Initiation au langage R

R est un langage de programmation open source et largement utilisé pour le traitement et l'analyse statistique des données. Il est spécialement utilisé pour l'analyse statistique, la visualisation et la modélisation de données.

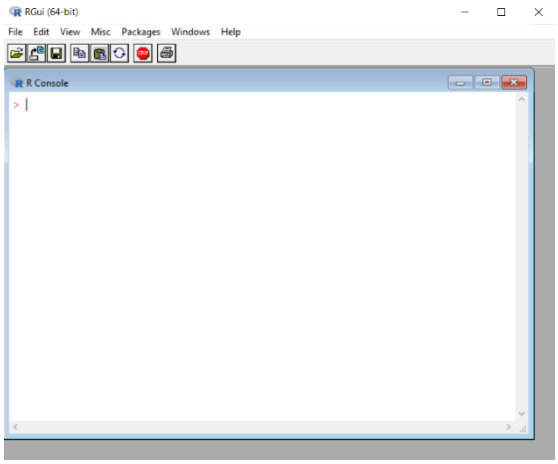
1. Installation

1.1. Installation de l'environnement R

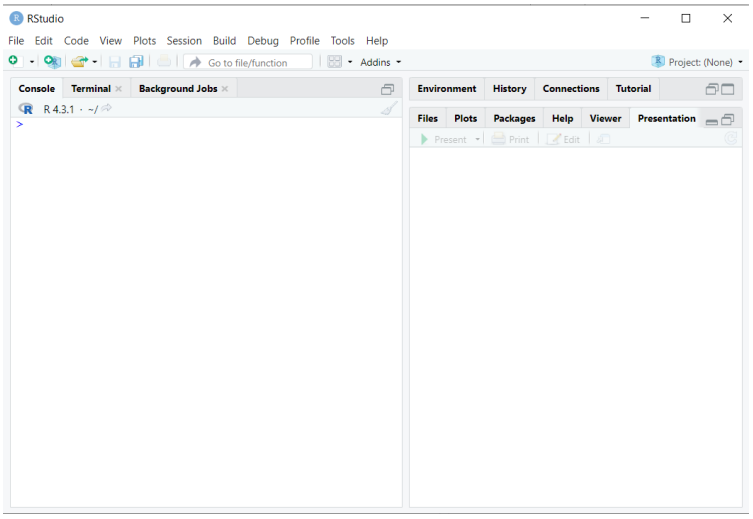
- Vous pouvez télécharger la dernière version de R à partir du site officiel : https://www.r-project.org/
- Une fois le fichier d'installation téléchargé, exécutez-le et suivez les instructions du programme d'installation.
- L'environnement de base de R est généralement une console R qui permet d'entrer et d'exécuter des commandes R directement. Il fournit un environnement de base pour exécuter des scripts et des commandes R.

1.2. Installation de l'environnement RStudio

- RStudio est un environnement de développement intégré (IDE) pour R qui fournit une interface utilisateur conviviale pour écrire, exécuter, déboguer et visualiser du code R.
- Vous pouvez télécharger l'IDE RStudio à partir du site officiel : https://posit.co/products/open-source/rstudio/



Interface de l'environnement de base de R



Interface de l'environnement RStudio

1.3. Installation de packages R

Les packages R sont des collections de fonctions, de données et de documentation qui peuvent être installées et chargées dans R pour ajouter de nouvelles fonctionnalités ou améliorer les fonctionnalités existantes.
Pour pouvoir utiliser un package il faut l'importer à l'aide du code **`library(nom_du_package)`**

Exemple :
```
library(expm)
```

Il existe deux façons d'installer des packages R : online et offline

1.3.1. *Installation online*

Dans R :

- Barre de menus > packages > install packages..
- Dans la boite de dialogue qui s'affiche, sélectionner le package à installer puis cliquer sur « OK »

Dans RStudio :

- Barre de menus > tools > install packages..
- Dans la boite de dialogue qui s'affiche, rechercher le package à installer puis cliquer sur « OK »

1.3.2. *Installation offline*

- Télécharger le package sur le site officiel CRAN. Exemple : le lien de télechargement du package expm : https://cran.r-project.org/bin/windows/contrib/4.4/expm_0.999-7.zip
- Dans R : Barre de menus > packages > install packages from local files..
- Dans la boite de dialogue qui s'affiche, sélectionner le fichier téléchargé puis clique sur « OK »

