Introduction au logiciel R

Laurence Viry

MaiMoSiNE - Collège des écoles doctorales Grenoble

7-16 Février 2017

Plan du cours

Introduction au logiciel R

Laurence Viry

MaiMoSiNE - Collège des écoles doctorales Grenoble

7-16 Février 2017

Les objets dans R

R utilise des fonctions et des opérateurs qui agissent sur des **objets**.

Les principaux **modes** ou **types** de ces objets sont:

Mode	Contenu de l'objet
numeric	nombres réels
complex	nombres complexes
logical	valeurs booléennes (vrai/faux)
character	chaînes de caractères
function	fonction
list	données quelconques
expression	expressions non évaluées

- pour connaîître le mode d'un objet de R, on utilise la fonction **mode()**
- La fonction **typeof()** permet d'obtenir une description plus précise de la **re-présentation interne d'un objet**.
- **Un objet** a **deux attributs intrinsèques**, son **mode** et sa **longueur**. Il peut avoir des attributs spécifiques qui diffèrent selon le type de l'objet (dim, dimnames,class,...).

```
> f <- "R"
> mode(f)
[1] "character"
> typeof(f)
[1] "character"
> ff <- c(3,"E")
> mode(ff)
[1] "character"
> mode(3L)
[1] "numeric"
> mode(3)
[1] "numeric"
> typeof(3L) # 3L entier 3
[1] "integer"
> typeof(3) # réel double précision
[1] "double"
> #3 == 3L;identical(3,3L) # égaux mais pas identique
```

```
> v <- c(1,5,98)
> mode(v)
[1] "numeric"
> length(v);
[1] 3
> attributes(v)
NULL
> mat <- matrix(2:8,ncol=4)
> mode(mat)
[1] "numeric"
> length(mat)
[1] 8
> attributes(mat)
$dim
[1] 2 4
```

Tester, convertir le mode d'un objet

- **Tester le mode d'un objet** avec les fonctions:
  - **is.null**(x)
  - **is.logical**(x)
  - **is.numeric**(x)
  - **is.complex**(x)
  - **is.character**(x)

  Le résultat est un **booléen** (TRUE,FALSE)

- on peut **convertir** un objet d'un mode à un autre de façon explicite grâce aux fonctions:
  - **as.logical**(x)
  - **as.numeric**(x)
  - **as.complex**(x)
  - **as.character**(x)

  Attention!!! R retourne un résultat de conversion, même si elle n'a pas de sens

NaN, INf et -INF

- **La division par zéro d'un nombre non nul** donne **Inf** ou **-Inf**suivant le signe du dénominateur

```
> 67/0
[1] Inf
> -10/0
[1] -Inf
> 0/0 # division par 0, indéfinie
[1] NaN
```

NaN comme Not a Number

- **R sait gérer les indéterminations**

```
> Inf-3
[1] Inf
> 1/Inf
[1] 0
> Inf/0
[1] Inf
> Inf-Inf
[1] NaN
```

NaN, INf et -INF (suite)

- **Les indéterminations se propagent**

```
> 5*log(-9)
[1] NaN
```

- **R sait gérer les données manquantes** NA pour Not Available

```
> 4*NA
[1] NA
```

- **Fonctions tests associées**

```
> is.finite(23.54)
[1] TRUE
> is.finite(5/0)
[1] FALSE
> is.nan(Inf-Inf)
[1] TRUE
> is.na(NA)
[1] TRUE
> x <- c(3,NA,6,9,7,NA)
> is.na(x)
[1] FALSE  TRUE FALSE FALSE FALSE  TRUE
```

Fonctions et opérateurs - package "base"

Group generic methods can be defined for four pre-specified groups of functions, Math, Ops, Summary and Complex. **help(Math)**

- Group "Math": abs, sign, sqrt,floor, ceiling, trunc,round, signif
  exp, log, expm1, log1p, cos, sin, tan, cospi, sinpi, tanpi, acos, asin, atan
  cosh, sinh, tanh, acosh, asinh, atanh
  ...
- Group "Ops"
- Group "Summary": all, any, sum, prod, min, max,range
- Group "Complex": Arg, Conj, Im, Mod, Re

mode(), typeof() et class()

- **mode(object)** et **typeof(objet)** nous renseignent sur la façon dont l'**objet est stocké en mémoire**.
- **class(object)** nous donne la **classe d'un objet**. Le principal intérêt de la classe est de savoir ce que font les fonctions génériques (comme print() ou plot()) sur cet objet.

```
> ai <- 4L
> mode(ai);class(ai)
[1] "numeric"
[1] "integer"
> ar <- 4
> mode(ar);class(ar)
[1] "numeric"
[1] "numeric"
> class(pi)
[1] "numeric"
```

