Notions de Programmation

sous R

Pr Roch Giorgi

roch.giorgi@univ-amu.fr

SESSTIM, Faculté des sciences médicales et paramédicale, Aix-Marseille Université, France
http://sesstim.univ-amu.fr/

Bases du Langage R (1)

- Commandes
  - ✓ Expression / Affectation

Expression : écrite dans la ligne de commande pour une évaluation immédiate du code

```
> 4*2
[1] 8
>
```

Affectation : création d'un objet contenant un résultat et stocké en mémoire dans l'espace de travail en cours de R

```
> x <- 4*2
> x
[1] 8
```

Bases du Langage R (2)

- Commandes
  - ✓ Expression / Affectation
    - Symbole d'affectation

      <-

      ou

      =
    - Commentaires : le signe = est source potentielle de confusion pour désigner une égalité (ce n'est pas le cas), dans les fonctions

Bases du Langage R (3)

- Commandes
  - ✓ Séparateur de commandes

Retour à la ligne

```
> x <- 2
> y <- 8
>
```

Utilisation de ;

```
> x <- 2 ; y <- 8
>
```

Bases du Langage R (4)

- Commandes
  - ✓ Regroupement de commandes

Retourne la dernière commande

```
> {
+   x <- 2
+   y <- 8
+ }
>
> {
+   x <- 2
+   y <- 8
+   x + y
+ }
[1] 10
```

Remarque : cette règle est importante dans le développement de fonctions

Noms d'Objets

- Lettres minuscules et lettres majuscules
  - ✓ Attention : R est sensible à la casse (temp ≠Temp ≠...)
- Chiffres de 0 à 9
  - ✓ Pas en premier
- Caractères « . » et « _ »
- A éviter, car utilisés par R
  - ✓ c, q, t, C, D, I, pi, var,...
- Interdit d'utiliser, car réservés pour R
  - ✓ break, else, for, in,...

Objets R

- Le langage R est orienté objet
- Sont des objets
  - ✓ Variables contenant des données
  - ✓ Les fonctions
  - ✓ Les opérateurs
  - ✓ … en fait tout !
- Caractérisation d'un objet
  - ✓ Mode
  - ✓ Longueur
  - ✓ Attribut(s) (pas tous)

Objets R – Modes et Types (1)

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

Simple : contiennent des données d'un seul type
Récursifs : peuvent contenir d'autres objets

Objets R – Modes et Types (2)

- Fonctions utiles
  - ✓ `mode`
  - ✓ `typeof`

```
> v1 <- c("a", "b")
> v1
[1] "a" "b"
> mode(v1)
[1] "character"
> typeof(v1)
[1] "character"
```

Objets R – Modes et Types (3)

- Fonctions utiles
  - ✓ `mode`
  - ✓ `typeof`

```
> v2 <- list(v1, 1:4)
> v2
[[1]]
[1] "a" "b"

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

> mode(v2)
[1] "list"
> typeof(v2)
[1] "list"
```

Objets R – Modes et Types (4)

- Fonctions utiles

Mode	Test si	Conversion en
`numeric`	`is.numeric`	`as.numeric`
`complex`	`is.complex`	`as.complex`
`logical`	`is.logical`	`as.logical`
`character`	`is.character`	`as.character`
`function`	`is.function`	`as.function`
`list`	`is.list`	`as.list`
`expression`	`is.expression`	`as.expression`

Objets R – Modes et Types (5)

- Que renvoient ?

```
> (v1 <- is.logical(5 > 6))
> is.character(v1)
> is.character("v1")
> is.logical(v1)
> as.character(v1)
```

Objets R – Longueur et Valeurs (1)

- Egale au nombre d'éléments qu'il contient
- Fonctions utiles pour déterminer la longueur
  - ✓ `length`
  - ✓ `nchar`

```
> v1 <- c("TRUE", "TRUE", " FALSE", "TRUE")
> v2 <- c(1, 4, 2, 6, 8, 10)
> nchar(v1)
[1] 4 4 6 4
> length(v1)
[1] 4
> nchar(v2)
[1] 1 1 1 1 1 2
> length(v2)
[1] 6
```

Objets R – Longueur et Valeurs (2)

- Egale au nombre d'éléments qu'il contient
- Fonctions utiles pour déterminer la longueur
  - ✓ `length`
  - ✓ `nchar`
- Fonctions utiles pour déterminer la valeur
  - ✓ `is.null` : mode NULL, longueur 0
  - ✓ `is.na` : donnée manquante
    - Certaine fonction ont un argument `na.rm` (`mean`, `sum`,...)
  - ✓ `is.infinite, is.finite`
  - ✓ `is.nan` : « not a number »

