Université Rennes 2
Master 1

Année 2019-2020
Ioana Gavra / Madison Giacofci

# Programmation sous R

Ouvrez une session R avec Rstudio et placez-vous dans votre répertoire personnel. Les questions commençant par Q.S. sont des questions supplémentaires qui, sans être essentielles à la suite du TD, vous permettent généralement de découvrir des fonctions en plus.

## 1 Structures de contrôle

### 1.1 Les conditions

On distingue 2 syntaxes principales pour la condition `if` suivant s'il existe ou non une instruction à effectuer dans le cas où la condition n'est pas respectée.

```
> if (condition==TRUE){
+ instructions
+ }
```

et

```
> if (condition==TRUE){
+ instructions
+ } else {
+ instructions
+ }
```

NB : Attention, lorsqu'on utilise la structure `if ... else`, il est nécessaire d'écrire le mot `else` sur la même ligne que la parenthèse fermante de l'instruction `if` précédente.

1. Créer une condition permettant d'afficher sur la console si un entier est pair ou impair (opérateur `%%`).
   Q.S. Reprendre la question précédente en utilisant `ifelse()` puis `switch()`.
2. Créer une condition permettant d'afficher sur la console si un entier est un carré ou non.
3. Créer une condition permettant d'afficher sur la console si un entier est le carré d'un nombre impair multiple de 3.

### 1.2 Les boucles

#### 1.2.1 Boucle `for`

La syntaxe de cette boucle se présente sous la forme suivante :

```
> for (variable in vector){
+ instructions
+ }
```

1. Le nombre $e$ peut être approché grâce à la limite suivante :

$$e = \lim_{n \to +\infty} \left(1 + \frac{1}{n}\right)^n$$

   À l'aide d'une boucle `for`, calculer les approximations successives obtenues pour $n$ entre 1 et 50. Représenter les résultats obtenus sur un graphe, en matérialisant la valeur cible par une ligne rouge pointillée.
2. Reprendre la question précédente en utilisant :

$$e = \sum_{n=0}^{+\infty} \frac{1}{n!}$$

3. On place un capital de 1000 euros pendant 10 ans au taux d'intérêt annuel de 0.75% (intérêts composés). À l'aide d'une boucle `for`, calculer le montant des intérêts produits. On rappelle que pour des intérêts composés, à la fin de chaque période, les intérêts générés sont ajoutés au capital de départ pour produire de nouveaux intérêts. Par exemple, le livret A fonctionne de cette manière. En 2008, son taux annuel était de 4%. Quel était alors le montant des intérêts générés par le même placement ?
4. Charger le jeu de données `UScereal`. En utilisant une boucle `for`, calculer la moyenne et l'écart-type du nombre de calories en fonction du producteur. Renvoyer le résultat sous forme de data-frame.

### 1.2.2 Boucle `while`

La syntaxe de cette boucle se présente sous la forme suivante :

```
> while (condition==TRUE){
+ instructions
+ }
```

Cette syntaxe est utile quand on ne connaît pas à l'avance le nombre d'opérations à effectuer.

1. Á l'aide d'une boucle `while`, déterminer la puissance de 2 dans la décomposition en facteurs premiers d'un entier naturel $n$.
2. Choisir un entier au hasard entre 1 et 100 et le stocker dans un objet `myst`. Créer une boucle (`while`) tirant un entier entre 1 et 100 à chaque itération (fonction `sample`) et le comparer à `myst`. La boucle continue tant que le nombre tiré est différent de `myst`. À la sortie de la boucle, le nombre de tirages effectués sera renvoyé.

### 1.2.3 Boucle `repeat`

La syntaxe de cette boucle se présente sous la forme suivante :

```
> repeat{
+ instructions
+ if (cond d'arret) break
+ }
```

Cette syntaxe est utile quand on ne connaît pas à l'avance le nombre d'opérations à effectuer, mais on sait que les instructions doivent être exécutés au moins une fois.
Q.S. Reprendre la question 2, de la section précédente en créant la boucle avec `repeat` à la place de `while`.

