Statistiques avec le logiciel R - TD1

18 septembre

Objectifs

— Découvrir le logiciel **R**
— Organiser son travail en **R**
— Savoir utiliser l'aide de **R**
— Connaître plusieurs types de données en **R**
— Utiliser des fonctions

1 Première prise en main de R

1.1 Introduction

Le logiciel **R** est à la fois un langage de programmation et un environnement de travail qui permet d'exécuter des commandes pour la réalisation d'analyses statistiques, et la production de graphiques. R a été créé par Ross Ihaka et Robert Gentleman à l'université d'Auckland (Nouvelle-Zélande). Il est actuellement développé par le R Development Core Team, qui comporte 20 membres. De plus, ce logiciel est complété par des centaines de packages, écrits par des centaines de contributeurs. C'est un logiciel collaboratif.

R est un logiciel gratuit et à code source ouvert (open source) sous licence GNU. Il est multi-plateforme et fonctionne sous Windows, Macintosh et Linux. Vous pourrez télécharger le logiciel depuis un site miroir du CRAN (The Comprehensive R Archive Network) par exemple `http://cran.univ-lyon1.fr/`.

Plus d'informations sont disponibles sur le site du logiciel `http://www.r-project.org/`

1.2 Démarrer R

1.2.1 Sous unbuntu

Sous ubuntu ou à partir d'une console Linux, dans un terminal, démarrer **R** en tapant `R`. La console de commande R (R Console) s'affiche avec le message d'introduction suivant :

```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.
```

On note que ce message indique la version utilisée, ici, 3.1.1, et comment obtenir de l'aide. Au-dessous du message, on remarque le caractère d'invite de commande '>', à la suite de ce symbole vous pourrez écrire vos instructions.

1.2.2 Sous windows

Dans le menu programme, cherchez le programme **R** et cliquez. Une fenêtre apparait avec un message similaire à la version Linux, vous observez également à la fin du message l'invite de commande '>'. Sur le bureau, l'icône **R** est la suivante :  .

1.3 Utiliser R comme une calculatrice

Dans un premier temps on peut utiliser **R** de la même façon qu'une calculatrice. On utilise le symbole `'+'` pour la somme, `'-'` pour la soustraction, `'*'` pour le produit et `'/'` pour la division. Voici quelques exemples.

```
> 5+2
[1] 7
> 6-5
[1] 1
> 5*6.7
[1] 33.5
> 32/4
[1] 8
```

R permet aussi de faire des calculs sur des vecteurs. On définit un vecteur avec la commande `c()`, par exemple

```
> c(1,2,3,4)
[1] 1 2 3 4
> c(0,0.01,-2.5)
[1]  0.00  0.01 -2.50
```

Les opérations se font terme à terme.

```
> c(1,2,3,4)*2
[1] 2 4 6 8
> c(0,0.01,-2.5)+5
[1] 5.00 5.01 2.50
```

1.4 Affectation des résultats et redirection dans des variables

`R` affiche le résultat de la requête dans la console. Ce résultat est affiché, puis perdu, on ne peut donc pas le réutiliser dans une autre requête. Pour pouvoir réutiliser le résultat d'une requête, on va le stocker dans une variable, cette opération s'appelle l'affectation du résultat dans une variable. Une affectation évalue une expression, mais n'affiche pas le résultat, celui-ci est stocké dans une variable. Pour afficher le résultat, il faut taper le nom de la variable et appuyer sur `ENTREE`.

Pour réaliser l'opération d'affectation, on utilise la flèche d'affectation `'<-'`, ou le signe `'='`. Pour créer un objet, on procédera toujours de la même façon : `Nom.de.l.objet = instructions`

Par exemple :

```
> x=1
> x
[1] 1
```

Choix du nom de variable : un nom de variable n'est constitué que de caractères alphanumériques et du point. **R** fait la distinction entre majuscules et minuscules, de plus un nom de variables ne peut pas commencer par un chiffre. Il est conseillé d'utiliser des noms de variables explicites, pour mieux se souvenir de ce que contient la variable.

1.5 Organisation du travail

Lorsqu'une instruction est écrite dans la console **R**, celle-ci est évaluée et son résultat est affiché. Cependant l'instruction n'est pas enregistrée, il est donc vivement conseillé d'écrire les instructions dans un éditeur et de les envoyer ensuite dans la console **R**. Le fichier d'instructions, nommé *script*, pourra

être enregistré au format `.R` ou texte `.txt`. Les instructions pourront ensuite être copiées-collées dans la console **R**.

