Chapitre 1

Prise en main du logiciel

1.1 Présentation et commandes élémentaires

R est un logiciel statistique apparu en 1996. Il propose à la fois un langage et un environnement permettant un traitement et une exploitation avancée de données statistiques (tests, régression, classification et apprentissage, analyse de variance, etc.).

Contrairement à d'autres logiciels couramment utilisés dans le monde de l'entreprise (SAS, ...), **R** est un logiciel libre et gratuit. Ce dernier peut-être téléchargé à l'adresse :

http ://cran.cict.fr/

et installé sur la plupart des systèmes d'exploitation présents sur le marché : Linux, Windows, Mac-Os, etc. Le site contient également une quantité non négligeable d'informations : documentation, forums, FAQ et aide en ligne.

L'interface **R** se présente sous la forme de consoles : commandes, programmes et graphiques. Les instructions sont tapées directement dans la console commande. Nous verrons dans les sections suivantes comment utiliser les graphiques et la programmation.

Les opérateurs arithmétiques usuels s'utilisent de manière très intuitive :

```
> 2+2
[1] 4
> 3*10
[1] 30
> 25/5
[1] 5
```

Pour assigner une valeur à un objet (nous aborderons cette notion plus en détail dans la section suivante), il suffit d'utiliser la commande $<-$ ou $->$:

```
> (10-3)*7 -> a
> a
[1] 49
> babar <- 12/3
> babar
[1] 4
```

R fait la distinction entre les majuscules et les minuscules. Un nom d'objet doit toujours commencer par une lettre mais peut comporter des chiffres ou encore des points.

La fonction *ls* permet d'afficher l'ensemble des objets en mémoire :

```
> ls()
[1] "a"    "babar"
```

Pour plus de détails, il est également possible d'utiliser ls.str :

```
> ls.str()
a :  num 49
babar :  num 4
```

Il est possible d'effacer des objets de la mémoire à l'aide de la commande rm : rm(a) pour effacer l'objet a ou rm(list=ls()) pour l'ensemble des objets contenus dans la mémoire.

Enfin, comme pour tout logiciel, la commande la plus utile est celle permettant d'afficher l'aide en ligne :

```
> help("ls")
ls                    package:base                    R Documentation

List Objects

Description:

     'ls' and 'objects' return a vector of character strings giving the
     names of the objects...
```

La commande *help* renvoie donc un certain nombre d'informations sur la fonction entre guillemets à savoir : une brève description de cette dernière, l'usage que l'on peut en faire, les arguments, le type d'objet retourné ainsi que quelques exemples. Bien souvent, cette rubrique est la plus intéressante.

1.2 Les objets sous R

1.2.1 Création d'objets

Le logiciel **R** fait la distinction entre différents types de données que l'on désignera sous le terme générique d'objets. Ces derniers sont caractérisés par leur nom, leur mode (numérique, logique, caractère ou nombre complexe) et leur longueur. Les objets les plus couramment utilisés sous **R** sont présentés ci-dessous :

Vecteur. Un vecteur est une variable dans le sens courant du terme. Pour créer un vecteur, on utilise la commande vector qui a deux arguments : le mode et la longueur de l'objet crée.

```
> a <- vector(mode="numeric", length=2)
> ls.str()
a :  num [1:2] 0 0
```

Cette commande crée un vecteur numérique de longueur 2. Par défaut, les coordonnées du vecteur valent 0.

Matrices. Une matrice est en fait un vecteur plus la donnée du nombre de lignes et de colonnes.

```
> x <- 1:20
> A <- matrix(x,4,5)
> A
     [,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
```

Le vecteur x est créé de manière implicite. Il contient tous les entiers compris entre 1 et 15. La matrice A est construite en remplissant les lignes et les colonnes avec les valeurs de x. Pour remplir les lignes en premier, on affectera la valeur TRUE à l'option byrow :

```
> B <- matrix(x,4,5,byrow=TRUE)
> B
     [,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
```

Data frames. Un data frame est un ensemble de vecteurs ayant tous la même longueur mais pouvant être de mode différent. Ce type d'objet peut par exemple contenir des données ainsi que leur dénominations respectives et sera donc très utile par la suite.

```
> voitures <- 1:4
> camions <- 3:6
> data.frame(voitures,camions)
  voitures camions
1        1       3
2        2       4
3        3       5
4        4       6
```

Pour donner un nom aux lignes, on utilisera l'option row.names qui doit être un vecteur de mode caractère et de longueur égale au nombre de lignes du data frame.

