Introduction à R

Chapitre 1: Les concepts de base, l'organisation des données

Une première session

R est une calculatrice

R remplace aisément les fonctionnalités d'une calculatrice. Il permet aussi de faire des calculs sur des vecteurs. Voici quelques exemples très simples.

```
> 5*(-3.2) # Attention, le separateur décimal doit # être un point (.)
```

```
## [1] -16
```

```
> 5*(-3,2) # sinon, l'erreur suivante est generée:
```

```
## Error: <text>:1:6: unexpected ','
## 1: 5*(-3,
##          ^
```

```
> 5^2 # Identique a 5**2
```

```
## [1] 25
```

```
> sin(2*pi/3)
```

```
## [1] 0.8660254
```

```
> sqrt (4) # Racine carree de 4.
```

```
## [1] 2
```

```
> log(1) # Logarithme neperien de 1.
```

```
## [1] 0
```

```
> c(1,2,3,4,5) # Crée un vecteur contenant les cinq premiers entiers
```

```
## [1] 1 2 3 4 5
```

```
> c(1,2,3,4,5)*2 # Calcul des cinq premiers nombres pairs
```

```
## [1]  2  4  6  8 10
```

Tout code **R** qui suit le caractere «#» est considéré par **R** comme un commentaire. En fait, il n'est pas interprété par **R**.

Pour quitter le logiciel R, il faut taper la commande `q()`.

Il vous est ensuite proposé de sauver une image de la session. En repondant oui, les commandes tapées précédemment seront de nouveau accessibles lors d'une prochaine réouverture de **R**, au moyen des flèches «haut» et «bas» du clavier.

Stratégie de travail

- Prenez l'habitude de stocker vos fichiers dans un dossier réservé à cet usage (nomme par exemple `TravauxR`). En outre, il est conseillé de taper toutes ses instructions **R** dans une fenêtre de script appelée *script* ou *R Editor*, accessible depuis le menu «Fichier/Nouveau script». Ouvrez une nouvelle fenêtre de script puis copiez le script suivant :

