Aide-mémoire R

Ce document présente quelques unes des fonctionnalités de R qui peuvent être utile dans le cadre des TP de méthodes quantitatives. Pour aller plus loin, il faut se reporter aux nombreux ouvrages et manuels renseignés à la fin de ce document.

Table des matières

Installer R

Installation sous Windows :

L'installateur windows est téléchargeable sur http://cran.r-project.org/bin/windows/. Une fois le téléchargement terminé, double-cliquez sur ce fichier. Suivez les instructions de l'installateur. Au moment de choisir les "composants à installer" et les "options de démarrage", laissez les options cochées par défaut.
Si l'ordinateur sur lequel vous souhaitez installer R n'est pas connecté à Internet, il est possible de l'installer à partir d'un CD – demandez à l'assistante.

Installation sous GNU/linux/ubuntu :

Installer r-base via le gestionnaire de paquets synaptic ou en tapant dans la console

```
sudo apt-get install r-base
```

Pour les autres distributions de Linux, voir http://cran.r-project.org/bin/linux/.

Installation sous Mac OS:

Downloader l'installateur sur la page http://cran.r-project.org/bin/macosx/ puis lancer l'installation.

Pour plus d'informations lire par exemple le manuel "R Installation and Administration" (disponible sur http://cran.r-project.org/doc/manuals/R-admin.pdf).

Lancer R

Sous Windows ou Mac OS:
Double-cliquez sur l'icône du programme. Lorsque vous lancez le programme, il se présente sous la forme d'une fenêtre appelée RGui (GUI pour "Graphical User Interface", interface graphique), qui permet l'accès à quelques menus de base, et d'une fenêtre R Console, dans laquelle l'utilisateur entre des instructions sous forme de texte. Après un message d'avertissement, l'invite de commande de R (>) apparaît. Toutes les étapes qui suivent se feront par l'intermédiaire de ce terminal.

Sous GNU/Linux/ubuntu:
Lancez R en tapant simplement

```
R
```

dans la console (menu Applications/Accessoires/Terminal).
Après un message d'avertissement, l'invite de commande de R (>) apparaît. Toutes les étapes qui suivent se feront par l'intermédiaire du même terminal.

Quitter R et récupérer votre travail plus tard

Pour quitter, fermer la fenêtre ou tapez

```
q()
```

A la fin de chaque session dans R, il vous sera proposé de "sauver une image" de cette session. Si vous souhaitez garder une trace de votre travail, répondez oui et vous pourrez retrouver une sauvegarde de votre session la prochaine fois que vous lancerez R. Vous pourrez donc retrouver vos anciennes commandes en utilisant la flèche vers le haut.

R crée dans votre répertoire de travail un fichier .RData, qui peut être ouvert dans R, et un fichier .Rhistory, qui peut être lu dans un éditeur de texte. Il faut (sous Windows comme sous Linux) afficher les fichiers cachés pour voir ces fichiers. Sous Windows, vous pouvez démarrer R en double-cliquant sur le fichier .RData. Sous Linux il faut charger l'historique en faisant

```
load(" .RData ")
loadhistory(" .Rhistory ")
```

Si vous souhaitez une interface un rien plus conviviale...

...c'est possible. Il existe des interfaces graphiques plus ou moins complètes, avec un certain nombre de fonctions accessibles via des menus, ce qui permet d'éviter de taper toutes les instructions en ligne de commande[1].
"R Commander" est une de ces interfaces, qui permet entre autres d'importer des données, de les éditer, de les visualiser, de faire quelques graphiques et analyses statistiques de base (statistiques descriptives, corrélations, régressions, classifications, analyses factorielles, ...), etc., sans passer par la ligne de commande. Les instructions s'affichent cependant sous forme de texte au moment où vous les lancez, ce qui vous permet de contrôler les fonctions utilisées et éventuellement de les modifier (pour relancer une commande, la sélectionner dans la "fenêtre de script" et cliquer sur le bouton "soumettre").

Pour installer R Commander sous Windows:
Dans la fenêtre RGui, menu "Packages" / "Install packages", choisissez "Rcmdr". Pour plus d'informations voir par exemple http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html
Pour installer R Commander sous Mac OS:

[1] Pour un aperçu des interfaces disponibles, voir par exemple http://en.wikipedia.org/wiki/R_%28programming_language%29#Graphical_user_interfaces

Suivez les instructions suivantes (extraites de
http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/installation-notes.html):

The procedure for installing the R Commander under Mac OS X is more complicated, so please read and follow these instructions carefully. These instructions and the associated files are intended for Mac OS X 10.5 (Leopard) or 10.6 (Snow Leopard) systems. I assume that you've already installed R.

- Check to see if the X11 windowing system (X Windows) has already been installed on your computer (as is probably the case): The file X11.app should appear in the Utilities folder under Applications in the finder. If X11.app is missing, you can install it from your Mac OS X installation disc as follows:
    - Insert your Mac OS X install disc. (If you have two discs it will be on the"Install Disc 1").
    - Double click on Optional Installs.
    - Double click on Optional Installs.mpkg, then click Continue and accept the license agreement.
    - Click the triangle next to Applications in order to expand the list of applications.
    - Check "X11", and then click Continue and Install. Click Close when the installation finishes.
- Download tcltk-8.5.5-x11.dmg, which is the installer for Tcl/Tk for X Windows.
    - Install Tcl/Tk for X Windows by double-clicking on the downloaded file tcltk-8.5.5-x11.dmg and then double-clicking on the installer file tcltk.pkg. Continue through the installation.
- Start R by running R.app. At the R > command prompt, type the following command and press the return key (to avoid errors, you can copy the command from this document and paste it at the R > command prompt):

  install.packages("Rcmdr", dependencies=TRUE)

  R will ask you to select a CRAN mirror; pick a mirror site near you. The install.packages command may report some warnings about missing packages, but these are not needed, and you can ignore the warnings. Note that many packages will be downloaded and installed, so be patient.

