Logiciels scientifiques : partie concernant `R`

Cécile Mercadier

2023-2024

Table des matières

Présentation globale

Liste des séances

— mardi 5 septembre 2023 GR1 et GR2 : 13h00-15h00 Quai 43 salle 101
— jeudi 7 septembre 2023 GR1 et GR2 : 13h00-15h00 Quai 43 salle 101

Objectifs

Les objectifs sont surtout d'apprendre ou de revoir les bases de `R`. L'intégration de ces notions reposera surtout sur votre travail personnel. Mais il est important de comprendre que R/Rstudio n'est pas qu'un moyen de puiser dans les bibliothèques (appelés packages) déjà existantes. C'est aussi un langage de programmation, qui permet de répondre à des problèmes et de structurer cette réponse. Il est aussi possible dans `Rstudio` de combiner différents langages. Les plus avancés pourront regarder l'intérêt d'appeler du code en `C++` pour accélérer une boucle par exemple ou d'inclure du LaTeX pour faire des rapports combinant formules mathématiques, codes et résultats.

Déroulé des séances et références

Le cours s'appuiera sur :

— une présentation de l'environnement Rstudio,
— trois chapitres (le premier présente les commandes de bases ; le second teste votre autonomie ; le troisième est facultatif, il va plus loin en vous proposant de considérer `Rcpp` ou d'essayer de mélanger `R` et LaTeX),

ainsi que sur les références externes suivantes :

— Le document Goulet (2016) présente la base de la programmation avec `R`. En parallèle de sa lecture, on peut consulter les fichiers de codes suivants fournis par l'auteur.
— Le document Louvet et Miele (2016) conduira à des réflexions et des raffinements autour de la programmation avec `R`. Le chapitre 1 énonce quelques motivations pour améliorer les performances. En particulier, on s'intéressera au package `Rcpp` qui permet de conjuguer `R` et `C++`.
— L'initiation à `Rcpp` pourra être regardée par les plus avancés via la lecture des slides de Eddelbuettel (2020) entre autres.

Résumé des liens utiles

http://math.univ-lyon1.fr/~mercadier/Goulet_introduction_programmation_R.pdf

https://cran.r-project.org/doc/contrib/Goulet_introduction_programmation_R.zip

http://pbil.univ-lyon1.fr/members/miele/pratiqueR/calcul-parallele-avec-R_extraits.pdf

http://dirk.eddelbuettel.com/papers/celebRtion_feb2020_rcpp.pdf

TP1 : Acquérir les bases de R

L'environnement Rstudio

- Créer un répertoire `LSR` dans votre espace de travail
- Télécharger le document `http://math.univ-lyon1.fr/~mercadier/chap1.Rmd` Il faut l'enregistrer (en faisant peut-être `CTRL-S`) dans votre répertoire `LSR`.
- Ouvrir Rstudio et se placer dans le bon répertoire puis via la fenêtre de droite cliquer sur `More > Set as working directory`

L'aide dans Rstudio

- `help.start()` pour faire apparaitre une page d'aide vers des manuels... A noter que `Search Engine & Keywords` permet d'obtenir la liste des fonctions associées à un mot clé donné
- `?as.integer` permet d'avoir l'aide de la fonction `as.integer` plus généralement appeler `?nomdelacommande`

Ouvrir le fichier `chap1.Rmd` dans `Rstudio`. Exécuter (cela renverra les mêmes résultats que ci-dessous), comprendre (c'est là tout l'intérêt), et me poser une question dès que cela est nécessaire. Ajouter un commentaire explicatif lorsqu'il n'y en a pas (vous pouvez le faire sur le fichier `chap1.Rmd` directement). La trace imprimée ci-dessous vous permettra d'écrire vos propres explications lorsque cela est utile.

Résumé : Dans `R` on trouvera des scalaires (vecteur de taille 1), des vecteurs, des matrices, des data.frame (sorte de matrice où les colonnes forment aussi une liste et peuvent être de classes différentes), des tableaux de matrices. On trouvera également des valeurs singulières et des syntaxes itératives. Tout ceci permettra l'écriture d'un code contenant des fonctions, des appels à ces fonctions ou à des fonctions existantes. Certaines sont présentes dans des bibliothèques spéciales qu'il faut charger...

Les vecteurs

```
val = 2 # crée l'objet val et lui attribue 2 en même temps

val1 <- val2 <- 1 # attribution simultanée de 1 à deux objets
val1
```

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

```
val2
```

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

```
x <- c(5.6,-2,78,42.3) # vecteur de numériques à 4 éléments
x
```

```
## [1]  5.6 -2.0 78.0 42.3
```

```
x <- c(x,3,c(12,8)) # vecteur à 7 éléments
x
```

```
## [1]  5.6 -2.0 78.0 42.3  3.0 12.0  8.0
```

```
c(1, 3, 8)
```

```
## [1] 1 3 8
```

```r
1:6
```

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

```r
seq(1,6,by=0.5)
```

```
##  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

```r
seq(1,6,length=5)
```

```
## [1] 1.00 2.25 3.50 4.75 6.00
```

```r
rep(1,4)
```

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

```r
rep(NA, 9) # NA est discutée un peu tard dans le TP. Il s'agit du code dans R pour dire que la valeur es
```

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

