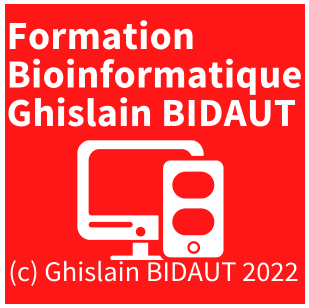


Introduction à R et Bioconductor

Ghislain Bidaut, Plateforme Cibi, CRCM, Aix-Marseille Université

24/03/2022

Introduction à R et Bioconductor

- Qu'est ce que **R** ?

R est à la fois un **langage de programmation** et un **environnement d'analyse de données**. Il est issu du monde de la statistique car librement inspiré du langage **S** développé par *Bell Labs* dans les années 70.

Il a été développé par Ross Ithaca et Robert Gentleman (Ithaca & Gentleman, 1996) et intégré au projet **GNU**. **R** est donc également un **Logiciel Libre**.

Il a progressivement remplacé **S-PLUS** et connaît maintenant un très grand nombre de domaines d'applications.

- Qu'est ce que **Bioconductor** ?

Bioconductor est un projet de développement de librairies dédiées à **l'analyse de données en biologie** pour **R**. Il est centré autour du site http://www.bioconductor.org

Introduction à R

Description

- R est **interprété** (pas de compilation)
- Il est particulièrement puissant pour les applications **mathématiques et statistiques**
- Il est basé sur la manipulation de **vecteurs** ce qui encourage d'**éviter** le recours aux boucles (de type for, while, …)
- Il ne demande pas de **typage** des variables
- De par sa puissance, il permet de développer des programmes avec seulement quelques lignes de code

Environnement et utilisation

Nous utilisons l'environnement **RStudio**, qui est une interface développée indépendamment de R. **RStudio** existe sous les principaux systèmes d'exploitation et consiste en un IDE complet comprenant:

- **Invite de commande R**
- Un **éditeur de texte** avec coloration syntaxique
- Un éditeur de différents formats de documents basées sous R (**Rapports** et **présentations** (**RMarkDown**)
- Un terminal *bash*
- Une aide **R intégrée**
- Un gestionnaire d'**historique**
- Un gestionnaire d'**objets R**
- Un **explorateur de fichiers**

Invite de commande

- On entre des instruction directement à l'invite de commande

```
(base) 19:41:14-bidaut@afterglow:~$ R

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

>
```

Stratégie de travail basique

On rentre directement les expression pour les valider directement:

```
2 + 3
```

```
## [1] 5
```

On peut assigner des valeurs à des **variables** grâce au symbole d'assignation **<-** (=marche aussi)

```
x <- 2^2
x
```

```
## [1] 4
```

```
y <- 2.5+4
y
```

```
## [1] 6.5
```

Fonctions

Il existe des fonctions préprogrammée. Ici est un exemple d'utilisation de la fonction *exponentielle*, que l'on invoque avec `exp()`.

```
x <- exp(2)
x
```

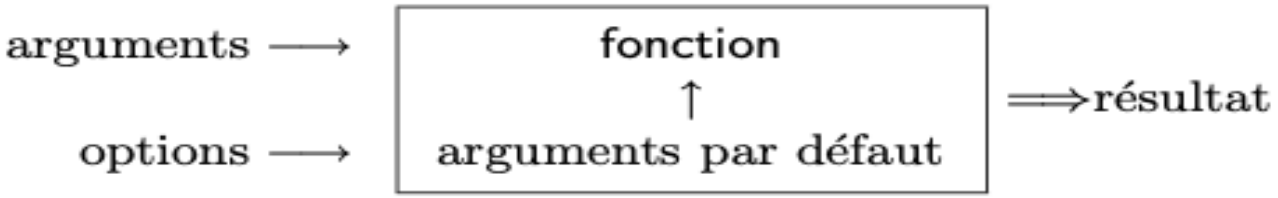
```
## [1] 7.389056
```

Si on tape le nom de la fonction sans parenthèses, R renvoie son contenu, c'est à dire l'objet qui contient le code de la fonction.

```
exp
```

```
## function (x)  .Primitive("exp")
```

Fonctions



Les **fonctions** prennent en entrée des *objets* de manière optionnelle. On peut également spécifier des **options**. Les options ont des valeurs par défaut. Les fonctions renvoient un **objet** en résultat. Les fonctions sont elle-même des **objets**. Un **opérateur** fait référence aux opérateurs mathématiques, tels que + qui ne sont pas invoqués avec des parenthèses.