Once it is installed, to load the Rcmdr package, simply issue the command

```
library(Rcmdr)
```

at the R > command prompt and press return.

Pour installer R Commander sous linux:
Une fois R lancé, tapez dans le terminal
`install.packages("Rcmdr", dependencies=TRUE)`
R Commander est alors disponible via le menu Applications.

Mode de fonctionnement général de R: commandes, objets, fonctions

Le symbole > vous invite à entrer une commande. C'est dans la même console que R affichera la majorité des résultats.

Raccourci pratique: la flèche vers le haut, sur votre clavier, rappelle la dernière instruction que vous avez tapée. R possède aussi une autre fonctionnalité pratique: lorsque vous appuyer sur la touche "tab", il propose l'autocomplétion des mots que vous êtes en train de taper.

Pour interrompre un processus, faites ctrl+C (sous Windows, *Escape* peut aussi fonctionner). Pour retrouver l'invite de commande, tapez `q`.

Extraits de Julien Barnier, "R pour les sociologues (et assimilés)" (disponible sur http://cran.r-project.org/doc/contrib/Barnier-intro_R.pdf):

2.2 Des objets

2.2.1 Objets simples

Faire des opérations arithmétiques, c'est bien, mais sans doute pas totalement suffisant. Notamment, on aimerait pouvoir réutiliser le résultat d'une opération sans avoir à le ressaisir ou à le copier/coller.

Comme tout langage de programmation, R permet de faire cela en utilisant des *objets*. Prenons tout de suite un exemple :

```
R> x <- 2
```

Que signifie cette commande ? L'opérateur <- est appelé *opérateur d'assignation*. Il prend une valeur quelconque à droite et la place dans l'objet indiqué à gauche. La commande pourrait donc se lire *mettre la valeur 2 dans l'objet nommé* x.

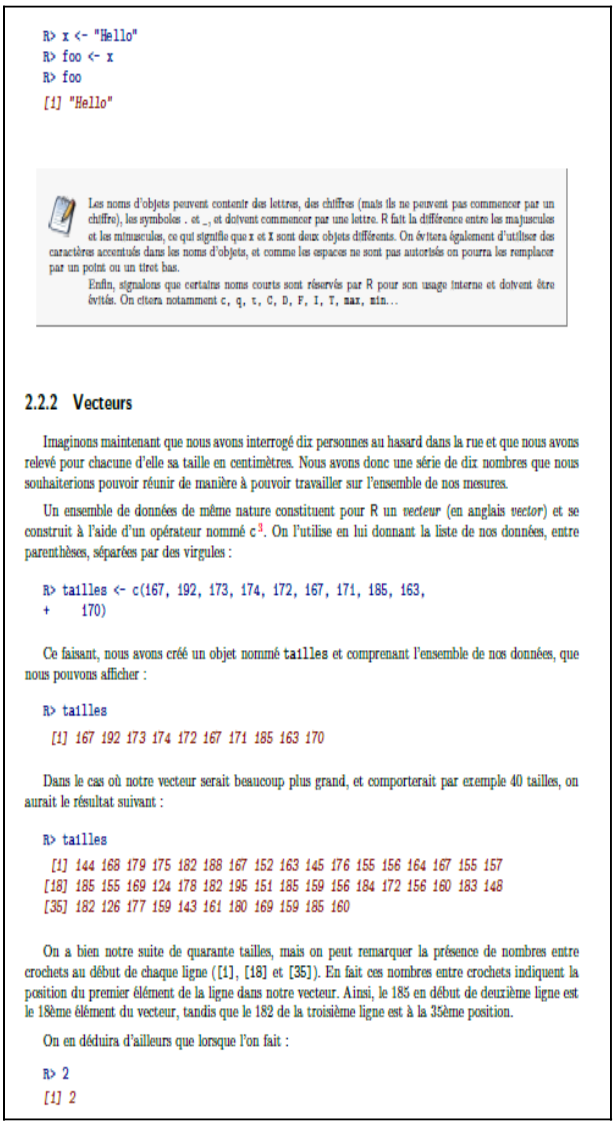
On va ensuite pouvoir réutiliser cet objet dans d'autres calculs ou simplement afficher son contenu :

```
R> x + 3
[1] 5
R> x
[1] 2
```

Par défaut, si on donne à R seulement le nom d'un objet, il va se débrouiller pour nous présenter son contenu d'une manière plus ou moins lisible.

On peut utiliser autant d'objets qu'on veut. Ceux-ci peuvent contenir des nombres, des chaînes de caractères (indiquées par des guillemets droits ") et bien d'autres choses encore :

```
R> x <- 27
R> y <- 10
R> foo <- x + y
R> foo
[1] 37
```



```
R> x <- "Hello"
R> foo <- x
R> foo
[1] "Hello"
```

Les noms d'objets peuvent contenir des lettres, des chiffres (mais ils ne peuvent pas commencer par un chiffre), les symboles . et _, et doivent commencer par une lettre. R fait la différence entre les majuscules et les minuscules, ce qui signifie que `x` et `X` sont deux objets différents. On évitera également d'utiliser des caractères accentués dans les noms d'objets, et comme les espaces ne sont pas autorisés on pourra les remplacer par un point ou un tiret bas.

Enfin, signalons que certains noms courts sont réservés par R pour son usage interne et doivent être évités. On citera notamment `c`, `q`, `t`, `C`, `D`, `F`, `I`, `T`, `max`, `min`...