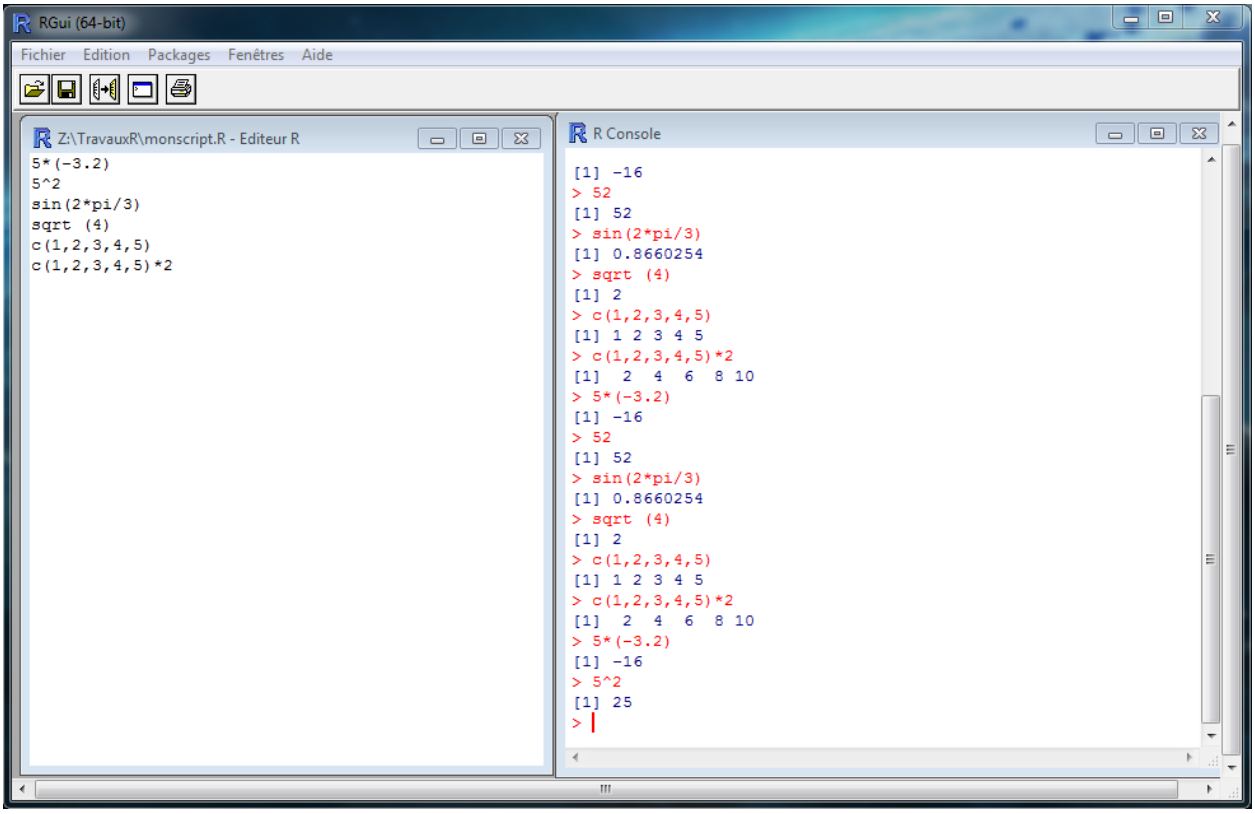

Figure 1: Vue de la fenêtre de script et de la consolde de commandes.

```
5*(-3.2)
5^2
sin(2*pi/3)
sqrt (4)
c(1,2,3,4,5)
c(1,2,3,4,5)*2
```

A la fin de la session, il est possible de sauvegarder ce script dans le dossier `TravauxR` par exemple, sous le nom `monscript.R`, et le rouvrir lors d'une session ulterieure depuis le menu «Fichier/Ouvrir un script».

Ensuite, vous pouvez taper successivement les combinaisons de touches `CTRL+A` pour sélectionner l'ensemble de ces instructions, puis `CTRL+R` pour les coller et les exécuter en une seule étape dans la console de **R**. Il est aussi possible d'exécuter une seule ligne d'instructions **R** du script en tapant `CTRL+R` lorsque le curseur clignotant se trouve sur la ligne en question dans la fenê tre de script.

Il est possible d'utiliser la fonction `source ()` depuis la console de R pour aller lire et exécuter le contenu du fichier script. Pour cela, il faut proceder ainsi :

1. cliquez une fois dans la fenetre *R Console*;
2. allez dans le menu «Fichier/Changer le répertoire courant» ;
3. explorez votre système de fichiers pour sélectionner le dossier `TravauxR`;
4. tapez dans la console: `source("monscript.R")` .

- Il faut avoir l'habitude d'utiliser le systerne d'aide en ligne de **R**. Cette aide est accessible au moyen de la fonction `help ()`. La commande `help (source)` permet par exemple d'obtenir de l'aide sur la fonction `source ()`.

Affichage des résultats et redirection dans des variables

R répond aux requêtes en affichant le résultat obtenu apres évaluation. **Ce résultat est affiché puis perdu**. Il apparaît plus intéressant de rediriger la sortie **R** d'une requête en la stockant dans une variable: cette opération s'appelle aussi **affectation du resultat dans une variable**. Une affectation évalue une expression, mais n'affiche pas le résultat qui est stocké dans un objet. Pour afficher ce résultat, il suffira de taper le nom de cet objet, suivi de la touche `ENTREE`.

Pour réaliser cette opération, on utilise la **flèche d'affectation** `<-`. La flèche `<-` s'obtient en tapant le signe inferieur (`<`) suivi du signe moins (`-`).

Pour créer un objet dans **R**, on utilise donc la syntaxe suivante : `nom.objet.a.creer <- instructions`

Par exemple

```
> x <- 1 # Affectation.
> x
```

```
## [1] 1
```

On dit que `x` vaut `1`, ou que l'on affecte `1` à `x`, ou encore que l'on met dans `x` la valeur `1`. Notez que l'on peut aussi utiliser l'operation d'affectation dans l'autre sens -> de la facon suivante :

Si une commande n'est pas complète à la fin d'une ligne, **R** affichera un signe d'invite different, par defaut le signe plus (`+`), sur la deuxième ligne ainsi que sur les lignes subséquentes. **R** continuera d'attendre des instructions jusqu'à ce que la commande soit syntaxiquement complète.

Utilisation des fonctions

En plus des fonctions qu'on a déjà vu (`sin()`, `sqrt()`, `exp()` et `log()`), **R** contient de nombreuses autres fonctions dans sa version de base, et on peut en ajouter des milliers d'autres (en installant des packages, ou même en les realisant soi-même).

Une fonction dans **R** est définie par son nom et par la liste de ses paramètres. La majorité des fonctions retournent une valeur, qui peut être un nombre, un vecteur, une matrice...