Objets R – Attributs (1)

- Eléments d'informations additionnels liés à l'objet
- Principaux attributs
  - ✓ `class` : classe d'un objet affectant son comportement
  - ✓ `dim` : dimensions des matrices et tableaux
  - ✓ `dimnames` : étiquettes des dimensions des matrices et tableaux
  - ✓ `names` : étiquettes des éléments d'un objet
  - ✓ `row.names` : étiquettes des lignes d'un data frames

Objets R – Attributs (2)

- Fonctions utiles
  - ✓ `attributes`
  - ✓ `attr`
  - ✓ Que renvoient-elles ?

```
> x <- cbind(a=1:3, t.pi=pi)
> x
     a     t.pi
[1,] 1 3.141593
[2,] 2 3.141593
[3,] 3 3.141593
> attributes(x)
```

Objets R – Attributs (3)

- Fonctions utiles
  - ✓ `attributes`
  - ✓ `attr`
  - ✓ Que renvoient-elles ?

```
> x <- cbind(a=1:3, t.pi=pi)
> attributes(x)
```

```
$dim
[1] 3 2

$dimnames
$dimnames[[1]]
NULL

$dimnames[[2]]
[1] "a"  "t.pi"
```

Objets R – Attributs (4)

- Fonctions utiles
  - ✓ `attributes`
  - ✓ `attr`
  - ✓ Que renvoient-elles ?

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> attr(x,"dim") <- c(2, 5)
```

Objets R – Attributs (5)

- Fonctions utiles
  - ✓ `attributes`
  - ✓ `attr`
  - ✓ Que renvoient-elles ?

```
> x <- 1:10
> attr(x,"dim") <- c(2, 5)
> x
```

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

Vecteurs (1)

- Dans R tout est un vecteur
  - ✓ Les tableaux, matrices sont linéarisés en interne dans R
- Fonctions utiles
  - ✓ `c()`
  - ✓ `numeric()`
  - ✓ `logical()`
  - ✓ `character()`
  - ✓ Que renvoie ?

①
```
> c("a", "b", "c")
```

Vecteurs (1)

- Dans R tout est un vecteur
  - ✓ Les tableaux, matrices sont linéarisés en interne dans R
- Fonctions utiles
  - ✓ `c()`
  - ✓ `numeric()`
  - ✓ `logical()`
  - ✓ `character()`
  - ✓ Que renvoie ?

①
②
```
> c("a", "b", "c")
> numeric(3)
```

Vecteurs (1)

- Dans R tout est un vecteur
  - ✓ Les tableaux, matrices sont linéarisés en interne dans R
- Fonctions utiles
  - ✓ `c()`
  - ✓ `numeric()`
  - ✓ `logical()`
  - ✓ `character()`
  - ✓ Que renvoie ?

①
```
> c("a", "b", "c")
```
②
```
> numeric(3)
```
③
```
> (v1 <- c(a=2, b=6, c=3))
```

Vecteurs (1)

- Dans R tout est un vecteur
  - ✓ Les tableaux, matrices sont linéarisés en interne dans R
- Fonctions utiles
  - ✓ `c()`
  - ✓ `numeric()`
  - ✓ `logical()`
  - ✓ `character()`
  - ✓ Que renvoie ?

①
②
③
④

```
> c("a", "b", "c")
> numeric(3)
> (v1 <- c(a=2, b=6, c=3))
> v2 <- c(2, 6, 3)
> names(v2) <- c("a", "b", "c")
> v2
```

Vecteurs (2)

- Indiçage dans un vecteur à l'aide des [ ]
  - ✓ Position fixe : valeur du(des) élément(s)
  - ✓ Position relative : nom du(des) élément(s)
  - ✓ Que renvoient ?

```
> v2 <- c(2, 6, 3)
> names(v2) <- c("a", "b", "c")
① > v2[c(1, 3)]
② > v2[c("c", "a")]
③ > v2[c(-1)]
④ > v2 < 5
⑤ > v2[v2 < 5]
⑥ > v2[ ]
```

Facteurs (1)

- Un « factor » est un vecteur d'entiers
  - ✓ Chaque valeur correspond à un code de modalité
  - ✓ On peut attribuer des noms aux modalités
  - ✓ Classe « factor »
  - ✓ Fonctions `factor`, `is.factor`, `as.factor`

```
> v1 <- c(1, 2, 1, 1, 2, 1, 1)
> (sexe <- factor(v1))
[1] 1 2 1 1 2 1 1
Levels: 1 2
> levels(sexe)
[1] "1" "2"
```

- Que renvoient ?