```r
rep(c(1,2),each=3)
```

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

```r
numeric(4)
```

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

```r
A = rep(1,1000) # création d'un vecteur de 1
typeof(A)
```

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

```r
object.size(A)
```

```
## 8048 bytes
```

```r
object.size(as.integer(A))
```

```
## 4048 bytes
```

+1L ajoute comme dans +1 la quantité 1 mais en précisant qu'on considère ce nombre comme un entier

```r
typeof(as.integer(A)+1)
```

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

```r
object.size(as.integer(A)+1)
```

```
## 8048 bytes
```

```r
typeof(as.integer(A)+1L)
```

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

```r
object.size(as.integer(A)+1L)
```

```
## 4048 bytes
```

```r
A = rep(1L,1000) # vecteur de 1 stocké en entiers
object.size(A)
```

```
## 4048 bytes
```

```r
x <- c("A","BB","C1")
x
```

```
## [1] "A"  "BB" "C1"
```

```r
x <- rep('A',5)
x
```

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

```r
paste("X",1:5,sep="-")
```

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

```r
paste(c("X","Y"),1:5,"txt",sep=".")
```

```
## [1] "X.1.txt" "Y.2.txt" "X.3.txt" "Y.4.txt" "X.5.txt"
```

```r
paste(c("X","Y"),1:5,sep=".",collapse="+")
```

```
## [1] "X.1+Y.2+X.3+Y.4+X.5"
```

```r
substr("livre",2,5) # `substr` extrait dans une chaine de caractères entre deux positions données
```

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

```r
txtvec <- c("arm","foot","lefroo", "bafoobar")
grep("foo", txtvec) # grep donne la position dans un vecteur d'expressions des composantes contenant un
```

```
## [1] 2 4
```

```r
gsub("foo", txtvec, replacement = "DON")
```

```
## [1] "arm"      "DONt"     "lefroo"   "baDONbar"
```

```r
1>0
```

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

```r
x <- c(-1,0,2)
test <- x>1
test
```

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

```r
(1+x^2)*test #Une opération numérique contenant des booléens les transforme de la façon suivante : `FAL
```

```
## [1] 0 0 5
```

```r
all(x>1)
```

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

```r
any(x>1)
```

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

```r
v <- 1:100
v[6] # donne le sixième élément de v
```

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

```r
v[6:8] # donne les 6ème, 7ème et 8ème éléments de v
```

```
## [1] 6 7 8
```

```r
v[c(6,6,1:2)] # donne les 6ème, 6ème, 1er et 2ème éléments de v
```

```
## [1] 6 6 1 2
```

```r
v[10:1] # donne les 10ème, 9ème, ..., 1er éléments de v
```

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

```r
v[-(1:5)] # donne v sans ses 5 premiers éléments
```

```
##  [1]   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
## [20]  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43
## [39]  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
## [58]  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
## [77]  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```r
v[-c(1,5)] # donne v sans le premier et le cinquième élément
```

```
##  [1]   2   3   4   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21
## [20]  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
## [39]  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
## [58]  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
## [77]  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
## [96]  98  99 100
```

`print` est pertinent dans une boucle. Comparer en effet `for(i in 1:3){i}` avec `for(i in 1:3){print(i)}`

```r
v <- 1:15
print(v)
```

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

```r
v[(v<5)]
```

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

```r
v[(v<5)&(v>=12)] # & signifie "et"
```

```
## integer(0)
```

```r
v[(v<5)|(v>=12)] # | signifie "ou"
```

```
## [1]  1  2  3  4 12 13 14 15
```

```r
Temp <- c(23, 28, 24, 32)
O3 <- c(80, 102, 87, 124)
O3[Temp>25]
```

```
## [1] 102 124
```

```r
x[is.na(x)] <- 0 # les éléments NA de x reçoivent la valeur 0
x[x<0]<- -x[x<0]
x <- abs(x)
```

```r
b=sample(c(TRUE,FALSE),8,replace = TRUE)
which(b)
```

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

```r
which.min(x)
```

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

```r
which(x==min(x))
```

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

Les matrices

```r
matrix(nrow=4,ncol=6)
```

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

```r
matrix(1:24,nrow=4)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    2    6   10   14   18   22
## [3,]    3    7   11   15   19   23
## [4,]    4    8   12   16   20   24
```

```r
O=matrix(1:24,nrow=4,byrow = TRUE)
O[1,3]
```

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

```r
m <- matrix(c(1,17,12,3,6,0),ncol=2)
m
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]   17    6
## [3,]   12    0
```

```r
m <- matrix(1:8,nrow=2,byrow=TRUE)
m
```

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

```r
m <- matrix(1:4,nrow=3,ncol=3)
```

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

```r
m
```

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

```r
un <- matrix(1,nrow=2,ncol=4)
un
```

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

```r
x <- seq(1,10,by=2)
x
```

```
## [1] 1 3 5 7 9
```

```r
as.matrix(x)
```

```
##      [,1]
## [1,]    1
## [2,]    3
## [3,]    5
## [4,]    7
## [5,]    9
```

```r
m[1,]
```

```
## [1] 1 4 3
```

```r
m[1,,drop=FALSE]
```

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

```r
m[,c(2,2,1)]
```

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

```r
m[-1,] # matrice m sans sa première ligne
```

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

```r
m[1:2,-1] # 2 premières lignes de m privée de sa 1ère colonne
```

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

```r
m <- matrix(1:8,ncol=4,byrow=TRUE)
m
```

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

```r
m[,m[1,]>2]
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    7    8
```

```r
m[m>2]
```

```
## [1] 5 6 3 7 4 8
```

```r
m[m>2] <- NA
m
```

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

```
m <- matrix(1:4,ncol=2)
m
```

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

```
n <- matrix(3:6,ncol=2,byrow=T)
n
```

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

```
m+n
```

```
##      [,1] [,2]
## [1,]    4    7
## [2,]    7   10
```

```
m*n # produit élément par élément
```

```
##      [,1] [,2]
## [1,]    3   12
## [2,]   10   24
```

```
sin(m) # sinus élément par élément
```

```
##           [,1]       [,2]
## [1,] 0.8414710  0.1411200
## [2,] 0.9092974 -0.7568025
```

```
exp(m) # exponentielle élément par élément
```

```
##          [,1]     [,2]
## [1,] 2.718282 20.08554
## [2,] 7.389056 54.59815
```

```
m^4 # puissance quatrième élément par élément
```

```
##      [,1] [,2]
## [1,]    1   81
## [2,]   16  256
```

```
A <- matrix(1:4,ncol=2)
B <- matrix(c(5,7,6,8),ncol=2)
D <- A%*%t(B)
D
```

```
##      [,1] [,2]
## [1,]   23   31
## [2,]   34   46
```

```
eig <- eigen(D)
eig
```

```
## eigen() decomposition
## $values
```

```
## [1] 68.9419802  0.0580198
##
## $vectors
##            [,1]       [,2]
## [1,] -0.5593385 -0.8038173
## [2,] -0.8289393  0.5948762
```

```
eig$vectors[,1]
```

```
## [1] -0.5593385 -0.8289393
```

```
V <- c(1,2)
solve(D,V)
```

```
## [1] -4  3
```

```
X <- matrix(1:6,ncol=3)
X
```

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

```
ncol(X)
```

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

```
nrow(X)
```

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

```
dim(X)
```

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

```
cbind(c(1,2),c(3,4))
```

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

```
apply(X,MARGIN=2,sum) # sommes par colonne
```

```
## [1]  3  7 11
```

```
apply(X,1,mean) # moyennes par ligne
```

```
## [1] 3 4
```

Les facteurs

`factor` ou `as.factor` définit une variable qualitative où les modalités possibles sont les `levels`.

`table` donne la table des effectifs. Faire `s=sample(1:10,20,replace=TRUE)` puis `table(s)` pour voir que la premiere ligne est equivalente à `sort(unique(s))` et seconde ligne donne les effectifs associés.

```
sexe <- factor(c("M","M","F","M","F","M","M","M"))
sexe
```

```
## [1] M M F M F M M M
## Levels: F M
```

```r
sexe <- factor(c(2,2,1,2,1,2,1),labels=c("femme","homme"))
sexe
```

```
## [1] homme homme femme homme femme homme femme
## Levels: femme homme
```

```r
niveau <- ordered(c("débutant","débutant","champion","champion",
"moyen","moyen","moyen","champion"),
levels=c("débutant","moyen","champion"))
niveau
```

```
## [1] débutant débutant champion champion moyen    moyen    moyen    champion
## Levels: débutant < moyen < champion
```

```r
salto <- c(1:5,5:1)
salto
```

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

```r
salto.f <- as.factor(salto)
salto.f
```

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

```r
levels(salto.f)
```

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

```r
nlevels(salto.f)
```

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

```r
table(salto.f)
```

```
## salto.f
## 1 2 3 4 5
## 2 2 2 2 2
```

```r
(ff <- factor(substring("statistics", 1:10, 1:10), levels = letters))
```

```
##  [1] s t a t i s t i c s
## Levels: a b c d e f g h i j k l m n o p q r s t u v w x y z
```

```r
as.numeric(ff)
```

```
##  [1] 19 20  1 20  9 19 20  9  3 19
```

```r
x <- factor(c(10,11,13))
as.numeric(x)
```

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

```r
as.numeric(as.character(x))
```

```
## [1] 10 11 13
```

Les listes

```r
liste1 = vector("list", 4)
liste1
```

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

```
liste1[[1]] = 3
liste1
```

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

```
liste2 = list(a = 2:10, b = "janvier", c = 2020)
liste2
```

```
## $a
## [1]  2  3  4  5  6  7  8  9 10
##
## $b
## [1] "janvier"
##
## $c
## [1] 2020
```

```
liste2$b
```

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

```
liste2[[2]]
```

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

```
str(liste2) # donne la structure de liste2
```

```
## List of 3
##  $ a: int [1:9] 2 3 4 5 6 7 8 9 10
##  $ b: chr "janvier"
##  $ c: num 2020
```

```
vecteur <- seq(2,10,by=3)
matrice <- matrix(1:8,ncol=2)
facteur <- factor(c("M","M","F","M","F","M","M","M"))
ordonne <- ordered(c("débutant","débutant","champion",
                     "champion","moyen","moyen","moyen","champion"),
```