Série temporelle. Les séries temporelles permettent de collecter des données qui évoluent avec le temps (population d'un pays, chiffre d'affaires d'une entreprise, etc.). Un série temporelle peut être crée à l'aide de la commande *ts()*. Nous reviendrons de manière plus détaillée sur cette dernière d'ici la fin du cours.

Il existe d'autres types d'objets reconnus par **R** mais ces derniers ne seront pas abordés dans ce cours.

1.2.2 Manipulation d'objets et fonctions élémentaires

Il peut parfois être utile d'accéder de manière sélective aux différents éléments d'un objet. Ce type de requête est très intuitive sous **R**. Par exemple, pour accéder à la troisième coordonnée d'un vecteur v, on utilisera v[3]. De même pour une matrice ou un data.frame A, A[i,j] renvoie l'élément situé à la i-ème ligne et la j-ème colonne. Reprenons la matrice B définie précédemment :

```
> B
     [,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
> B[2,5]
[1] 10
> B[,2]
[1]  2  7 12 17
> B[3,]
[1] 11 12 13 14 15
```

La quasi totalité des fonctions usuelles est disponible sous **R** ($\log, \exp, \cos, \sin, \tan, \ldots$). Nous aurons également besoin de fonctions permettant l'exploitation de données vectorielles :

- sum(x) : somme des coefficients du vecteurs x,
- prod(x) : produit des éléments de x,
- min(x) : minimum des coefficients de x (idem pour max(x)).

Les quantités statistiques usuelles (variance, médiane, covariance) sont obtenues par les commandes respectives var, median et cov.

Certaines commandes et fonctions sont dédiées au calcul matriciel. Le logiciel **R** offre à ce titre de nombreuses possibilités. Pour la multiplication de deux matrices, on utilisera l'opérateur %*%.

```
> A <-matrix(2,2,2) ; B <- matrix(3,2,2)
> A;B
     [,1] [,2]
[1,]    2    2
[2,]    2    2
     [,1] [,2]
[1,]    3    3
[2,]    3    3
> A %*% B
     [,1] [,2]
[1,]   12   12
[2,]   12   12
```

On notera également l'existence de la fonction diag permettant la création et la manipulation de matrice diagonales (cf l'aide en ligne pour plus de détails).

1.3 Générer et exploiter des données

1.3.1 Générer des données

Il peut parfois être utile de pouvoir directement générer des données de manière "régulière" ou encore aléatoire. Nous illustrerons en TP quelques théorèmes probabilistes fondamentaux de convergence (Théorème de la limite centrale ou la loi des grands nombres).

Nous avons vu précédemment qu'il était possible de générer un vecteur contenant tous les entiers compris entre deux nombres a et b à l'aide de la commande a :b :

```
> 5:18 -> x
> x
 [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18
```

Pour générer des séquences de nombres réels dans le même esprit, on utilisera la commande seq :

```
> seq(2,5,0.25)
 [1] 2.00 2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00 4.25 4.50 4.75 5.00
```

Le vecteur créé contient tous les nombres réels compris entre 2 et 5 et ayant pour incrément 0.25. On pourra également construire des séquences en rentrant manuellement les valeurs de ces dernières :

```
> c(2,7,7.4,7.5,7.9,8,10)
[1]  2.0  7.0  7.4  7.5  7.9  8.0 10.0
```

Enfin, la commande rep définit un vecteur dont tous les coefficients sont identiques :

```
> rep(2,12)
 [1] 2 2 2 2 2 2 2 2 2 2 2 2
```

Étant donné un vecteur, une matrice, ou encore un data.frame appelé par exemple X, il est possible de modifier chacune des valeurs de cet objet à l'aide de l'instruction data.entry(X). Cette dernière ouvre une console de type *tableur*. Il est alors très facile d'accéder au différentes composantes de l'objet X.

L'ensemble des commandes présentées précédemment permet de créer des séquences déterministes. Il peut être parfois utile de travailler avec des données aléatoires (illustration de Théorème

probabilistes, approximation par méthode Monte-Carlo,etc.). Le logiciel **R** permet de générer une large gamme de variables aléatoires. Ci-dessous, un récapitulatif des différentes densités de probabilité disponibles :

- normale : rnorm(n,mean=0,sd=1)
- exponentielle : rexp(n,rate=1)
- Poisson : rpois(n, lambda=1)
- Student : rs(n,df=2)
- uniforme : runif(n,min=0,max=1)
- binomiale : rbinom(n,size=5,proba=0.5)

Pour chaque commande, n renseigne la taille du vecteur souhaité. Les paramètres spécifiques à chaque densité de probabilité doivent ensuite être spécifiés. Par exemple, si l'on souhaite créer un vecteur de taille 10 dont les coefficients suivent une loi uniforme sur l'intervalle $[2, 3]$, on utilisera :

```
> runif(10,min=2, max=3)
 [1] 2.883957 2.450405 2.362389 2.889726 2.844320 2.838443 2.490955 2.123026
 [9] 2.264480 2.069791
```

Toutes les fonctions présentées ci-dessus peuvent être utilisées pour obtenir la densité de probabilité en un point ou le quantile en remplaçant respectivement le r par d ou q.

