R pour les séries temporelles (compléments du Chapitre 2)

Yves Aragon*
Université Toulouse Capitole

14 novembre 2022

2.1 Obtenir de l'aide sur R

Le moteur de recherche intégré à R permet d'obtenir facilement de l'aide. On l'a indiqué dans l'introduction, R comporte de nombreux packages, certains font partie de la distribution, d'autres doivent être *installés* par l'utilisateur, généralement depuis un site miroir de CRAN. Mais il ne peut s'en servir que s'ils sont *chargés* et l'aide en ligne ne concerne que les fonctions des packages chargés. Examinons quelques situations.

1 Si l'on connaît le nom de la fonction sur laquelle on veut de l'aide et si elle fait partie d'un package chargé sur la machine, on utilise indifféremment `?` ou `help()`, ainsi :

```
> ?aggregate
> help(aggregate)
```

donnent l'un ou l'autre la syntaxe et des exemples de `aggregate()` ainsi que des liens vers des questions connexes. Si une fonction appartient à un package non installé, il faut d'abord l'installer pour avoir de l'aide sur ses fonctions.

2 Si on n'a pas de nom de fonction mais qu'on veut avoir la liste des fonctions s'intéressant à une certaine notion, on utilise `help.search()` ou `??`. Par exemple :

```
> help.search("date")
> # ou
> ??date
```

donnent la liste de toutes les fonctions, classées par package, chargés ou seulement installés, où "date" apparaît.

3 Enfin quand on ne sait pas s'il existe une fonction pour une tâche, on peut, depuis R, se renseigner sur le site de R. Exemple :

*yves.aragon@gmail.com

```
> RSiteSearch("gini")
```

En dernier recours on peut chercher sur le Net. Ainsi en tapant sur un moteur de recherche les mots :
`CRAN R date month example`, on obtient des renseignements pertinents sur la façon d'extraire le mois d'une date. Le mot `example` permet souvent de limiter les pages ; de même
`CRAN package inequality` donne, parmi beaucoup de choses sans intérêt, la liste des packages traitant de mesures d'inégalité.

Il faut systématiquement examiner l'aide en ligne des fonctions rencontrées dans ce livre. Même si l'usage en paraît évident, la fonction a souvent des options fort utiles dans d'autres circonstances. De plus, les exemples contenus dans l'aide sollicitent la réflexion du lecteur et sont très instructifs. On en tire le plus grand profit en les exécutant ligne à ligne.

Une erreur courante. Un nom de fonction est toujours suivi de parenthèses `(...)`, et inversement, *seul un nom de fonction est suivi de parenthèses.* Considérons le code suivant :

```
> num=which.min(lait)
> t.lait = time(lait)
> cat('temps collecte minimale : ',t.lait(num),'\n')
Erreur dans cat("temps collecte minimale : ", t.lait(num), "\n") :
  impossible de trouver la fonction "t.lait"
```

Dans ce code, `lait` est une série temporelle étudiée au chapitre 11. On repère le numéro de l'observation où la série est minimum. Par `time()`, on extrait le temps/date de cette série et on veut imprimer le temps du minimum, à savoir, `t.lait[num]`. Par erreur on a tapé des `( )` au lieu de `[ ]`, R croit donc que `t.lait` est une fonction, qu'il ne trouve évidemment pas. Le message d'erreur est donc clair.

2.2 Comprendre et changer la structure d'un objet

Dans R, les données multidimensionnelles sont stockées sous différentes formes : matrice, array, dataframe... ou sous des combinaisons de tels objets. Un dataframe est une collection de vecteurs de même longueur qu'on peut imaginer comme une matrice dont les colonnes peuvent être de types variés alors que les colonnes d'une matrice doivent être toutes de même type. Une liste est une structure plus générale qu'un dataframe. C'est un groupe d'objets reliés entre eux, qui peuvent être des vecteurs, des matrices, des listes... Pour savoir les noms des objets qui composent une liste, on utilise `names()`.
Les fonctions de R construisent et manipulent des objets. Ils peuvent être assez simples comme une matrice, un vecteur, ou complexes. C'est le cas des objets fabriqués par les fonctions statistiques telles que `lm()`, `ARIMA()`, qui effectuent un

ajustement linéaire, une modélisation ARIMA... Les objets en sortie de ces fonctions peuvent être des listes, par exemple la liste de tous les résultats d'un ajustement linéaire, ou des objets plus structurés appartenant à de grandes catégories appelées classes. `plot()`, `summary()`, `print()`, `coef()` sont des fonctions génériques, elles adaptent leur action à la classe de l'objet auquel on les applique. `class(a)` donne la classe de l'objet `a`. Il est parfois utile d'abandonner la classe d'un objet. On utilise pour cela `unclass()`. Nous avons très souvent recours à `str()`, pour voir comment R a importé un fichier de données (les entiers sont-ils restés des entiers, comment les dates ont-elles été comprises....), pour trouver facilement les composantes d'une sortie d'un traitement statistique qui nous intéressent. `as.vector()`, `as.numeric()`, `as.Date()`... permettent de changer la structure d'un objet, `unlist()` permet d'abandonner la structure de liste d'un objet. Ces fonctions sont d'un usage fréquent. Par exemple, un vecteur et une matrice colonne ne sont généralement pas interchangeables et on aura parfois besoin de changer leur structure, et ce genre de code est d'un emploi fréquent. Dans le code ci-dessous, `x` et `y` sont des vecteurs, et `xmat` une matrice colonne :

```
> x <- rnorm(10)
> xmat <- as.matrix(x, ncol = 1)
> y <- as.vector(xmat[, 1])
```