2.2.2 Vecteurs

Imaginons maintenant que nous avons interrogé dix personnes au hasard dans la rue et que nous avons relevé pour chacune d'elle sa taille en centimètres. Nous avons donc une série de dix nombres que nous souhaiterions pouvoir réunir de manière à pouvoir travailler sur l'ensemble de nos mesures.

Un ensemble de données de même nature constituent pour R un *vecteur* (en anglais *vector*) et se construit à l'aide d'un opérateur nommé c[3]. On l'utilise en lui donnant la liste de nos données, entre parenthèses, séparées par des virgules :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163,
+     170)
```

Ce faisant, nous avons créé un objet nommé `tailles` et comprenant l'ensemble de nos données, que nous pouvons afficher :

```
R> tailles
 [1] 167 192 173 174 172 167 171 185 163 170
```

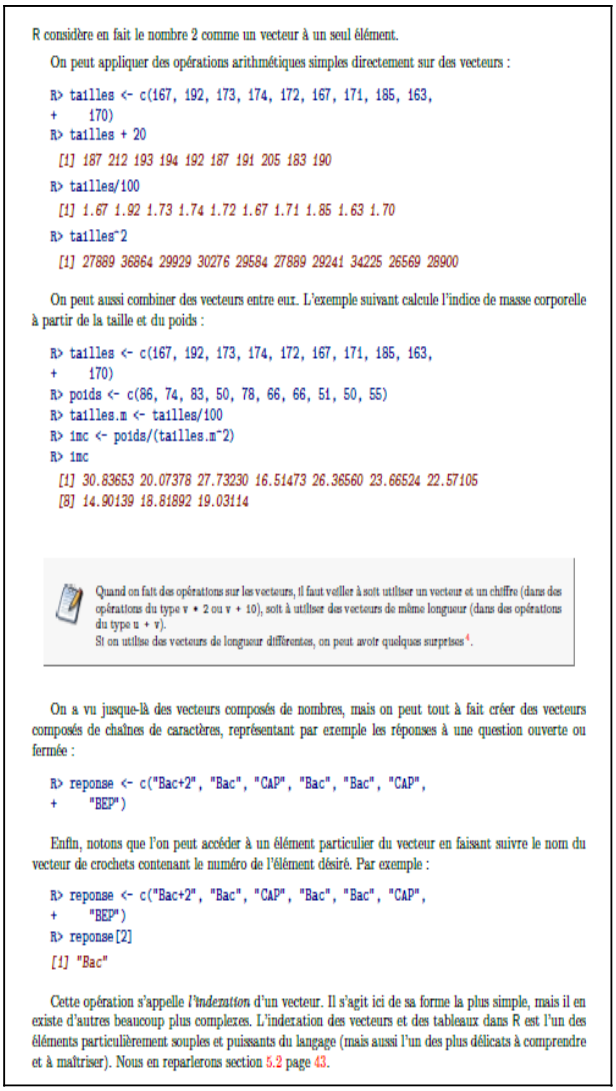
Dans le cas où notre vecteur serait beaucoup plus grand, et comporterait par exemple 40 tailles, on aurait le résultat suivant :

```
R> tailles
 [1] 144 168 179 175 182 188 167 152 163 145 176 155 156 164 167 155 157
[18] 185 155 169 124 178 182 195 151 185 159 156 184 172 156 160 183 148
[35] 182 126 177 159 143 161 180 169 159 185 160
```

On a bien notre suite de quarante tailles, mais on peut remarquer la présence de nombres entre crochets au début de chaque ligne ([1], [18] et [35]). En fait ces nombres entre crochets indiquent la position du premier élément de la ligne dans notre vecteur. Ainsi, le 185 en début de deuxième ligne est le 18ème élément du vecteur, tandis que le 182 de la troisième ligne est à la 35ème position.

On en déduira d'ailleurs que lorsque l'on fait :

```
R> 2
[1] 2
```



R considère en fait le nombre 2 comme un vecteur à un seul élément.

On peut appliquer des opérations arithmétiques simples directement sur des vecteurs :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163,
+     170)
R> tailles + 20
 [1] 187 212 193 194 192 187 191 205 183 190
R> tailles/100
 [1] 1.67 1.92 1.73 1.74 1.72 1.67 1.71 1.85 1.63 1.70
R> tailles^2
 [1] 27889 36864 29929 30276 29584 27889 29241 34225 26569 28900
```

On peut aussi combiner des vecteurs entre eux. L'exemple suivant calcule l'indice de masse corporelle à partir de la taille et du poids :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163,
+     170)
R> poids <- c(86, 74, 83, 50, 78, 66, 66, 51, 50, 55)
R> tailles.m <- tailles/100
R> imc <- poids/(tailles.m^2)
R> imc
 [1] 30.83653 20.07378 27.73230 16.51473 26.36560 23.66524 22.57105
 [8] 14.90139 18.81892 19.03114
```

Quand on fait des opérations sur les vecteurs, il faut veiller à soit utiliser un vecteur et un chiffre (dans des opérations du type `v * 2` ou `v + 10`), soit à utiliser des vecteurs de même longueur (dans des opérations du type `u + v`).

Si on utilise des vecteurs de longueur différentes, on peut avoir quelques surprises[4].

On a vu jusque-là des vecteurs composés de nombres, mais on peut tout à fait créer des vecteurs composés de chaînes de caractères, représentant par exemple les réponses à une question ouverte ou fermée :

```
R> reponse <- c("Bac+2", "Bac", "CAP", "Bac", "Bac", "CAP",
+     "BEP")
```

Enfin, notons que l'on peut accéder à un élément particulier du vecteur en faisant suivre le nom du vecteur de crochets contenant le numéro de l'élément désiré. Par exemple :