2. Concepts de base de R

On utilise généralement **R** interactivement, selon un cycle question-et-réponse :

- Vous entrez une commande et tapez la touche "Enter".
- **R** exécute cette commande (avec affichage d'un résultat si besoin)

R attend une autre commande

2.1. Types de données en R

- **Nombres (numériques) :** Ils représentent les valeurs numériques et peuvent être des entiers ou des nombres à virgule flottante.
- **Caractères** : Ils représentent des chaînes de texte, délimitées par des guillemets " ".
- **Logiques** : Ils représentent des valeurs booléennes (FALSE ou TRUE) utilisées pour l'évaluation conditionnelle.
- etc.

2.2. Structures de données en R

- **Vecteurs** : Ils sont des séquences ordonnées d'éléments de même type.
- **Matrices** : Elles sont des tableaux bidimensionnels de valeurs du même type.
- etc.

2.3. Opérations de base

Opérateur	Fonction
<- ou =	Assignation
* / + -	Multiplication division addition soustraction
< <= == >= > !=	plus petit, plus petit ou égal, égal, plus grand ou égal, plus grand, différent de
^	Puissance
:	Génération de suites
%*%	Produit matriciel
%%	Modulo
%/%	Division entière
!	Négation logique
& &&	« et » logique
| ||	« ou » logique
<-	assignation

2.4. Fonctions courantes de R

Fonction	*Description*
Fonctions élémentaires	
`sqrt(x)`	Racine carrée de x.
`abs(x)`	Valeur absolue de x.
`round(x, digits = n)`	Arrondi de x à n décimales.
`exp(x)`	Exponentielle de x.
`log(x)`	Logarithme naturel de x.
`log10(x)`	Logarithme base 10 de x.
Fonctions trigonométriques	
`sin(x)`	Sinus de x (en radians).
`cos(x)`	Cosinus de x (en radians).
`tan(x)`	Tangente de x (en radians).
`asin(x)`	Arc sinus de x (en radians).
`acos(x)`	Arc cosinus de x (en radians).
`atan(x)`	Arc tangente de x (en radians).
Fonctions statistiques	
`sum(x)`	Somme des éléments de x.
`prod(x)`	Produit des éléments de x.
`min(x, y, ...)`	Valeur minimale parmi x, y, etc.
`max(x, y, ...)`	Valeur maximale parmi x, y, etc.
`mean(x)`	Moyenne des éléments de x.
`median(x)`	Médiane des éléments de x.
`var(x)`	Variance des éléments de x.
`sd(x)`	Écart type des éléments de x.

3. Vecteurs

Les vecteurs sont des structures de données utilisées pour stocker des éléments de données de même type. Les vecteurs est essentiel pour effectuer des opérations de base et avancées sur les données dans R.

3.1. Création de vecteurs

Différentes façons pour créer des vecteurs en R :

- Utilisation de la fonction **`c()`** pour créer des vecteurs.
- Création de séquences de nombres avec la fonction **`seq().`**
- Génération de vecteurs aléatoires avec les fonctions **`sample(), runif(),`** etc.
- etc.

Exemple

```
#Création d'un vecteur numérique
numeric_vector <- c(1, 2, 3, 4, 5)

#Création d'un vecteur de caractères
character_vector <- c("apple", "banana", "cherry", "date", "elderberry")

# Création d'un vecteur logique
logical_vector <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

# Création d'une séquence de nombres
sequence_vector <- seq(1, 10, by = 2)
```

3.2. Accès et modification aux éléments d'un vecteur

```
# Création d'un vecteur
v <- c(10, 20, 30, 40, 50)

# Accès aux éléments individuels d'un vecteur
v[1]  # Accès au premier élément du vecteur
v[2]  # Accès au deuxieme élément du vecteur

# Utilisation de plages d'indices pour accéder à des sous-ensembles de vecteurs
v[2:4]   # Accès aux éléments du deuxième au quatrième indice du vecteur

# Modification des éléments d'un vecteur
v[3] <- 35  # Modification du troisième élément du vecteur

# Suppression d'éléments d'un vecteur avec la fonction subset()
subset_v <- subset(v, v > 30)  # Extraction des éléments du vecteur supérieurs à 30
```