Classes S3. Une liste avec un attribut `class` associé indiquant de quel type de liste il s'agit, est une classe S3. On accède aux éléments d'une classe S3 par `$`. Une classe S3 est une liste avec un attribut supplémentaire. Si un objet d'une certaine classe est passé comme argument à une fonction, R cherche une fonction nommée de façon appropriée pour les objets de cette classe. Ainsi `summary()` est une fonction qui donne un résumé adapté aux sorties de différentes fonctions.
Classes S4. Elles ont été ajoutées assez récemment à R. Ces classes contiennent généralement à la fois des données et des fonctions, comme les classes S3, mais elles ont quelques avantages techniques. Pour notre usage il nous suffit de savoir qu'on fait référence à leurs éléments par @ et non `$`. Les données boursières récupérées à l'aide de **its** sont de classe S4. Nous examinons leur structure à la section 2.2. Les séries temporelles de type `timeSeries` sont des objets de classe S4, on en verra un exemple notamment au chapitre 8, *Trafic mensuel de l'aéroport de Toulouse-Blagnac.*

2.3 Exemple

Nous nous intéresserons un peu plus loin aux structures de données temporelles. Examinons maintenant la structure d'une sortie d'ajustement d'un modèle combinant une tendance linéaire et une erreur qui a une dynamique autorégressive simple. Au chapitre 1, on a utilisé le code suivant :

```
> require("forecast")
> temps <- time(LakeHuron)
```

```
> mod.lac <- Arima(LakeHuron, order = c(1, 0, 0),
+                  xreg = temps, method = "ML")
```

pour estimer le modèle :

$$y_t = \beta_0 + \beta_1 x_t + u_t, \ t = 1, \cdots, T. \tag{2.1}$$

$$u_t = \phi u_{t-1} + z_t, \qquad |\phi| < 1. \tag{2.2}$$

On obtient les résultats de l'estimation, valeur des coefficients notamment, par `summary(mod.lac)`, opération que nous ferons souvent ; mais ici examinons la structure de l'objet `mod.lac`.

```
> str(mod.lac, width = 60, strict.width = "cut")
```

```
List of 19
 $ coef     : Named num [1:3] 0.7835 618.2956 -0.0204
  ..- attr(*, "names")= chr [1:3] "ar1" "intercept" "xreg"
 $ sigma2   : num 0.512
 $ var.coef : num [1:3, 1:3] 4.01e-03 -1.48e-01 7.81e-05 -..
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "ar1" "intercept" "xreg"
  .. ..$ : chr [1:3] "ar1" "intercept" "xreg"
 $ mask     : logi [1:3] TRUE TRUE TRUE
 $ loglik   : num -105
 $ aic      : num 218
 $ arma     : int [1:7] 1 0 0 0 1 0 0
 $ residuals: Time-Series [1:98] from 1875 to 1972: 0.1908..
 $ call     : language Arima(y = LakeHuron, order = c(1, "..
 $ series   : chr "LakeHuron"
 $ code     : int 0
 $ n.cond   : num 0
 $ nobs     : int 98
 $ model    :List of 10
  ..$ phi  : num 0.783
  ..$ theta: num(0)
  ..$ Delta: num(0)
  ..$ Z    : num 1
  ..$ a    : num 1.86
  ..$ P    : num [1, 1] 0
  ..$ T    : num [1, 1] 0.783
  ..$ V    : num [1, 1] 1
  ..$ h    : num 0
  ..$ Pn   : num [1, 1] 1
 $ aicc     : num 219
 $ bic      : num 229
 $ xreg     : num [1:98, 1] 1875 1876 1877 1878 1879 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
```

```
  .. ..$ : chr "xreg"
 $ x        : Time-Series [1:98] from 1875 to 1972: 580 58..
 $ fitted   : Time-Series [1:98] from 1875 to 1972: 580 58..
 - attr(*, "class")= chr [1:3] "forecast_ARIMA" "ARIMA" ""..
```

Les options `width=60, strict.width="cut"` limitent la largeur du texte imprimé ; elles ne sont là que pour les besoins de la mise en page du livre et ne sont généralement pas utilisées dans la pratique. On voit que `mod.lac` est une liste de 15 objets dont `mod.lac$model` est lui-même une liste. On peut alors récupérer une composante de la liste, le coefficient du temps dans la régression... On voit par exemple, que le nom de ce coefficient est `temps`, on peut donc, et c'est plus sûr que de repérer la position d'un terme dans un vecteur, le récupérer par son nom :

```
> residus <- mod.lac$residuals
> (coeftemps <- mod.lac$coef[names(mod.lac$coef) == "temps"])
```

```
named numeric(0)
```

`fitted()`, `residuals()`, `coefficients()` sont des fonctions génériques qui permettent également de récupérer certains résultats. Il arrive que les sorties d'une fonction soient très abondantes et dépassent la capacité de la console, si bien qu'on en perd le début. `sink()` permet d'orienter la sortie vers un fichier. Par exemple :

```
sink('d:/outmod.txt')
mod.lac
sink()
```

La première instruction oriente la sortie vers le fichier texte `outmod.txt` du répertoire `D:`, la deuxième écrit la structure de l'objet `mod` dans ce fichier et la troisième, qu'il ne faut pas oublier, redirige la sortie vers la console.