Dans une utilisation classique de **R** en statistiques, vous serez amené à travailler sur plusieurs fichiers. Par exemple, lors des prochains TD, nous verrons comment lire une table de données, ou comment exporter des sorties graphiques ou des tableaux numériques. Il est donc judicieux de créer un répertoire dédié au projet, dans lequel seront enregistrés :

— le script **R**
— les tableaux de données, ou autres fichiers sources
— les sorties graphiques.

1.6 L'environnement RStudio

L'environnement de développement pour **R**, RStudio, fournit une interface graphique attractive pour la programmation **R**. Il fournit notamment une console **R**, un éditeur de code et une fenêtre graphique. De plus, des boutons permettent d'envoyer directement une sélection dans la console **R**. On prendra l'habitude de travailler avec RStudio.

Pour démarrer RStudio, quittez **R**. Si vous êtes sous Ubuntu, taper `q()` dans la console. Puis démarrez RStudio, dans **Démarrer**, **Programme**, cliquez sur **Rstudio**.

L'écran est divisé en quatre fenêtres, avec en haut à gauche un éditeur de code, dans lequel on écrira les instructions, en bas à gauche la console **R**, en haut à droite une fenêtre où apparaitra l'ensemble des variables définies dans la session, et en bas à droite une fenêtre graphique ou d'affichage de l'aide. On remarque que le code est écrit dans le fichier `Exemple1.R`. Dans le menu fichier, on pourra enregistrer, ouvrir ou créer un nouveau script. RStudio fournit aussi des outils pour faciliter le développement de code **R**, comme `run` pour envoyer une ligne de commande ou une sélection de l'édition vers la console ou `source` pour envoyer l'ensemble du script, dans ce cas toutes les lignes seront exécutées une par une. Le raccourcis clavier de la touche `run` est **Ctrl+Entree**.

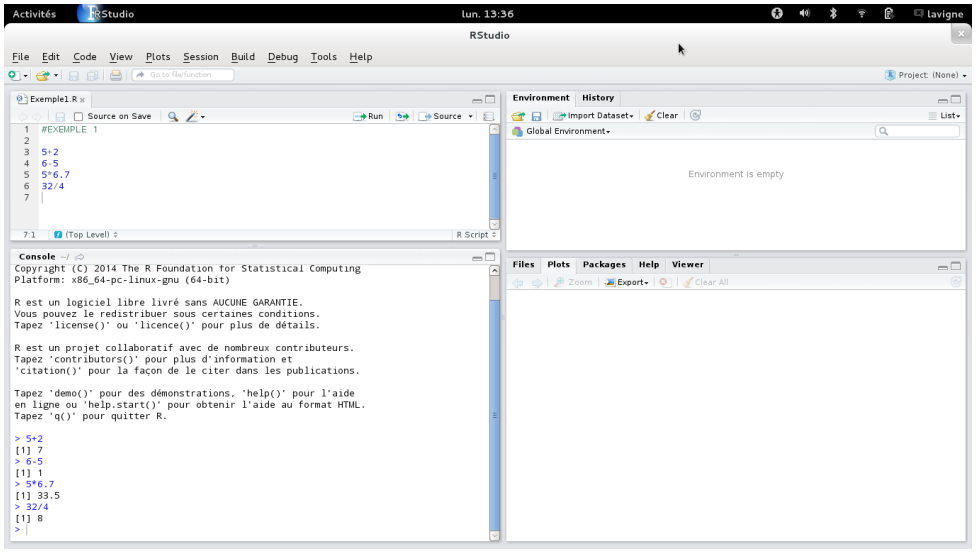

FIGURE 1 – Environnement RStudio

1.7 L'environnement de travail R

L'environnement de travail regroupe l'ensemble des variables et objets **R** définis pendant une session **R**. On ne peut pas "voir" cet environnement, mais il existe bien. L'instruction `ls()` affiche la liste de tous les objets présents dans l'environnement. Dans RStudio, la fenêtre en haut à droite liste aussi tous les objets de l'environnement. On peut enregistrer cet environnement en cliquant sur l'icône "disquette" de l'onglet "Environment", ou en utilisant directement la fonction `save`. De même, en cliquant sur l'onglet "fichier" ou avec la fonction `load`, on peut charger un environnement existant. L'instruction `rm()` permet

