Petit manuel d'utilisation de R à destination des étudiants du MASTER IMEA 2

Université de Nice Sophia-Antipolis
Année Universitaire 2011/2012

Table des matières

Chapitre 1

Présentation générale

Le but de ce document est de fournir les éléments de base permettant une prise en main rapide du logiciel R afin de faciliter le déroulement de certains TD mais aussi de vous aider lors des ateliers auxquels vous participerez.
Ce document est largement inspiré d'un manuel produit par Emmanuel Paradis. Pour une documentation plus complète, vous pourrez vous référer aux différentes ressources que vous trouverez dans le chapitre **??** de ce présent document.

R est un système d'analyse statistique et graphique développé par Ross Ihaka et Robert Gentleman. Ce logiciel constitue une alternative au logiciel S-PLUS, même si de nombreuses différence dans la conception existent. Cependant, de nombreux programmes écrits pour S-PLUS sont directement utilisables sous R.

Un point fort de R réside dans le fait que ce logiciel est distribué librement. Son installation peut être mise en œuvre à partir du site internet du *Comprehensive R Archive Network (CRAN)* qui d'une part met à disposition les exécutables et d'autres part donne des informations relatives à la procédure d'installation.

Au cours de ces différents chapitres, ce document permet tout d'abord de connaître les commandes essentielles à la réalisation d'une étude statistique puis d'apprendre à rechercher de l'aide sur les fonctions utilisées, de manière à savoir les arguements à intégrer ou à comprendre les résultats obtenus.

Chapitre 2

Pour démarrer

2.1 Fonctionnement de R

La syntaxe associée à R est relativement simple même si quelques règles sont à connaître. Ainsi, il faut savoir que, pour être exécutée, une fonction doit toujours être suivie de parenthèses, même si ces dernières ne contiennent aucun argument. En effet, leur absence entraîne l'affichage, par R, des lignes de commande de la fonction appelée.

Le symbole ">" qui apparaît en début de ligne montre que R est prêt à être utilisé.
Quand R est utilisé, les variables, données, fonctions, résultats, etc. sont stockés, dans la mémoire de l'ordinateur, sous forme d'objets qui possèdent un nom. L'utilisateur va pouvoir interagir sur ces objets au moyen d'opérateurs ou de fonctions.

De nombreuses fonctions sont déjà stockées dans une bibliothèque localisée sur le disque dans le répertoire R_HOME/library (R_HOME désignant le répertoire où R est installé). Ce répertoire contient des "packages" de fonctions, eux-mêmes présents sur le disque sous forme de répertoire. Le package **base** constitue le cœur du logiciel et contient, comme son nom l'indique, les fonctions de base. Si l'on souhaite utiliser des fonctions appartenant à un autre package, il suffit de charger ce dernier à l'aide de la commande **library**. Par exemple **library(rpart)** permet de charger les fonctions dédiées à CART.

Si l'on souhaite trouver les fonctions qui composent un package défini, il suffit de se rendre dans le répertoire afférent. Par exemple, pour le package R, il suffit de visiter le fichier R_HOME/library/base/R/base.

Une des commandes les plus simples consister à afficher le contenu d'un objet en tapant tout simplement son nom. Par exemple, si l'objet **n** contient la valeur 10 :

```
> n
[1] 10
```

Le chiffre 1 qui apparaît entre crochets indique que l'affichage commence au premier élément élément de **n**. Cette commande est similaire à **print(n)**. Cette dernière sera régulièrement

utilisée pour l'affichage de résultats intermédiaires lors du recours à des boucles.

En ce qui concerne la syntaxe des objets, il faut savoir que le nom d'un objet doit débuter par une lettre et qu'ensuite, il peut comporter des lettres, des chiffres, des points et des espaces soulignés. D'autre part, R effectue une distinction entre les minuscules et les majuscules. Ainsi "A" et "a" désignent deux objets différents.

2.2 Assignation et aide

Afin de créer un objet utilisable par le logiciel, diverses commandes existent.
La plus simple consiste à utiliser l'opérateur d'"assignation" qui correspond à une flèche.

```
> n <- 3
> n
[1] 3
```

ou

```
> 3 -> n
> n
[1] 3
```

Une assignation peut également prendre la forme d'une opération ou d'une fonction, mais seul le résultat sera conservé dans l'objet créé.