1. (Suite de Syracuse). Choisir un entier $n$ quelconque. On modifie cet entier comme suit :
   - si $n$ est pair, on le divise par 2,
   - sinon, on le multiplie par 3 et on lui ajoute 1.

   On répète ensuite l'opération avec le nombre obtenu jusqu'au moment où on attend le nombre 1 : en effet, à partir de ce nombre, on tombe sur un cycle se répétant indéfiniment 1,4,2,1,4,2,1...

   Créer une boucle `repeat` reproduisant cette suite et s'arrêtant lorsque l'on atteint 1 pour un nombre entier $n$ choisi. Conserver la suite des nombres obtenus, et imprimer le nombre d'opérations nécessaires.

## 2 Création de fonctions

Nous avons déjà vu que le logiciel R possède de nombreuses fonctions basiques ou plus complexes. Il est en outre possible de construire simplement de nouvelles fonctions. Par définition, une fonction est destinée à renvoyer une ou plusieurs valeurs en sortie calculées à partir de paramètres donnés en entrée. Au sein du logiciel, la syntaxe est la suivante :

```
> mafonction = function(variable1,variable2,...){
+ instructions
+ return(resultat)
+ }
```

Les paramètres de la fonction peuvent être de types différents : par exemple, `variable 1` peut être un vecteur, `variable 2` une liste, `variable 3` un facteur et ainsi de suite...
Le résultat doit par contre être constitué d'un seul objet : si l'on souhaite renvoyer plusieurs objets de types différents, on peut mettre ces objets au sein d'une liste.

1. Reprendre la question 3 de la Section 1.2.1. Créer une fonction calculant le montant des intérêts perçu pour un capital $C$, placé pendant $n$ années au taux annuel $taux$ (on considère des intérêts composés).
2. Reprendre la question 2 de la Section 1.2.2.
   a) Créer une fonction renvoyant, à partir d'un nombre $n$ donné, le nombre de tirages nécessaires pour retrouver ce nombre.
   b) Choisissez un nombre entre 1 et 100. En utilisant une boucle `for`, appliquer votre fonction $m = 10$ fois et faire la moyenne du nombre de tirages nécessaires.
   c) Faire la même chose pour $m = 1000$. Que constatez-vous ? Quel est le résultat attendu ?
3. Reprendre la question 2 de la Section 1.2.3.
   a) Créer une fonction donnant les termes de la suite de Syracuse partant d'un entier $n$ quelconque, ainsi que sa longueur avant d'atteindre 1. Le résultat sera renvoyé sous forme de liste.

   b) Appliquer cette fonction aux entiers de 2 à 100. Représenter sur un graphique les longueurs des suites associées à chaque nombre.
   c) Parmi les suites créées, quel est le nombre maximum atteint ? Pour quel entier est-il atteint ?

   *Remarque : On pourrait se poser la question de savoir si partant d'un nombre entier quelconque, on atteint toujours le nombre 1 à un certain rang. La réponse est loin d'être triviale et ce problème, connu sous le nom de* problème 3n+1*, a défié plus d'un chercheur au cours du siècle dernier et plus particulièrement dans les années 60. Pour une introduction simple :* `http://images.math.cnrs.fr/Le-probleme-3n-1-elementaire-mais.html`

### 2.1 Comment gérer des cas particuliers

Des exceptions peuvent être gérés à l'intérieur d'une fonction avec différentes structures de contrôles : `break`, `stop`, `warning`,...

