

Introduction à R

Modélisation Stochastique & Simulation



2023/2024

Table of contents





- ❶ Pourquoi utiliser R?
- ❷ R et sa documentation
- ❸ Structure de données
- ❹ Manipulation de données
- ❺ Initiation à la programmation en R
- ❻ Lois de probabilité

Pourquoi utiliser R?


Pourquoi utiliser R?

- Tout d'abord  est un logiciel gratuit et à code source ouvert (open source). Il fonctionne sous différentes plates formes : UNIX (et Linux), Windows et Macintosh. Tout le monde peut d'ailleurs contribuer à son amélioration en y intégrant de nouvelles fonctionnalités ou méthodes d'analyse non encore implémentées. Cela en fait donc un logiciel en rapide et constante évolution.
- C'est aussi un outil très puissant et très complet, particulièrement bien adapté pour la mise en oeuvre informatique de méthodes statistiques.
- Le logiciel  est conçu pour être un outil très efficace lorsque l'on arrive à le maîtriser, puisque l'on devient alors capable de créer ses propres outils, ce qui permet ainsi d'opérer des analyses très sophistiquées sur les données.


R et les Statistiques

-  est un **langage de programmation** utilisé pour le calcul statistique, l'analyse des données et la recherche scientifique. Il s'agit de l'un des langages les plus utilisés par les statisticiens, analystes de données et chercheurs pour gérer, manipuler, analyser et visualiser des données.
- Rstudio est un **environnement de développement intégré** (IDE) pour  qui permet aux utilisateurs d'interagir plus facilement avec  en intégrant différents aspects de scriptage, de la complétion de code au débogage.
- RStudio ne peut fonctionner que si  a été installé au préalable.

R et les graphiques

- Une des grandes forces de  réside dans ses capacités, bien supérieures à celles des autres logiciels courants du marché, à combiner un langage de programmation avec la possibilité de réaliser des graphiques de qualité. Ces dernières possèdent de très nombreux paramètres permettant par exemple d'ajouter des titres, des légendes, des couleurs, etc.
- Mais il est également possible d'effectuer des graphiques plus sophistiqués permettant de représenter des données complexes telles que des courbes de surface ou de niveau, des volumes affichés avec un effet 3D, des courbes de densité, et bien d'autres choses encore.


R et les graphiques

- Vous pouvez obtenir une démonstration des possibilités graphiques de  en tapant successivement les commande suivantes:

```
> demo(graphics)
> demo(persp)
> demo(image)
> example(plot)
```

R et sa documentation

La commande help()

-  contient une aide en ligne (en anglais) très complète et très bien structurée sur toutes les fonctions que vous pouvez utiliser ainsi que sur les différents symboles du langage. Cette aide est accessible au moyen de plusieurs commandes dont la principale est `help()`. Tapez par exemple:

```
> help(help)
```

- La commande `help()` possède un alias qui est le point d'interrogation: `?`

```
> ?sum  
> ?mean
```

La commande help()

- Il peut arriver que cet alias ne fonctionne pas, et il est alors nécessaire d'utiliser la fonction `help()` avec des guillemets.

```
> ?function      # Ne fonctionne pas  
> help(function) # Renvoie une erreur  
> help("function") # Appel correct
```

Structure de données

Les vecteurs (vector)

- Cette structure de données est la plus simple. Elle représente une suite de données de même type. La fonction permettant de créer ce type de structure (c'est-à-dire les vecteurs) est la fonction `c()` (pour collection).
- D'autres fonctions comme `seq()` ou bien les deux points `" : "` permettent aussi de créer des vecteurs.
- Notez que lors de la création d'un vecteur, il est possible de mélanger des données de plusieurs types différents.