L'**utilsiation** d'une fonction (on dit aussi **appeler** ou **exécuter**) se fait en tapant le nom de celle-ci, suivi, entre une paire de parenthèses, de la liste des paramètres que l'on veut utiliser. Les paramètres sont séparés par des virgules. Chacun des paramètres peut être suivi du signe `=` et de la valeur que l'on veut donner au paramètre. Cette valeur du paramètre sera appelée paramètre effectif, paramètre d'appel ou parfois paramètre d'entrée.

Exemple d'un appel d'une fonction:

`nomfonction(par1=valeur1,par2=valeur2,par3=valeur3)`

où `par1`, `par2`, ... sont appelés les paramètres de la fonction tandis que `valeur1` est la valeur que l'on donne au parametre `par1`, etc. On peut toutefois noter qu'il n'est pas forcément nécessaire d'indiquer les paramètres mais seulement leurs valeurs, pour autant que l'on respecte leur ordre. Pour toute fonction présente dans **R**, certains paramètres sont obligatoires et d'autres sont facultatifs (car une valeur par defaut est déjà fournie dans le code de la fonction).

Exemple d'utilsiation des paramètres d'une fonction:

La fonction `log(x ,base=exp(1))` admet deux paramètres : `x` et `base`.

- `x` est obligatoire, c'est le nombre dont on veut calculer le logarithme.
- `base` est un paramètre optionnel puisqu'il est suivi du signe `= et de la valeur par défaut`exp(1)‘.

Dans l'appel suivant, le calcul effectué sera celui du logarithme népérien du nombre 1 puisque la valeur de base n'est pas renseignée :

```
> log(1)
```

```
## [1] 0
```

On peut appeler une fonction en jouant sur les paramètres de plusieurs façons différentes. Ainsi, pour calculer le logarithme népérien de 3, on peut utiliser n'importe laquelle des expressions suivantes.

- `log(3)`
- `log(3,base=exp(1))`
- `log(x=3)`
- `log(3 ,exp(1)))`
- `log(x=3,base=exp(1))`
- `log(base=exp(1),3)`
- `log(x=3, exp(1))`
- `log(base=exp(1),x=3)`

Les données dans R

Nature (ou type, ou mode) des données

Les fonctions `mode()` et `typeof()` permettent de gérer le type des données.

Enumérons maintenant les divers types de données (aussi appelés modes).

Type numérique (`numeric`)

Il y a deux types numériques : les entiers (`integer) et les reels` (realoudouble‘). Lorsque vous saisissez :

```
> a <- 1
> b <- 3.4
> c <- as.integer(a)
> typeof(c)
```

```
## [1] "integer"
```

Les variables `a` et `b` sont du type `"double"` et la variable `c` a la même valeur que `a` excepté qu'elle a été forcée à être du type `"integer"`. L'intérêt est que son stockage prend moins de place en mémoire. Les instructions commençant par `as.` sont très courantes en **R** pour convertir une donnée en un type différent. Nous verrons après comment vérifier que le type d'un objet est numérique.

Type complexe (`complex`)

Type booléen ou logique (`logical`)

Données manquantes (`NA`)

Type chaînes de caractères (`character`)

Données brutes (‘raw)

Vecteurs

Un vecteur est un groupe de valeurs primitives du même type. Il peut s'agir d'un groupe de nombres, de valeurs vraies / fausses, de textes et de valeurs d'un autre type. C'est l'un des éléments constitutifs de tous les objets R. Il existe plusieurs types de vecteurs dans **R**. Ils sont distincts les uns des autres par le type d'éléments qu'ils stockent. Dans les sections suivantes, nous verrons les types de vecteurs les plus couramment utilisés, notamment les vecteurs numériques, les vecteurs logiques et les vecteurs de caractères.

Vecteur numérique

Un vecteur numérique est un vecteur de valeurs numériques. Un nombre scalaire est le vecteur numérique le plus simple. Voici un exemple de vecteur numérique:

```
> 1.5
```

```
## [1] 1.5
```

Le vecteur numérique est le type de données le plus fréquemment utilisé et constitue la base de presque tous les types d'analyse de données. Dans d'autres langages de programmation courants, il existe certains types scalaires tels que entier, double et chaîne, et ces types scalaires constituent les blocs de construction des types de conteneur tels que les vecteurs. En **R**, cependant, il n'existe pas de définition formelle des types scalaires. Un nombre scalaire n'est qu'un cas particulier de vecteur numérique, et il n'est spécial que parce que sa longueur est 1.

Lorsque nous créons une valeur, il est naturel de penser à la manière de la stocker pour une utilisation future. Pour stocker la valeur, nous pouvons utiliser `<-` pour affecter la valeur à un symbole. En d'autres termes, nous créons une variable nommée `x` de la valeur `1.5`: