UNIVERSITÉ DE NANTES

**Master 1 EKAP - Travaux dirigés d'introduction au logiciel R - TD n°7**

Responsable du cours : Camille Aït-Youcef

Année : 2020 - 2021

# Quelques notions de programmation en R

La progrommation en R est basée sur les mêmes principes que d'autres logiciels de calcul scientifique. En effet on retrouve à la fois les structures classiques de la programmation (boucles, condition if else, etc.) et des fonctions prédéfinies propres à la pratique statistique.

**Commandes groupées**

Un groupe de commandes est comme une parenthèse en mathématiques : les commandes groupées sont effectuées ensemble. Sous R, le groupe de commandes est délimité par des accolades : {

expression1

expression2

...

}

Ou bien {expression1 ; expression2 ; ... }

**Les boucles (for ou while)**

Les boucles classiques peuvent être utilisées avec le logiciel R.

La boucle **for** :

Exemple : afficher tous les entiers de 1 à 99 à l'écran. une solution peut être :

for (i in 1:99) print (i)

L'indice i prend comme valeurs toutes les coordonnées du vecteur choisi. Si nous souhaitons balayer les entiers de deux en deux, il faut constituer un vecteur qui démarre à 1 et qui va jusqu'à 99, de deux en deux. La fonction **seq** (seq est une fonction qui permet de sélectionner un sous ensemble dans un vecteur conditionnellement à ses arguments) permet de constituer un tel vecteur. La boucle devient :

for (i in seq (1,99, by=2)) print (i)

En général, plusieurs ordres à effectuer sont nécessaire à chaque itération dans une boucle. Pour cela il est donc nécessaire de grouper les commandes. En général la boucle for s'écrit :

```
for (i in vecteur) {
expression1
expression2
...
}
```

Une autre possibilité de boucle est la condition **while**. Sa syntaxe générale est la suivante :

```
while (condition) {
expression1
expression2
...
}
```

Les ordres expression1, expression2, etc. sont effectués tans que la condition est vraie. La condition est évaluée en début de boucle. Dès que la condition est fausse, la boucle est arrêtée. Ainsi,

```
i <- 1
while (i< 3) {
print (i)
i <- i + 1 }
```

Permet d'afficher i et de l'incrémenter de 1 tant que i est inférieur à 3. Une dernière possibilité

de boucle est l'ordre repeat. Il se comprend comme : répéter indéfiniment des ordres. La sortie de boucle est assurée par l'ordre break. Cet ordre peut être utilisé quelle que soit la boucle. Un exemple est donné dans le paragraphe suivant.

**Les conditions (if, else)**

Il s'agit d'exécuter un ordre sous condition : l'ordre est exécuté si et seulement si la condition est vraie. Dans sa forme simple la boucle s'écrit :

```
if (condition) {
expression1
expression2
...
}
```

Par exemple, si la boucle **repeat** est utiliser pour afficher i variant de 1 à 3 compris, il faut sortir de la boucle, avant l'affichage, quand i est supérieur à 3.

```
    i <- 1
repeat {
print(i)
i <- i+1
if (i>3) break }
```

Ici, si i est supérieur à 3, un seul ordre est exécuté, break.

Une autre commande peut être ajoutée à la suite du **if**, la condition **else**, qui permet de séparer les deux cas : si la condition est vraie alors l'ordre (ou le groupement d'ordre) après if est exécuté, si elle n'est pas vraie, alors l'ordre (ou le groupement d'ordres) après else est exécuté. Sous sa forme générale la condition if, else s'écrit :

if (condition) { expression1

expression2

...

}

else {

expression3

expessionr4

...

}

**Remarque :** l'ordre else doit être sur la même ligne que l'accolade fermante de la clause if.

**Les clauses prédéfinies**

Certaines fonctions ont été prédéfinies dans R afin d'éviter d'avoir recours à des boucles coûteuses en temps de calcul. La plus utilisée d'entre elles est sûrement la fonction **apply**. Elle permet d'appliquer une même fonction à toutes les marges d'un tableau.
Son expression est la suivante

apply(x, MARGIN= i, FUN = y).