Il sera parfois utile de convertir un vecteur, une matrice ou n'importe quel type de donnée sous la forme d'une série temporelle. Cette opération sera réalisé à l'aide de la commande ts (pour *time serie*). Cette fonction admet plusieures options :

- data : un vecteur ou une matrice
- start : date de la première observation
- end : date de la dernière observation
- frequency : fréquence des observations par unité de temps
- names : précise le nom des différentes séries dans le cas d'une série multiple

Exemple de création de série temporelle :

```
> X <- c(1,4,7,9,5,34,25,1,4,0,9)
> ts(X,frequency=12, start=c(2007,9))
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2007                                   1   4   7   9
2008   5  34  25   1   4   0   9
```

Remarque : dans l'exemple précédent, les observations commencent au mois de septembre. On a donné la valeur c(2007,9) à l'option start : la série commence donc au neuvième mois de l'année 2007.

1.3.2 Lecture de données

Le principal intérêt statistique de **R** est la possibilité de créer ou d'exploiter des données de grande taille. Ces dernières pourront être récupérées sous la forme de fichier texte (sources : INSEE, bureau d'études, etc.) et exploitées (tests, régression, graphiques, etc.). Ces aspects seront traités en détail dans la suite de ce cours.

Pour pouvoir récupérer des données, il est utile de connaître le répertoire de travail, c'est-à-dire le répertoire sous lequel les divers résultats seront sauvegardés par défaut. Ce dernier s'obtient à l'aide la commande getwd() :

```
> getwd()
[1] "C:/Program Files/R/R-2.5.1" (Windows)
[1] "/home/Enseignements/R" (Environnement Sun)
```

Pour changer le répertoire de travail, on utilisera la commande *setwd* :

```
> setwd("C:/Documents and Settings/Mes documents/Cours R")
> getwd()
[1] "C:/Documents and Settings/Mes documents/Cours R"
```

Remarque : R ne reconnaît que le caractère "/" pour spécifier le chemin d'accès d'un répertoire (même sous Windows).

Nous nous intéresserons uniquement à la lecture de fichiers textes, bien que **R** puisse lire des fichiers sous des formats aussi différents que SAS, excel, etc., ou encore accéder à des bases de données de type SQL.

La fonction read.table convertit un tableau de données dans un fichier texte en data.frame. Pour un fichier nommé "Tableau.dat", on utilisera la commande :

```
> Tab1 <- read.table("Tableau.dat")
```

qui crée un data.frame "Tab1". Si le fichier n'est pas placé dans le répertoire courant, on pourra spécifier le chemin d'accès de ce dernier directement dans la commande read.table. Cette dernière admet un certaines d'options :

- header : indique si la première ligne contient les noms de variables. Par défaut, la valeur de cette option est FALSE.
- sep : précise le séparateur de champ dans le fichier entre guillemets (" " par défaut).
- dec : le caractère utilisé pour les décimales ("." par défaut).
- row.names : indique par l'intermédiaire d'un vecteur de mode caractère le nom des lignes (par défaut : $1, 2, 3, \ldots$).
- col.names : idem pour les colonnes.
- na.strings : précise la valeur des données manquantes. Par défaut, la valeur de cette option est "NA".
- nrows : nombre maximum de lignes à lire.
- skip : nombre de lignes à sauter avant de commencer à lire des données. Cette option est utile quand le fichier texte contient par exemple un préambule.
- blank.lines.skip : si "TRUE", ignore les lignes blanches.
- comment.char : précise la caractère utilisé pour faire des commentaires. Toutes les lignes commençant par ce caractère ne seront pas prises en compte.

Cette liste d'options n'est pas exhaustive (cf l'aide en ligne) mais permet déjà d'analyser un nombre conséquent de fichiers textes. Il existe également un certain nombre de variantes de la commande read.table dont la valeur des options par défaut est quelque peu différente : read.csv, read.csv2, read.delim, read.delim2.

Imaginons que l'on souhaite récupérer les données d'un fichier intitulé donnees.dat dont le contenu est indiqué ci-dessous :

```
A B C D E
12 50 65 32 2
 8 45 13 5 1
7 36 85 21 26
```

Si ce dernier est placé dans le répertoire courant, on utilisera :

```
> exemple <- read.table("donnees.dat" , header=TRUE)
> exemple
   A  B  C  D  E
1 12 50 65 32  2
2  8 45 13  5  1
3  7 36 85 21 26
```

Par défaut, le nom des variables est $V1, V2, V3, \ldots$.

On peut également mentionner l'existence de la commande read.fwf permettant de lire des données dans un format à largeur fixé. Nous reviendrons plus en détail sur cette dernière dans une planche de TP.

1.3.3 Enregistrer des données dans un fichier

Nous avons vu dans les sections précédentes comment créer et lire des données. Après d'éventuelles modifications, il sera parfois utile de pouvoir sauvegarder ces dernières dans un fichier texte. C'est le chemin inverse de l'étape de lecture. Cette procédure peut être réalisée à l'aide de la commande write.table. Les différentes options disponibles sont détaillées ci-dessous :

- file : nom du fichier dans lequel écrire (vérifier le répertoire courant)
- append : prend une valeur logique. Si TRUE, **R** ajoute les données dans le fichier concerné sans effacer les précédentes. La valeur par défaut est FALSE.
- sep : précise le séparateur à utiliser.
- eol : caractère de fin de ligne. Par défaut prend la valeur "n" (retour chariot)
- na : caractère à utiliser pour les données manquantes.
- dec : précise le caractère à utiliser pour les décimales.

De nombreuses autres options sont disponibles pour cette fonction (cf aide en ligne pour plus de détails).

1.4 Les graphiques

1.4.1 Fonctions graphiques

On distingue sous **R** deux types de commandes graphiques : les fonctions de premier et de second ordre. Les commandes de premier ordre permettent de créer des graphiques à partir d'objets (vecteur, série temporelle, data.frame, etc.). Les commandes de second ordre ont une influence sur des graphiques déjà existants : modifications du titre, de la couleur du fond, etc. et servent essentiellement à améliorer le rendu général.

Voici une liste des principales commandes pour la création de graphiques :

- plot(x) : graphe des valeurs de x (sur l'axe des y) ordonnée par leur position dans le vecteur.
- plot(x,y) : crée un graphe bivarié de x (sur l'axe des x) et de y (sur l'axe des y).
- pie(x) : création d'un graphe en "camembert". Suivant les versions de **R**, on utilise plutôt la commande piechart.
- boxplot(x) : graphique de type "boites et moustaches".
- plot.ts(x) : permet de tracer la graphique d'une série temporelle. Cette dernière peut-être multi variée mais les différentes séries doivent avoir dans ce cas les mêmes fréquences et dates.
- ts.plot(x) : mêmes propriétés que la commandes précédentes sauf que les séries peuvent avoir des dates différentes.
- hist(x) : histogramme des fréquences de x.
- barplot(x) : histogramme des valeurs de x.

La commande lines est également très utile pour tracer des fonctions connues (densité de probabilité, *log*, exponentielle, etc.). Nous aurons souvent l'occasion de l'utiliser dans les planches de TP.

Il est possible de préciser certaines options lors de la création de graphiques. Les options présentées ci-dessus sont communes à la plupart des fonctions introduites ici :

- add si TRUE, superpose le graphique crée à celui existant (valeur FALSE par défaut).
- type : précise la forme des points utilisée : "p" pour des points, "l" pour des lignes, "b" pour des points reliés par des lignes, etc.
- xlim et ylim précise les limites inférieures des axes.
- xlab et ylab : variable de mode caractère, précise le nom à donner aux axes.
- main : précise le titre du graphique créé.

Pour plus de détails : cf l'aide en ligne...

Ces options permettent d'intervenir sur le rendu d'un graphique dès sa création. Il est également possible d'intervenir sur un graphique après sa création. Les commandes permettant de réaliser de telles actions sont dites de second ordre. Voici une liste non-exhaustive des fonctions de ce type les plus utilisées :

- title() : ajoute un titre au graphique courant.

- points(x,y) : ajoute un point à la coordonnée (x, y).
- abline(a,b) : trace un droite de coefficient directeur b et d'ordonnée à l'origine a. Pour un droite horizontale, passant par l'ordonnée y on utilisera abline(h=y). Idem pour une droite vertical en remplaçant h par v.
- mtext(blabla,side=3,...) : ajoute le texte *blabla* dans la marge spécifiée par side. D'autres options sont disponibles.