Attention, si un objet existe déjà et qu'une nouvelles valeur lui est assignée, la valeur précédente est automatiquement effacée.
Ainsi, il peut être bon d'afficher la liste des objets utilisés. Ceci s'effectue au moyen de la commande **ls**. Si l'on souhaite connaître quelques détails sur les objets créés, on peut aussi recourir à la commande **ls.str()**.

Remarque 2.2.1
Différentes options sont disponibles dans ces fonctions. Par exemple, **pattern** de ne rechercher que les objets contenant un caractère donné **(ls(pattern="n")** retourne la liste des objets dont le nom contient le caractère n, **ls(pattern="n̂")** restreint la recherche au objet commençant par n).

□

Afin de connaître avec précision les options d'une fonction, il suffit de consulter l'aide en en ligne accessible grâce à la commande **help()**. Par exemple **help(ls)** affiche l'aide relative à la fonction **ls**. On y trouve ainsi des informations sur les arguments possibles mais aussi sur le résultat retourné.
D'autres syntaxes équivalentes existent : **help("ls")** ou **?ls**.

Remarque 2.2.2
Dans certains cas, seule la seconde syntaxe existe (par exemple pour les opérateurs logiques, **help("*")**).

Par ailleurs, il peut être utile d'ajouter l'option **try.all.packages=TRUE** dans la fonction **help**. Ainsi si l'on recherche de l'aide sur la fonction **rpart** qui ne fait pas partie du package **base**, **help(raprt)** nous informera simplement que dans les packages chargés l'information demandée n'existe pas. La commande **help("rpart",try.all.packages=TRUE)** permet alors de savoir que la fonction **brpart** est localisée dans le package **rpart** puis **help("rpart", package="rpart")** permet d'obtenir l'affichage de l'aide désirée.

□

Il existe une autre possibilité pour obtenir de l'aide. Elle s'obtient via la commande **help.start()** qui ouvre une aide au format html. Une recherche par mots-clés est alors possible, recherche également disponible sur R par l'intermédiaire de **help.search()**.
Ainsi, si vous tapez **help.search("distribution")**, vous aurez une liste de "fonctions" où aller chercher davantage d'aide. Ainsi, il n'est pas obligatoire de connaître les noms de fonction pour faire une recherche, des mots-clés suffisent.

Chapitre 3

Les données

3.1 Objets R

Le logiciel R manipule des objets. Ces derniers sont caractérisés d'une part par leur nom et d'autre part par leurs attributs qui servent à spécifier la nature ou le type des objets.
Par exemple, il peut être bon de différentier le vecteur $(1, 2, 3)$ suivant que celui-ci représente une variable entière (ex : le nombre d'enfants par famille) ou le codage d'une variable catégorique (ex : oui/non/peut-être).
Les attributs liés à un objets sont de deux types : le mode et la longueur. Le *mode* correspond à la nature même de l'objet ; il en existe principalement 4 (numérique, caractère, complexe et logique). La *longueur* représente le nombre d'éléments constitutifs de l'objet. Afin d'obtenir ces différentes informations, il suffit de faire appel aux commandes **mode** et **length**.

```
> x <- 1
>mode(x)
[1] ‘‘numeric’’
> length(x)
[1] 1
```

Les valeurs manquantes sont toujours représentées par la chaîne de caractères **NA**.
D'autre part, R représente correctement les valeurs infinies (**Inf** et $-$**Inf**) et celles qui ne sont pas des nombres (**NaN**).

3.2 Lecture et enregistrement de données

Il est possible de créer des données directement dans R, mais il est aussi possible d'en importer comme d'en exporter.
Quelle que soit l'opération choisie, il faut faire attention à la localisation du répertoire de travail. Ainsi, la commande **getwd()** permet de savoir le répertoire où l'on se trouve. S'il s'avère nécessaire d'en changer, **setwd** peut nous y aider (ex : **setwd(home/christinet/R)** permet d'accéder au répertoire R du dossier christinet).

3.2.1 Lecture de données

Différentes fonctions permettent de lire des données. Ainsi, **read.table**, **scan** et **read.fwf** permettent d'accéder à des données stockées dans des fichiers de type ASCII. D'autres formats peuvent être lus simplement, ils font alors appel à des fonctions qui ne sont pas de base.