de supprimer certains objets de l'environnement, et l'instruction `rm(list=ls())` permet de supprimer tous les objets de l'environnement.

```
> ls()
[1] "x"
> rm(list=ls())
> ls()
character(0)
```

```
> a=3
> b=5
> d=9
> e=a+b
> ls()
[1] "a" "b" "d" "e"
> rm(e)
> ls()
[1] "a" "b" "d"
```

2 Organisation des objets de R

2.1 Types de données

Comme la plupart des logiciels de programmation, **R** dispose des types de données classiques. Il existe six types de données dans **R**, dans ce TD nous ne verrons que les quatre types que nous utiliserons lors des prochaines séances. Lorsqu'une variable est créée, **R** lui affecte automatiquement un type de donnée. On peut avoir accès à ce type avec la fonction `typeof`, de plus de nombreuses conversions entre types sont possibles avec les fonctions commençant par `as.`.

2.1.1 Le type numérique (`numeric`)

Il y a deux types numériques, le type entier `integer` et le type réel `double`.

```
> a=3
> b=a*4.1
> typeof(a)
[1] "double"
> typeof(b)
[1] "double"
```

Par défaut, `a` est un réel, mais il peut être converti en entier grâce à la fonction `as.integer`.

```
> int.a=as.integer(a)
> typeof(int.a)
[1] "integer"
```

2.1.2 Le type booléen ou logique (`logical`)

Le type logique est le résultat d'une assertion logique et ne peut prendre que les valeurs `TRUE` ou `FALSE`. Une assertion peut-être créée par les opérateurs de la Table **??**. Voici comment créer une variable logique.

```
> a==3
[1] TRUE
> b<=2
[1] FALSE
> is.numeric(a)
[1] TRUE
```

On peut aussi directement définir une variable logique à l'aide des lettres capitales `T` pour *true* et `F` pour *false*.

```
> A=T
> B=F
> (A==B)==F
[1] TRUE
```

Opérateur en **R**	Description
`y==x`	Est-ce que $x_i = y_i$?
`y!=x`	Est-ce que $x_i \neq y_i$?
`x>y`	Est-ce que $x_i > y_i$?
`x>=y`	Est-ce que $x_i \geq y_i$?
`x<y`	Est-ce que $x_i < y_i$?
`x<=y`	Est-ce que $x_i \leq y_i$?
`|`	OU
`&`	ET

TABLE 1 – Opérateurs logiques, `x` et `y` sont des vecteurs.

2.1.3 Le type chaîne de caractères (`character`)

Toute suite de caractères mise entre guillemets simples `''` ou doubles `""` est une chaîne de caractères.

```
> c='bonjour'
> typeof(c)
[1] "character"
```

Les conversions des types numériques ou logiques vers le type chaîne de caractères sont possibles grâce à la fonction `as.character`.

```
> as.character(b) #conversion du type numérique
[1] "12.3"
> as.character(A) #conversion du type logique
[1] "TRUE"
```

2.1.4 Le type données manquantes (`NA`)

Une donnée manquante est indiquée par l'instruction `NA` pour *non available* c'est-à-dire non disponible. `NA` est un type logique constant indiquant si la valeur est manquante ou non. Généralement, tout calcul numérique impliquant des données manquantes a pour résultat une donnée manquante. Cependant, on peut souvent enlever les données manquantes avec l'option `na.rm=T`. La fonction `is.na` renvoie un logique indiquant si la donnée `A` est manquante ou non.

```
> d=c(2.1,NA,3.6, 7.4,3.2, NA)
> is.na(d)
[1] FALSE  TRUE FALSE FALSE FALSE  TRUE
> sum(d)
[1] NA
> sum(d,na.rm=TRUE)
[1] 16.3
```

3 Les fonctions

Nous avons jusqu'ici utilisé quelques fonctions de **R**, comme `sum`, `is.numeric`, `as.character`, mais il existe des centaines de fonctions dans la version de base de **R** et des milliers d'autres si on considère ses extensions (*packages*). La tableau Tab. 2 vous en donne quelques exemples classiques. Enfin il est aisé de définir soi-même sa propre fonction.