3.3. Fonctions de manipulation de vecteurs

Fonction	Description
`length(x)`	Renvoie la longueur du vecteur x.
`sum(x)`	Calcule la somme des éléments du vecteur x.
`prod(x)`	Calcule le produit des éléments du vecteur x.
`mean(x)`	Calcule la moyenne arithmétique des éléments du vecteur x.
`median(x)`	Calcule la médiane des éléments du vecteur x.
`min(x)`	Renvoie la valeur minimale du vecteur x.
`max(x)`	Renvoie la valeur maximale du vecteur x.
`range(x)`	Renvoie la plage des valeurs du vecteur x (min et max).
`sort(x)`	Trie les éléments du vecteur x par ordre croissant.
`rev(x)`	Inverse l'ordre des éléments du vecteur x.
`seq(from = d, to = f, by = i)`	Genere un vecteur de valeurs de d à f avec incrémentation i
`sample(x, size, replace = FALSE)`	Sélectionne aléatoirement des éléments du vecteur x.
`which(x == value)`	Renvoie les indices où les éléments du vecteur x sont égaux à la valeur spécifiée.

4. Matrices

4.1. Création et manipulation de matrices

La fonction **`matrix(data, nrow, ncol)`**: crée une matrice avec les données **`data`** de dimensions **`nrow`** (nombre de lignes) **`ncol`** (nombre de colonnes).

Exemple (création) :

```
# Création d'une matrice m
m<-matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
print(m)
 [,1] [,2] [,3]
 [1,] 1 3 5
 [2,] 2 4 6
# Création d'une matrice n. On spécifie uniquement le nombre de lignes
n<-matrix(c(1,2,3,4,5,6), nrow = 3)
print(n)
 [,1] [,2]
 [1,] 1 4
 [2,] 2 5
 [3,] 3 6
```

Exemple (accès) :

```
# Création d'une matrice
m <- matrix(1:12, nrow = 3, ncol = 4)

# Accès aux éléments d'une matrice
print(m[2, 3])  # Accès à l'élément à la 2ème ligne et 3ème colonne

# Utilisation de plages d'indices pour extraire des sous-matrices
print(m[1:2, 2:4])  # Extraction d'une sous-matrice des lignes 1 à 2 et
des colonnes 2 à 4
```

```
# Modification des éléments d'une matrice
m[3, 4] <- 20  # Modification de l'élément à la 3ème ligne et 4ème colonne
```

```
# Création d'une matrice
m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

# m[,i]: renvoie la colonne i de la matrice m.
m[,1]
[1] 1 2
> m[,2]
[1] 3 4
> m[,3]
[1] 5 6

# m [i,]: renvoie la ligne i de la matrice m.
> m [1,]
[1] 1 3 5
# m[,c(1,2)]: renvoie les deux premières colonnes de la matrice m.
> m[,c(1,2)]
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Remarque. pour calculer la puissance matricielle, il est possible d'utiliser l'opération `%^%` du package `expm`

Exemple

```
>library(expm)
> p=matrix(c(0.9,0.15,0.25,0.075,0.8,0.25,0.025,0.05,0.5), nrow = 3, ncol
= 3)
> p
     [,1]  [,2]  [,3]
[1,] 0.90 0.075 0.025
[2,] 0.15 0.800 0.050
[3,] 0.25 0.250 0.500
> p%^%100
      [,1]   [,2]   [,3]
[1,] 0.625 0.3125 0.0625
[2,] 0.625 0.3125 0.0625
[3,] 0.625 0.3125 0.0625
```

4.2. Fonctions de manipulation de matrices

Fonction	Description
dim()	Retourne les dimensions de la matrice.
nrow()	Retourne le nombre de lignes de la matrice.
ncol()	Retourne le nombre de colonnes de la matrice.
rownames()	Retourne ou assigne les noms des lignes de la matrice.
colnames()	Retourne ou assigne les noms des colonnes de la matrice.
cbind()	Concatène des matrices ou des vecteurs par colonne.
rbind()	Concatène des matrices ou des vecteurs par ligne.
t()	Transpose la matrice.
solve()	Calcule l'inverse ou la pseudo-inverse de la matrice.
det()	Calcule le déterminant de la matrice.
colSums()	Calcule la somme des colonnes de la matrice.
rowSums()	Calcule la somme des lignes de la matrice.
colMeans()	Calcule la moyenne des colonnes de la matrice.
rowMeans()	Calcule la moyenne des lignes de la matrice.
max()	Retourne la valeur maximale de la matrice.
min()	Retourne la valeur minimale de la matrice.
sort()	Trie les éléments de la matrice.
…	

Exemple