La fonction **read.table** permet de lire très facilement un fichier de donner. Par exemple :

```
> messonnees<- read.table(‘‘mesdonnees.txt’’)
```

crée un tableau de données nommé "mesdonnees" et les variables sont, par défaut, nommées **V1**, **V2**, ... POur accéder à ces dernières, il suffit de taper **mesdonnees$V1** ou **mesdonnees["V1"]** ou encore **mesdonnees[,1]**.
Différentes options sont disponibles dans cette fonction. Prenez le temps d'aller consulter l'aide afférente de façon à obtenir l'effet escompté.

En ce qui concerne la fonction **scan**, elle offre un peu plus de flexibilité puisqu'elle permet, entre autre, de spécifier le mode des variables. Elle permet également de créer différents objets grâce à l'option **what**.
Par exemple :

```
> mesdonnees2 <-scan(‘‘mesdonnees.txt’’,what=list(‘‘ ’’,0,0))
```

crée une liste de trois objets, dont le premier est de mode caractère et les deux suivants de mode numérique.
Dans le cas où les modes spécifiés dans la fonction scan ne correspondent aux données, un message d'erreur apparaîtra.
Pour de plus amples informations, consulter l'aide en ligne.

3.2.2 Enregistrement des données

La façon la plus simple d'écrire des données créées dans R dans un fichier consiste à utiliser la fonction **write**. La syntaxe par défaut **write(x,file="donnees.txt")** permet de stocker l'objet **x** (un vecteur, une matrice ou un tableau) dans le fichier **donnees.txt**. Des options permettent d'enregistrer les données correctement, comme l'option **ncol**.
En ce qui concerne l'enregistrement d'objets de tout type, on peut utiliser la commande **save**.

Quelle que soit l'option retenue, il est préférable d'utiliser le format d'enregistrement ASCII (option **ascii=TRUE)**, car cela permet une meilleure compatibilité entre les différents systèmes d'exploitation.

S'il s'avère que l'on a besoin de charger ces données, on pourra taper ultérieurement **load(nomdufichier)**.

3.3 Création de données

Il existe différentes manière de créer des données quelles soient régulières ou aléatoires.

3.3.1 Données régulières

Une première solution pour créer une séquence de données régulières est la suivante :

```
> a:b
```

qui crée, à partir de la valeur a et par pas de 1, une suite de nombres inférieurs ou égaux à b.

Remarque 3.3.1
La commande : prévaut sur toute autre opération.

□

Si le pas diffère de 1, la commande à utiliser est **seq** dont la syntaxe est la suivante :

```
> seq(a,b,p)
```

où a est la valeur de départ, b la valeur maximale à ne pas dépasser et p le pas. Il se peut que la valeur b ne soit jamais atteinte.
Bien d'autres syntaxes utilisant **seq** sont possibles, elles sont discutées dans l'aide.

Enfin, il y a toujours la possibilité de taper chacune des valeurs comme ci-suit :

```
> c(n_1,n_2,n_3)
```

qui produit le vecteur contenant les valeurs n_1, n_2, n_3.

D'autres fonctions sont également disponibles telles :

- **rep** qui permet de répéter un objet un certain nombre de fois ;
- **sequence** qui crée une suie de séquences de nombres entiers se terminant chacune par les valeurs spécifiées en argument ;
- **gl(k,n)** qui génère une série régulière dans un facteur comprenant k niveaux et n répétitions.

3.3.2 Données aléatoires

En statistique, on apprécie de pouvoir générer des données aléatoires. Il s'avère que R est capable de le faire, et ce pour un grand nombre de densité de probabilité. Ces fonctions sont de la forme *rfunc(n,p_1,p_2, ...)* où *func* désigne la loi de probabilité, n le nombre de données à générer et p_1, p_2, ..., les paramètres de la loi.

Voici un tableau qui fait apparaître la dénomination et les paramètres de certaines lois.

nom de la loi	fonction dans R
Gauss(normale)	rnorm(n,mean=μ,sd=σ)
exponentielle	rexp(n, rate=λ)
gamma	rgamma(n, shape = a, scale = s)
poisson	rpois(n, λ)
weibull	rweibull(n, shape = a, scale = s)
cauchy	rcauchy(n, location = a, scale = s)
student	rt(n, df)
fisher	rf(n,df1,df2)
binomiale	rbinom(n,size,prob)
géométrique	rgeom(n,prob)
uniforme	runif(n, min=a, max=b)

Afin d'obtenir la densité de probabilité, on utilise **dfunc** en lieu et place de rfunc. En ce qui concerne la densité de probabilité cumulée, ce sera **pfunc** et **qfunc** pour la valeur du quantile. Par contre, les arguments diffèrent selon la fonction utilisée, il est donc nécessaire de consulter l'aide associée.