Exemple: la fonction **ls** permet de lister les objets en mémoire:

```
ls()
```

```
## [1] "x" "y"
```

Fonction ls

```
ls(pattern = "x")
```

```
## [1] "x"
```

```
m = data.frame(x, y)
```

```
ls.str()
```

```
## m : 'data.frame':    1 obs. of  2 variables:
##  $ x: num 7.39
##  $ y: num 6.5
## x :  num 7.39
## y :  num 6.5
```

Aide

Aide en ligne pour la fonction *lm* (linear model)

```
?lm
```

Cet appel ouvre une page d'aide dans l'onglet "help" de RStudio, contenant les sections suivantes:

- **Description**: Description brève
- **Usage**: Pour une fonction, donne le nom avec ses arguments et les éventuelles options (et leurs valeurs par défaut)
- **Arguments**: Liste des arguments de la fonction
- **Details**: Description détaillée
- **Value**: Le type d'objet éventuellement retourné par la fonction ou l'opérateur

Aide

- **See Also**: D'autres rubriques proches ou similaire
- **Examples**. Des exemples qui peuvent être généralement exécutés

La fonction `example`:

```
example(ls)
```

```
##
## ls> .Ob <- 1
##
## ls> ls(pattern = "O")
## character(0)
##
## ls> ls(pattern= "O", all.names = TRUE)    # also shows ".[foo]"
## [1] ".Ob"
##
## ls> # shows an empty list because inside myfunc no variables are defined
## ls> myfunc <- function() {ls()}
##
## ls> myfunc()
## character(0)
##
## ls> # define a local variable inside myfunc
## ls> myfunc <- function() {y <- 1; ls()}
##
## ls> myfunc()                # shows "y"
## [1] "y"
```

Aide

Recherche de l'aide contextuelle:

- Soit directement sous RStudio
- Soit en ligne de commande:

```
help("bs")
```

```
## No documentation for 'bs' in specified packages and libraries:
## you could try '??bs'
```

```
help("bs", try.all.packages = TRUE)
```

```
## Help for topic 'bs' is not in any loaded package but can be found in
## the following packages:
##
##   Package               Bibliothèque
##   splines               /Library/Frameworks/R.framework/Versions/4.1/Resources/library
```

```
help.start()
```

Données sous R

Objets et attributs.

Considérons une variable qui prendrait les valeurs 1,2 ou 3.

- Est-ce que cette variable est un entier ?
- Est ce que cette variable est le codage d'une variable catégorielle ?

Deux attributs intrinsèque: *mode* et *longeur*:

- Le **Mode**: `numeric`, `character`, `complex`, `logical`, `function`, `list`, `expression`
- La **Longueur**: Un entier correspondant au nombre d'éléments de l'objet

List ou *expression* sont des objets *récursifs* (Qui peuvent contenir eux même d'autres objets).

Données sous R

Un peu d'exploration…

```
x <- 1
mode(x)
```

```
## [1] "numeric"
```

```
A <- "Linux"; compar <- TRUE; z <- 1i
mode(A); mode(compar); mode(z)
```

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

```
## [1] "logical"
```

```
## [1] "complex"
```

Représentations des valeurs numériques

```
n <- 1
x <- 1.3
X <- 2.5e25

n
```

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

```
x
```

```
## [1] 1.3
```

```
X
```

```
## [1] 2.5e+25
```

Représentation des valeurs infinies

```
x <- 1/0
x
```

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

```
exp(x)
```

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

```
exp(-x)
```

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

```
x - x
```

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

Représentations des caractères

```
my_c <- "Ceci est une valeur en mode caractère."
length(my_c)
```

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

```
nchar(my_c)
```

```
## [1] 38
```

Caractère d'échappement

Comme en *bash*, on a la notion de caractère d'échappement:

```
my_c <- "Ceci est un guillemet double: \"."
cat(my_c)
```

```
## Ceci est un guillemet double: ".
```

On peut utiliser aussi des guillemets simples:

```
my_c <- 'Ceci est un guillemet double: "'
cat(my_c)
```

```
## Ceci est un guillemet double: ".
```

Noms de valeurs à éviter

- Il vaut **éviter l'utilisation de caractères accentués** pour les noms de variable (et ce, quel que soit le langage !)
- Les noms d'objet ne **peuvent pas commencer par un chiffre**.
- R est **sensible à la casse**, donc **truc**, **Truc** et **TRUC** sont trois objets distincts.
- Les noms suivants **sont utilisés par R**, donc à éviter: **c, q, t, C, D, I, diff, length, mean, pi, range, var**.
- Les mots suivants **sont réservés** et il est interdit de les utiliser comme nom d'objet:
  - `break`, `else`, `for`, `function`, `if`, `in`, `next`, `repeat`, `return`, `while`
  - `TRUE`, `FALSE`
  - `Inf`, `NA`, `NaN`, `NULL`
  - `NA_integer_`, `NA_real_`, `NA_complex_`, `NA_character_`
  - `...`, `..1`, `..2`, etc