```
> is.factor(v1)
> is.factor(sexe)
```

Facteurs (2)

- Comment obtenir ?

①

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

```
> levels(sexe) <- c("homme", "femme")
> sexe
```

```
> sexe <- factor(v1, labels=c("homme", "femme"))
> sexe
```

②

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

```
> factor(v1, labels=c("femme", "homme"), levels=c(2, 1))
```

```
> relevel(sexe, ref="femme")
```

Matrices et Tableaux (1)

- Correspondent à des vecteurs ayant un attribut `dim`
  - ✓ Matrice
    - `dim` =2
    - Classe « matrix »
    - Contenu de même type
    - Fonctions `matrix`, `is.matrix`, `as.matrix`

```
> (amat <- matrix(1:8, nrow=2, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> is.matrix(amat)
[1] TRUE
```

    - Quels sont la classe, le mode et le type de la matrice suivante ?

```
> class(matrix(c(1, "2", 3, "4"), nrow=2, ncol=2))
```

Matrices et Tableaux (2)

- Correspondent à des vecteurs ayant un attribut `dim`
  - ✓ Tableau
    - Matrice à plus de 2 dimension
    - Classe « array »
    - Fonctions `array`, `is.array`, `as.array`

```
> (bmat <- array(1:12, dim = c(2, 3, 2)))
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
> is.matrix(bmat)
[1] FALSE
```

Matrices et Tableaux (3)

- Dimensions des tableaux

1ère dimension : lignes

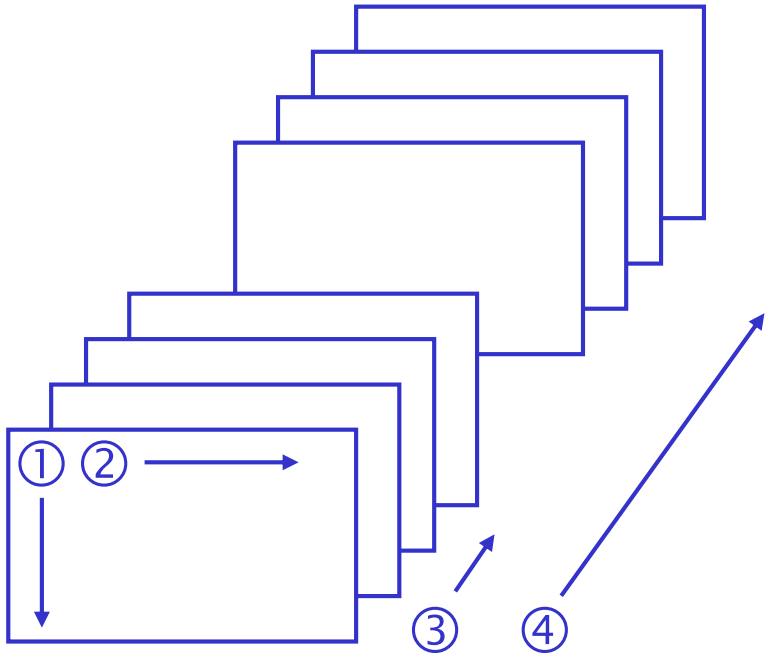
2ème dimension : colonnes

3ème dimension : strates

```
> (bmat <- array(1:12, dim = c(2, 3, 2)))
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Matrices et Tableaux (4)

- Ordre de remplissage des tableaux



Matrices et Tableaux (5)

```
> (amat <- matrix(1:8, nrow=2, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

- Comment obtenir ?

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

```
> (amat <- matrix(1:8, nrow=2, ncol=4, byrow=TRUE))
# ou...
> (amat <- matrix(c(1, 5, 2, 6, 3, 7, 4, 8), nrow=2, ncol=4))
```

Matrices et Tableaux (6)

- Opérations sur des tableaux
  - ✓ Indiçage dans un tableau à l'aide des [ ]
    - Avec des « , » pour séparer les dimensions
    - Indiçage par une(des) position(s) fixe(s) ou relative(s)
      - Cf. vecteur pour le principe
  - ✓ Fusion de matrices
    - Fonctions `rbind`, `cbind`

Matrices et Tableaux (7)