3.4 Manipulation des objets

Nous venons de créer des objets par l'intermédiaire de l'oprateur d'assignation. Cependant, il est possible de créer des objets en définissant leur mode et leur longueur. Cette option permet, par exemple, de créer des objets "vide" puis de les remplir au fur et à mesure.

vecteur :
La commande **vector**, qui comprend deux arguments (mode et length), crée un vecteur composé de 0 si le vecteur est numérique, FALSE s'il est logique et " " s'il s'agit d'un caractère.

facteur :
Le codage d'une variable catégorielle fait appel à la commande **factor** qui inclue les valeurs de la variables mais aussi toutes les modalités possibles.

```
> factor(1:7)
[1] 1 2 3 4 5 6 7
Levels 1 2 3 4 5 6 7
> factor(1:7;levels=1:10)
[1] 1 2 3 4 5 6 7
Levels 1 2 3 4 5 6 7 8 9 10
> factor(1:4;levels=c(``A'',``B'',``F'',``G'')
[1] A B F G
Levels A B F G
```

On constate que **levels** permet d'indiquer les modalités d'une variable dans le cas présent, mais elle peut aussi être utiliser pour connaître les diverses modalités liées à un facteur.

```
> s <- factor(1:7;levels=1:10)
> levels(s)
[1]  ‘‘1’’ ‘‘2’’ ‘‘3’’ ‘‘4’’ ‘‘5’’ ‘‘6’’ ‘‘7’’ ‘‘8’’ ‘‘9’’ ‘‘10’’
```

Matrice :
Une matrice est un tableau comportant des lignes et des colonnes. Par conséquent, la "longueur" associée à une matrice comprend deux informations, celle relative au nombre de lignes et la seconde au nombre de colonnes. Cette information est alors donnée à R sous la forme :

```
> matrix(data=NA,nrow=l;ncol=j)
```

Ceci a le mérite de produite une matrice comptant l lignes et j colonnes avec que des éléments NA. Si l'on souhaite directement remplir une matrice avec des valeurs spécifiées, il faut alors les inscrire dans un vecteur en tant qu'argument data, et faire attention si l'on souhaite que le remplissage de la matrice s'opère par ligne ou par colonne. Voici la différence sur un exemple.

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

> matrix(data=x, ncol=5,nrow=2,byrow=FALSE)
     [,1] [,2] [,3] [,4] [,5]
[1,]  1    3    5    7    9
[2,]  2    4    6    8    10
```

Ainsi, selon que l'option **byrow** est active ou non, le mode de remplissage de la matrice ne s'opère pas de la même façon. Donc, prenez garde.

Data frame :
Un data.frame est un objet implicitement crée lors du recours à la commande **read.table**. On peut fabriquer un tel objet de manière explicite au moyen de **data.frame** comme ci-suit :

```
> x <- 1:4
> y <- 1:3
> z <- c(1,7)
> t <- 3
> data.frame(x,t)
  [x] [t]
1  1   3
2  2   3
3  3   3
4  4   3
```

```
> data.frame(x,z)
  [x] [z]
1   1   1
2   2   7
3   3   1
4   4   7

> data.frame(x,y)
```

Un message d'erreur est alors renvoyé car il y a une incompatibilité au niveau de la longueur des vecteurs.

Remarque 3.4.1
La lecture des messages d'erreur qui apparaissent à l'écran donne, dans la plupart des cas, la clé de l'erreur commise. Donc, lisez attentivement ces messages car ils sont d'une aide précieuse en cas de difficultés.

□

Liste :
Une liste permet de regrouper dans un même objet des éléments différents, sans aucune contrainte liée à la taille ou au mode de ces derniers. Si l'on reprend les notations de l'exemple ci-dessus, voilà ce que l'on peut obtenir.