TRUE et FALSE

Les variables T et F sont affectés par défaut à TRUE et FALSE.

```
T
```

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

```
F
```

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

```
T <- 3 # A NE PAS FAIRE /!\
T
```

```
## [1] 3
```

```
TRUE <- 3
Error in TRUE <- 3 : membre gauche
de l'assignation
(do_set) incorrect
```

Lecture de données d'un fichier

Répertoires de travail

```
setwd(dir = "/Users/bidaut/data/")
getwd()
```

```
## [1] "/Users/bidaut/data"
```

Lecture de fichiers texte et importation dans l'environnement R

La fonction la plus commune est `read.table`

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names,
           col.names, as.is = FALSE, na.strings = "NA", colClasses = NA, nrows = -1,
          skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE,
          blank.lines.skip = TRUE, comment.char = "#")
```

Exemple: un fichier de description d'experimentation RNA-seq

```
targets <- read.table(file='targets_GSE130657.txt', header = T)
targets
```

```
##                     SRA_ID     GEO_ID CellType Sample
## 1 SRR9005674.trim.fastq.gz GSM3746500     THP0      1
## 2 SRR9005675.trim.fastq.gz GSM3746501     THP0      2
## 3 SRR9005676.trim.fastq.gz GSM3746502     THP0      3
## 4 SRR9005677.trim.fastq.gz GSM3746503        M      1
## 5 SRR9005678.trim.fastq.gz GSM3746504        M      2
## 6 SRR9005679.trim.fastq.gz GSM3746505        M      3
```

Enregistrement de données dans un fichier

Enregistrement d'un tableau dans un fichier texte délimité.

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
                        eol = "\n", na = "NA", dec = ".", row.names = TRUE,
                        col.names = TRUE, qmethod = c("escape", "double")))
```

Enregistrement d'un objet dans un fichier de type `.RData`.

```
save(x, y, z, file="xyz.RData")
```

Générér des données

Générer une séquence régulière

```
x <- 1:30
```

Plus de paramètres

```
seq(1, 5, 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```
seq(length=9, from=1, to=5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Fonction `c`

```
c(1,3,12,15)
```

```
## [1]  1  3 12 15
```

Répétition d'éléments

```
rep(1, 30)
```

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

Création d'une suite de séquences

```
sequence(5:4)
```

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

```
sequence(c(10, 5))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5
```

Génération de niveaux (gl)

`gl(k,n)` génère **k** niveaux de données dans un facteur répétées **n** fois.

```
gl(3,5)
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
## Levels: 1 2 3
```

```
gl(3,5, length = 30)
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
## Levels: 1 2 3
```

```
gl(2,6, label=c("Male", "Femelle"))
```

```
##  [1] Male    Male    Male    Male    Male    Male    Femelle Femelle Femelle
## [10] Femelle Femelle Femelle
## Levels: Male Femelle
```

Création d'un tableau avec toutes les combinaisons

Fonction `expand.grid`:

```
expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
```

```
##    h   w    sex
## 1 60 100   Male
## 2 80 100   Male
## 3 60 300   Male
## 4 80 300   Male
## 5 60 100 Female
## 6 80 100 Female
## 7 60 300 Female
## 8 80 300 Female
```

Générer des données aléatoires

R permet de générer des données aléatoires à partir d'un grand nombre de lois de densité de probabilités.

Loi	Fonction
Gauss(normale)	`rnorm(n, mean=0, sd=1)`
exponentielle	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpos(n, lambda)`
binomiale	`rbinom(n, size, prob)`
hypergéométrique	`rhyper(nn, m, n, k)`

Générer des données aléatoires

Ces fonctions ont des compagnes obtenues en replaçant la lettre `r` par `d`, `p` ou `c` pour obtenir la **densité**, la **densité de probabilité cumulée**, et la **valeur de quantile**.

Par exemple, pour obtenir les valeurs critiques au seuil de 5% pour un test bilatéral suivant une loi normale:

```
qnorm(0.025)
```

```
## [1] -1.959964
```

```
qnorm(0.975)
```

```
## [1] 1.959964
```

Créer des objets: Vecteurs

```
vector(mode = "numeric", 5)
```

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

```
vector(mode = "logical", 5)
```

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

```
vector(mode = "character", 5)
```

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

Créer des objets: Facteurs

```
factor(1:3)
```

```
## [1] 1 2 3
## Levels: 1 2 3
```

```
factor(1:3, levels = 1:5)
```

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

```
factor(1:3, labels=c("A", "B", "C"))
```

```
## [1] A B C
## Levels: A B C
```

```
factor(1:5, exclude=4)
```

```
## [1] 1    2    3    <NA> 5
## Levels: 1 2 3 5
```

```
(ff <- factor(1:3, levels = 1:5))
```

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

```
levels(ff)
```

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

`

Créer des objets: Matrices

Une matrice est un vecteur qui possède un argument supplémentaire (*dim*) qui est lui même un vecteur.

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

L'option `dimname` permet de donner des noms aux lignes et colonnes. l'option `byrow` permet de définir le sens de remplissage.