```
> (amat <- matrix(1:8, nrow=2, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> (bmat <- matrix(21:24, nrow=4, ncol=1))
     [,1]
[1,]   21
[2,]   22
[3,]   23
[4,]   24
> (cmat <- matrix(9:12, nrow=1, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    9   10   11   12
```

```
> rbind(amat, cmat)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
[3,]    9   10   11   12
```

Matrices et Tableaux (8)

- Comment obtenir ?

①

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7   21
[2,]    2    4    6    8   22
```

```
cbind(amat, bmat[1:2, ])
```

②

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
[3,]   21   22   23   24
```

```
rbind(amat, t(bmat))
```

Listes (1)

- Vecteur de type spécial dont les éléments peuvent être de n'importe quel ordre
  - ✓ Classe « list »
  - ✓ Fonctions `list`, `is.list`, `as.list`

```
> (alist <- list(x=runif(5), y=matrix(1:8, nrow=2, ncol=4),
+  z=c("MQERS", "Aix-Marseille Université")))
$x
[1] 0.9569821 0.8808573 0.7476618 0.1148537 0.4046859

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

$z
[1] "MQERS"                    "Aix-Marseille Université"
```

Listes (2)

- ✓ Type des éléments de la liste

```
> str(alist)
List of 3
 $ x: num [1:5] 0.957 0.881 0.748 0.115 0.405
 $ y: int [1:2, 1:4] 1 2 3 4 5 6 7 8
 $ z: chr [1:2] "MQERS" "Aix-Marseille Université"
```

- ✓ Indiçage des éléments d'une liste
  - Obtention du 2ème élément de la liste

```
> alist[[2]]  # ou alist[["y"]]
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> # Comparez au résultat obtenu avec : alist[2]
> # Comparez au résultat obtenu avec : alist$y
```

Listes (3)

- ✓ Indiçage des éléments d'une liste

```
> alist$y
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

  - Comment obtenir ?

```
     [,1] [,2]
[1,]    1    7
[2,]    2    8
```

```
> alist$y[ , c(1, 4)]
> # ou
> alist[["y"]][ , c(1, 4)]
```

Listes (4)

```
> (matrix(c(1,5,2,6,3,7,4,8), nrow=2, ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

- Comment obtenir ?

```
  A B C D
1 1 2 3 4
2 5 6 7 8
```

```
> matrix(c(1,5,2,6,3,7,4,8), nrow=2, ncol=4,
+ dimnames=list(1:2, c("A", "B", "C", "D")))
> # ou
> amat <- matrix(c(1,5,2,6,3,7,4,8), nrow=2, ncol=4)
> (dimnames(amat) <- list(1:2, c("A", "B", "C", "D")))
```

Data Frame

- C'est une liste de classe « data.frame »
  - ✓ Tous les éléments ont la même longueur
  - ✓ Tableau à 2 dimensions, chaque élément de la liste correspond à une colonne (accès avec `$`)
  - ✓ Les colonnes peuvent être de modes différents
  - ✓ Fonctions `data.frame`, `is.data.frame`, `as.data.frame`
  - ✓ Indiçage possible avec [ ]
  - ✓ Opérations de fusion possibles (`rbind`, `cbind`)
  - ✓ Ses colonnes sont rendues visibles dans l'espace de travail avec la fonction `attach`; masquées avec la fonction `detach`

Opérations dans R (1)

- Arithmétiques (aussi logiques – non traitées)
  - ✓ Rappel : le vecteur est l'unité de base dans R
  - ✓ Les opérations sur les vecteurs sont effectuées élément par élément

```
> 5 + 4
[1] 9
> c(5, 3) + c(4, 1)
[1] 9 4
>
> 5*4
[1] 20
> c(5, 3)*c(4, 1)
[1] 20  3
```

Opérations dans R (2)

- ✓ Attention à la longueur des vecteurs lors d'opérations !!!
  - Réplication des vecteurs les plus courts pour correspondre au plus long vecteur
  - L'erreur n'est pas toujours visible
  - Si le plus long est un multiple de la taille du plus court

```
> 1:10 + 2
[1]  3  4  5  6  7  8  9 10 11 12
> # Ce qui correspond à
> 1:10 + rep(2, 10)
[1]  3  4  5  6  7  8  9 10 11 12
```

  - Si le plus long n'est pas un multiple de la taille du plus court