```
R> reponse <- c("Bac+2", "Bac", "CAP", "Bac", "Bac", "CAP",
+     "BEP")
R> reponse[2]
[1] "Bac"
```

Cette opération s'appelle *l'indexation* d'un vecteur. Il s'agit ici de sa forme la plus simple, mais il en existe d'autres beaucoup plus complexes. L'indexation des vecteurs et des tableaux dans R est l'un des éléments particulièrement souples et puissants du langage (mais aussi l'un des plus délicats à comprendre et à maîtriser). Nous en reparlerons section 5.2 page 43.

D'autres types d'objets existent: listes, matrices, tableaux, etc.
A tout moment, vous pouvez taper

```
ls()
```

pour voir quels objets votre environnement de travail contient. Cette fonction est très utile si vous avez de nombreuses tables et listes, et que vous avez tendance à oublier leurs noms.

2.3 Des fonctions

Nous savons désormais faire des opérations simples sur des nombres et des vecteurs, stocker ces données et résultats dans des objets pour les réutiliser par la suite.

Pour aller un peu plus loin nous allons aborder, après les *objets*, l'autre concept de base de R, à savoir les *fonctions*. Une fonction se caractérise de la manière suivante :

- elle a un nom ;
- elle accepte des arguments (qui peuvent avoir un nom ou pas) ;
- elle retourne un résultat et peut effectuer une action comme dessiner un graphique, lire un fichier, etc. ;

En fait rien de bien nouveau puisque nous avons déjà utilisé plusieurs fonctions jusqu'ici, dont la plus visible est la fonction `c`. Dans la ligne suivante :

```
R> reponse <- c("Bac+2", "Bac", "CAP", "Bac", "Bac", "CAP",
+     "BEP")
```

on fait appel à la fonction nommée `c`, on lui passe en arguments (entre parenthèses et séparées par des virgules) une série de chaînes de caractères, et elle retourne comme résultat un vecteur de chaînes de caractères, que nous stockons dans l'objet `tailles`.

Prenons tout de suite d'autres exemples de fonctions courantes :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163,
+     170)
R> length(tailles)
[1] 10
R> mean(tailles)
[1] 173.4
R> var(tailles)
[1] 76.71111
```

Ici, la fonction `length` nous renvoit le nombre d'éléments du vecteur, la fonction `mean` nous donne la moyenne des éléments du vecteur, et la fonction `var` sa variance.

2.3.1 Arguments

Les arguments de la fonction lui sont indiqués entre parenthèses, juste après son nom. En général les premiers arguments passés à la fonction sont des données servant au calcul, et les suivants des paramètres influant sur ce calcul. Ceux-ci sont en général transmis sous la forme d'argument nommés.

Reprenons l'exemple des tailles précédent :

```
R> tailles <- c(167, 192, 173, 174, 172, 167, 171, 185, 163,
+     170)
```

Imaginons que le deuxième enquêté n'ait pas voulu nous répondre. Nous avons alors dans notre vecteur une valeur manquante. Celle-ci est symbolisée dans R par le code `NA` :

```
R> tailles <- c(167, NA, 173, 174, 172, 167, 171, 185, 163,
+     170)
```

Recalculons notre taille moyenne :

```
R> mean(tailles)
[1] NA
```

Et oui, par défaut, R renvoit `NA` pour un grand nombre de calculs (dont la moyenne) lorsque les données comportent une valeur manquante. On peut cependant modifier ce comportement en fournissant un paramètre supplémentaire à la fonction `mean`, nommé `na.rm` :

```
R> mean(tailles, na.rm = TRUE)
[1] 171.3333
```

Positionner le paramètre `na.rm` à `TRUE` (vrai) indique à la fonction `mean` de ne pas tenir compte des valeurs manquantes dans le calcul.

Lorsqu'on passe un argument à une fonction de cette manière, c'est-à-dire sous la forme `nom=valeur`, on parle d'*argument nommé*.

2.3.2 Quelques fonctions utiles

Récapitulons la liste des fonctions que nous avons déjà rencontrées :

Fonction	Description
`c`	construit un vecteur à partir d'une série de valeurs
`length`	nombre d'éléments d'un vecteur
`mean`	moyenne d'un vecteur de type numérique
`var`	variance d'un vecteur de type numérique
`+, -, *, /`	opérateurs mathématiques de base
`^`	passage à la puissance

On peut rajouter les fonctions de base suivantes :

Fonction	Description
`min`	valeur minimale d'un vecteur numérique
`max`	valeur maximale d'un vecteur numérique
`sd`	écart-type d'un vecteur numérique
`:`	génère une séquence de nombres. `1:4` équivaut à `c(1,2,3,4)`

Obtenir de l'aide

Extrait de Julien Barnier, "R pour les sociologues (et assimilés)" (disponible sur http://cran.r-project.org/doc/contrib/Barnier-intro_R.pdf):

10.1.1 Aide sur une fonction

La fonction la plus utile est sans doute celle qui permet d'afficher la page d'aide liée à une ou plusieurs fonctions. Celle-ci permet de lister les arguments de la fonction, d'avoir des informations détaillées sur son fonctionnement, les résultats qu'elle retourne, etc.