```
# Création d'une matrice
m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

# nrow, ncol: nombre de lignes et de colonnes d'une matrice
> nrow(m); ncol(m)
[1] 2
[1] 3

# rowSums, colSums: sommes par ligne et par colonne, respectivement, des éléments d'une matrice
> rowSums(m);colSums(m)
[1] 9 12
[1] 3 7 11

# rowMeans, colMeans: moyennes par ligne et par colonne, respectivement, des éléments d'une matrice
> colMeans(x)
[1] 1.5 3.5
#  dim(m): renvoie les dimensions de la matrice m.
> dim(m)
[1] 2 3
```

5. Structures de contrôle

5.1. Structures conditionnelles

Syntaxes

```
if(condition) {
 # code qui s'exécute si condition est TRUE
}
```

```
if(condition) {
 # code qui s'exécute si condition est TRUE
}else {
 # code qui s'exécute si condition est FALSE
}
```

Exemple

```
a = 25
if(a %% 2 ==0) {
 print("Nombre pair")
} else {
 print("Nombre imapir")
}
```

6. Les structures itératives

6.1. La boucle for

Syntaxe

```
for(e in range) {
 #instructions à exécuter
}
```

Exemples :

```
n = 5
fact = 1
for(i in 1:n) {
 fact = fact * i
}
print (fact)
for(e in c(1,2,3,4,5)) {
 print(e)
}
```

```
v<-c(1,2,3,4,5)
sum<-0
for(i in v) {
 sum = sum + v[i]
 i<-i+1
}
print (sum)
```

6.2. La boucle while

Syntaxe

```
while(e in range) {
 #instructions à exécuter
}
```

Exemple

```
n = 5
fact = 1
i=1
while(i <=n) {
 fact = fact * i
 i<-i+1
}
print (fact)
```

7. Génération de lois de probabilités

Distribution	Fonction R	Description
Uniforme	**runif(n, min, max)**	Génère des nombres aléatoires uniformément distribués.
Normale	**rnorm(n, mean, sd)**	Génère des nombres aléatoires selon une distribution normale.
Binomiale	rbinom(n, size, prob)	Génère des nombres aléatoires selon une distribution binomiale.
Poisson	rpois(n, lambda)	Génère des nombres aléatoires selon une distribution de Poisson.
Exponentielle	rexp(n, rate)	Génère des nombres aléatoires selon une distribution exponentielle.
Gamma	rgamma(n, shape, rate)	Génère des nombres aléatoires selon une distribution gamma.
Beta	rbeta(n, shape1, shape2)	Génère des nombres aléatoires selon une distribution bêta.
Log-normale	rlnorm(n, meanlog, sdlog)	Génère des nombres aléatoires selon une distribution log-normale.
Weibull	rweibull(n, shape, scale)	Génère des nombres aléatoires selon une distribution de Weibull.

Exemples