Les vecteurs (vector)

```
> c(1,1,7,-2,1)
```

```
[1] 1 1 7 -2 1
```

```
> c(1,1,7,-2,1,TRUE,1,"R")
```

```
[1] "1" "1" "7" "-2" "1" "TRUE" "1" "R"
```

```
> seq(from=0,to=4,by=1)
```

```
[1] 0 1 2 3 4
```

```
> x <- 0:4
```

```
[1] 0 1 2 3 4
```

Les matrices (matrix) et les tableaux (arrays)

Ces deux notions généralisent la notion de vecteur puisqu'elles représentent des suites à double indice pour les matrices et à multiples indices pour les tableaux (array). Ici aussi, les éléments doivent avoir le même type.

```
> X <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE)
> X
```

	[,1]	[,2]	[,3]
[1,]	1	2	3
[2,]	4	5	6
[3,]	7	8	9
[4,]	10	11	12

permet de créer une matrice comportant quatre lignes (row signifie ligne) et trois colonnes remplies par lignes successives (byrow=TRUE) avec les éléments du vecteur 1:12.

Les matrices (matrix) et les tableaux (arrays)

De la même manière, il est possible de créer une matrice remplie par colonnes successives (`byrow=FALSE`).

```
> X <- matrix(1:12,nrow=4,ncol=3,byrow=FALSE)
> X
```

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

Les matrices (matrix) et les tableaux (arrays)

Remarque: Par défaut, les éléments d'une matrices sont de même type (numeric). Le changement du type d'un élément, induira le changement du type de la matrice.

```
> Y <- matrix(1:9,nrow=3,ncol=3,byrow=T)
> typeof(Y)
```

```
[1] "double"
```

```
> Y[1,1] <- "A"
> Y
```

```
      [,1] [,2] [,3]
[1,]  "A"  "2"  "3"
[2,]  "4"  "5"  "6"
[3,]  "7"  "8"  "9"
```

```
> typeof(Y)
```

```
[1] "character"
```


Les matrices (matrix) et les tableaux (arrays)

La fonction `array` permet de créer des matrices multidimensionnelles à plus de deux dimensions comme cela est illustré (pour un array ayant trois dimensions).

```
> X <- array(1:12,dim=c(2,2,3))  
> X
```

Les matrices (matrix) et les tableaux (arrays)

```
, , 1  
      [,1] [,2]  
[1,]     1     3  
[2,]     2     4
```

```
, , 2  
      [,1] [,2]  
[1,]     5     7  
[2,]     6     8
```

```
, , 3  
      [,1] [,2]  
[1,]     9    11  
[2,]    10    12
```

Les listes (list)

La structure du langage R la plus souple et à la fois la plus riche est sans doute l'objet de type "liste". Contrairement aux structures précédentes, les listes permettent de regrouper dans une même structure des données de types différents.

```
> A <- list(TRUE,-1:3,matrix(1:4,nrow=2),"3Y- ENSIA")
```

Les listes (list)

```
> A
```

```
[[1]]
```

```
[1] TRUE
```

```
[[2]]
```

```
[1] -1 0 1 2 3
```

```
[[3]]
```

```
  [,1] [,2]
```


```
[1,]    1    3
```

```
[2,]    2    4
```

```
[[4]]
```

```
[1] "3Y- ENSIA"
```

Le tableau individus \times variables (data.frame)

Le tableau individus \times variables est la structure par excellence en statistique. Cette notion est exprimée dans  par le `data.frame`. Conceptuellement, c'est une matrice dont les lignes correspondent aux individus et les colonnes aux variables (ou caractères) mesurées sur ces derniers. Chaque colonne représente une variable particulière dont tous les éléments sont du même type. Les colonnes de la matrice-données peuvent être nommées. Voici un exemple de création d'un `data.frame` :

```
> DATA <- data.frame(Sexe=c("H", "F", "H", "F", "H", "F"),  
                      Taille=c(1.83, 1.76, 1.82, 1.60, 1.90, 1.66),  
                      Poids=c(67, 58, 66, 48, 75, 55),  
                      row.names=c("R", "L", "P", "D", "B", "C"))
```

Le tableau individus \times variables (data.frame)


```
> DATA
```

	Sexe	Taille	Poids
R	H	1.83	67
L	F	1.76	58
P	H	1.82	66
D	F	1.60	48
B	H	1.90	75
C	F	1.66	55

```
> is.data.frame(DATA)
```

```
[1] TRUE
```

