

DÉSAISONNALISER UNE SÉRIE TEMPORELLE



1 - Rapide introduction à R

ALAIN QUARTIER-LA-TENTE



 : Logiciel statistique

RStudio : IDE le plus connu

CRAN : Plateforme centralisant un ensemble de packages R sous un format normalisé permettant une installation facile

GitHub : Plateforme de partage de code où l'on retrouve de nombreux packages en développement

Aide

- Si vous ne connaissez pas les fonctions, Google/ChatGPT est votre ami
- Sinon `help(ma_fonction)` ou `?ma_fonction` pour chercher l'aide associée à `ma_fonction`. Voir aussi vignettes (documentation long format). Exemple :

```
# Pour voir l'ensemble des vignettes du package grid
vignette(package = "grid")
# Pour afficher une vignette en utilisant son nom
vignette("moveline", package = "grid")
```

- Cran Task Views (<https://cran.r-project.org/web/views/>) regroupement de packages selon des thèmes particuliers. Exemple pour ce cours : <https://cran.r-project.org/web/views/TimeSeries.html>

Quelques conseils

- Éviter les chemins absolus en utilisant les projets RStudio
- En cas d'erreur, prenez le temps de lire ce que le logiciel vous indique et cherchez à avoir un exemple minimal reproductible.

Sommaire

1. Les types de base

1.1 Les vecteurs

1.2 Les matrices

1.3 Les listes

1.4 Le data.frame et tibble

2. Importation des données

3. Les séries temporelles

Les vecteurs (1)

Les vecteurs sont les objets les plus simples : créés avec fonction `c()` et leurs éléments peuvent être manipulés avec l'opérateur `[]`

```
v1 <- c(1, 2, 3); v2 <- c("a", "b")  
v1
```

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

```
v2
```

```
[1] "a" "b"
```

```
# v1 peut aussi se créer de façon équivalente avec :  
1:3
```

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

```
# Pour concaténer deux vecteurs, notez le changement de type  
v3 <- c(v1, v2)  
v3
```

Les vecteurs (2)

```
[1] "1" "2" "3" "a" "b"
```

```
v3[c(4, 1)] # 4e puis 1er élément
```

```
[1] "a" "1"
```

```
v3[-c(4, 1)] # on enlève 1er et 4e éléments
```

```
[1] "2" "3" "b"
```

```
# Les éléments peuvent également être nommés
```

```
v4 <- c(elem1 = 1, elem2 = 2, 4)
```

```
v4
```

```
elem1 elem2
```

```
1      2      4
```

```
names(v4)
```

```
[1] "elem1" "elem2" ""
```

Les vecteurs (3)

```
names(v4)[1] <- "toto"  
v4
```

```
  toto elem2  
    1      2      4
```

```
v4[c("toto", "elem2")]
```

```
  toto elem2  
    1      2
```


Les matrices (1)

Matrices : vecteurs à deux dimensions créés avec fonction `matrix()`

```
m1 <- matrix(1:12, ncol = 3); m2 <- matrix(1:12, nrow = 3)
m1; t(m1); m1 * 2
```

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

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

Les matrices (2)

	[,1]	[,2]	[,3]
[1,]	2	10	18
[2,]	4	12	20
[3,]	6	14	22
[4,]	8	16	24

```
m1 %*% m2 # multiplication matricielle
```

	[,1]	[,2]	[,3]	[,4]
[1,]	38	83	128	173
[2,]	44	98	152	206
[3,]	50	113	176	239
[4,]	56	128	200	272

```
m1[, 1] # 1ere colonne : c'est un vecteur
```

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

Les matrices (3)

```
m1[-2, ] # Tout sauf 2ème ligne
```

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

```
# Nombre de lignes et de colonnes :  
nrow(m1); ncol(m1); dim(m1)
```

```
[1] 4
```

```
[1] 3
```

```
[1] 4 3
```

Les matrices (4)

```
# De la même façon que pour les vecteurs on peut nommer lignes/col  
colnames(m1) <- paste0("col", 1:ncol(m1))  
rownames(m1) <- paste0("row", 1:nrow(m1))  
m1
```

	col1	col2	col3
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12

```
m1[, "col2"]
```

row1	row2	row3	row4
5	6	7	8

```
# Pour combiner des matrices, on peut utiliser cbind et rbind:  
cbind(m1, 1:4)
```

Les matrices (5)

	col1	col2	col3
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12

```
rbind(m1, m1)
```

	col1	col2	col3
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12

Les matrices (6)

