TP1: Introduction à R

Objectif

L'objectif de cette séance est une première prise en main du logiciel R et de présenter un certain nombre de commandes de base.

1 Introduction à R

Cette section introduit le logiciel libre R et décrit les premières commandes nécessaires au démarrage d'une utilisation de méthodes statistiques avec ce logiciel. D'autres tutoriels disponibles sur le site wikistat.fr abordent des fonctionnalités plus complexes.

- Démarrer rapidement avec R
- Initiation à R
- Fonctions graphiques de R
- Programmation en R
- MapReduce pour le statisticien

Les aspects statistiques sont développés dans les différents scénarios de Wikistat et d'autres ressources sont disponibles.

1.1 Pourquoi R ?

Le logiciel R sous licence GNU est facile à installer à partir de la page du CRAN ou d'un site miroir ; ils contiennent toutes les ressources nécessaires à l'utilisateur de R, débutant ou expérimenté : fichiers d'installation, mises à jour, librairies, FAQ, newsletter, documentation... R fait partie des logiciels les plus utilisés par la communauté statistique académique et aussi de plus en plus dans les services R&D des entreprises, en concurrence avec les logiciels commerciaux. Son utilisation nécessite un apprentissage à travers des tutoriels comme par exemple ceux listés dans le résumé mais il est facile de démarrer à partir de quelques notions de base sur son utilisation ; c'est l'objectif de ce start-R.

Dans sa structure, R est un langage de programmation d'une syntaxe voisine à celle du langage C et capable de manipuler des objets complexes sous forme de matrice, scalaire, vecteur, liste, facteur et aussi *data frame*. Proposant donc une programmation matricielle, il offre des fonctionnalités analogues à Matlab, ou à Python, et dispose également d'une très riche librairie de quasiment toutes les procédures et méthodes statistiques de la littérature. Plus précisément, de très nombreuses recherches récentes en Statistique sont d'abord développées et diffusées à l'aide de ce logiciel par la communauté scientifique, sous forme de packages.

1.2 Premiers pas avec R

Nous allons utiliser RStudio qui est un environnement de programmation multiplateforme pour R.

1.2.1 Ouvrir, fermer RStudio

- Sous Windows, cliquer sur l'icône `RStudio` ou lancer le programme à partir du menu `Démarrer`. Lors de la première exécution, il est utile de préciser le répertoire de travail (menu `Session` de RStudio). Il est aussi possible de lancer R à partir de la session précédente en cliquant sur un fichier de sauvegarde (`.RData`).
- Sous Linux ; à partir d'un terminal, se placer dans le bon répertoire pour taper la commande `rstudio`. Il est aussi possible de lancer RStudio à partir du menu déroulant.

Enfin, quelque soit le système d'exploitation, il est également possible d'ouvrir RStudio en cliquant sur un script (extension `.R`) ou un document R Markdown (extension `.Rmd`) introduits ci-dessous.

Ensuite, à chaque ouverture de RStudio, le logiciel ouvre une fenêtre avec 4 sous-fenêtres :

- en haut à gauche : un éditeur de texte pour gérer le script à exécuter, sauvegarder,
- en haut à droite : une liste des objets (environnement) créés dans R ou l'historique des commandes,
- en bas à gauche : la console d'exécution classique de R,
- en bas à droite : une fenêtre avec menu pour visualiser le répertoire, les graphiques, la liste des *packages* installés pour les charger, mettre à jour, en installer d'autres, l'aide en ligne.

La fonction `quit()` fait quitter RStudio après une question proposant de sauvegarder l'environnement de travail. Y répondre favorablement crée le fichier de sauvegarde `.RData` dans le répertoire courant.

1.2.2 Gestion des packages

À la première ouverture de `RStudio`, un dossier `R/libs` est créé dans votre `home` pour l'installation des librairies.

Vous pouvez directement gérer l'installation, et la mise à jour des packages à partir de l'onglet "Package" accessible dans la sous-fenêtre en bas à droite (ou dans l'onglet "Tools"). Vous pouvez aussi les gérer depuis la console grâce aux commandes suivantes

```
> install.packages(...) # pour installer un package
> update.packages(...) # pour mettre à jour un package
> library(...)  # pour charger un package déjà installé
```

Exercice : Installez (après avoir vérifié qu'ils ne le sont pas déjà) les packages suivants, utiles dans les prochains TP : `vcd, BioStatR, corrplot, shape`.
Pour la prochaine séance, installez le package `FactoMineR`[1].