Méthode génériques

- La classe d'un objet **détermine le comportement d'une méthode générique sur cet objet**.
- L'utilisateur qui crée ses propres objets pourra ou devra **réécrire les méthodes génériques adaptée à cet objet**.
- Pour connaître la spécialisation d'une méthode générique on utilisera la fonction **method()**

```
> x <- 1:12
> class(x)
[1] "integer"
> y <- matrix(1:12, nrow = 2, ncol = 6)
> class(y)
[1] "matrix"
> print(x)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> print(y)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
```

Principaux objets prédéfinis dans R

On distinguera **les objets atomiques**, d'un type unique et **les objets hétérogènes** composés de plusieurs composantes de **mode** et de **longueurs différents**.

- **Vecteurs**: **Objet atomiques** composé d'ensemble de valeurs appelées composantes, coordonnées ou éléments (tableau 1D).
- **Matrices**: **Objet atomique**, chaque valeur de la matrice peut être repérée par son **numéro de ligne** et son **numéro de collone** (tableau 2D).

  On peut construire des tableaux de dimension supérieure à 2, avec la fonction **array**. La manipulation de ces tableaux est identique à celle des matrices.
- **Facteurs**: **vecteurs** permettant de traiter les données qualitatives.
- **Listes**: **Objet hétérogène**. C'est un ensemble ordonné d'objets ou composantes qui n'ont pas tous le même mode et la même longueur.
- **Data-frame**: **Listes particulières** dont les composantes sont **de même longueur**, mais elles peuvent être de **mode différent**. Les **tableaux de données** utilisés en statistiques sont souvent considérés comme des data-frames.
- **Fonctions**: une **fonction** est **un objet R** (prédéfinie, utlisateur).

Vecteurs dans R - Constructeurs

Les **vecteurs** sont des **objets atomiques** avec **deux attributs intrinséques**, **mode**(mode()) et **length**(length()).

- **Les vecteurs numériques**: **plusieurs constructeurs sont fournis par R**:
  - Construction par la fonction collecteur **c**:
    ```
    > x <- c(1,0,76)
    > x <- c(2,6,-1,c(3,4))
    > y <- c(3,0,x)
    ```
  - Construction par l'opérateur séquence **:**:
    ```
    > 1:6
    ```
  - Construction par la fonction **seq**:
    ```
    > seq(1,6,by=0.5)
    > seq(1,6,length=5)
    ```
  - Construction par la fonction **rep**:
    ```
    > rep(1,4)
    > rep(c(1,2),each=3); rep(c(1,2),3)
    ```
  - Construction par la fonction **scan**: R demande l'entrée des données au fur et à mesure. Si n est absent, la fin des entrées est provoquée par une valeur vide.
    ```
    > mesures <- scan(n=4)
    ```
- **Les vecteurs de caractères**
- **Les vecteurs logiques**

Vecteurs dans R (suite constructeurs)

- **Les vecteurs numériques**
- **Les vecteurs de caractères**: il est possible de créer des vecteurs caractères en utilisant les fonctions **c** et **rep**.

```
> x <- c("A","L","I","C","E")
> y <- rep("A",4)
```

- **Les vecteurs logiques**: les vecteurs booléens peuvent être générés par **des opérateurs logiques** (voir help(Math), Group "Ops")

```
> a <- 1>0
> x <- c(-1,2,3,-7,5,0)
> test <- x > 0
> test
[1] FALSE  TRUE  TRUE FALSE  TRUE FALSE
> y <-(1+x^2)*(x>0)
> all(x>0) # TRUE si tous >0
[1] FALSE
> any(x>0) # TRUE siun des éléments > 0
[1] TRUE
```

Manipulation vecteurs

- **Sélection d'une partie d'un vecteur**: elle s'opère avec l'opérateur de sélection [] et **un vecteur de sélection** qui peut être un vecteurs d'entiers positifs, négatifs ou de logiques:

```
> v <- 1:100
> v[6]      # 6 ième élément
> v[6:15]   # les éléments de 6 à 15
> v[3,3,c(1,5)] # 3 ième, 3ième et premier et cinquième élément
> v[4:1]    # 4 ième, 3ième,...
> v[-(1:5)] # v sans ses 5 premiers éléments
> v[(v<5)]
> v[(v<5) & (v>=12)] # & signifie "et"
> T <- c(23,28,21,32) ; xx <- c(45,67,32,1)
> xx[T>24]
```