```
                  levels=c("débutant","moyen","champion"))
maliste <- list(vecteur,matrice,facteur,ordonne)
length(maliste)
```

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

```
mode(maliste)
```

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

```
names(maliste)
```

```
## NULL
```

```
names(maliste) <- c("vec","mat","sexe","ski")
names(maliste)
```

```
## [1] "vec"  "mat"  "sexe" "ski"
```

```
maliste[[3]]
```

```
## [1] M M F M F M M M
## Levels: F M
```

```
maliste[[1]]
```

```
## [1] 2 5 8
```

```
maliste$sexe
```

```
## [1] M M F M F M M M
## Levels: F M
```

```
maliste[["sexe"]]
```

```
## [1] M M F M F M M M
## Levels: F M
```

```
maliste[c(1,3)]
```

```
## $vec
## [1] 2 5 8
##
## $sexe
## [1] M M F M F M M M
## Levels: F M
```

```
X <- matrix(1:12,nrow=4,ncol=3)
nomligne <- c("ligne1","ligne2","ligne3","ligne4")
nomcol <- c("col1","col2","col3")
dimnames(X) <- list(nomligne,nomcol)
X
```

```
##        col1 col2 col3
## ligne1    1    5    9
## ligne2    2    6   10
## ligne3    3    7   11
## ligne4    4    8   12
```

```
X[c("ligne4","ligne1"),c("col3","col2")]
```

```
##        col3 col2
```

```
## ligne4   12    8
## ligne1    9    5
```

```
dimnames(X) <- list(NULL,dimnames(X)[[2]])
X
```

```
##      col1 col2 col3
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Les data-frames

```
vec1 <- 1:5
vec2 <- c("a","b","c","c","b")
df <- data.frame(nom.var1 = vec1, nom.var2 = vec2)
x = data.frame(a = 1:10, colonne2 = LETTERS[7 + 1:10])
x
```

```
##     a colonne2
## 1   1        H
## 2   2        I
## 3   3        J
## 4   4        K
## 5   5        L
## 6   6        M
## 7   7        N
## 8   8        O
## 9   9        P
## 10 10        Q
```

```
x[, 1]
```

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

```
x[[1]]
```

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

Les arrays (tableau de matrices)

```
tab=array(1:24,dim=c(2,3,4))
  tab[,,1]
```

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

```
  tab[,,2]
```

```
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

```
  tab[,,3]
```

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

```
## [1,]   13   15   17
## [2,]   14   16   18
```

La classe d'un objet

```
df <- data.frame(x = 1:2, y = c(3, 5))
print(df)
```

```
##   x y
## 1 1 3
## 2 2 5
```

```
print.default(df)
```

```
## $x
## [1] 1 2
##
## $y
## [1] 3 5
##
## attr(,"class")
## [1] "data.frame"
```

```
#  install.packages("data.table") si besoin installer le package au préalable
library(data.table)
dt <- data.table::data.table(df)
dt
```

```
##    x y
## 1: 1 3
## 2: 2 5
```

```
class(dt)
```

```
## [1] "data.table" "data.frame"
```

Valeurs singulières sous R : Valeur manquante ; Valeur infinie ; Valeur logique

```
NA
```

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

```
x = c(1, NA, 7)
is.na(x)
```

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

```
sum(x)
```

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

```
sum(x, na.rm = TRUE)
```

```
## [1] 8
```

```
Inf
```

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

```
NaN
```

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

```
Inf * 0
```

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

```
is.finite(x)
```

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

```
is.infinite(x)
```

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

```
is.nan(x)
```

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

```
TRUE
```

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

```
T # éviter de donner le nom T à une variable
```

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

```
FALSE
```

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

```
F # éviter de donner le nom F à une variable
```

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

Syntaxes itératives : Boucle `for` ; Comparaison ; Condition ; Boucle `while`

```
for(var in seq){commandes}
```

Les expressions suivantes renvoient `TRUE` ou `FALSE`.

```
a == b
a != b
a < b
a > b
a <= b
a >= b
```

```
if(condition){commandes}
```

qui exécute `commandes` si `condition == TRUE`. De même

```
    while(condition){commandes}
```

exécute `commandes` tant que `condition == TRUE`.

Les fonctions

Construction d'une fonction sur `R`

```
# exemple de fonction
# créer une fonction semblable à table,
# ajouter ensuite une option modifiant l'ordre
```

```r
table.rapido=function(v){
  valeur=sort(unique(v))
  effectif <- rep(NA,length(valeur))
  for(i in 1:length(valeur)){
    effectif[i]=length(which(v==valeur[i]))
  }
  return(rbind(valeur,effectif))
}
v=sample(1:5,12,replace=TRUE)
table.rapido(v)
```

```
##          [,1] [,2] [,3] [,4] [,5]
## valeur      1    2    3    4    5
## effectif    3    2    2    3    2
```

```r
# Ajouter une option :
# ajouter ensuite une option modifiant l'ordre croissant en décroissant pour la liste des valeurs renco
# sort = TRUE -> ordre croissant, sort = FALSE -> décroissant
table.rapido=function(v,sort=TRUE){
  valeur=sort(unique(v))
  if(!sort){valeur=sort(unique(v),decreasing=TRUE)}
  effectif <- rep(NA,length(valeur))
  for(i in 1:length(valeur)){
    effectif[i]=length(which(v==valeur[i]))
  }
  return(rbind(valeur,effectif))
}
table.rapido(v,sort=FALSE)
```