```
matrix(seq(1:4), nr=2, nc=2)
```

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

```
matrix(seq(1:4), nr=2, nc=2, byrow = TRUE)
```

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

Créer des objets: Matrices

On peut aussi créer une matrice à partir d'un vecteur en assignant les valeurs voulues à `dim`.

```
x <- 1:15
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
dim(x)
```

```
## NULL
```

```
dim(x) <- c(5,3)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

Tableaux de données : Data frames

Un tableau de données (*Data frame*) peut être créé directement à partir d'un fichier par un appel à `read.table`.

On peut aussi créer un Data frame par assemblage de vecteurs. Les **noms** des objets sont repris dans les colonnes.

```
x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
data.frame(x, n)
```

```
##   x  n
## 1 1 10
## 2 2 10
## 3 3 10
## 4 4 10
```

Tableaux de données : Data frames

Si un vecteur est plus court, il sera **recyclé un nombre entier** de fois.

```
data.frame(x, M)
```

```
##   x  M
## 1 1 10
## 2 2 35
## 3 3 10
## 4 4 35
```

On peut aussi inclure des **Facteurs**.

```
ff <- factor(1:4); df <- data.frame(x, ff); summary(df)
```

```
##        x        ff
##  Min.   :1.00   1:1
##  1st Qu.:1.75   2:1
##  Median :2.50   3:1
##  Mean   :2.50   4:1
##  3rd Qu.:3.25
##  Max.   :4.00
```

Créer des objets: Listes

Une liste est créée de la même façon (fonction `list`). Il n'y a pas de contraintes sur le type des objets qui y sont inclus. Les noms des objets ne sont pas repris par défaut mais peuvent être spécifiés.

```
L1 <- list(x, y); L2 <- list(A=x, B=y)
names(L1)
```

```
## NULL
```

```
names(L2)
```

```
## [1] "A" "B"
```

Créer des objets: Séries temporelles

La fonction `ts` va créer une série temporelle à partir d'un vecteur (série temporelle simple) ou d'une matrice (série temporelle multiple).

```
ts(1:10, start = 1959)
```

```
## Time Series:
## Start = 1959
## End = 1968
## Frequency = 1
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
ts(1:47, frequency = 12, start = c(1959, 2))
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1959       1   2   3   4   5   6   7   8   9  10  11
## 1960  12  13  14  15  16  17  18  19  20  21  22  23
## 1961  24  25  26  27  28  29  30  31  32  33  34  35
## 1962  36  37  38  39  40  41  42  43  44  45  46  47
```

Créer des objets: Séries temporelles

```
ts(1:10, frequency = 4, start = c(1959,2))
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1959         1    2    3
## 1960    4    5    6    7
## 1961    8    9   10
```

```
ts(matrix(rpois(36, 5), 6, 3), start = c(1961,1), frequency = 12)
```

```
##          Series 1 Series 2 Series 3
## Jan 1961        8        3        4
## Feb 1961        3        4        5
## Mar 1961        7        3        4
## Apr 1961        2        7        7
## May 1961        3        7        5
## Jun 1961        4        6        5
```

Créer des objets: Expression

Toutes les commandes valides sont des expressions. Or, il est possible en R de stocker une expression dans un objet sans l'évaluer (fonction `expression`). Ces objets représentent des expressions littérales que l'on peut ensuite évaluer avec la fonction `eval`.

```
x <- 3; y <- 2.5; z <- 1
exp1 <- expression(x / (y + exp(z)))
exp1
```

```
## expression(x/(y + exp(z)))
```

```
eval(exp1)
```

```
## [1] 0.5749019
```

```
D(exp1, "z") # Calcul de la dérivée partielle d'exp1 en z
```

```
## -(x * exp(z)/(y + exp(z))^2)
```

Conversion d'objets

Il existe plusieurs fonctions de conversions,

- Conversions entre *modes*: **as.numeric**, **as.logical**, **as.character**
- Conversion entre objets: **as.matrix**, **as.data.frame**, **as.list**, **as.ts**:

```
(fac <- factor(c(1, 10)))
```

```
## [1] 1  10
## Levels: 1 10
```

```
as.numeric(fac)
```