```
> L1 <- list(x,y)
> L1
[[1]]
[1] 1 2 3 4

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

Par défaut, les noms des objets regroupés dans une liste n'est pas conservé. Si l'on veut pouvoir les nommer, il faut opérer comme ci-dessous, où figure à gauche du signe =, le nom donné à l'objet :

```
> L1 <- list(x=x,y=y)
> L1
$x
[1] 1 2 3 4

$y
[1] 1 2 3
```

Expression :
Ce dernier mode est essentiel pour R. Il consiste en une suite de caractère qui peut être interprétable par le logiciel et évaluer par la suite au moyen de la commande **eval**. C'est bien là un point non négligeable. Mais regardons cela plus précisément sur un exemple :

```
> x <- 1; y <- 2; z <- 3
> exp1 <- expression(x/(y+z))
> exp1
expression(x/(y+z))
> eval(exp1)
[1] 0.2
```

Ce moyen de conserver en mémoire une expression sous sa forme littérale peut s'avérer essentielle puisque certaine fonction, à l'image de **D** (calcul des dérivées partielles), prennent de tels objets comme argument.

Remarque 3.4.2
Il existe un autre objet qui ne sera pas détaillé ici ; il s'agit des séries temporelles qui peuvent être obtenues à l'aide de la commande **ts**, ou encore des fonctions (**function**) que l'on peut créer ou tout simplement de tableau appelé **array**.

Il existe une commande qui permet de convertir un objet en un autre ; il suffit d'utiliser **as.quelquechose** (as.numeric par exemple). Cependant, cette manipulation répond à certaines règles de codage qui peuvent par conséquent modifier la forme même des données.

□

3.5 Opérations élémentaires

3.5.1 Opérateurs usuels

Voici une liste des principaux opérateurs qui sont regroupés en trois grandes classes.

opérateurs		
Arithmétique	Comparaison	Logique
+ addition	< inférieur	! x NON logique
- soustraction	> supérieur	x & y ET logique
* multiplication	<= inférieur ou égal	x && y idem
/ division	>= supérieur ou égal	x | y OU logique
puissance	== égal	x || y idem
%% modulo	!= différent	xor(x,y) OU exclusif
%/% division entière		

Il est bon de savoir que pour indiquer une double inégalité du genre $0 < x < 1$, on écrira $0 < x$ & $x < 1$. Attention, l'omission du symbole logique "ET" entraîne une réponse mais qui n'est pas celle attendue.

Une autre de fonction de comparaison existe, il s'agit de **identical** et sa soeur **all.equal**.

3.5.2 Indexation

Que l'on soit en présence d'un vecteur ou d'une matrice, on peut vouloir extraire un élément ou une suite d'éléments. Une façon d'opérer consiste à recourir à la syntaxe suivante :

```
> x<-c(1,5,7,2)
> x[3]
[1] 7
```

On a ainsi extrait le 3ème élément du vecteur x. Ceci peut également permettre de modifier une des composantes du vecteur.

```
> x[3]<-2
> x
[1] 1 5 2 2
```

Cependant, on peut également extraire un ensemble d'éléments par :

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

Dans le cas où x est une matrice, on peut faire la même chose comme l'illustre les séquences suivantes.

```
> x<-matrix(1:6,2,3)
> x
     [,1] [,2] [,3]
[1,]  1    3    5
[2,]  2    4    6

> x[,3]
[1] 5 6
> x[,3]<-c(1,1)
     [,1] [,2] [,3]
[1,]  1    3    1
[2,]  2    4    1

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

Ce qui vient d'être réalisé sur les colonnes se fait tout aussi bien sur les lignes.

Une autre manière de procéder à l'extraction d'éléments consiste à combiner ce qui précède avec des opérateurs logiques.
Par exemple, si l'on souhaite extraire les valeurs paires d'une suite de valeurs contenues dans x, on tapera $\mathbf{x[x\%\%2 == 0]}$.

3.5.3 Calcul matriciel

Le logiciel offre des facilités dans la manipulation des matrices. Ainsi, les commandes **cbind** et **rbind** permettent de juxtaposer des matrices en conservant les lignes ou les commandes.