Avec x le nom du tableau, i = 1 ou 2 en fonction de si on veut appliquer la fonction apply aux marges en colonne ou en ligne et y est la fonction à appliquer à toutes les marges.
**Remarque 1 :** S'il y a une donnée manquante dans X, on peut utiliser l'argument na.rm=TRUE de la fonction, ce qui permet de calculer la moyenne uniquement sur les données présentes.
**Remarque 2 :** Il existe aussi des fonctions qui permettent d'obtenir des statistiques par colonne et par ligne pour les tableaux, comme par exemple les fonctions **colMeans** ou **rowMeans**.

La fonction **lapply**, ou son équivalent **sapply**, applique une même fonction à chaque élément

d'une liste. La différence entre ces deux fonctions est le stockage dans des objets différents de l'exécution de la fonction. La fonction lapply retourne par défaut une liste quand la fonction sapply retourne une matrice ou un vecteur.

**Construire une fonction**

Une fonction permet d'effectuer un certain nombre d'ordres. Ces ordres peuvent dépendre d'arguments fournis en entrée, mais cela n'est pas obligatoire. La fonction fournit un objet "résultat" unique. Cet objet "résultat" est désigné à l'intérieur de la fonction par la fonction **return**. Par défaut, si la fonction écrite ne renvoi pas de résultat, le dernier résultat obtenu avant la sortie de la fonction est renvoyé comme résultat.

Commençons par un ordre simple, la somme des n premiers entiers. Le nombre n est un entier qui est l'argument d'entrée,, le résultat est simplement la somme demandée : som <- function(n) {

resultat <- sum (1:n)

return (resultat)

}

La fonction est appelée grâce à son nom (som), suivi des arguments en entrée entre parenthèses. Ici, la fonction possède un argument en entrée, elle est donc appelée simplement par :

som(5)

## Exercice 9 : Programmer des fonctions

### Questions

1. Créez un vecteur de caractère qui prend comme valeur les trois premiers jours de la semaine, puis créez une boucle permettant d'afficher dans la console les trois premiers jour de la semaine.

2. Créez un vecteur i qui prend comme valeur uniquement 0. Créez une boucle qui affiche dans la console la valeur prise par l'indice i en incrémentant i de 1 tant que i est inférieur à 10.

3. Créez un vecteur y de longueur 10 qui vaut 0 si x vaut 4, 1 sinon. L'expression de y est la suivante :
   x <- sample(1:10,10, rep = T)

4. Créez une variable X qui prend la valeur 1984. Vous avez obtenu votre baccalauréat en 1991. Créer une boucle permettant d'afficher la valeur de x et qui augmente de 1 tant que X est inférieur ou égal à 1991.

5. Considérez le tableau nommé hasard qui est constitué de 25 nombres entiers tirés au hasard sans remise entre 1 à 25 (utiliser la fonction sample). Ce tableau correspond à une matrice carrée. Calculez la moyenne par ligne en utilisant la fonction apply. Puis utilisez la fonction rowMeans pour aboutir au même résultat.

6. Créez une fonction permettant pour chaque élément du tableau hasard de retourner l'inverse de la sommes de hasard + $y^2$. Utilisez la fonction apply pour appliquer votre fonction à chaque élément de votre tableau sachant que y vaut 2.

7. Exécutez les commandes data(iris) puis str(iris). Vous venez de charger en mémoire l'un des jeux de données distribués avec R. Tous les jeux de données disponibles avec l'installation de base de R sont accessibles en tapant data().

8. Créez la fonction moyennesd qui calcule pour chaque colonne du dataframe iris la moyenne

et l'écart-type. Appliquer votre fonction à la première colonne du jeu de donnée. Utilisez la fonction apply pour calculer les mêmes statistiques en colonne. Qu'en concluez vous ?