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

Conversions

```
fac2 <- factor(c("Male", "Female"))
fac2
```

```
## [1] Male   Female
## Levels: Female Male
```

```
as.numeric(fac2)
```

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

Opérateurs

Il y a trois principaux types d'opérateurs dans R:

- Opérateurs arithmétiques: `+`, `-`, `*`, `/`, `^`, `%%` (modulo), `%/%` (Division entière)
- Opérateurs de comparaison: `<` `>` `<=` `>=` `==` `!=`
- Opérateurs logiques: `!` (non), `&` ou `&&` (et), | ou || (ou), `xor(x,y)` (ou exclusif)

```
x <- 0.5
0 < x & x < 1
```

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

```
x <- 1:3; y <- 1:3; z <- c(1,2,2.9)
x == y
```

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

Comparaison de valeurs numériques

```
identical(x,y)
```

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

```
all.equal(x, z)
```

```
## [1] "Mean relative difference: 0.03333333"
```

```
0.9 == (1.1 - 0.2)
```

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

```
identical(0.9, 1.1 - 0.2)
```

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

```
all.equal(0.9, 1.1 - 0.2)
```

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

```
all.equal(0.9, 1.1 - 0.2, tolerance = 1e-16)
```

Accéder aux valeurs des objets

Nous accédons de façon sélective aux éléments d'un objet par l'indexation: `x[3]`

```
x <- seq(1,5)
x[3]
```

```
## [1] 3
```

```
x[3] <- 20
x
```

```
## [1]  1  2 20  4  5
```

L'indice lui-même peut être un vecteur:

```
i <- c(1,3)
x[i]
```

```
## [1]  1 20
```

```
x <- matrix(1:6, 2,3)
x[, 3]
```

```
## [1] 5 6
```

```
x[,3, drop = FALSE]
```

```
##      [,1]
## [1,]    5
## [2,]    6
```

Accéder aux valeurs des objets

On peut accéder aux valeurs d'une matrice à l'aide d'une expression de comparaison:

```
x <- rnorm(10, 1, 4)
x [x>=1] <- 20
x
```

```
##  [1] -0.8177400 20.0000000  0.6104186 20.0000000 -2.2648174 -3.2740743
##  [7] 20.0000000 20.0000000 20.0000000  0.1288336
```

Cas des doubles crochets avec les listes

```
li <- list(x, y, z)
li[[1]][1] # extrait x puis x[1]
```

```
## [1] -0.81774
```

Accéder aux valeurs des objets avec les noms

```
x <- matrix(1:4, 2, 2)
colnames(x) = c("a", "b")
rownames(x) = c("c", "d")
x["c", "a"]
```

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

```
dimnames(x)
```

```
## [[1]]
## [1] "c" "d"
##
## [[2]]
## [1] "a" "b"
```

Accéder aux valeurs d'un tabeau de données (Data frame)

```
df <- as.data.frame(x)
df
```

```
##   a b
## c 1 3
## d 2 4
```

```
df$a
```

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

Calcul sur les vecteurs

```
x <- rnorm(10, 1, 4)
```

Tri

```
sort(x)
```

```
##  [1] -4.594990 -3.492456 -2.858577 -1.019206  1.348459  3.284613  4.210135
##  [8]  4.663632  5.377681  6.197535
```

Rang

```
rank(x)
```

```
##  [1]  7  5  2  9  4  3  1  8  6 10
```

Calcul sur les vecteurs

Moyenne, variance

```
mean(x)
```

```
## [1] 1.311683
```

```
var(x)
```

```
## [1] 16.08077
```

Extraction de valeurs sur test

```
which(x>3)
```

```
## [1]  1  4  8  9 10
```

Comparaison

```
x <- 1:3
y <- 1:10
match(x, y)
```

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

Extrait les éléments uniques

```
x <- rep(1:6, 3)
x
```

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

```
unique(x)
```

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

Calcul matriciel

rbind, cbind

```
m1 <- matrix(1, nr = 2, nc = 2)
m2 <- matrix(2, nr = 2, nc = 2)
rbind(m1, m2)
```

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

```
cbind(m1, m2)
```

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

Produit de matrices

Produit élément par élément (`*`)

```
m1 * m2
```

```
##      [,1] [,2]
## [1,]    2    2
## [2,]    2    2
## [3,]    4    4    8    8
## [4,]    4    4    8    8
```

Produit matriciel (`%*%`)

```
pm <- rbind(m1, m2) %*% cbind(m1, m2)
pm
```

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

Transposition (`t`)

```
m <- matrix(rep(1:4, 3), 3, 4)
t(m)
```

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

Stratégie de travail

Une fois notre session terminée, on peut quitter R et sauver notre session

```
q()
Save workspace image? [y/n/c]: y
```

Des fichiers `.RData` (contenant les **objets R** créés) et `.Rhistory` sont créés.