Une fonction est définie par son nom et la liste de ses arguments. On fait appel à une fonction en tapant le nom de la fonction, suivi entre parenthèses par la liste de ses arguments, `NomDeLaFonction(Arg1=ValeurDeArg1, Arg2=ValeurDe2, Par3=ValeurDuPar3)`.

Exemple 1

```
> sum(c(1.2,9,NA,5,-10),na.rm=TRUE)
[1] 5.2
```

Exemple 2

```
> sum(c(1.2,9,NA,5,-10))
[1] NA
```

Dans cet exemple, la fonction `sum` prend deux arguments, le premier est le vecteur dont on veut sommer les éléments, le second indique comment traiter les données manquantes. On remarque que le premier argument n'a pas de nom, il est obligatoire et doit se placer en premier. Dans l'exemple 1, l'argument `na.rm=TRUE` indique qu'il ne faut pas tenir compte des données manquantes. Dans l'exemple 2, on n'a pas précisé la valeur de l'argument `na.rm`, c'est donc la valeur par défaut qui est utilisé, ici il s'agit d'un `TRUE`, ce qui signifie qu'il faut tenir compte de la valeur manquante, et donc la somme n'est pas définie, **R** renvoie `NA`.

Pour savoir ce que fait une fonction et comment l'utiliser, il **faut** se référer à son aide. **Toutes** les fonctions sont documentées sur **R**, pour obtenir l'aide relative à une fonction il suffit d'écrire son nom précédé du point d'interrogation dans la console. Par exemple en tapant `?sum` dans la console, vous observez l'apparition de la fenêtre d'aide, en bas à droite de l'écran dans RStudio. Dans la section *usage*, la fonction ainsi que la liste de ses arguments sont décrits, la valeur par défaut de chaque argument est écrite à la suite du signe `=`.

Vous pouvez créer de nouvelles fonctions qui ne seront actives que dans l'environnement de travail avec la fonction `function`. On l'utilise de la manière suivante

```
NomDeLaFonction = function(Arg1,Arg2) {

instructions

return(resultat)

}
```

```
> MaSomme = function(a,b){
+ res=a+b
+ return(res)
+ }
> MaSomme(1,5)
[1] 6
```

Nom de la fonction	Définition
`log`	logarithme (par défaut logarithme Népérien)
`exp`	exponentiel
`sqrt`	racine carrée
`sum`	somme des éléments
`sin`	fonction sinus
`floor`	partie entière

TABLE 2 – Exemples de fonctions classiques

4 Exercices

Exercice 1

1. Créer deux variables **a** et **b** prenant respectivement les valeurs 1 et 0.
2. Créer une nouvelle variable **c** somme de `a` et`b`. Quel est le type de `c` ?
3. Créer une nouvelle variable **d** division de `a` par `b`. Quel est le type de `d` ? Cela vous parait-il normal ?
4. Quelles sont les variables de l'environnement de travail ?

Exercice 2

1. Que produit la fonction `paste` ?
2. Créer deux variables prenant pour valeur les chaînes de caractères `"vendredi"` et `"septembre"`.
3. Avec la fonction paste, produire une nouvelle variable contant la date d'aujourd'hui.
4. Que spécifie l'argument `sep` de la fonction paste ?
5. Créer une fonction qui retourne la date, et prend en entrée le jour de la semaine, le jour du mois et le mois.

Exercice 3

1. Que produit la fonction `nchar` ?
2. Créer une nouvelle variable contenant l'Article premier de la constitution universelle des droits de l'homme. Quel est le type de cette variable ? Vérifiez-le ?
3. A l'aide de la fonction `nchar` compter le nombre de caractères dans cet article.
4. Donner l'instruction indiquant si le nombre de caractères est
   — supérieur à 100,
   — supérieur strictement à 100,
   — compris entre 20 et 100
   — inférieur à 30 ou supérieur à 70.
5. Répondre sans utiliser **R**. Que renvoie

   ```
   (((T==F)|(T==T))&((F==F)==T))==F
   ```

   . Vérifiez à l'aide de **R**.

Exercice 3

1. Charger l'environnement `Surprise.RData`.
2. Que contient cet environnement ? Détaillez-le avec précision.