```
##          [,1] [,2] [,3] [,4] [,5]
## valeur      5    4    3    2    1
## effectif    2    3    2    2    3
```

```r
set.seed(13) # fixe la graine du générateur
rnorm(n=3)
```

```
## [1]  0.5543269 -0.2802719  1.7751634
```

```r
set.seed(13) # fixe la graine du générateur
rnorm(n=4,mean=5,sd=0.5)
```

```
## [1] 5.277163 4.859864 5.887582 5.093660
```

```r
args(rnorm)
```

```
## function (n, mean = 0, sd = 1)
## NULL
```

```r
args(plot)
```

```
## function (x, y, ...)
## NULL
```

Les packages

Comment installer un package ?

— soit en cliquant via onglet `Tools > Installer un package >` taper le nom dans la zone de recherche

— soit en ligne de commandes : `install.packages("nom_du_package")`

N'oubliez pas de faire `library(nom_du_package)` ou `require(nom_du_package)` car l'avoir téléchargé sur l'ordinateur et le rendre accessible sur `R` sont deux choses différentes.

```
set.seed(45) # fixe la graine du générateur aléatoire
require(MASS) # ou library(MASS)
mvrnorm(3,mu=c(0,1),Sigma=matrix(c(1,0.5,0.5,1),2,2))
```

```
##            [,1]        [,2]
## [1,]  0.6681649  0.92211747
## [2,] -0.1600569 -0.05816423
## [3,] -0.1612923  0.50391363
```

```
set.seed(45) # fixe la graine du générateur aléatoire
MASS::mvrnorm(3,mu=c(0,1),Sigma=matrix(c(1,0.5,0.5,1),2,2))
```

```
##            [,1]        [,2]
## [1,]  0.6681649  0.92211747
## [2,] -0.1600569 -0.05816423
## [3,] -0.1612923  0.50391363
```

```
if (!require(rpart)) install.packages("rpart") # installe rpart si besoin
require(rpart)
```

```
update.packages(ask=FALSE)
packs = as.data.frame(installed.packages(.libPaths()[1]), stringsAsFactors = F)
install.packages(packs$Package) ## réinstallation des packages
```

TP2 : Tester son autonomie

Exercices extraits de la Section 2.10 de Goulet (2016)

Exercice 2.1

Ecrire une expression `x` afin de créer la liste suivante

```
x
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## $data
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## [[3]]
## [1] 0 0 0
##
## $test
## [1] FALSE FALSE FALSE FALSE
```

- extraire les étiquettes de la liste
- trouver le mode et la longueur du quatrième élément de la liste
- extraire les dimensions du second élément de la liste
- extraire les deuxième et troisième éléments du second éléments de la liste
- remplacer le troisème élément de la liste par le vecteur d'entiers 3, 4, . . ., 8

Exercice 2.2 Définir maintenant `x` un vecteur contenant les valeurs de l'échantillon suivant :

```
##  [1]  1 18  2  1  5  2  6  1 12  3 13  8 20  1  5  7  7  4 14 10
```

Ecrire ensuite des expressions `R` permettant d'extraire les éléments suivants :

- le deuxième élément de `x`
- les cinq premiers éléments de `x`
- les éléments de `x` strictement supérieurs à 14
- tous les éléments de `x` sauf ceux des positions 6, 10 et 12.

Exercice 2.3 Définir aléatroirement une matrice `x` de taille `10*7` en utilisant `matrix` et `sample`.

Ecrire ensuite des expressions `R` permettant d'obtenir les éléments de la matrice demandés ci-dessous :

- l'élément de la quatrième ligne et troisième colonne
- tout le contenu de la sixème ligne
- les première et quatrième colonnes (simultanément)
- les lignes dont le premier élément est supérieur à 50

Exercices extraits de la Section 3.8 de Goulet (2016)

Exercice 3.1 A l'aide de `rep`, `seq` et `c` seulement, générer les séquences suivantes

```
## [1] 0 6 0 6 0 6
```

```
## [1]  1  4  7 10
```

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

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

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

```
## [1]  1.0  5.5 10.0
```

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

Exercice 3.2 A l'aide de `:` et `rep` seulement, générer les séquences suivantes

```
##  [1] 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

```
##  [1] -2 -1  0  1  2 -2 -1  0  1  2
```

```
##  [1] -2 -2 -1 -1  0  0  1  1  2  2
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

Exercice 3.3 A l'aide de la commande `apply`, écrire des expressions `R` qui remplaceraient les fonctions suivantes :

- `rowSums`
- `colSums`
- `rowMeans`
- `colMeans`

Exercice 3.4 Générer la suite `1!, 2!, ..., 10!` en évitant l'appel des fonctions `factorial` ou `gamma`, sans recopier les résultats suivants non plus.