1.2.3 Gérer un script

Un nouveau script peut être créé et enregistré (ex. `monScript.R`) dans le répertoire courant, depuis le menu

File –> New file –> R Script

Vous pouvez alors l'ouvrir grâce au menu **File –> Open file...** ou en double cliquant sur le fichier `monScript.R`. Il est également possible d'exécuter un script directement depuis la console (en bas à gauche dans RStudio) grâce à la commande `source()`, en ayant vérifié au préalable que vous êtes bien dans le bon répertoire.

```
> getwd()    # pour connaitre le répertoire courant
> setwd(".../monDossier") # pour se placer dans monDossier
> source("monScript.R")    # pour exécuter le script R
```

1.2.4 Éditer des rapports avec R Markdown

Vous pouvez également créer directement un rapport grâce à R Markdown, avec la possibilité d'éditer directement des documents au format pdf[2] ou html (en incluant les commandes R, les sorties et les graphes).

Pour cela, il faut créer un nouveau document (d'extension `.Rmd`) :

File –> New file –> R Markdown...

Il faut alors préciser le type de document final souhaité (document au format html ou pdf ou presentation, etc), ainsi que les informations du document (titre et auteur). N'oubliez pas d'enregistrer votre document. Pour l'éditer, vous pouvez taper du texte (dont des commandes LaTeX), et insérer du code R à l'intérieur de blocs délimités par ```` ```{r} ```` et ```` ``` ````, appelés `R chunck` ou *balises R*. Il ne reste plus qu'à compiler le document en cliquant sur le bouton `Knit`. Le fichier de sortie s'affiche alors.

Remarquons qu'il est possible de donner un nom à chaque bloc (pour les retrouver plus facilement), et d'ajouter des options d'affichage pour chaque bloc de code R en mettant par exemple ```` ```{r nom, eval=FALSE} ````. Voici les options les plus courantes :

- `eval` (`TRUE` par défaut, ou `FALSE`) : détermine si le code R doit être évalué ou non.

1. L'installation de `FactoMineR` prend du temps, donc merci de ne pas l'installer pendant ce TP afin de ne pas rester bloqués.

2. D'ailleurs, la partie Statistique Descriptive du cours sera évaluée par projet. Il vous sera demandé de rendre votre rapport au format pdf compilé à l'aide de RMarkdown.

- `echo` (`TRUE` par défaut, ou `FALSE`) : détermine si le code R doit être affiché ou non.
- `results` ('`markup`' par défaut, ou '`hide`' ou '`hold`' ou '`asis`') : détermine comment les sorties doivent être affichées.
- `error` (`FALSE` par défaut, ou `TRUE`) : détermine si les messages d'erreur doivent être affichés.
- `warning` (`TRUE` par défaut, ou `FALSE`) : détermine si les messages d'avertissement doivent être affichés.

Plus d'informations peuvent être trouvées sur le site internet http://rmarkdown.rstudio.com/ ou ici [3].

Exercice : Créez un nouveau document R Markdown. Enregistrez le document pré-rempli que Rstudio vous propose et compilez-le au format pdf. Faites le lien entre le code source (`.Rmd`) et le rapport édité. À quoi correspondent les "#" hors des balises R (essayez avec #, ## ou ### et comparez) ? Modifiez les options d'affichage ci-dessus et comparez les sorties.

De manière générale, il est recommandé de

- travailler dans le document R Markdown (en lançant balise R par balise R, avec la petite flèche verte en haut à droite de chaque bloc, ou en lançant la ligne de code contenant le curseur en tapant `Ctrl Enter`),
- commenter les résultats obtenus (en dehors des balises R) au fur et à mesure,
- ne compiler le document (`Knit`) qu'à la fin.

1.2.5 L'aide du logiciel

Il ne vous sera pas demandé de connaître par cœur toutes les commandes de R. Cependant, il est très important de savoir utiliser l'aide pour les retrouver. Cela vous permettra d'acquérir une réelle autonomie. On peut accéder à l'aide sur un objet (par exemple `plot`) des manières suivantes :

```
> help.start()
> help(plot)
> ?plot
```

3. https ://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

2 Commandes de base de R

2.1 Une calculatrice

R fonctionne comme une calculette. Vous pouvez taper les lignes ci-dessous après l'invite de commande (>). Afin de garder une trace de votre travail vous pouvez enregistrer vos commandes dans un Script R, ou éditer un document R Markdown.

```
> # Ceci est un commentaire
> 2+2*(5-1)
> 3^2
> pi
```

Pour affecter une valeur à un nom, on peut utiliser soit "=", soit "<-" :

```
> x <- 4
> y = x+2 ; y
> ls()   # affiche la liste des objets crées et installés
> rm(list=ls()) # efface l'environnement courant
> y
```

Il est possible d'appliquer des fonctions usuelles :

```
> sqrt(4)
> exp(log(2))
> log(8,base=2)
> cos(pi/3)
> abs(-3.5)
> round(2.75) ; ceiling(3.00001) ; floor(3.99999)
```

R manipule les opérateurs logiques ;

- les inégalités strictes : "<" ou ">",
- les inégalités larges : "<=" ou ">=",
- l'égalité "==" (attention à ne pas confondre avec "=" pour l'assignation),
- la différence "!=".

```
> 2>=3
> 10/2 == 5; 6/2 != 2
> F
```

Pour afficher une chaîne de caractères, on emploie des guillemets (ou des apostrophes) :

```
> "x"
> 'x+y'
```

2.2 Types de variables

La principale difficulté dans l'utilisation de R est de bien identifier les types d'objets manipulés.

- Vecteur

```
> a = c(1,2,4) ; a
> b = c(a,8) ; b

> x = 1:10 # définition d'une séquence

> y = 2*x ; y
> length(y)                # taille d'un vecteur
> y[5] ; y[1:3] ; y[-3]    # composants d'un vecteur
> which(y >=3)             # indices pour lesquels...
> y[y<=5]                  # valeurs de y vérifiant
> is.vector(y)

> seq(1,10,by=.5) ; seq(1,10,length.out=5)
> rep(c(1,2),times=3) ; rep(c(1,2),each=3)
```

- Matrice

```
> A = matrix(1:15,ncol=5); A
> B = matrix(1:15,ncol=5,byrow=TRUE) ; B
> cbind(B,(1:nrow(B)))
> rbind(B,(1:ncol(B)))

> dim(A)
> rownames(A) = c("ligne1","ligne2","ligne3") ; A
> A[1,3] ; A[,2] ; A[2,] ; A[1:3,1:3] # composants
> diag(A) ; diag(4) ; diag(1:4)
> t(A)             # transposée d'une matrice
> A*A              # produit terme à terme
> t(A)%*%c(1,2,3) # produit matriciel
> class(A)
```

- Liste (permet de stocker des variables de différents types)

```
> L=list(mat=A, texte="test liste", vec=y)
> L[[2]] ; L$vec # composants
> class(L)
```

- Base de donnée (ou *data frame*) : ce sont des tableaux contenant des vecteurs de types éventuellement différents (mais de même taille)

```
> t = c(147, 132, 156, 167, 156, 140)
> p = c( 50, 46, 47, 62, 58, 45)
> yx = factor(c(1,2,1,3,2,1),
+             labels=c("marron","bleu","vert"))
> s = factor(c("M","F","F","M","M","F"))
> labels(yx) ; levels(yx)
> H = data.frame(taille=t,poids=p,sexe=s,yeux=yx) ; H
> H$taille
> class(H)
```

  Lorsque vous avez de grands jeux de données avec beaucoup de lignes, vous pouvez utiliser les commandes `head` (resp. `tail`) afin d'afficher uniquement les premières (resp. les dernières) lignes du tableau.

D'autres fonctionnalités de R sont vues dans les différents scénarios proposés sur le site Wikistat.

2.3 Quelques paramètres graphiques

La fonction principale pour tracer un graphe est `plot()`. Elle possède de nombreuses options dont certaines sont listées ci-dessous.

- `type` : points (`"p"`), lignes (`"l"`), escalier (`"s"`), bâtons (`"h"`), etc,
- `lwd` : épaisseur du trait,
- `lty` : type de trait (continu, pointillé, etc),
- `cex` : taille des points,
- `pch` : forme des points,
- `col` : couleur (`""red"`, `"blue"`, `"rainbow(...)"`),
- `main` et `sub` : titre et sous-titre du graphe,
- `xlab` et `ylab` : titres des axes,
- `xlim` et `ylim` : limite des axes.