```
> 1:10 + c(2, 5, 6)
 [1]  3  7  9  6 10 12  9 13 15 12
Message d'avis :
In 1:10 + c(2, 5, 6) :
  la taille d'un objet plus long n'est pas multiple de la
taille d'un objet plus court
```

Opérateurs dans R (1)

Opérateur	Fonction
`$`	extraction d'une liste
`^`	puissance
`-`	changement de signe
`:`	génération de suites
`%*% %% %/%`	produit matriciel, modulo, division entière
`* /`	multiplication, division
`+ -`	addition, soustraction
`< <= == >= > !=`	inférieur, inférieur ou égal, égal, supérieur ou égal, supérieur, différent de
`!`	négation logique
`& &&`	« et » logique
`| ||`	« ou » logique
`<-`	assignation

Opérateurs mathématique, logiques, d'assignation, d'extraction

Opérateurs dans R (2)

- Evaluation de droite à gauche
  - ✓ Opérateurs puissance et assignation à gauche
- Tous les autres sont évalués de gauche à droite

```
> (a <- c(2, 3))
[1] 2 3
> 3^2^4 # correspond à 3^16 et non à 9^4
[1] 43046721
> 2 - 2 - 2 # correspond à (2 - 2) - 2 et non à 2 - (2 - 2)
[1] -2
```

Fonctions

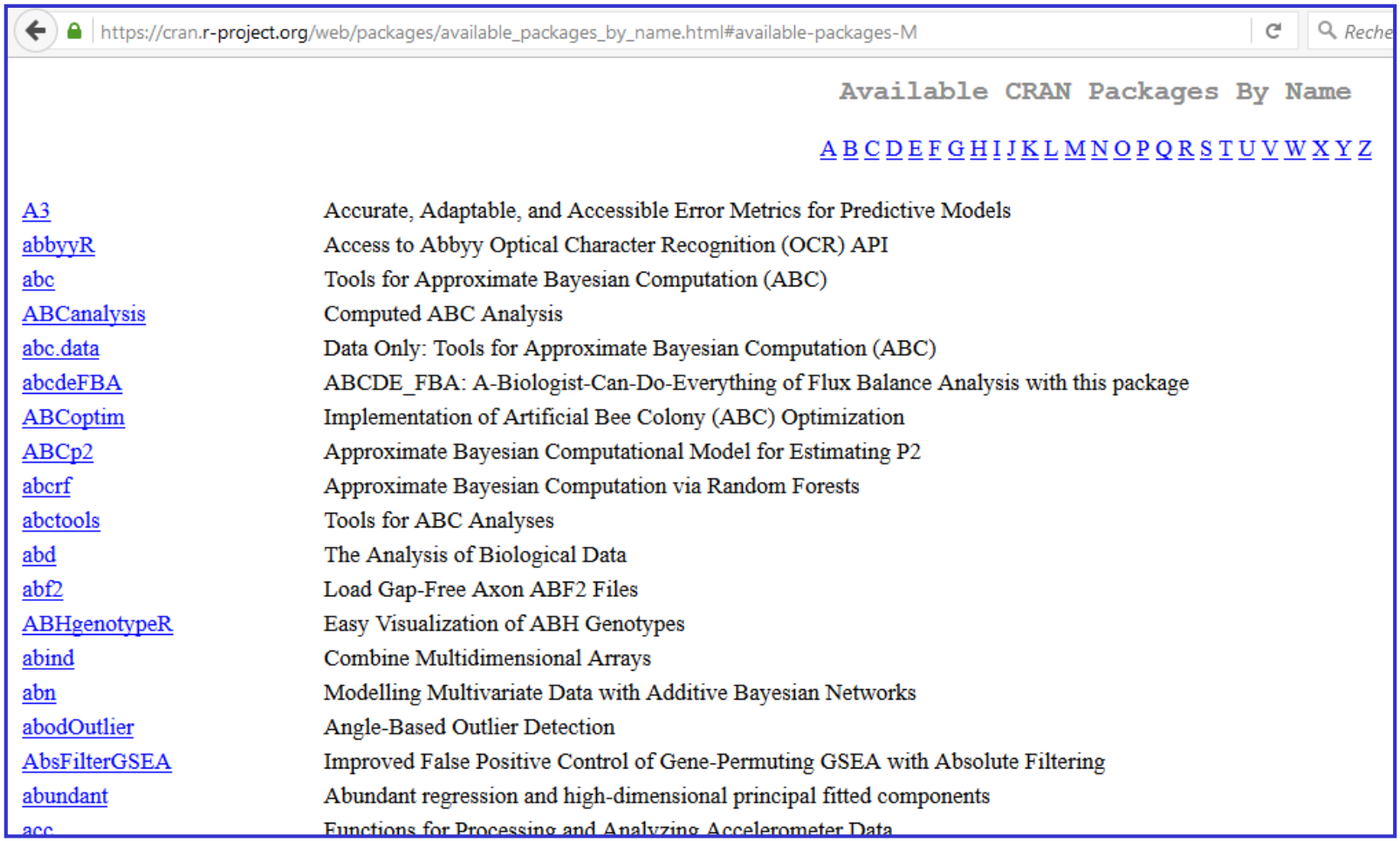
- Internes de R, ou personnelles
- Structure
  - ✓ `nom.fonct(nom.arg1=val1, nom.arg2=val2,…)`
- Arguments
  - ✓ Nombre non limité
  - ✓ Certains ont une valeur par défaut
  - ✓ Ordre important si l'on n'utilise pas la forme `nom.arg1=valeur` (ce qui est recommandé)

Packages et Fonctions

- Les fonctions R peuvent être contenues dans des packages R
  - ✓ publics, déposés sur le site CRAN, en téléchargement libre
  - ✓ personnels
- Package
  - ✓ Sous windows, OS, Unix
  - ✓ Toujours documenté (PDF, HTML aide sous R)
- Souvent centrés sur un domaine particulier
  - ✓ `survival` : analyses de survie
  - ✓ `mice` : pour faire de l'imputation multiple
  - ✓ ...

Packages – CRAN

- https://cran.r-project.org/web/packages/available_packages_by_name.html


https://cran.r-project.org/web/packages/available_packages_by_name.html#available-packages-M

Available CRAN Packages By Name

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

A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
abbyyR	Access to Abbyy Optical Character Recognition (OCR) API
abc	Tools for Approximate Bayesian Computation (ABC)
ABCanalysis	Computed ABC Analysis
abc.data	Data Only: Tools for Approximate Bayesian Computation (ABC)
abcdeFBA	ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package
ABCoptim	Implementation of Artificial Bee Colony (ABC) Optimization
ABCp2	Approximate Bayesian Computational Model for Estimating P2
abcrf	Approximate Bayesian Computation via Random Forests
abctools	Tools for ABC Analyses
abd	The Analysis of Biological Data
abf2	Load Gap-Free Axon ABF2 Files
ABHgenotypeR	Easy Visualization of ABH Genotypes
abind	Combine Multidimensional Arrays
abn	Modelling Multivariate Data with Additive Bayesian Networks
abodOutlier	Angle-Based Outlier Detection
AbsFilterGSEA	Improved False Positive Control of Gene-Permuting GSEA with Absolute Filtering
abundant	Abundant regression and high-dimensional principal fitted components
acc	Functions for Processing and Analyzing Accelerometer Data

Packages – sur son Poste de Travail

- Packages et utilisation de leurs fonctions
  - ✓ Un package doit être installé sur son poste de travail pour pouvoir l'utiliser
  - ✓ Un package doit être chargé dans sa session pour pouvoir utiliser ses fonctions
- Détail des packages installés