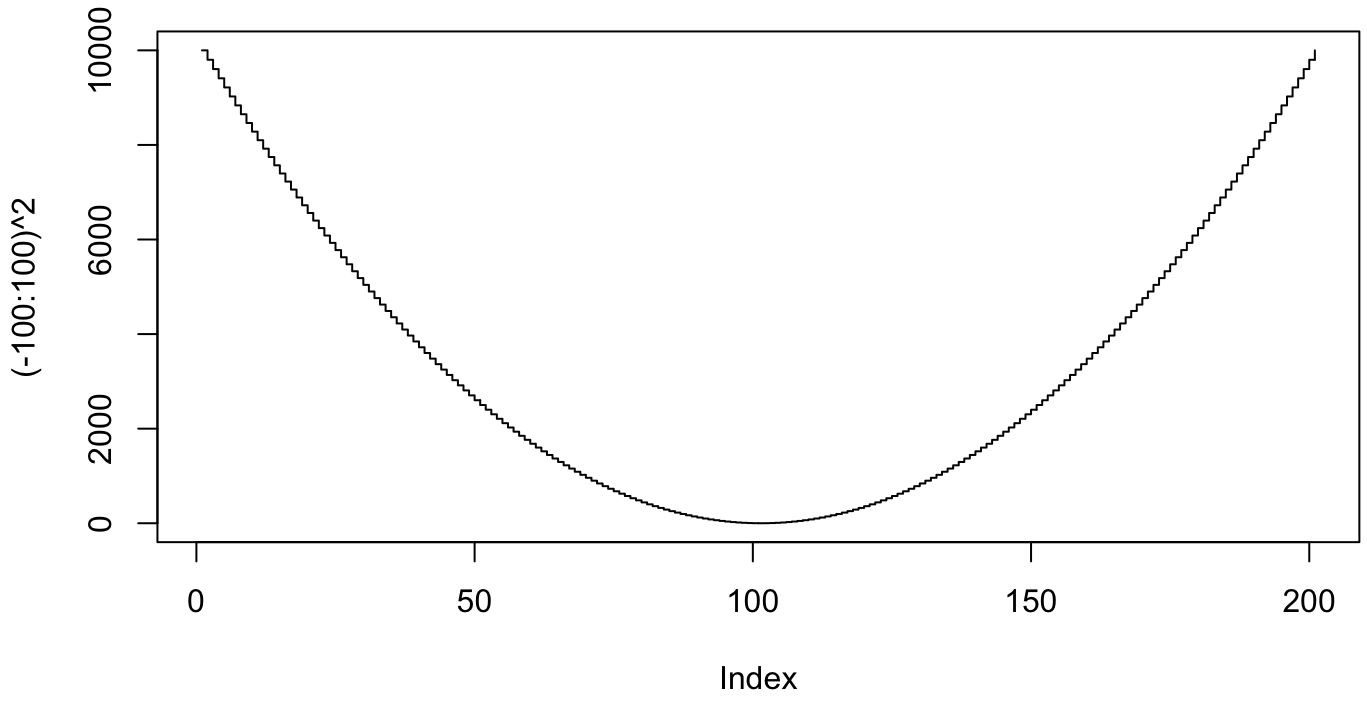
Le problème de cette approche est que nous ne **conservons pas** les instructions qui ont permis de créer les fichiers utilisés **de manière organisée**.

Il est important de consolider le code qui permet de créer les objets en conservant les instructions dans un *script*.

Sous R, on peut créer des scripts en enregistrant des instructions dans des fichiers portant l'extension `.R`.