```
##  [1]       1       2       6      24     120     720    5040   40320  362880
## [10] 3628800
```

Exercice 3.5 Tester aléatoirement `x == (x %% y) + y * (x %/% y)` en attribuant des valeurs aléatoires à `x` et `y`. Expliquer.

Exercice 3.6 Soit `x` le vecteur de taille `20` de l'exercice 2.2. Ecrire des expressions `R` permettant d'extraire les éléments suivants :

- les cinq premiers éléments de l'échantillon
- la valeur maximale
- la moyenne des cinq premiers éléments
- la moyenne des cinq derniers éléments

Exercice 3.7

- Trouver une formule pour calculer la position, dans le vecteur sous-jacent, de l'élément (i, j) d'une matrice $I \times J$ remplie colonne par colonne.
- De même pour l'élément (i, j, k) d'un tableau de matrices $I \times J \times K$.

Exercice 3.8 Simuler une matrice `mat` de taille `9 * 8`, puis écrire des expressions `R` permettant d'effectuer les tâches demandées ci-dessous :

- calculer la sommes des éléments de chacune des lignes
- calculer la moyenne des éléments de chacune des colonnes
- calculer la valeur maximale de la sous-matrice formée par les trois premières lignes et les trois premières colonnes
- extraire toutes les lignes dont la moyenne des éléments est supérieure à 7

Exercice 3.9 On dispose des meilleurs temps au 100 mètres homme entre 1964 et 2005 enregistrés de façon chronologique (la première valeur est la plus ancienne).

```
temps = c(10.06, 10.03, 10.02, 9.95, 10.04, 10.07, 10.08, 10.05, 9.98, 10.09, 10.01, 10.00,
          9.97, 9.93, 9.96, 9.99, 9.92, 9.94, 9.90, 9.86, 9.87, 9.85, 9.91, 9.84, 9.89,
          9.79, 9.80, 9.82, 9.78, 9.77)
```

Que donne la commande ci-dessous

```r
temps[match(unique(cummin(temps)), temps)]
```

A noter que le Jamaïcain Usain Bolt est détenteur du record (9.58) depuis le 16 août 2009.

Exercices extraits de la Section 4.6 de Goulet (2016)

Exercice 4.2 Etant donné un vecteur d'observations `x`= $(x_1, \ldots, x_n)$ et un vecteur de poids `w`= $(w_1, \ldots, w_n)$, écrire une fonction qui renvoie la moyenne pondérée des observations,

$$\sum_{i=1}^{n} \frac{w_i}{wt} x_i$$

avec $wt = \sum_{i=1}^{n} w_i$. Tester votre fonction pour différents vecteurs `x` et `w`.

Exercice 4.3 Soit un vecteur d'observations `x`= $(x_1, \ldots, x_n)$. Calculer, par une fonction qu'on testera, la moyenne harmonique de ce vecteur définie par

$$\frac{n}{\frac{1}{x_1} + \ldots + \frac{1}{x_n}}$$

Exercice 4.4 Rappeler l'expression de la f.d.r. d'une loi de Poisson avec paramètre $\lambda = 2$ au point x. Ecrire une fonction effectuant ce calcul et la tester en différents points x.

Exercices extraits de la Section 5.9 de Goulet (2016)

Exercice 5.2 Expliquer la fonction suivante

```r
variance <- function(x, biased = FALSE)
{
if (biased) {
        n <- length(x)
        (n - 1)/n * var(x)
    }
else var(x)
}
```

Exercice 5.3 Ecrire une fonction `phi` qui donne la valeur de la densité d'une gaussienne centrée réduite.

Exercices sur Fibonacci

- La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. Elle commence ici avec deux valeurs initiales données par `1`. A l'aide d'un vecteur et d'une boucle `for` calculer les dix premiers termes de cette suite.
- De la suite précédente, tester à l'aide de `if` et transformer les valeurs strictement plus petite que `10` en valeurs manquantes grâce à `NA`.
- Calculer la suite de Fibonacci `tant que` les termes sont inférieurs à `300`. On utilisera donc une boucle `while` associée à une condition.

TP3 facultatif : Mesurer et challenger `R` !

Lecture du chapitre 1 de Louvet et Miele (2016) intitulé "A la recherche de performances"

Une partie de l'ouvrage est accessible ici Louvet et Miele (2016)

Toute la lecture est instructive. Il faut tout lire mais certains passages peuvent l'être plus en diagonale. Pendant le TP, on commencera à faire du code seulement à partir de la section 1.2 "Les différentes approches en R pour gagner en performances". On continuera à lire mais toutes les parties codées peuvent être exécutées à partir du fichier suivant : miele.R

Résumons la Section 1.1.

Ecriture : - viser la simplicité, la clarté - importance de la véracité au dessus de tout - privilégier le découpage pour retomber sur des fonctions connues et optimisées par ailleurs - faire les bons choix dans la labellisation

Optimisation : - test - système de gestion (versions, partage)

Remarque sur le terme *refactoring* : vraisemblablement synonyme de refonte, remaniement, du coup, ce doit être un terme général pour dire qu'on retravaille la structure d'un code.

Remarque sur `self.time` versus `total.time`

`self.time` tient compte uniquement des exécutions à l'intérieur même de la fonction ; celles associées à des fonctions extérieures/appelées sont considérées comme nulles en temps.

`total.time` somme tous les temps d'exécution, intérieures comme extérieures (via les sous-fonctions appelées).

Lecture des slides de Dirk Eddelbuettel

Document de travail

http://dirk.eddelbuettel.com/papers/celebRtion_feb2020_rcpp.pdf

Résumé des premiers slides (pages 1 à 29) :

- Avant on travaillait sur `S`, `SPlus`, "remplacé" aujourd'hui `R`. `R` n'est pas qu'une boîte à outils statistique, c'est un langage.
- Il n'est pas le seul mais reste complet car il permet de gérer des éléments spécifiques en statistique.
- `R` a évolué, il a « toujours » été présent dans la communauté des chercheurs, il est de plus en plus présent dans les entreprises, il intègre désormais facilement des liens avec d'autres langages, et via Rstudio il est facile d'éditer le travail et les résultats d'un projet.
- `R` permet de créer de nouveaux packages.
- Focus est fait ensuite sur `Rcpp`. On voit que le nombre d'utilisateurs est croissant et qu'il apparait dans de nombreux packages récents.

Lecture des slides suivants :

Dans un second temps, on lira de la page 30 à la page 68 du même document. Les solutions des quatre premiers exercices sont données dans le fichier

de.R

On recopie maintenant les parties de codes données dans ces pages