```
> installed.packages()
```

- Liste des packages chargés

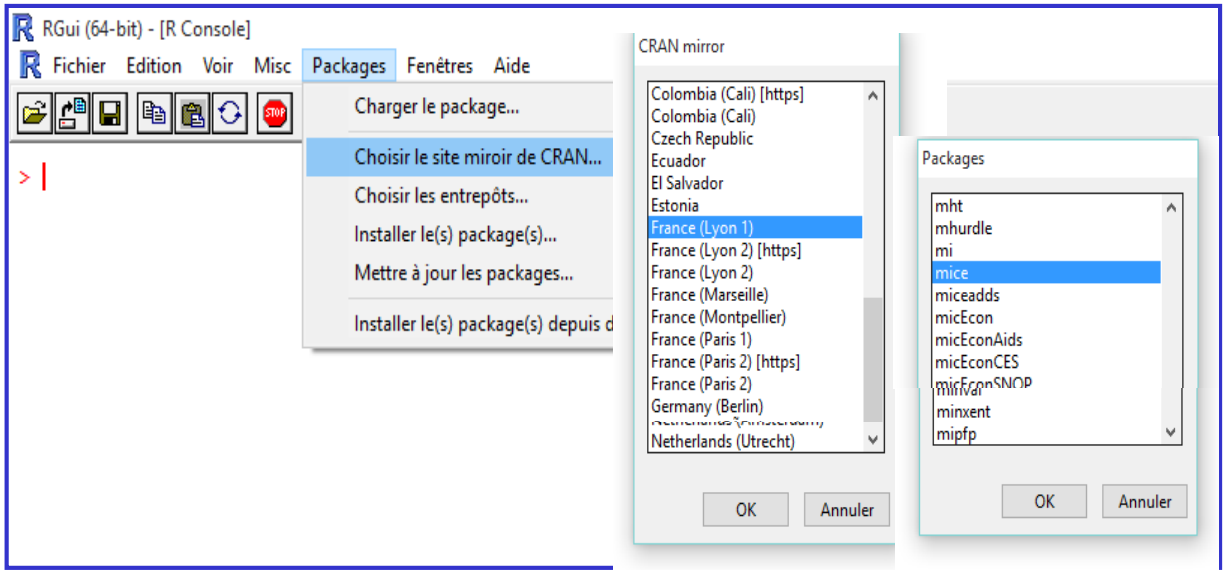
```
> search()
```

- Aide sur un package chargé