```
> x<-matrix(1,ncol=2,nrow=2)
> y<-matrix(2,ncol=2,nrow=2)
> rbind(x,y)
     [,1] [,2]
[1,]  1    1
[2,]  1    1
[3,]  2    2
[4,]  2    2

> x<-matrix(1,ncol=2,nrow=2)
> y<-matrix(2,ncol=2,nrow=2)
> cbind(x,y)
     [,1] [,2] [,3] [,4]
[1,]  1    1    2    2
[2,]  1    1    2    2
```

Un autre opérateur très utile est l'opérateur de multiplication matriciel qui s'écrit $\& * \&$.

D'autres fonctions notables existent telles **diag**, **solve**, **qr**, **eigen** ou **svd**. Mais, nous ne les détaillerons pas ici.

Chapitre 4

Représentation graphique

4.1 fenêtre graphique

En ce qui concerne cette section, une multitude de notions pourraient être abordées car beaucoup de choses sont réalisables. Dans un soucis de concision, nous ne donnerons ici que l'essentiel et le reste sera acquis au fur et à mesure de la manipulation et de vos besoins, notamment en étudiant l'aide sur les fonctions que nous allons mentionner.

Afin d'ouvrir une fenêtre graphique, on peut utiliser les commandes **X11()**, **postscript()** ou **pdf()** selon que l'on souhaite un simple affichage ou un enregistrement. Mais, la fenêtre graphique active, autrement dit celle sur laquelle s'affichera les graphiques, est la dernière appelée.
Afin de connaître les fenêtres ou device ouverts on utilise **dev.list()** qui nous donne le numéro associé à chacune des fenêtres. Ce numéro sera très utile pour modifier la fenêtre graphique au moyen de **dev.set(numéro)**.
Enfin, la commande **dev.off()** permet de fermer la fenêtre active.

Après avoir ouvert un device, on peut vouloir le diviser en plusieurs sous fenêtres. Plusieurs options sont possibles, mais nous n'en évoquerons simplement deux.

Une première méthode consiste à utiliser **split.screen** de la façon suivante :

```
> split.screen(c(1,2))
```

qui permet de diviser la fenêtre en deux parties qui seront appelées par **screen(1)** et **screen(2)**.

Une solution équivalent consiste à recourir à la fonction **par** :

- **par(mfrow=c(2,2)** permet de scinder la fenêtre en quatre sous-fenêtres (2 lignes et 2 colonnes). Le mode de remplissage s'effectue par ligne.
- **par(mcol=c(2,2)** permet de scinder la fenêtre en quatre sous-fenêtres (2 lignes et 2 colonnes). Le mode de remplissage s'effectue par colonne.

Remarque 4.1.1
Une aute fonction utile est **layout** qui peut conduire à des partitions assez complexes.

□

4.2 fonctions graphiques

Dans ce paragraphe ne sera mentionné que les fonctions qui pourront être utiles en statistique. Cependant, nous n'en donnerons qu'un petit aperçu, les détails et options figurant dans l'aide.

plot(x)	graphe des valeurs de x (sur l'axe des y) ordonnées sur l'axe x
plot(x,y)	graphe de y en fonction de x
boxplot(x)	boîte à moustaches
hist(x)	histogramme des fréquences de x
barplot(x)	histogramme des valeurs de x
qqnorm(x)	quantiles de x en fonction des valeurs attendues selon une loi normale
qqplot(x,y)	quantiles de y en fonction de ceux de x

Pour toutes ces fonctions, il existe un nombre impressionnant d'option qu'il est bon de consulter.
Par ailleurs, des commandes comme **legend**, **title**, **text** permettent de commenter des graphiques ce qui est essentiel à la compréhension d'un graphe.

Remarque 4.2.1
R permet une représentation graphique très précise dont les règles sont assez intuitives, d'où le peu d'informations donné dans ce document. Pour compléter cette section, vous pouvez vous reporter au document rédiger par E. Paradis.

□

Chapitre 5

Ressources bibliographiques

Voici quelques références qui pourront compléter votre apprentissage du logiciel R.

Manuels : ils sont pour la plupart distribués avec R et se situent dans R_HOME/doc/manual

- An introduction to R [R-intro.pdf]
- R Data Import/Export [R-data.pdf]
- Writing R Extensions [R-exts.pdf]
- R Language Definition [R-lang.pdf]

Ressources en ligne : de nombreuses documentations sont disponibles sur le site de CRAN dans l'onglet documentation puis contributed. En voici quelques unes.

- Emmanuel Paradis : R pour les débutants
- John Maindonald : Using R for data Analysis and Graphics - Introduction, Examples and Commentary

pour aller plus loin Voici une dernière référence qui est totalement orientée vers l'utilisation de R à des fins statistiques. Le site est très bien fait mais un peu complexe.
Voici l'adresse : http ://pbil.univ-lyon1.fr/R/enseignement.html