```
## evaluate simple expression
## ... as C++
Rcpp::evalCpp("2 + 2")
```

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

```r
## more complex example
set.seed(42)
Rcpp::evalCpp("Rcpp::rnorm(2)")
```

```
## [1]  1.3709584 -0.5646982
```

```r
## As seen, evalCpp() evaluates a single C++ expression. Includes and dependencies can be declared.
## This allows us to quickly check C++ constructs.
library(Rcpp)
evalCpp("2 + 2") # simple test
```

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

```r
evalCpp("std::numeric_limits<double>::max()")
```

```
## [1] 1.797693e+308
```

Exercice 1 : Evaluate an expression in C++ via `Rcpp::evalCpp()`

```r
## cppFunction() creates, compiles and links a C++ file, and creates an R function to access it.
cppFunction("
int exampleCpp11() {
auto x = 10; // guesses type
return x;
}", plugins=c("cpp11")) ## turns on C++11
## R function with same name as C++ function
exampleCpp11()
```

```
## [1] 10
```

```r
library(Rcpp)
cppFunction("int f(int a, int b) { return(a + b); }")
f(21, 21)
```

```
## [1] 42
```

Exercice 2 : Write a C++ function on the R command-line via `cppFunction()`. Should the above work ? Yes ? No ? What can you see examining it ? Can you "break it" ?

Lorsqu'on fait sous R studio `File > New File > C++ File` on obtient la création d'un fichier `.cpp` contenant les éléments suivants :

```r
#include <Rcpp.h>
using namespace Rcpp;
// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). Learn
// more about Rcpp at:
//
//   http://www.rcpp.org/
//   http://adv-r.had.co.nz/Rcpp.html
//   http://gallery.rcpp.org/
//

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
  return x * 2;
}
```

```
// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.
//

/*** R
timesTwo(42)
*/
```

Ce fichier a créé une fonction très simple multiplication par 2. Le paramètre d'entrée doit être un vecteur de `numeric`. Avant sa définition on demande à `R` d'incorporer cette fonction via `Rcpp::export`. Le package `Rcpp` travaille alors pour nous... Dans les commandes `R` du bas, on accède bien à cette nouvelle fonction par son nom sous `C++`. Pour exécuter du `R` dans un fichier `.cpp`, on en déduit donc qu'il suffit de l'insérer entre la balise ouvrante `/*** R` et la balise fermante `*/`.

Exercice 3 : Modify the `timesTwo` function used via `Rcpp::sourceCpp()`. Use the `RStudio File -> New File -> C++ File` template.

Suggestions d'exercices : Piocher dans les exercices du chapitre 2. Par exemple, répondre via `Rcpp` aux exercices 4.2 ou 4.3 ou améliorer les performances de la fonction Fibonacci via `C++`.

Conclusion

Concernant `R` déjà, il y a encore beaucoup de choses à apprendre. Par ailleurs, il est important de distinguer ici `Rcpp` et `C++`. Si vous avez regardé le chapitre 3, il concerne le package `Rcpp`. Il ne s'agit pas d'un cours de `C++`. Une fois la porte `C++` ouverte, il y a beaucoup à apprendre ! Le but premier était de vous faire connaître cette possibilité. Laissez les choses mûrir un peu et *codez, codez, codez* reste quand même le meilleur conseil. Pour aller plus loin :

https://cran.r-project.org/web/packages/Rcpp/vignettes/Rcpp-quickref.pdf

http://www2.stat.duke.edu/~cr173/Sta790_Sp19/slides/Lec08.html#12

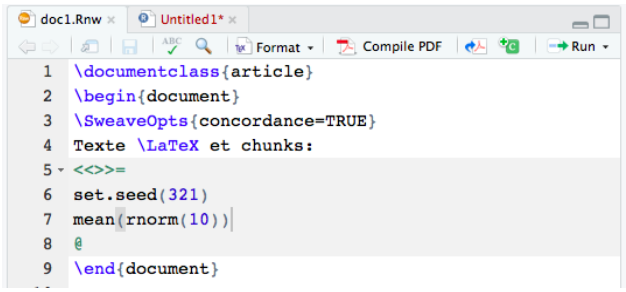
Autre possibilité, mixer `R` et LaTeX

Ce qui suit est largement inspiré du documen *Utilisation avancee de R avec rstudio* (écrit par Eric Marcon). Un rapport `Sweave` ou `knitr` est un document LaTeXqui contient du code `R`. En ce qui me concerne, je choisis `knitr`. Je le fais en allant dans Tools puis Global Options puis Sweave puis sur le premier onglet sélectionner `knitr`.

Les lignes de codes sont exécutées *au moment* de la compilation TeX ce qui assure *l'adéquation* entre les résultats présentés dans un rapport (sorties de R, figures) et le code qui le précède. Dans R, créer un tel document via **File > New File > R Sweave**. Le document créé dispose de l'extension **.Rnw**.

Concernant les parties de texte, la syntaxe est celle de LaTeX : on peut utiliser des packages précisés dans le préambule, de la bibliographie. On peut donc aussi utiliser un ancien fichier .tex qui tourne déjà bien et sauvegarder **une copie avec l'extension .Rnw**. La compilation du document se fait en deux temps :

$$\text{.Rnw} \xrightarrow[Sweave]{} \text{.tex} \xrightarrow[pdfLaTeX]{} \text{.pdf}$$



donnera

Texte LaTeXet chunks:

```
> set.seed(321)
> mean(rnorm(10))

[1] 0.148663
```

On peut choisir d'afficher le code (option **echo=**) et le résultat (**eval=**) et/ou bien juste le code ou encore seulement le résultat...

```
Options d'affichage\\
Version TRUE/TRUE
<<NomDuChunk, echo=TRUE, eval=TRUE>>=
set.seed(321)
mean(rnorm(10))
@
Version FALSE/TRUE
<<NomDuChunk, echo=FALSE, eval=TRUE>>=
set.seed(321)
mean(rnorm(10))
@
Version TRUE/FLASE
<<NomDuChunk, echo=TRUE, eval=FALSE>>=
set.seed(321)
mean(rnorm(10))
@
Version FALSE/FALSE
<<NomDuChunk, echo=FALSE, eval=FALSE>>=
set.seed(321)
mean(rnorm(10))
@
```

Options d'affichage
Version TRUE/TRUE

```
> set.seed(321)
> mean(rnorm(10))

[1] 0.148663
```

Version FALSE/TRUE

```
[1] 0.148663
```

Version TRUE/FLASE

```
> set.seed(321)
> mean(rnorm(10))
```

Version FALSE/FALSE

Concernant les figures, la compilation génèrera une figure si la commande R contient **fig =TRUE** comme par exemple

```
<<fignhtemp, fig =TRUE , echo =FALSE >>=
plot(nhtemp, lag(nhtemp, 1), cex = .8, col = "blue",main = "Lag plot of New Haven temperatures")
@

\begin{figure}
\begin{center}
<<figSpray, fig =TRUE , echo =FALSE >>=
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")
@
\end{center}
\caption{L\'egende}
\label{figOzone }
\end{figure}
```

permettant soit de mettre un titre avec R (premier exemple), soit de le faire avec LaTeX.

On peut aussi continuer d'utiliser **\includegraphics** avec une figure externe déjà sauvegardée en pdf, ou bien avec une figure à produire lors de la compilation. Pour cette dernière possibilité, on donnera un nom à l'insertion `R` correspondante (par exemple `barpl`) et on y fera référence en ajoutant `NomDuFichierRnw-NomDeLinsertionR.pdf` (ici, ce serait `doc1-barpl.pdf`). Ceci est à appliquer en parallèle de l'option **include =FALSE** pour que le graphe soit produit sans qu'il soit inclus au moment de sa création.

Options : **\SweaveOpts{concordance=TRUE}** est une option pour la création d'un fichier de concordance des numéros de lignes entre le fichier `.Rnw` et le fichier `.tex` produit. L'insertion

```
<<Declarations, echo=FALSE>>=
options(prompt = "R> ", width = 50)
@
```

permet de modifier le texte dont toute entrée `R` sera précédée. Ici ce sera préfixé par `R>` au lieu de `>`. L'autre option nous dit que le résultat `R` affiché sera coupé au bout de 50 caractères. La suite du résultat est mise à la ligne suivante.

Rappel des options :

`echo=FALSE` Don't include the code

`results='hide'` Don't include the output

`include=FALSE` Don't show code or output

`eval=FALSE` Don't evaluate the code at all

`warning=FALSE` Don't show R warnings

`message=FALSE` Don't show R messages

`fig.width=#` Width of figure

`fig.height=#` Height of figure

`fig.path=Figs/` Path for figure files

Références

Eddelbuettel, D. 2020. "Introduction to Rcpp from Simple Examples to Machine Learning." http://dirk.eddelbuettel.com/papers/celebRtion_feb2020_rcpp.pdf.

Goulet, V. 2016. "Introduction à La Programmation En R." http://math.univ-lyon1.fr/~mercadier/Goulet_introduction_programmation_R.pdf.

Louvet, V., and V. Miele. 2016. "Calcul Parallèle Avec R." EDP Sciences. http://pbil.univ-lyon1.fr/members/miele/pratiqueR/calcul-parallele-avec-R_extraits.pdf.