```
> help(package=survival)
```

Packages – Installation

- Installation d'un package
  - ✓ En utilisant le menu

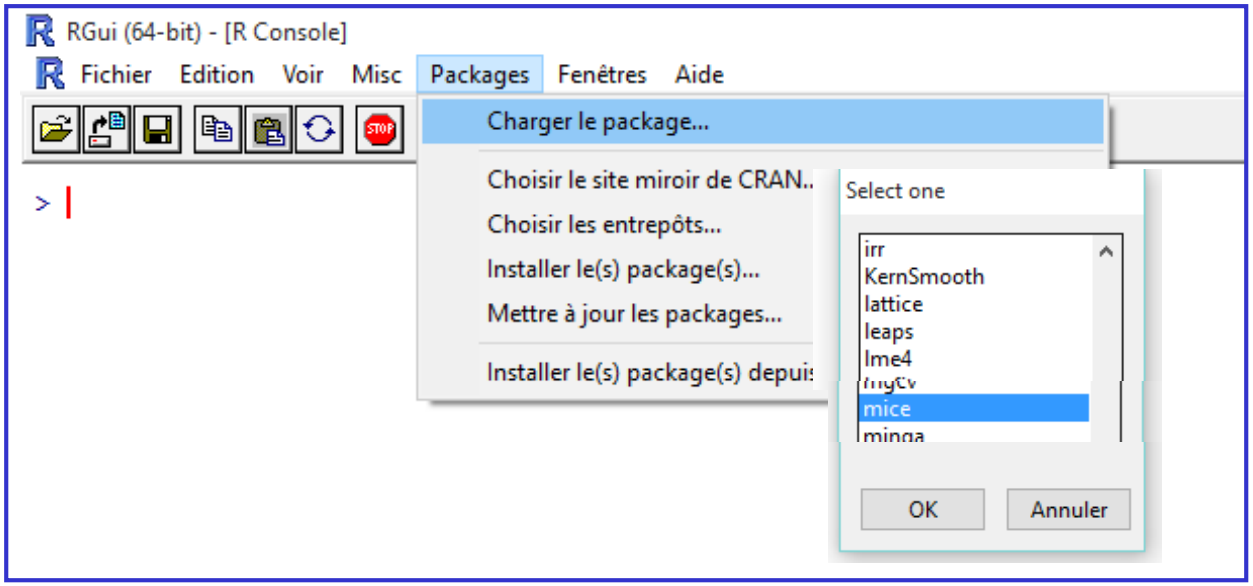


  - ✓ A partir de la console R

```
> install.packages(c("mice"),…)
```

Packages – Chargement

- Chargement d'un package
  - ✓ En utilisant le menu



  - ✓ A partir de la console R