Les facteurs (factor) et les variables ordinales (ordered)

 permet d'organiser les chaînes de caractères de façon plus astucieuse au moyen de la fonction `factor()`, est donc celle à utiliser pour stocker des variables qualitatives.

```
> x <- factor(c("bleu", "vert", "bleu", "rouge", "bleu"))  
> x
```

```
[1] bleu vert bleu rouge bleu  
Levels: bleu rouge vert
```

Les facteurs (factor) et les variables ordinales (ordered)

Pour les variables ordinales, il est plutôt conseillé d'utiliser la fonction `ordered()`.

```
> x <- ordered(c("Petit", "Grand", "Moyen", "Grand", "Moyen",  
"Petit", "Petit"), levels=c("Petit", "Moyen", "Grand"))  
> x
```


```
[1] Petit Grand Moyen Grand Moyen Petit Petit  
Levels: Petit < Moyen < Grand
```


Les différentes structures de données en R

Structure des données	Instruction R
vecteur	<code>c()</code>
matrice	<code>matrix()</code>
tableau multidimensionnel	<code>array()</code>
liste	<code>list()</code>
tableau individus \times variables	<code>data.frame()</code>
facteur	<code>factor()</code> , <code>ordered()</code>

Manipulation de données


Arithmétique vectorielle

- Le logiciel  présente l'avantage de pouvoir opérer sur des vecteurs ou des matrices. Ainsi, l'appel suivant

```
> x <- c(1,2,4,6,3)
> y <- c(4,7,8,1,1)
> x+y
```

```
[1] 5 9 12 7 4
```



Arithmétique vectorielle

-  opère de la même façon pour de très nombreuses fonctions telles que: $+$, $*$, $-$, $/$, \exp , \log , \sin , \cos , \tan , $\sqrt{}$... etc.
- Par exemple, l'instruction suivante calcule l'exponentielle de tous les éléments de la matrice M :

```
> M <- matrix(1:9,nrow=3)
> exp(M)
```


	[,1]	[,2]	[,3]
[1,]	2.718282	54.59815	1096.633
[2,]	7.389056	148.41316	2980.958
[3,]	20.085537	403.42879	8103.084

Le recyclage

Un point important à noter à ce stade est la façon dont  se comporte lorsque les deux vecteurs fournis à l'une des fonctions ci-dessus ne sont pas de la même longueur.  va alors compléter le vecteur le plus court en réutilisant les valeurs de ce vecteur. L'exemple suivant devrait permettre de bien comprendre ce fonctionnement:

```
> x <- c(1,2,3,4,5,6,7,8,9,10)
> y <- c(1,2,3,4,5)
> x + y
```

```
[1] 2 4 6 8 10 7 9 11 13 15
```

 a donc complété le vecteur y ainsi: `c(1,2,3,4,5,1,2,3,4,5)` en se resserrant de ses propres valeurs de façon circulaire.

Le recyclage

Voici un autre exemple de recyclage utilisé lors de la création d'une matrice. Le vecteur $1:4$ est ainsi réutilisée pour remplir la matrice qui est déclarée de taille 3×3 .

```
> M <- matrix(1:4,ncol=3,nrow=3)
> M
```

```
      [,1] [,2] [,3]
[1,]     1     4     3
[2,]     2     1     4
[3,]     3     2     1
```

Warning message:

```
In matrix(1:4, ncol = 3, nrow = 3) :
  data length [4] is not a sub-multiple or multiple of the
  number of rows [3]
```

Fonctions basiques

Voyons maintenant quelques fonctions basiques de manipulation de données qui sont très souvent utilisées et qu'il est donc indispensable de connaître.

```
> length(c(1,3,6,2,7,4,8,1,0)) # renvoie la longueur d'un vecteur
```

```
[1] 9
```

```
> sort(c(1,3,6,2,7,4,8,1,0)) # permet d'ordonner les éléments  
# d'un vecteur (croissantes)
```

```
[1] 0 1 1 2 3 4 6 7 8
```

```
> sort(c(1,3,6,2,7,4,8,1,0),decreasing=TRUE) # (décroissantes)
```

```
[1] 8 7 6 4 3 2 1 1 0
```

