représentation ASCII des données est écrite. La valeur par défaut de ascii est 'FALSE'.

Pour sauver les objets *X* et *Y* dans le fichier mes_donnees.Rdata, à l'adresse /Documents/donnees, on tapera la commande suivante

```
save(X, Y, file = "/Documents/donnees/mes_donnees.Rdata")
```

La fonction save.image est un raccourci de `save(list = ls(all = TRUE), file = ".Rdata")`. Elle permet de sauver tous les objets de la mémoire. Ces données peuvent alors être chargées en mémoire via la fonction load()⊡`load(".Rdata")`.

Remarque⊡Si on ne spécifie pas le chemin de sauvegarde, sauve les données à l'adresse données par la fonction `getwd()`. On peut modifier le chemin par défaut grâce à la fonction `setwd()` Par exemple⊡ `(setwd("/Documents/donnees")`.