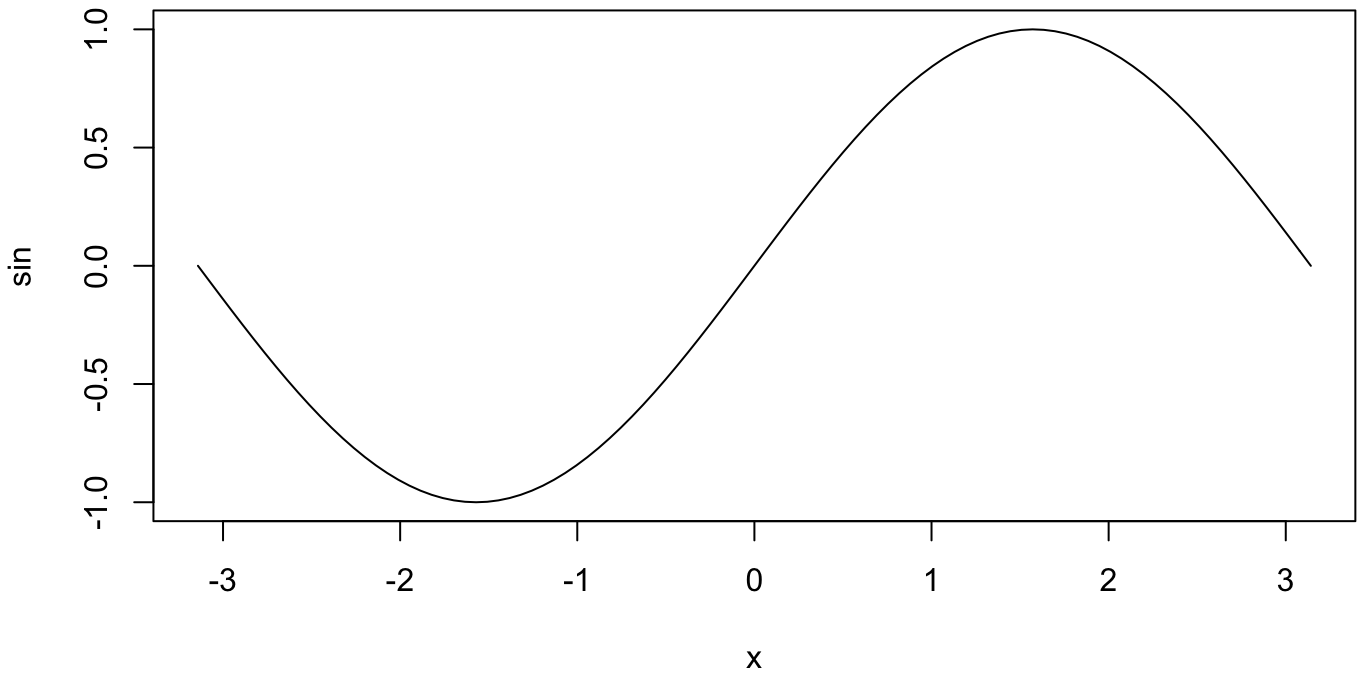
Graphique simple

```
plot((-100:100)^2, type = "s") # s = stair steps
```



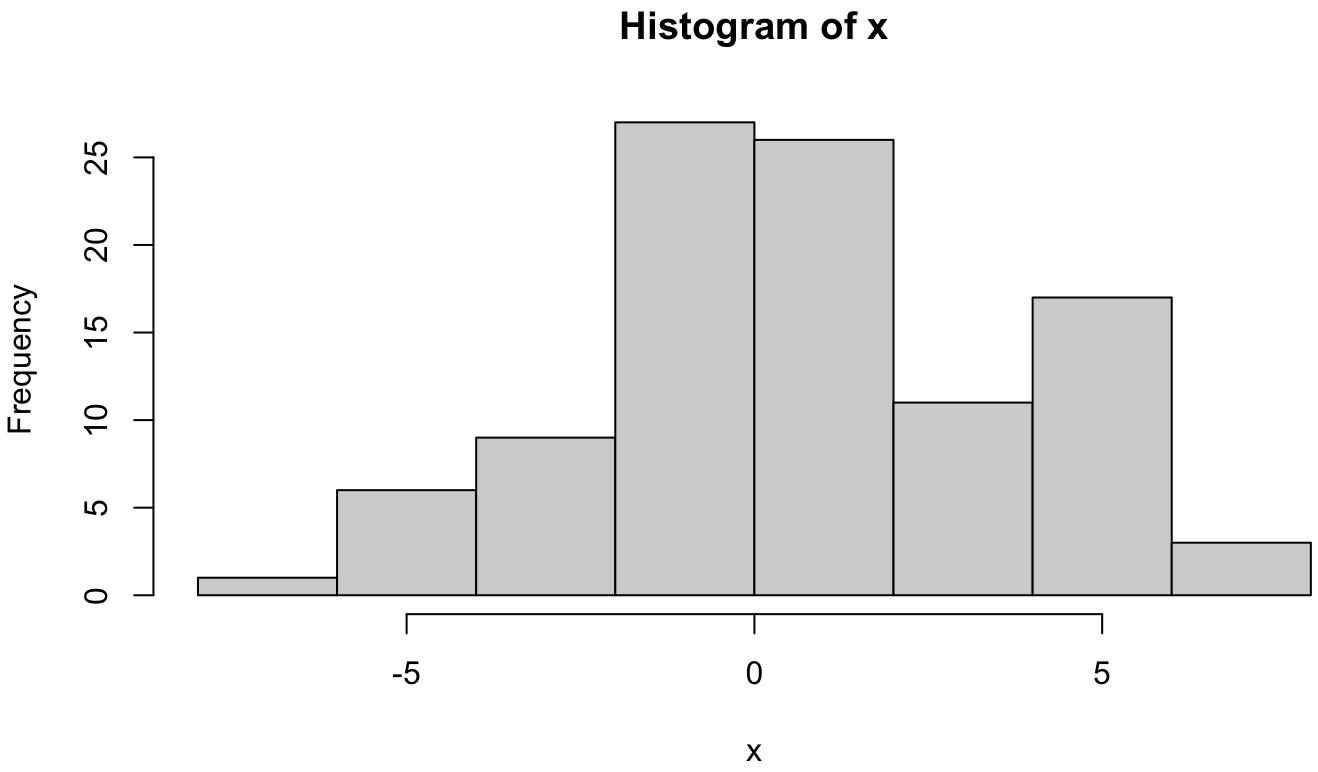
Plot d'une fonction

```
plot(sin, -pi, pi) # ou curve(sin(x), -pi, pi)
```