Fonctions basiques

```
> rev(c(1,3,6,2,7,4,8,1,0)) # réarrange les éléments d'un  
                             # vecteur en sens inverse
```

```
[1] 0 1 8 4 7 2 6 3 1
```

```
> order(c(1,3,6,2,7,4,8,1,0)) # renvoie le vecteur des rangs  
                              # de classement des éléments
```

```
[1] 9 1 8 4 2 6 3 5 7
```

```
> unique(c(1,3,6,2,7,4,8,1,0)) # enlève les doublons d'un vecteur
```

```
[1] 1 3 6 2 7 4 8 0
```


Initiation à la programmation en R

Instructions de condition

Instruction switch

Cette commande permet de basculer d'une liste de commandes vers une autre selon la valeur de l'expression qui se trouve juste après la commande `switch`. Elle s'utilise de la façon suivante:

```
> switch(<expr:test>, <expr:cas 1> = <code1>, <expr:cas 2> = <code2>,  
        etc...)
```

Dans l'instruction ci-dessus, `<expr:test>` est soit un nombre, soit une chaîne de caractères. Cette instruction renvoie :

- `<code1>` si `<expr:test>` vaut `<expr:cas 1>`,
- `<code2>` si `<expr:test>` vaut `<expr:cas 2>`, ..., ect.

Si `<expr:test>` n'est égal à aucun des `<expr:cas i>`, la fonction `switch` renvoie alors `NULL`

Instruction switch

```
> x <- c(5,2,3,1,4,2)
> entree <- "moyenne"
> switch(entree, moyenne = mean(x), mediane = median(x))
```

```
[1] 2.833333
```

```
> entree <- "mediane"
> switch(entree, moyenne = mean(x), mediane = median(x))
```

```
[1] 2.5
```

Instruction `if...else`

L'instruction conditionnelle `if` est utilisée sous les deux formes suivantes:

```
if (cond) {expr:vrai}
```

ou

```
if (cond) {expr:vrai} else {expr:faux}
```

Voici deux exemples d'utilisation:

```
> x <- 4  
> if (x < 0) {y = x+2}  
> if (x == 0) {y = -x}  
> if (x > 0) {y = sqrt(x)}  
> y
```

```
[1] 2
```

Instruction if...else

```
> x <- 2
> y <- -3
> if (x <= y) { z <- y-x
+           paste("x plus petit que y ==> z =", z)
+       } else { z <- x-y
+           paste("x plus grand que y ==> z =", z)}
```

```
[1] "x plus grand que y ==> z = 5"
```

La fonction `ifelse`

La fonction `ifelse()` permet d'exécuter l'un ou l'autre de deux instruction appliquée à un vecteur suivant que les valeur d'une condition logique sont vrais ou fausses. Sa syntaxe est:

```
ifelse (condition, expression.vrai, expression.faux)
```

Par exemple:

```
> x <- c(-3:2)
> y <- ifelse(x > 0, x , -x)
> y
```

```
[1] 3 2 1 0 1 2
```


```
> y <- ifelse(x > 0, log(x) , NA)
```

Warning message:

```
In log(x) : NaNs produced
```

Instructions de boucles

Instruction de boucles

 possède trois instructions de boucle: `for`, `while` et `repeat`. Les mots réservés `next` et `break` fournissent par ailleurs un contrôle supplémentaire de l'exécution d'un code:

- L'instruction `break` provoque une sortie immédiate de la boucle en cours d'exécution.
- L'instruction `next` amène le curseur d'exécution du programme au départ de la boucle. La prochaine itération de la boucle (s'il y en a une) est ensuite exécutée. Aucune instruction après `next` dans la boucle courante n'est exécutée.

La boucle for

- La boucle `for` permet d'effectuer des opérations pour un nombre d'itérations définis, possède la syntaxe suivante:

```
for (i in val_init:val_fin) { liste des instructions }
```

La boucle for

- La boucle `for` permet d'effectuer des opérations pour un nombre d'itérations définis, possède la syntaxe suivante:

```
for (i in val_init:val_fin) { liste des instructions }
```

- Voici deux exemples:

```
for (i in 1:3) { print(i) } # Affichage
```

La boucle for

- La boucle `for` permet d'effectuer des opérations pour un nombre d'itérations définis, possède la syntaxe suivante:

```
for (i in val_init:val_fin) { liste des instructions }
```