1. Créer une fonction renvoyant la factorielle d'un nombre $n$ donné (sans utiliser `factorial`...). Modifier ensuite la fonction pour :
   — renvoyer une erreur si le nombre donné en entrée est négatif (on peut utiliser `stop`).
   — calculer la factorielle de la partie entière de $n$ si $n \in \mathbb{R}_+ \setminus \mathbb{N}$ et afficher un avertissement le cas échéant (`warning()`).
2. Reprendre la fonction cherchant un entier mystère (question 2, Section 1.2.2). Ajouter une condition `if` pour autoriser 100 itérations au maximum à l'aide de la fonction `break`. On affichera le cas échéant la mention "l'algorithme n'a pas convergé" et on donnera la valeur `NULL` au résultat.
3. Créer une boucle `for` pour calculer la moyenne sur 50 essais du nombre de tirages nécessaires pour retrouver le nombre mystère, tout en autorisant 100 itérations au maximum. Utiliser la structure de contrôle `next` pour passer à l'itération suivante lorsque l'algorithme ne trouve pas le nombre mystère à temps.

### 2.2 Opérateurs

R a plusieurs opérateurs binaires prédéfinis. Exemples :

```
+,-,*, /, ^, %%, %/%, &, |, ==, !=, <, <=, >=, >, %in% ...
```

Ces opérateurs agissent comme une fonction avec 2 paramètres. On peut définir un nouvel opérateur `%MonOperateur%` en utilisant une structure de la forme :

```
"%MonOperateur%" <- function(par1, par2){ intructions; return(Resultat)}
```

Exemple : On peut définir un opérateur `%m%`, calculant la moyenne de deux nombres de la manière suivante :

```
"%m%" <- function(x,y) return(mean(c(x,y)))
#pour calculer la moyenne de 2 et 5:
2%m%5
```

Q.S. Créer un opérateur `%+%` permettant de concaténer deux chaînes de caractères (obtenir donc une forme plus synthétique et simplifié de la fonction `paste()`)

## 3 Famille des fonctions `apply`

Nous avons déjà vu que le langage de R est vectorisé, c'est à dire que ses objets sont basés sur la notion de vecteurs. Pour cette raison, il est préférable d'éviter autant que possible l'utilisation de boucles afin d'optimiser les performances du logiciel. Cela se fait au moyen des fonctions de la famille `apply` dont les plus courantes sont recensées dans le tableau ci-dessous.

| **Fonction** | **Entrée** | **Sortie** |
|---|---|---|
| `apply(x,marg,FUN)` | Matrice ou data-frame | Vecteur, matrice ou liste |
| `lapply(x,FUN)` | Liste ou vecteur | Liste |
| `sapply(x,FUN)` | Liste ou vecteur | Vecteur, matrice ou liste |
| `tapply(x,factor,FUN)` | Data-frame et facteurs | Data-frame ou liste |

1. Calculer le cosinus des entiers entre 1 et 10000 à l'aide d'une boucle `for`, d'une des fonctions `apply` et enfin en utilisant simplement la fonction `cos`. Comparer les temps de calcul nécessaires à chaque fois à l'aide de la fonction `system.time`.
2. Importer le jeu de données `bladder`.
   — Calculer la moyenne par colonne, puis par ligne à l'aide de la fonction `apply`. Comment gérer la présence de données manquantes ?
   Remarque : On peut également utiliser les raccourcis `colMeans()` et `rowMeans()`.
   — Faire de même pour l'écart-type, sans utiliser la fonction `sd` (*i.e.* en la programmant par vous-même). Vérifiez votre résultat en utilisant la fonction `sd`.
   Q.S. En utilisant `sweep()`, centrer puis normaliser les données (par colonne).
3. Reprendre les boucles et fonctions construites au cours du TP et les optimiser à l'aide des fonctions `apply` quand il est possible de le faire.
4. On définit les vecteurs $x$ et $y$ de la manière suivante :

   ```
   > x = seq(0,100,by=10)
   > y = sample(1:100,15)
   ```

   Créer une fonction permettant de donner pour chaque élément de $y$ l'élément de $x$ qui lui est le plus proche.
5. Calculer le produit d'une matrice $n \times p$ et d'un vecteur de taille $p$ à l'aide d'une des fonctions de la famille `apply`.
6. Charger le jeu de données mtcars (`data(mtcars)`). Calculer la consommation moyenne (`mpg`) en fonction de la transmission (`am`) et du nombre de cylindres (`cyl`) à l'aide de `tapply`.