Pour accéder à l'aide de la fonction `mean`, par exemple, il vous suffit de saisir directement :

```
R> help("mean")
```

Ou sa forme abrégée :

```
?mean
```

Chaque page d'aide comprend plusieurs sections, en particulier :

Description donne un résumé en une phrase de ce que fait la fonction

Usage indique la ou les manières de l'utiliser

Arguments détaille tous les arguments possibles et leur signification

Value indique la forme du résultat renvoyé par la fonction

Details apporte des précisions sur le fonctionnement interne de la fonction

Note pour des remarques éventuelles

References pour des références bibliographiques ou des URL associées

See Also *très utile*, renvoit vers d'autres fonctions semblables ou liées, ce qui peut être très utile pour découvrir ou retrouver une fonction dont on a oublié le nom

Examples série d'exemples d'utilisation

Les exemples peuvent être directement exécutés en utilisant la fonction `example` :

```
R> example(mean)
```

Définir un répertoire de travail

Si vous tapez
`getwd()`
R affiche le chemin vers son répertoire de travail courant, c'est-à-dire le dossier où il va aller chercher les fichiers que vous lui demanderez d'importer, déposer les fichiers que vous lui demanderez d'exporter, et sauver l'historique de la session.

Vous pouvez définir votre propre répertoire de travail pour R en utilisant la commande
`setwd()`
par exemple
`setwd("/home/brigittebardot/acp")`
ce qui implique de connaître le chemin vers votre dossier depuis la racine du système.

Vous pouvez vérifier que R travaille dans le bon répertoire en entrant à nouveau
`getwd()`

Lorsque vous travaillez dans R sous windows, il est préférable que les extensions de tous les fichiers s'affichent: allez dans le menu démarrer/panneau de configuration/option des dossiers/affichage et décochez l'option "masquer les extensions des fichiers dont le type est connu". De même, afficher les fichiers cachés peut parfois s'avérer utile. Si nécessaire, faites-le depuis l'explorateur de fichiers: sous GNU/Linux/Ubuntu, Ctrl+H ; sous Windows, menu outils/option des dossiers.

Installer un package

Un package (en français “extension”, "bibliothèque" ou "librairie logicielle") est un paquet de fonctions supplémentaires . En effet, R permet de faire toutes sortes d’analyses, et pour ne pas surcharger le programme (et l’utilisateur) toutes ces fonctions ne sont pas installées "par défaut". Il y a une trentaine de librairies installées dans la version de base, et plus de 450 librairies additionnelles.

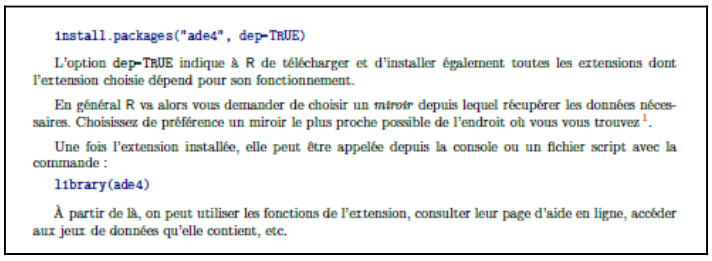
Extrait de Julien Barnier, "R pour les sociologues (et assimilés)” (disponible sur http://cran.r-project.org/doc/contrib/Barnier-intro_R.pdf):

B.2 Installation des extensions

Les interfaces graphiques sous Windows ou Mac OS X permettent la gestion des extensions par le biais de boîtes de dialogues (entrées du menu *Packages* sous Windows par exemple). Nous nous contenterons ici de décrire cette gestion *via* la console.

On notera cependant que l'installation et la mise à jour des extensions nécessite d'être connecté à l'Internet.

L'installation d'une extension se fait par la fonction `install.packages`, à qui on fournit le nom de l'extension. Ici on souhaite installer l'extension `ade4` :



```
install.packages("ade4", dep=TRUE)
```

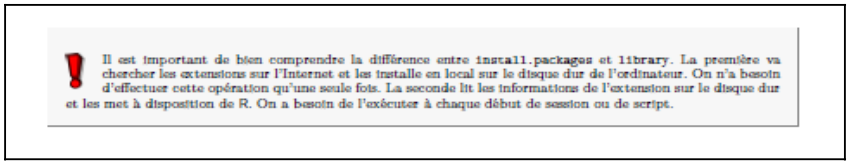
L'option `dep=TRUE` indique à R de télécharger et d'installer également toutes les extensions dont l'extension choisie dépend pour son fonctionnement.

En général R va alors vous demander de choisir un *miroir* depuis lequel récupérer les données nécessaires. Choisissez de préférence un miroir le plus proche possible de l'endroit où vous vous trouvez [1].

Une fois l'extension installée, elle peut être appelée depuis la console ou un fichier script avec la commande :

```
library(ade4)
```

À partir de là, on peut utiliser les fonctions de l'extension, consulter leur page d'aide en ligne, accéder aux jeux de données qu'elle contient, etc.



Il est important de bien comprendre la différence entre `install.packages` et `library`. La première va chercher les extensions sur l'Internet et les installe en local sur le disque dur de l'ordinateur. On n'a besoin d'effectuer cette opération qu'une seule fois. La seconde lit les informations de l'extension sur le disque dur et les met à disposition de R. On a besoin de l'exécuter à chaque début de session ou de script.

Importer vos données

Vos variables (colonnes) doivent avoir des intitulés courts, explicites et si possible sans espaces ni caractères spéciaux.

Si vos données sont au format texte, l'importation se fait avec la fonction
`read.table("nomdufichier.csv")`
N'oubliez pas les guillemets.
Cette fonction[2] accepte différents arguments, qui se mettent entre les parenthèses, séparés par des virgules. Le premier argument est le nom de votre fichier. Si votre fichier se trouve dans votre répertoire de travail, il n'y a pas besoin de spécifier l'endroit où il se trouve. Les autres arguments concernent les modalités d'importation: `sep` pour spécifier le type de séparateur de champs, `dec` pour spécifier le symbole décimal, `header` pour indiquer la présence d'entêtes (titres) de colonnes (s'il y en a, faire `header=TRUE`), `row.names` pour indiquer où se trouvent les identifiants des lignes (càd des observations, càd les noms ou les codes des

[2] Voir aussi les fonctions read.csv() et read.csv2()

communes, pays, secteurs statistiques, ou autre).