Histogramme

```
x <- rnorm(100, 1, 3)
hist(x)
```



Programmmation

On peut mettre les commandes dans un fichier **RScript**, ce qui permet de les exécuter plus tard. Ce type de fichier porte l'extension *.R*.

On peut directement en créer un dans RStudio, puis l'exécuter soit **pas à pas (commande ALT-Entrée)**, soit dans son intégralité dans la **console**.

Pour structurer son code on doit:

- le commenter à l'aide du caractère **#**

```
# Ceci est un commentaire
```

- Utiliser des fonctions

```
my_square_function <- function(x) {
  return(x*x)
}
my_square_function(2)
```

```
## [1] 4
```

Ressources

- *Introduction à la programmation en R* par *Vincent Goulet*: https://cran.r-project.org/doc/contrib/Goulet_introduction_programmation_R.pdf
- *R pour les débutants*, par *Emmanuel Paradis*: https://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

Bioconductor

Bonconductor possède son propre dépôt et sa propre fonction pour installer des paquets.

https://www.bioconductor.org/install/

Installation sous R:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Paquets disponibles: https://www.bioconductor.org/packages/release/BiocViews.html#___Software

Recherche et installation de packages Bioconductor

Exemple: le paquet **edgeR**:

https://www.bioconductor.org/packages/release/bioc/html/edgeR.html

Installation:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("edgeR")
```

Vignette:

```
browseVignettes("edgeR")
```

```
sessionInfo()
```

Appendice

Installation de R 4.0.0 et Rstudio sous Debian 10 “Buster”

Mise à jour du système

```
sudo apt-get autoremove
sudo apt-get update
sudo apt-get upgrade
```

Les instructions suivantes sont disponibles à https://cran.r-project.org/bin/linux/debian/#debian-buster-stable

1: Ajout de la clé de dépôt dans apt

2: Search for 0xE19F5F87128899B192B1A2C2AD5F960A256A04AF at https://keyserver.ubuntu.com, and copy the key block shown when klicking on the link in the line starting with pub into a plain text file, named, for instance, jranke.asc which you add to apt

```
with (sudo) apt-key add jranke.asc.
```

Installation de R 4.0.0 (Suite)

Ajout du dépôt dans apt

```
sudo nano /etfc/apt/sources.list
# deb http://cloud.r-project.org/bin/linux/debian buster-cran40/
```

On peut ensuite installer R 4.0.0

```
apt update
apt install -t buster-cran40 r-base
```

Installation de rstudio 1.4

```
sudo apt --fix-broken install
sudo apt-get install libclang-dev
wget http://security.debian.org/debian-security/pool/
     updates/main/o/openssl1.0/libssl1.0.2_1.0.2u-1~deb9u5_amd64.deb
sudo dpkg -i libssl1.0.2_1.0.2u-1~deb9u5_amd64.deb
wget https://download1.rstudio.org/desktop/
     debian9/x86_64/rstudio-1.4.1717-amd64.deb
sudo dpkg -i rstudio-1.4.1717-amd64.deb
sudo apt -f install # en cas de problème de dépendance
```

Appendice

Mise à jour des librairies R/Bioconductor après une mise à jour de R

```
update.packages(lib.loc="/usr/local/lib/R/site-library", ask = FALSE,
  checkBuilt = TRUE, Ncpus = 16)
```

Licence