- **Quelques sélections**

```
> x[is.na(x)] <- 0 # les éléments NA de x, reçoivent la valeur 0
> y[y<0] <- -y[y<0]  #remplacement des éléments negatifs par leu
> x <- c(10:15,45:50,1:4)
> which.min(x ); which(x==min(x)); (1:length(x))[x == min(x)]
```

Manipulation vecteurs de caractères

- Pour **la concaténation** on utilise la fonction **paste**:

```
> paste("A",1:5,sep="_")
[1] "A_1" "A_2" "A_3" "A_4" "A_5"
> paste(c("X","Y"),1:4,"txt",sep=".")
[1] "X.1.txt" "Y.2.txt" "X.3.txt" "Y.4.txt"
> paste(c("X","Y"),1:4,sep=".",collapse = "+")
[1] "X.1+Y.2+X.3+Y.4"
```

- Pour **l'extraction**, on utilise la fonction **substr**

```
> substr("formater",4,8)
[1] "mater"
```

Matrices dans R

Les **matrices** sont des **objets atomiques** (dimension 2)

- **attributs**: **mode**, **length**, **un attribut** "**dim**" qui fournit le nombre de lignes et de colonnes et **un attribut optionnel** "**dimnames**", liste qui contient le nom des lignes et des colonnes.
- **Constructeurs**: utilisation de la fonction **matrix**
  Par défaut, R range les valeurs par colonne.

```
> m1 <- matrix(c(3,4,5,10,23,6,0,-1),ncol=4)
> dim(m1)
```

```
[1] 2 4
```

```
> m2 <- matrix(1:8,nrow=4)
> dim(m2)
```

```
[1] 4 2
```

Pour les **ranger par lignes** en mémoire, on utilise l'argument **byrow**:

```
> m <- matrix(1:8,nrow=2,byrow=TRUE) # stockage par ligne
> dim(m)
```

```
[1] 2 4
```

Matrices dans R

- **Constructeurs**: utilisation de la fonction **matrix**

  Si la longueur du vecteur est différente du nombre d'éléments de la matrice, R remplit la matrice (indique un warning):

```
> m <- matrix(1:4,nrow=3,ncol=3) # répète le vecteur
> m <- matrix(1:10,nrow=2,ncol=4) # prend les premiers éléments
> un <- matrix(0,nrow=2,ncol=2)  # initialise à 0 toute la matri
```

  Un vecteur n'est pas de class "matrix", il peut être transformé en matrice uni-colonne.

```
> v <- seq(1,10,by=2)
> class(v)
> mv <- as.matrix(v)
> class(mv)
```

- **Constructeurs**: utilisation des fonctions **rbind** et **cbind**

```
> rbind(c(1, 2, 3), c(4, 5, 6)) # chaque vecteur une ligne
> cbind(c(1, 2, 3), c(4, 5, 6)) # chaque vecteur une colonne
```

Sélection d'élément ou d'une partie de matrice

Sélection par des entiers positifs

```
> m <- matrix(1:8, ncol=4)
> a <-m[2,] # 2ième ligne sous forme de vecteur
> class(a)
[1] "integer"
> a <-m[2,,drop=F] # 2ième ligne sous forme matrice
> m[,c(2,2,1)] # 2ième, 2ième, 1ère collonnes
     [,1] [,2] [,3]
[1,]    3    3    1
[2,]    4    4    2
```

Sélection par des entiers négatifs

```
> m[-1,] # m sans sa première ligne
[1] 2 4 6 8
> m[1:2,-1] # 2 premières lignes sans la 1ère colonne
     [,1] [,2] [,3]
[1,]    3    5    7
[2,]    4    6    8
```

Sélection d'élément ou d'une partie de matrice (suite)

Sélection par des logiques

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> m[,m[1,]>2] # colonne dont le premier élément > 2
     [,1] [,2] [,3]
[1,]    3    5    7
[2,]    4    6    8
> m[m[,3]> 5,] # ligne dont la troisème colonne > 5
[1] 2 4 6 8
> m[m>2] # vecteur contenant les valeurs de m >2
[1] 3 4 5 6 7 8
> m[m>2] <- NA # remplace les valeurs de m >2 par NA
> m
     [,1] [,2] [,3] [,4]
[1,]    1   NA   NA   NA
[2,]    2   NA   NA   NA
```

Calcul sur les matrices - opérations termes à termes