Par exemple, pour importer dans R un fichier .txt avec des tabulations comme séparateur de champs, des virgules comme séparateurs de décimales, des entêtes de colonnes et les codes/noms des lieux dans la première colonne, entrez :
```
read.table("monfichier.txt", sep="\t", dec=",", row.names=1)
```

Vous pouvez utiliser d'autres séparateurs de champs en remplaçant "\t" par une virgule, un point-virgule, ou n'importe quoi d'autre. La virgule est cependant à déconseiller si vos données contiennent des décimales.

Si vous avez utilisé l'argument `row.names`, votre colonne d'identifiants n'est pas votre première colonne. Il est important de le vérifier car cela influencera les colonnes à prendre en compte lors de vos analyses.

R remplacera les cellules vides de votre fichier de départ par des "NA" ("not available"). Si les valeurs manquantes sont représentées d'une autre manière (le caractère "/" par exemple) dans votre fichier de départ, spécifiez-le à l'aide de l'argument `na.strings`, par exemple:
```
read.table("monfichier.csv", na.strings="/", dec=".")
```

Si R a pu charger correctement votre table, vous devriez voir apparaître vos données dans la console. Vérifiez que tout y est, que les entêtes de colonnes sont à la bonne place, que les symboles décimaux ont été correctement interprétés (càd remplacés par des points), et que les valeurs manquantes ont été remplacées par des NA.

Vos données peuvent être mises dans un objet utilisable par les fonctions de R, objet auquel vous donnez un nom ("matable" dans l'exemple qui suit). Entrez:
```
matable<-read.table(...)
```
(en intégrant les arguments nécessaires selon les caractéristiques de votre fichier). En utilisant la flèche <- vous faites ce qu'on appelle une affectation.

Pour plus d'informations voir par exemple les pages 33 et suivantes de Julien Barnier, "R pour les sociologues (et assimilés)" (disponible sur http://cran.r-project.org/doc/contrib/Barnier-intro_R.pdf) ou le manuel "R Data Import/Export" (disponible sur http://cran.r-project.org/doc/manuals/R-data.pdf).

Manipulation de votre table de données

Vous pouvez à tout moment afficher le contenu de cet objet en tapant simplement son nom. Entrez simplement:

```
matable
```

Pour voir seulement quelques lignes de la table, tapez

```
head(matable)
```

Pour éditer cette table tapez

```
fix(matable)
```

Pour modifier cette table et sauver le résultat dans une nouvelle table sans modifier la table de départ

```
nouvelletable<-edit(matable)
```

Si vous avez importé des données sous forme de table, cet objet est un "data frame". Vous pouvez le vérifier en tapant:

```
is.data.frame(matable)
```

Les entêtes de colonnes devraient apparaître lorsque vous entrez:

```
names(matable)
```

Vous pouvez voir le contenu d'une colonne en utilisant le caractère $. Par exemple, si vous voulez voir le contenu de la colonne "macolonne", entrez:

```
matable$macolonne
```

Pour accéder à une valeur précise, vous pouvez aussi utiliser la notation tab[i,j], où i sont les lignes et j les colonnes. Par exemple si vous voulez voir la valeur de la troisième variable pour la onzième observation, entrez:

```
matable[11, 3]
```

Vous pouvez invoquer vos variables de différentes façons, par exemple pour voir les valeurs des variables "densite_pop" et "nbr_log" pour les 20 premières observations, entrez:

```
matable[1:20, c("densite_pop", "nbr_log")]
```

Pour voir toute la table sauf la première colonne, entrez:

```
matable[,-1]
```

(rien à gauche de la virgule car on veut voir toutes les lignes; " -1 " à droite de la virgule pour exclure la première colonne)

Pour voir toute la table sauf les trois premières colonnes:

```
matable[,-1:-3]
```

Toute la table sauf les colonnes 1 et 5:

```
matable[,c(-1,-5)]
```

et ainsi de suite.

Lorsque vous faites appel à une fonction qui va travailler sur votre table, vous devez en général lui préciser sur quelles colonnes travailler. Faites par exemple

```
cor(matable[,-1:-2])
```

pour calculer les coefficients de corrélation entre toutes les colonnes sauf les deux premières.
N'oubliez jamais cette étape car travailler sur les mauvaises colonnes peut fausser complètement vos résultats.

Gérer les valeurs manquantes – NA - NaN

Dans R, les valeurs manquantes sont représentées par les lettres “NA” (“not available”). Beaucoup de calculs seront impossibles sur des données comportant des valeurs manquantes. Si vous avez des valeurs manquantes, une réflexion s'impose concernant leur raison d'être et le traitement qu'il convient de leur appliquer (exclure les lignes incomplètes? les colonnes incomplètes? remplacer les valeurs manquantes par la moyenne de la ligne? de la colonne? par la moyenne des valeurs encadrantes? etc).

Il y a différents outils qui permettent de traiter les valeurs manquantes:

- la fonction is.na() qui permet de tester la présence de valeurs manquantes
- la plupart des fonctions acceptent un paramètre na.rm=TRUE pour ne pas inclure les NA dans le calcul.
- la fonction complete.cases() sélectionne uniquement les lignes complètes d'une table
- na.omit()
- na.exclude()
- etc.

Exporter des résultats depuis R

Il y a d'innombrables fonctions d'exportation dans R. Ci-dessous quelques exemples.

Exporter un tableau au format html
Pour exporter un tableau pas trop grand avec une mise en forme de base:
Installer et charger le package "xtable". Pour exporter votre table, faire
```
print(xtable(matable), type= "html", file= "matable.html")
```
qui crée un document html dans votre répertoire de travail.