```
# Générer 20 nombres uniforme entre 5 et 10
runif(20, min = 5, max = 10)
# Générer 20 nombres de distribution normale avec une moyenne de 2 et un écart-type de
0.5
rnorm(20, mean = 2, sd = 0.5)
# Générer 20 nombres de distribution exponentielle avec un taux de 0.2
rexp(20, rate = 0.2)
```

8. Graphiques

R offre de nombreuses fonctions pour créer une variété de graphiques :

8.1. Fonction plot()

La fonction plot() est utilisée pour créer un large éventail de graphiques de base. Elle est utilisée pour afficher des nuages de points, des courbes, des histogrammes, etc.

Paramètres courants :

- **x :** Vecteur contenant les valeurs de l'axe des abscisses.
- **y :** Vecteur contenant les valeurs de l'axe des ordonnées.
- Paramètres optionnels :
- **type :** Type de graphique. Les valeurs possibles sont : "p" pour un graphique en points, "l" pour un graphique en lignes, "b" pour un graphique en points et en lignes, "h" pour un histogramme, etc.
- **main** : Titre principal du graphique. Exemple : main = "Titre du graphique"
- **xlab** : Étiquette de l'axe des abscisses. **xlab**= " Étiquette de l'axe X"
- **ylab** : Étiquette de l'axe des ordonnées. **ylab**= " Étiquette de l'axe Y"
- **xlim** : Limite de l'axe des abscisses. Exemple : **xlim = c(-3,3)**
- **ylim** : Limite de l'axe des ordonnées. Exemple : **xlim = c(0,1)**
- **col** : Couleur des points ou des lignes. Exemple : col = "red"
- **pch** : Code de forme des points (valeurs de 0 à 25) . Exemple : pch= 15 des point carrés, pch= 15 pour des point cercles, pch= 17 pour des point triangles, etc.
- **lty** : Type de ligne. **Exemple** : lty = 1 pour une ligne continue, lty = 2 pour une ligne en tirets, lty = 3 pour une ligne en points, etc.
- **lwd** : Épaisseur de la ligne. **Exemple :** lwd = 2

8.2. La fonction lines()

La fonction ***lines()*** est utilisée pour ajouter des lignes à un graphique. Elle prend deux arguments obligatoires :
x: un vecteur contenant les abscisses des points à tracer.
y: un vecteur contenant les ordonnées des points à tracer.
La fonction lines()prend également plusieurs arguments optionnels qui permettent de contrôler l'apparence de la ligne : ***type***, ***col***, ***lty***, etc.

8.3. Fonction axis()

La fonction **axis() est u**tilisée pour ajouter ou modifier des axes dans un graphique : personnaliser la position des graduations, les couleurs, les styles de ligne, etc. Voici les principaux paramètres de la fonction `axis()` :

- **side** : un entier spécifiant de quel côté l'axe doit être dessiné. L'axe est placé comme suit : 1 (en bas), 2 (à gauche), ...
- **at** : Un vecteur spécifiant les positions des graduations sur l'axe.
- **labels** : Un vecteur de chaînes de caractères spécifiant les étiquettes à afficher aux positions spécifiées par at.
- **pos** : La position de l'axe spécifiée en termes de la position du côté du graphique.

8.4. Fonction *legend()*

La fonction "legend()" permet d'ajouter une boîte de légende à un graphique, fournissant des informations sur les différentes séries de données ou éléments représentés dans le graphique.

Exemple

```
x <- seq(-3, 3, by = 0.01) # generation
y<- exp(-x^2)
y1<- exp(x)
x2<- seq(0, 3, by = 0.01)
y2<- log(x2)
plot(x, y,
 main = "Exemples de graphiques",
 xlab = "x",
 ylab = "y = f(x)",
       type = "l",
 xlim = c(-3, 3), # Limite de l'axe des abscisses
 ylim = c(-3, 3), # Limite de l'axe des ordonnées
 lwd = 2 # Largeur de ligne

 )
lines(x, y1,
 type = "l", # Type de graphique (points et lignes)
 col = "blue", # Couleur des points et des lignes
 lwd = 2, # Largeur de ligne
 )
lines(x2, y2,
 type = "l", # Type de graphique (points et lignes)
 col = "red", # Couleur des points et des lignes
 lwd = 2, # Largeur de ligne
 )

# Ajout des axes personnalisés se croisant au point 0 avec valeurs
axis(side = 1, at = seq(-3, 3, by = 1.0), labels = seq(-3, 3, by = 1.0), pos = 0)
axis(side = 2, at = seq(-3, 3, by = 1.0), labels = seq(-3, 3, by = 1.0), pos = 0)
legend("topleft", # Position de la légende dans le graphique:topright, bottomright, topleft, bottomleft,…
 legend = c("y = exp(-x^2)", "y = exp(x)", "y = log(x)"), # Noms des séries de données
 col = c("black", "blue", "red"), # Couleurs des lignes dans la légende
 lty = 1, # Type de ligne pour chaque entrée dans la légende
 lwd = 2) # Épaisseur de ligne pour chaque entrée dans la légende
```