On peut utiliser la fonction `apply` pour appliquer une fonction à toutes les lignes ou toutes les colonnes. Exemple :

```
apply(m1, 1, sum) # somme sur toutes les lignes (dimension 1)
```

```
row1 row2 row3 row4  
15    18    21    24
```

```
apply(m1, 2, sum) # somme sur toutes les colonnes (dimension 2)
```

```
col1 col2 col3  
10    26    42
```

Les listes (1)

Une liste peut contenir tout type d'objet

```
l1 <- list(v1, m1, v4); l1
```

```
[[1]]
```

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

```
[[2]]
```

	col1	col2	col3
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12

```
[[3]]
```

toto	elem2
1	2

4

Les listes (2)

```
length(l1) # nombre d'éléments d'une liste
```

```
[1] 3
```

```
# On peut encore nommer les éléments de la liste :
```

```
names(l1) <- c("vect1", "mat", "vect2")
```

```
l1
```

```
$vect1
```

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

```
$mat
```

	col1	col2	col3
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12

Les listes (3)

```
$vect2
```

```
  toto elem2
```

```
    1      2      4
```

```
# Pour accéder à un élément d'une liste utiliser [[,  
# autrement on a encore une liste
```

```
l1[1] # liste d'un seul élément : v1
```

```
$vect1
```

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

```
l1[[1]] # premier élément de la liste
```

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

```
# On concatène deux listes avec fonction c:
```

```
c(l1, l1[-2])
```

Les listes (4)

```
$vect1
```

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

```
$mat
```

	col1	col2	col3
row1	1	5	9
row2	2	6	10
row3	3	7	11
row4	4	8	12

```
$vect2
```

```
  toto elem2
```

```
    1      2      4
```

```
$vect1
```

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

Les listes (5)

```
$vect2  
  toto elem2  
    1      2      4
```

Le data.frame (1)

Entre les listes et matrices : comme un tableur, souvent utilisé pour stocker des données

```
d1 <- data.frame(col1 = c("a", "b", "c"), col2 = 1:3)  
d1
```

	col1	col2
1	a	1
2	b	2
3	c	3

Le tibble (1)

tibble : comme un data.frame réinventé, plus permissif

```
library(tibble)
t1 <- tibble(col1 = c("a", "b", "c"), col2 = 1:3)
t1 # ou as.tibble(d1)
```

```
# A tibble: 3 x 2
  col1    col2
  <chr> <int>
1 a         1
2 b         2
3 c         3
```

Le tibble (2)

```
# On peut aussi les définir ligne par ligne :
```

```
tribble(  
  ~col1, ~col2,  
  "a", 1,  
  "b", 2,  
  "c", 3  
)
```

```
# A tibble: 3 x 2
```

```
  col1    col2  
  <chr> <dbl>  
1 a           1  
2 b           2  
3 c           3
```

Sommaire

1. Les types de base

2. Importation des données

3. Les séries temporelles

Importer des données

Soyez fainéants et commencez par utiliser l'interface de RStudio (Environnement > Import Dataset).

Sommaire

1. Les types de base

2. Importation des données

3. Les séries temporelles

ts()

Il existe de nombreux formats pour gérer les séries temporelles. Les deux principales sont :

- `ts()` : format de base R simple à utiliser mais des difficultés à gérer les fréquences non-entières (journalières, hebdomadaires, etc.)
- `tsibble()` : inspiré du tidyverse (`tidyverts` <https://tidyverts.org>) mais pour la gestion des séries temporelles (non compatible avec les packages liés à JDemetra+)

ts() (1)

On peut créer un objet avec la fonction `ts(data = ., start = ., frequency = .)`

```
x = ts(c(1:12), start = 2020, frequency = 4)
x; class(x)
```

	Qtr1	Qtr2	Qtr3	Qtr4
2020	1	2	3	4
2021	5	6	7	8
2022	9	10	11	12

```
[1] "ts"
```

```
mts <- ts(matrix(rnorm(30), 10, 3), start = c(1961, 1),
               frequency = 12)
mts; class(mts)
```

ts() (2)

	Series 1	Series 2	Series 3
Jan 1961	-0.6264538	1.51178117	0.91897737
Feb 1961	0.1836433	0.38984324	0.78213630
Mar 1961	-0.8356286	-0.62124058	0.07456498
Apr 1961	1.5952808	-2.21469989	-1.98935170
May 1961	0.3295078	1.12493092	0.61982575
Jun 1961	-0.8204684	-0.04493361	-0.05612874
Jul 1961	0.4874291	-0.01619026	-0.15579551
Aug 1961	0.7383247	0.94383621	-1.47075238
Sep 1961	0.5757814	0.82122120	-0.47815006
Oct 1961	-0.3053884	0.59390132	0.41794156

```
[1] "mts"      "ts"       "matrix" "array"
```

Pour manipulations : voir TP