```
> A <- matrix(1:4,ncol=2)
> B <- matrix(3:6,ncol=2,byrow=T) # optimise les accès mémoire
> A+B # somme élément par élément
     [,1] [,2]
[1,]    4    7
[2,]    7   10
> A*B # produit élément par élément
     [,1] [,2]
[1,]    3   12
[2,]   10   24
> sin(A) # sinus élément par élément
          [,1]       [,2]
[1,] 0.8414710  0.1411200
[2,] 0.9092974 -0.7568025
> exp(A) # exponentielle élément par élément
         [,1]     [,2]
[1,] 2.718282 20.08554
[2,] 7.389056 54.59815
```

Calcul sur les matrices - opérations algébriques

Quelques fonctions:

Fonction	Description
X%*%Y	produit de matrices
t(X)	transposition d'une matrice
diag(6)	matrice diagonale de vecteur v
crossprod(X,Y)	produit t(X)%*%Y
det(X)	déterminant de la matrice X
svd(X)	décomposition en valeurs singulières
eigen(X)	diagonalisation d'une matrice
solve(A,b)	résolution d'un système linéaire
chol(Y)	décomposition de cholesky
qr(Y)	Décomposition QR

Utiliser **help.search("linear")**

Exemples d'utilisation eigen et solve

```
> A <- matrix(1:4,ncol=2)
> B <- matrix(c(5,7,6,8),ncol=2)
> D <- A%*%t(B) # produit des matrices
> eig <- eigen(D) # diagonalisation de D
> eig
$values
[1] 68.9419802  0.0580198

$vectors
           [,1]       [,2]
[1,] -0.5593385 -0.8038173
[2,] -0.8289393  0.5948762
> attributes(eig)
$names
[1] "values"  "vectors"
> V <-c(1,2)
> solve(D,V)
[1] -4  3
```

Opérations sur les lignes et colonnes

- Les fonctions **ncol**,**nrow** et **dim**

```
> A <- matrix(1:6,ncol=3)
> ncol(A)
> nrow(A)
> dim(A)
```

- la fonction **apply** permet d'appliquer une fonction f aux lignes (MARGIN=1) ou aux colonnes (MARGIN=2) de la matrice

```
> A <- matrix(1:6,ncol=3)
> apply(A,MARGIN=2,sum) # somme par colonne
> apply(A,MARGIN=1,mean) # moyenne par ligne
```

- **Concatéation** par colonne avec **cbind**, par ligne avec **rbind**

```
> cbind(c(1,2),c(-3,7))
```

On peut construire des tableaux de dimensions supérieures grâce à la fonction **array**. La manipulation de ces tableaux est similaire à celle des matrices.

Les facteurs

Les **facteurs** sont des **vecteurs** (objet atomique) permettant le traitement des **données qualitatives**.

- **attributs**: les deux attributes intrinséques **mode** et **length** et l'attribut **levels** qui contient les modalités du facteur.

  Les modalités du facteur sont obtenues grâce à la fonction **levels**()

- **Constructeur**: **trois fonctions** permettent de créer des **facteurs**:
  - La fonction **factor**

```
> sexe <- factor(c("M","F","M","M","M","F"))
> sexe
[1] M F M M M F
Levels: F M
> levels(sexe)
[1] "F" "M"
> mode(sexe)
[1] "numeric"
> typeof(sexe)
[1] "integer"
> class(sexe)
[1] "factor"
```

Les facteurs

- **Constructeur**: **trois fonctions** permettent de créer des **facteurs**:
  - la fonction **ordered**

```
> niveau <- ordered(c("deb","deb","cha","moyen","cha"),
+                   levels=c("deb","moyen","cha"))
> niveau
[1] deb   deb   cha   moyen cha
Levels: deb < moyen < cha
> class(niveau)
[1] "ordered" "factor"
```

  - La fonction **as.factor**

```
> salto <- c(1:5,7:3)
> summary(salto)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       3       4       4       5       7
> salto.f <- as.factor(salto)
> class(salto.f)
[1] "factor"
> summary(salto.f)
1 2 3 4 5 6 7
1 1 2 2 2 1 1
```

Les facteurs -

Pour retrouver **les modalités**, **le nombre de modalités**, **l'effectif par modalité**, on utilisera respectivement les fonctions **levels()**, **nlevels()** et **table()**:

```
> salto <- c(1:5,7:3)
> salto.f <- as.factor(salto)
> levels(salto.f)
[1] "1" "2" "3" "4" "5" "6" "7"
> nlevels(salto.f)
[1] 7
> table(salto.f)
salto.f
1 2 3 4 5 6 7
1 1 2 2 2 1 1
```

La fonction **table** permet égalemet de construire des tablraux croisés.

Pour convertir les facteurs en numérique:

```
> x <- factor(c(10,11,13))
> t <- as.numeric(x )
> tc <- as.numeric(as.character(x) )
```

Les listes

Les **listes** sont des objets **hétérogènes**. C'est un ensemble ordonné d'objets ou **composantes** qui n'ont pas toujours le même mode ou la même longueur.

- **attributs**: les deux attributes intrinséques **mode** et **length** et l'attribut supplémentaires **names** qui contient les modalités du facteur.

  Les noms des composantes sont obtenues grâce à la fonction **names**()
- Les **listes** sont utilisées par les fonctions qui retournent plusieurs objets.
- **Constructeur**: la fonction **list** permet de créer une **liste**:

```
> vecteur <- c(2,10,by=2)
> matrice <- matrix(1:8,ncol=2)
> facteur <- factor(c("M","F","M","M","M","F"))
> maListe <- list(vecteur,matrice,facteur)
> mode(maListe)
[1] "list"
> length(maListe)
[1] 3
> names(maListe)
NULL
```

```
> attributes(maListe)

NULL

> names(maListe) <- c("vec","mat","genre")
> attributes(maListe)

$names
[1] "vec"   "mat"   "genre"

> maListe

$vec
      by
 2 10  2


$mat
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8

$genre
[1] M F M M M F
Levels: F M
```

Les listes: extraction

Pour extraire une composante de la liste, on utilise l'opérateur **[[]]** en indiquant **la position de la composante** qu'on veut extraire.

```
> maListe[[1]]
```

```
      by
 2 10  2
```

On peut aussi utiliser **le nom de la composante**.

```
> maListe$genre
> maListe[["genre"]]
```

On peut extraire plusieurs élément d'une même liste en utilisant l'opérateur **[]**.

```
> maListe[c(1,3)]
```

```
$vec
      by
 2 10  2

$genre
[1] M F M M M F
Levels: F M
```

Quelques fonctions sur les listes

- la fonction **lapply**: applique une fonction(moyenne, variance,...) à chacune des composantes.

```
> x <- list(a = 1:10, beta = exp(-3:3),
+           logic = c(TRUE,FALSE,FALSE,TRUE))
> # compute the list mean for each list element
> lapply(x, mean)# median and quartiles for each list element
$a
[1] 5.5

$beta
[1] 4.535125

$logic
[1] 0.5
```

- Pour **concaténer** deux listes, on utilise la fonction **c()**

```
> l1 <- list(a = 1:10, beta = exp(-3:3))
> l2 <- list(ch=c("M","A","A"), re=seq(0,1,20))
> ll <- c(l1,l2)
```

la liste dimnames

La liste **dimnames** est un attribut optionnel d'une matrice, elle a deux composantes, le nom des lignes et le nom des colonnes.

```
> A <- matrix(1:12,nrow=4)
> nomligne <- c("ligne1","ligne2","ligne3","ligne4")
> nomcolonne <- c("col1","col2","col3")
> dimnames(A) <- list(nomligne,nomcolonne)
> A
```

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

Observez les attributs de la matrice A.

Les data-frames

Les **data-frame** sont des listes particulières dont les composantes sont de **même longueur** mais dont **les modes peuvent être différents**. Les **tableaux de données** usuels utilisés en statistique sont considérés comme des **data-frame**.

- Les principales manières de construire un **data-frame** est d'utiliser une des fonctions:
  - **data.frame** qui permet de concaténer des vecteurs de même taille.

```
> vec1 <- 1:5
> vec2 <- c("a","f","s","o","p")
> df <- data.frame(nomVar1=vec1,nomVar2=vec2)
> df
```

```
  nomVar1 nomVar2
1       1       a
2       2       f
3       3       s
4       4       o
5       5       p
```

  - En utilisant la fonction **read.table** lit un tableau de donnée et crée un **data-frame**
  - En utilisant la fonction **as.data.frame** pour effectuer une conversion explicite.

Les data-frames

- Pour extraire les composantes des data-frame, on peut utiliser les méthodes pour les listes ou pour les matrices.
- Pour transformer une **matrice** en **data-frame**, on utilise la fonction **as.data.frame**.
- Pour transformer un **data-frame** en une **matrice** en , on utilise la fonction **data.matrix**.