Les fonctions `lines`, `abline` et `points` permettent d'ajouter des lignes ou des points à un graphe déjà existant. On peut ajouter du texte ou une légende grâce aux commandes `text` et `legend`. Il est également possible de tracer plusieurs graphes sur `n` lignes et `p` colonnes grâce à la commande `par(mfrow=c(n,p))`.

Exercice : Tracez sur un graphe intitulé "Fonction sinus", la courbe de la fonction sinus sur $[0, 2\pi]$. Représentez en rouge sur le même graphe les extrema de la fonction. Tracez en pointillés sur le graphe les droites d'équation $y = 1$ et $y = -1$.

2.4 Fonctions

Il est possible de créer des fonctions dans R (de plusieurs variables), en utilisant la syntaxe suivante :

```
> mafonction <- function(x,a,b)
+ {
+   y=a*x+b
+   return(y)
+ }
> mafonction(1:10,a=2,b=3)
```

Il est possible de fixer des paramètres par défaut (par exemple en remplaçant la première ligne ci-dessus par :
`mafonction <- function(x,a=2,b=0){...`

On peut donc appeler la fonction sans préciser les valeurs des paramètres fixés, tout en gardant la possibilité de les modifier :

```
> mafonction(1:10)
> mafonction(1:10,2,3)
```

Exercice : Écrivez une fonction `ma.var` permettant de calculer la variance (empirique, non corrigée) d'une série statistique. Quelle est la variance de la variable "taille" dans la base de données `H` créée ci-dessus ? Quelle est la variance du vecteur `rep(5,times=18)` ?

3 R et Statistique

3.1 Simulation de variables aléatoires

Le logiciel R permet d'effectuer des calculs avec toutes les lois de probabilité usuelles, et aussi de simuler des échantillons issus de ces lois. Par exemple, pour la loi normale, on obtient, grâce aux commandes :

- `dnorm` : fonction de densité,
- `pnorm` : fonction de répartition,
- `qnorm` : fonction quantile,
- `rnorm` : pour simuler des variables aléatoires.

On peut remplacer "`norm`" dans les commandes ci-dessus par les éléments de la colonne "`law`" pour obtenir n'importe quelle distribution du tableau ci-dessous.

Loi	`law`
Gaussienne	`norm`
Binomiale	`binom`
Poisson	`pois`
Uniforme continue	`unif`
Exponentielle	`exp`
Student	`t`
Khi-deux	`chisq`
Fisher	`f`

Il est également possible de faire des tirages avec ou sans remise grâce à la commande `sample`.

Exercice : En vous aidant de l'aide du logiciel,

i. Tracez la densité d'une variable aléatoire gaussienne de moyenne 10 et de variance 4 sur l'intervalle $[0, 20]$.
ii. Tracez, sur un autre graphe, sa fonction de répartition.
iii. Simulez un échantillon de `n` variables aléatoires gaussiennes de moyenne 10 et de variance 4, pour `n` valant `2, 5, 10, 100, 1000`. Pour chaque valeur de `n`, calculez la moyenne empirique et la variance empirique (grâce à votre fonction `ma.var`) de cet échantillon. Que remarquez-vous ?

3.2 Importation/Exportation de données

Vous pouvez importer des données sous forme de `data.frame`, soit par l'onglet "Import Dataset" dans la sous-fenêtre en haut à droite, soit directement dans la console grâce aux commandes `read.table()`, `read.csv()` ou `read.csv2()` selon le format du fichier de données.

Il est également possible d'enregistrer un jeu de données (simulés par exemple) grâce à la commande `write.table()`.

Exercice : Importez les données `Afrique` (disponibles sur Moodle) depuis la console. Extrayez-en la première ligne. Que remarquez-vous ? Que se passe-t-il si vous rajoutez l'option `header=TRUE` ?

Il existe également des jeux de données directement dans R :

```
> data()
> data(cars)
```

Que contient ce jeu de données ? (vous pourrez vous aider de la commande `help`).

Exercice : Chargez la librairie `MASS` et enregistrez le jeu de données `UScrime` dans un fichier **UScrime.txt**. Que contient ce jeu de données ?

3.3 Étude descriptive du jeu de données...

S'il vous reste du temps, faites l'étude descriptive (univariée et bivariée) du jeu de données "Afrique" en vous aidant du cours.