Exporter un tableau au format texte (.csv)
La fonction write.table() est l'équivalent de read.table(). Elle permet d'exporter des tables au format texte et accepte de nombreux arguments.
La fonction `write.csv2()` permet de générer un fichier texte (.csv) dont les séparateurs sont des points-virgules plutôt que des virgules, ce qui permet de ne pas créer de confusion avec les séparateurs de décimales. Par exemple:
```
write.csv2(matable, "nomdufichier.csv", row.names=TRUE)
```

Exporter un tableau au format dbf
Installer et charger le package "foreign". Pour exporter:
```
write.dbf(matable, "nomdufichier.dbf")
```

Exporter un graphique au format jpg
commencer l'exportation en entrant
```
jpeg("nomdufichier.jpg", width=10, height=10, units= "cm", res=300)
```
pour une image de 10 cm de côté par exemple.
Indiquer ensuite ce que ce fichier doit contenir en entrant la fonction qui crée votre graphique, par exemple
```
plot(...)
```
Finir ensuite l'exportation en entrant
```
dev.off()
```

Pour plus d'informations lire le manuel “R Data Import/Export” (disponible sur http://cran.r-project.org/doc/manuals/R-data.pdf).

Faire des graphiques

Les fonctions graphiques dans R sont innombrables.
Le résultat d'une fonction graphique ne peut pas être "mis" dans un objet R; il est envoyé dans un "graphical device" qui est par défaut une fenêtre, mais qui peut aussi être un fichier si vous souhaiter l'exporter.

Extraits de Emmanuel Paradis, "R pour les débutants" (disponible sur http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf):

4.2 Les fonctions graphiques

Voici un aperçu des fonctions graphiques principales de R.

`plot(x)`	graphe des valeurs de `x` (sur l'axe des y) ordonnées sur l'axe des x
`plot(x, y)`	graphe bivarié de `x` (sur l'axe des x) et `y` (sur l'axe des y)
`sunflowerplot(x, y)`	idem que `plot()` mais les points superposés sont dessinés en forme de fleurs dont le nombre de pétales représente le nombre de points
`pie(x)`	graphe en camembert
`boxplot(x)`	graphe boites et moustaches
`stripchart(x)`	graphe des valeurs de `x` sur une ligne (une alternative à `boxplot()` pour des petits échantillons)
`coplot(x~y | z)`	graphe bivarié de `x` et `y` pour chaque valeur (ou intervalle de valeurs) de `z`
`interaction.plot (f1, f2, y)`	si `f1` et `f2` sont des facteurs, graphe des moyennes de `y` (sur l'axe des y) en fonction des valeurs de `f1` (sur l'axe des x) et de `f2` (différentes courbes) ; l'option `fun` permet de choisir la statistique résumée de `y` (par défaut `fun=mean`)
`matplot(x,y)`	graphe bivarié de la $1^{ère}$ colonne de `x` contre la $1^{ère}$ de `y`, la $2^{ème}$ de `x` contre la $2^{ème}$ de `y`, etc.
`dotchart(x)`	si `x` est un tableau de données, dessine un graphe de Cleveland (graphes superposés ligne par ligne et colonne par colonne)
`fourfoldplot(x)`	visualise, avec des quarts de cercles, l'association entre deux variables dichotomiques pour différentes populations (`x` doit être un tableau avec `dim=c(2, 2, k)` ou une matrice avec `dim=c(2, 2)` si $k = 1$)
`assocplot(x)`	graphe de Cohen–Friendly indiquant les déviations de l'hypothèse d'indépendance des lignes et des colonnes dans un tableau de contingence à deux dimensions
`mosaicplot(x)`	graphe en 'mosaïque' des résidus d'une régression log-linéaire sur une table de contingence
`pairs(x)`	si `x` est une matrice ou un tableau de données, dessine tous les graphes bivariés entre les colonnes de `x`
`plot.ts(x)`	si `x` est un objet de classe `"ts"`, graphe de `x` en fonction du temps, `x` peut être multivarié mais les séries doivent avoir les mêmes fréquence et dates

`ts.plot(x)`	idem mais si `x` est multivarié les séries peuvent avoir des dates différentes et doivent avoir la même fréquence
`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 des quantiles de `x`
`contour(x, y, z)`	courbes de niveau (les données sont interpolées pour tracer les courbes), `x` et `y` doivent être des vecteurs et `z` une matrice telle que `dim(z)=c(length(x), length(y))` (`x` et `y` peuvent être omis)
`filled.contour (x, y, z)`	idem mais les aires entre les contours sont colorées, et une légende des couleurs est également dessinée
`image(x, y, z)`	idem mais les données sont représentées avec des couleurs
`persp(x, y, z)`	idem mais en perspective
`stars(x)`	si `x` est une matrice ou un tableau de données, dessine un graphe en segments ou en étoile où chaque ligne de `x` est représentée par une étoile et les colonnes par les longueurs des branches
`symbols(x, y, ...)`	dessine aux coordonnées données par `x` et `y` des symboles (cercles, carrés, rectangles, étoiles, thermomètres ou « box-plots ») dont les tailles, couleurs, etc, sont spécifiées par des arguments supplémentaires
`termplot(mod.obj)`	graphe des effets (partiels) d'un modèle de régression (`mod.obj`)

Pour chaque fonction, les options peuvent être trouvées via l'aide-en-ligne de R. Certaines de ces options sont identiques pour plusieurs fonctions graphiques; voici les principales (avec leurs éventuelles valeurs par défaut) :