- Voici deux exemples:

```
for (i in 1:3) { print(i) } # Affichage
```

```
[1] 1
```

```
[1] 2
```

```
[1] 3
```

La boucle for

- La boucle `for` permet d'effectuer des opérations pour un nombre d'itérations définis, possède la syntaxe suivante:

```
for (i in val_init:val_fin) { liste des instructions }
```

- Voici deux exemples:

```
for (i in 1:3) { print(i) } # Affichage
```

```
[1] 1
```

```
[1] 2
```

```
[1] 3
```

```
> x <- seq(0,6,by=2)
```

```
> for (j in x) { print(2*j) }
```

La boucle for

- La boucle `for` permet d'effectuer des opérations pour un nombre d'itérations définis, possède la syntaxe suivante:

```
for (i in val_init:val_fin) { liste des instructions }
```

- Voici deux exemples:

```
for (i in 1:3) { print(i) } # Affichage
```

```
[1] 1  
[1] 2  
[1] 3
```

```
> x <- seq(0,6,by=2)  
> for (j in x) { print(2*j) }
```

```
[1] 0  
[1] 4  
[1] 8  
[1] 12
```

La boucle `for`

Écrire un code qui calcule la somme $S = \sum_{i=1}^n i$.

La boucle for

Écrire un code qui calcule la somme $S = \sum_{i=1}^n i$.

```
> n = 30; S = 0  
> for(i in 1:n) { S=S+i }  
> S
```

```
[1] 465
```


La boucle `for`

On désire calculer les valeurs du vecteur x de dimension n dont les composantes sont définies par l'expression suivante:

$$x(i) = \sum_{j=1}^m \exp(i) \log(j^2) \quad i = 1, \dots, n$$

Solution avec les boucles `for`:

La boucle for

On désire calculer les valeurs du vecteur x de dimension n dont les composantes sont définies par l'expression suivante:

$$x(i) = \sum_{j=1}^m \exp(i) \log(j^2) \quad i = 1, \dots, n$$

Solution avec les boucles for:

```
> n=5; m=20
> x <- rep(0,n)
> for(i in 1:n){
  for(j in 1:m){
    x[i] <- x[i] + exp(i)*log(j^2)
  }
}
```

Solution sans les boucles for:

La boucle for

On désire calculer les valeurs du vecteur x de dimension n dont les composantes sont définies par l'expression suivante:

$$x(i) = \sum_{j=1}^m \exp(i) \log(j^2) \quad i = 1, \dots, n$$

Solution avec les boucles for:

```
> n=5; m=20
> x <- rep(0,n)
> for(i in 1:n){
  for(j in 1:m){
    x[i] <- x[i] + exp(i)*log(j^2)
  }
}
```

Solution sans les boucles for:

```
> n=5; m=20
> x <- exp(1:n)*sum(log((1:m)^2))
```

La boucle `while`

La boucle est exécutée tant que la condition qui suit la commande `while` (tant que) est vraie.
Sa syntaxe est:

```
while (condition) { liste des instructions }
```

La boucle `while`

La boucle est exécutée tant que la condition qui suit la commande `while` (tant que) est vraie.
Sa syntaxe est:

```
while (condition) { liste des instructions }
```

Voici des exemples:

```
> N = 3; i = 1  
> while (i <= N) { print(i); i=i+1 }
```

```
[1] 1
```

```
[1] 2
```

```
[1] 3
```

La boucle `while`

La boucle est exécutée tant que la condition qui suit la commande `while` (tant que) est vraie.
Sa syntaxe est:

```
while (condition) { liste des instructions }
```

Voici des exemples:

```
> N = 3; i = 1  
> while (i <= N) { print(i); i=i+1 }
```

```
[1] 1  
[1] 2  
[1] 3
```

```
> x = 0 ; y = 1  
> while (x+y <= 6) { x <- x+y }  
> x
```

```
[1] 6
```

La boucle `while`

Calculer la somme $S = \sum_{i=1}^n i$, en utilisant l'instruction `while` ?