1.4.2 Manipulation de graphiques

L'ensemble des commandes présentées précédemment permet d'intervenir sur un graphique à la fois. Il est également possible de "configurer" **R** par l'intermédiaire d'un certain nombre de paramètres graphiques. Ce paramétrage peut être réalisé par l'intermédiaire de la fonction par dont une partie des options disponibles est listée ci-dessous :

- bg : permet de choisir la couleur de l'arrière plan par défaut. Il existe 657 couleurs disponibles (colors()) pour toutes les afficher.
- lty : spécifie le type des lignes tracé, 1 pour des lignes continues (par défaut), 2 pour des tirets, 3 pour des points, etc.
- las : permet de paramétrer la façon dont sont disposés les noms des axes, 0 pour parallèles aux axes, 1 pour une notation horizontale, ...

Pour finir, nous allons voir comment gérer les fenêtres graphiques. Il peut en effet être utile de pouvoir travailler sur plusieurs dispositifs graphiques à la fois. Pour ouvrir une fenêtre graphique, on utilise la commande x11() (si aucune fenêtre n'est ouverte, l'exécution d'une des commandes présentées dans la section précédente ouvrira automatiquement une fenêtre).

Il est possible d'ouvrir autant de fenêtre que l'on souhaite. Dans ce cas, il est tout de même nécessaire de connaître le dispositif actif, c'est-à-dire celui qui sera utilisé lors de la prochaine instruction. La commande dev.list affiche la liste de tous les dispositifs ouverts, dev.cur renseigne sur le dispositif courant et enfin dev.set permet de changer de fenêtre.

```
> x11();x11();x11()
> dev.list()
x11 x11 x11
  2   3   4
> dev.cur()
x11
  4
> dev.set(2)
x11
  2
```

Lorsque l'on souhaite gérer plusieurs graphiques à la fois, il peut être gênant d'avoir à manipuler plusieures fenêtres. Une alternative consiste donc à partitionner le dispositif courant en plusieurs morceaux. La fonction layout prend en argument une matrice et partitionne la fenêtre en plusieurs parties.

```
> x11()
> layout(matrix(1:4,2,2))
> layout.show(4)
```

La commande layout.show(4) permet de visualiser la partition ainsi créée. La forme de la partition est complètement conditionnée par la nombre de lignes et de colonnes de la matrice. On pourra également penser à utiliser l'option byrow pour "remplir" différemment le dispositif. L'ensemble des graphiques construits seront affichés successivement sur la partition.

1.5 Fonctions et programmation

Le logiciel **R** permet également à l'utilisateur de créer ses propres fonctions. Cet aspect est particulièrement utile pour effectuer des tâches répétitives.

La création d'une fonction sous **R** est très intuitive. La syntaxe utilisée est très proche de celles du langage de programmation **C**.

```
> exemple1 <- function(n)
+   {
+     x <- 3*n+2
+     print(x)
+ }
> exemple1(1)
[1] 5
> exemple1(10)
[1] 32
```

La fonction print permet d'afficher les informations souhaitée. Comme pour tout langage de programmation, la présentation d'une fonction a énormément d'importance. Pour sauter une ligne sans exécuter les instructions précédentes, on utilisera "SHIFT+ENTREE".

Il est possible de créer des fonctions admettant plusieurs entrées et de différent types : vecteurs, caractères, matrices, data frame, etc. Il sera parfois utile de faire appel à des boucles pour automatiser des tâches longues et répétitives. La fonction permettant de faire la somme de deux vecteurs pourrait par exemple être écrite de la manière suivante :

```
> addition <- function(x,y)
+   {
+    if (length(x) != length(y)) print("erreur") else {
+         z <- numeric(length(x))
+         for (i in 1:length(x)) z[i]<-x[i]+y[i]
+         print(z) }
+   }
> x <- c(1,2,3)
> y <- c(4,5)
> addition(x,y)
[1] "erreur"
> y <- c(4,5,6)
> addition(x,y)
[1] 5 7 9
```

Il est également de faire des boucles "while" dont la syntaxe est :

```
while ( i>min) {
    ...
      }
```

On répète les instructions entre accolades tant que la condition logique i>min est satisfaite.

Une fonction doit être "compilée" à chaque nouveau démarrage de **R**. Si l'on est amené à utiliser régulièrement une fonction, il sera utile de sauvegarder cette dernière dans un fichier texte et d'utiliser un "copier-coller" à chaque nouvelle utilisation.