`add=FALSE`	si `TRUE` superpose le graphe au graphe existant (s'il y en a un)
`axes=TRUE`	si `FALSE` ne trace pas les axes ni le cadre
`type="p"`	le type de graphe qui sera dessiné, `"p"` : points, `"l"` : lignes, `"b"` : points connectés par des lignes, `"o"` : idem mais les lignes recouvrent les points, `"h"` : lignes verticales, `"s"` : escaliers, les données étant représentées par le sommet des lignes verticales, `"S"` : idem mais les données étant représentées par le bas des lignes verticales
`xlim=, ylim=`	fixe les limites inférieures et supérieures des axes, par exemple avec `xlim=c(1, 10)` ou `xlim=range(x)`
`xlab=, ylab=`	annotations des axes, doivent être des variables de mode caractère
`main=`	titre principal, doit être une variable de mode caractère
`sub=`	sous-titre (écrit dans une police plus petite)

4.3 Les fonctions graphiques secondaires

Il y a dans R un ensemble de fonctions graphiques qui ont une action sur un graphe déjà existant (ces fonctions sont appelées *low-level plotting commands* dans le jargon de R, alors que les fonctions précédentes sont nommées *high-level plotting commands*). Voici les principales :

`points(x, y)`	ajoute des points (l'option `type=` peut être utilisée)
`lines(x, y)`	idem mais avec des lignes
`text(x, y, labels, ...)`	ajoute le texte spécifié par `labels` au coordonnées `(x,y)`; un usage typique sera : `plot(x, y, type="n")` ; `text(x, y, names)`
`mtext(text, side=3, line=0, ...)`	ajoute le texte spécifié par `text` dans la marge spécifiée par `side` (*cf.* `axis()` plus bas); `line` spécifie la ligne à partir du cadre de traçage
`segments(x0, y0, x1, y1)`	trace des lignes des points `(x0,y0)` aux points `(x1,y1)`
`arrows(x0, y0, x1, y1, angle=30, code=2)`	idem avec des flèches aux points `(x0,y0)` si `code=2`, aux points `(x1,y1)` si `code=1`, ou aux deux si `code=3`; `angle` contrôle l'angle de la pointe par rapport à l'axe
`abline(a,b)`	trace une ligne de pente `b` et ordonnée à l'origine `a`
`abline(h=y)`	trace une ligne horizontale sur l'ordonnée `y`
`abline(v=x)`	trace une ligne verticale sur l'abcisse `x`
`abline(lm.obj)`	trace la droite de régression donnée par `lm.obj` (*cf.* section 5)
`rect(x1, y1, x2, y2)`	trace un rectangle délimité à gauche par `x1`, à droite par `x2`, en bas par `y1` et en haut par `y2`
`polygon(x, y)`	trace un polygone reliant les points dont les coordonnées sont données par `x` et `y`
`legend(x, y, legend)`	ajoute la légende au point de coordonnées `(x,y)` avec les symboles donnés par `legend`
`title()`	ajoute un titre et optionnellement un sous-titre
`axis(side, vect)`	ajoute un axe en bas (`side=1`), à gauche (2), en haut (3) ou à droite (4); `vect` (optionnel) indique les abcisses (ou ordonnées) où les graduations seront tracées
`box()`	ajoute un cadre autour du graphe
`rug(x)`	dessine les données `x` sur l'axe des x sous forme de petits traits verticaux
`locator(n, type="n", ...)`	retourne les coordonnées (x, y) après que l'utilisateur ait cliqué `n` fois sur le graphe avec la souris; également trace des symboles (`type="p"`) ou des lignes (`type="l"`) en fonction de paramètres graphiques optionnels (...); par défaut ne trace rien (`type="n"`)

Pour plus d'informations lire par exemple les pages 38 et suivantes d'Emmanuel Paradis, "R pour les débutants" (disponible sur http://cran.univ-lyon1.fr/doc/contrib/Paradis-rdebuts_fr.pdf).

Ressources sur R

Les livres et manuels renseignés ci-dessous seront tenus à disposition des étudiants lors des séances de travail.

Livres

- Crawley, M. (2007), The R Book, ed. Wiley, 942 p.
- Muenchen, R. (2009), R for SAS and SPSS Users, ed. Springer, 466 p. (version résumée en 80 pages disponible également).
- Husson, F., Le, S. et Pages, J. (2009), Analyse de données avec R, ed. PU Rennes.
- Milot, G. (2009), Comprendre et réaliser des tests statistiques à l'aide de R, ed. De Boeck Université.
- Bivand, R., Pebesma, E. et Gomez-Rubio, V. (2008), Applied Spatial Data Analysis with R, ed. Springer. Full text disponible sur http://www.springerlink.com/content/978-0-387-78170-9 sur les ordinateurs de l'ULB.
- Cornillon, P.A., Guyader, A., Husson, F., Jégou, N., Josse, J., Kloareg, M., Matzner-Lober, E. et Rouvière, L. (2009), Statistiques avec R, PU Rennes.

Manuels en ligne

Plusieurs manuels dans toutes sortes de langues sont disponibles sur http://cran.r-project.org/other-docs.html.

On consultera en particulier:

- Barnier, J. (2010), R pour les sociologues (et assimilés), disponible sur http://cran.r-project.org/doc/contrib/Barnier-intro_R.pdf
- Paradis, E. (2005), R pour les débutants, disponible sur http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf
- Kaufmann, M. (2009), Aide-mémoire R, disponible sur http://cran.r-project.org/doc/contrib/Kauffmann_aide_memoire_R.pdf, qui liste en 4 pages les fonctions les plus courantes et la façon de les utiliser.

Sites web

- http://www.statmethods.net/
- http://r4stats.com/
- quelques tutoriels utiles se trouvent sur http://www2.agrocampus-ouest.fr/math/livreR/
- etc.