La boucle while

Calculer la somme $S = \sum_{i=1}^n i$, en utilisant l'instruction while ?

```
> n = 30; S = 0; i = 1  
> while (i <= n) { S=S+i; i=i+1 }  
> S
```

```
[1] 465
```


La boucle `repeat`

La boucle `repeat` (répéter) est arrêtée si la condition qui suit la commande `if` (si) est vraie.
En général sa syntaxe est:

La boucle repeat

La boucle `repeat` (répéter) est arrêtée si la condition qui suit la commande `if` (si) est vraie. En général sa syntaxe est:

```
repeat {  
  liste des instructions  
  if (condition) break  
}
```

La boucle repeat

La boucle `repeat` (répéter) est arrêtée si la condition qui suit la commande `if` (si) est vraie. En général sa syntaxe est:

```
repeat {  
  liste des instructions  
  if (condition) break  
}
```

Voici un exemple simple:

```
> i <- 1  
> repeat {  
+   print(i); i <- i+1  
+   if (i==4) break  
+ }
```

La boucle repeat

La boucle `repeat` (répéter) est arrêtée si la condition qui suit la commande `if` (si) est vraie. En général sa syntaxe est:

```
repeat {  
  liste des instructions  
  if (condition) break  
}
```

Voici un exemple simple:

```
> i <- 1  
> repeat {  
+   print(i); i <- i+1  
+   if (i==4) break  
+ }
```

```
[1] 1  
[1] 2  
[1] 3
```

La boucle repeat

Calculer la somme $S = \sum_{i=1}^n i$, en utilisant l'instruction `repeat` ?

La boucle repeat


Calculer la somme $S = \sum_{i=1}^n i$, en utilisant l'instruction `repeat` ?

```
> n = 30; S = 0; i = 1  
> repeat {  
+ S = S+i; i = i+1  
+ if (n < i) break  
+ }  
> S
```


```
[1] 465
```

Lois de probabilité

Lois de probabilité

-  a quatre fonctions principales pour travailler avec les lois de probabilité. Chaque fonction a un préfixe d'une lettre, suivi du nom de la distribution avec laquelle nous voulons travailler.

Préfixe	Description
d	Masse/ densité de probabilité
p	Fonction de répartition
q	Quantiles
r	Générateur aléatoire

- Parmi les lois de probabilité supportées par , on cite:

Suffixe	Loi
binom	Binomiale
geom	Géométrique
pois	Poisson
nbinom	Binomiale négative

Suffixe	Loi
unif	Uniforme
norm	Normale
exp	Exponentielle
gamma	Gamma

Lois de probabilité

- Calculer $P(X = 5)$, où $X \sim \mathcal{B}(10, 0.8)$

```
> dbinom(5, 10, 0.8)
```

```
[1] 0.02642412
```

Lois de probabilité

- Calculer $P(X = 5)$, où $X \sim \mathcal{B}(10, 0.8)$

```
> dbinom(5, 10, 0.8)
```

```
[1] 0.02642412
```

- Calculer $P(X < 1.20)$ où $X \sim \mathcal{N}(0, 1)$

Lois de probabilité

- Calculer $P(X = 5)$, où $X \sim \mathcal{B}(10, 0.8)$

```
> dbinom(5,10,0.8)
```

```
[1] 0.02642412
```

- Calculer $P(X < 1.20)$ où $X \sim \mathcal{N}(0, 1)$

```
> pnorm(1.20,mean=0,sd=1)
```

```
[1] 0.8849303
```

Lois de probabilité

- Trouver x tel que $P(X < x) = 0.975$, et $X \sim \mathcal{N}(0, 1)$.

Lois de probabilité

- Trouver x tel que $P(X < x) = 0.975$, et $X \sim \mathcal{N}(0, 1)$.

```
> qnorm(0.975, mean=0, sd=1)
```

```
[1] 1.959964
```

Lois de probabilité

- Trouver x tel que $P(X < x) = 0.975$, et $X \sim \mathcal{N}(0, 1)$.

```
> qnorm(0.975, mean=0, sd=1)
```

```
[1] 1.959964
```

- Générer 10 observation d'une loi $\mathcal{P}(4)$.

```
> rpois(10, 4)
```

```
[1] 4 5 2 3 3 4 5 8 3 6
```