```
> library(mice)
```

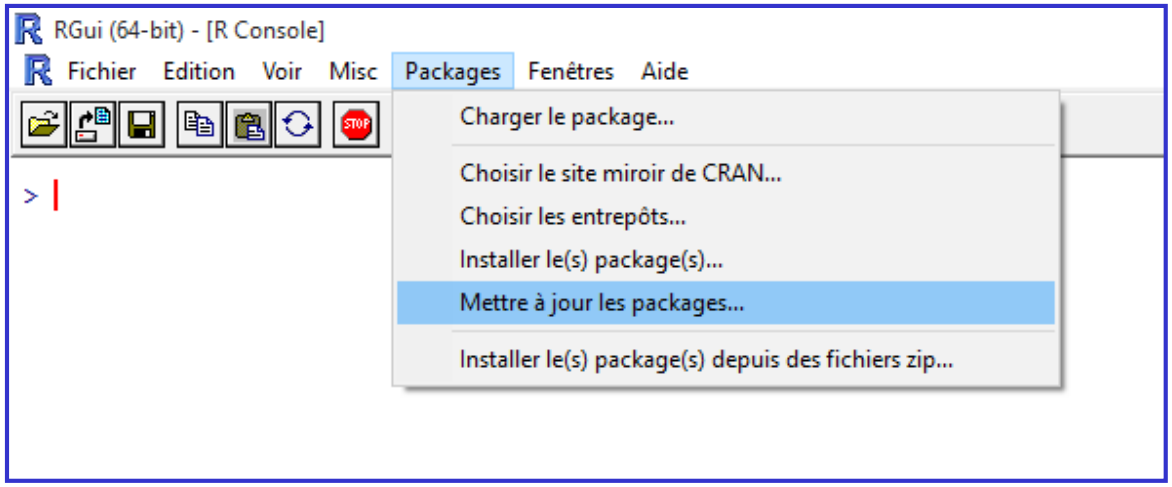
Packages – Utilisation

- Utilisation des fonctions d'un package chargé

```
> ?mice
> data(nhanes)
> head(nhanes)
  age  bmi hyp chl
1   1   NA  NA  NA
2   2 22.7   1 187
3   1   NA   1 187
4   3   NA  NA  NA
5   1 20.4   1 113
6   3   NA  NA 184
> imp <- mice(nhanes)
…
```

Packages – Mise à Jour

- Mise à jour d'un package installé
  - ✓ En utilisant le menu



  - ✓ A partir de la console R

```
> update.packages(checkBuilt=TRUE, ask=FALSE)
```

Espace de Travail (1)

- Important de connaître dans quel espace on effectue un travail
  - ✓ Lister les objets contenus

```
> ls()
[1] "amat" "bmat" "cmat" "v1"
```

  - ✓ Nettoyer l'espace

```
> rm(c("bmat", "v1"))
> ls()
[1] "amat" "cmat"
> rm(list=ls()) # Nettoie tout
> ls()
[1] character(0)
```

Espace de Travail (2)

- Parfois utile de conserver un espace dans lequel on a effectué un travail
  - ✓ Enregistrer son espace suite à un travail

```
> save.image("c:/stat/calcul.RData")
```

    - Seuls les objets sont sauvegardés
    - Pas le code ayant été utilisé pour créer ces objets
  - ✓ Charger un espace pour reprendre un travail

```
> load.image("c:/stat/calcul.RData")
```

    - Seuls les objets sont récupérés
    - Pas le code servant à reprendre le travail

Programmation et Organisation (1)

- Etre organisé, rigoureux, attentif, méthodique,…
- Réfléchir à ce que l'on veut faire, comment s'y prendre pour le faire, avant de se lancer dans l'écriture du script R
- Ecrire un code lisible, aéré, compréhensible pour vous (dans le temps) et pour les autres (collaborateurs)

Programmation et Organisation (2)

- ✓ Insérer des espaces là où cela est nécessaire

```
> (amat <- matrix(1:8, nrow=2, ncol=4))
> # plutôt que
> (amat<-matrix(1:8,nrow=2,ncol=4))
```

- ✓ Ecrire des commentaires, commenter son script, écrire des commentaires, commenter son script, commenter,…

```
> # Il est toujours plus facile d'écrire ce que l'on fait
> # plutôt que d'essayer de retrouver ce que l'on a fait
> # ou voulu faire
> # que ce soit pour nous ou pour une autre personne…
```

Programmation et Organisation (3)

- Organiser ses dossiers (exemple)
  - C:
    - INF-STAR
      - Data
      - Program
      - Result

Programmation et Organisation (3)

- Systématiser son script en conséquences (exemple)

```
# Paramétrages généraux
#---------------------------
# Dossier principal
MyPath <- "c:/INF-STAR/"

# Mes fichiers de données et mes résultats
PathIn <- paste(MyPath, "Data/", sep="")
PathOut <- paste(MyPath, "Result/", sep="")

# Eventuelle library courante
# library(survival)

# Nom de mon fichier de données
FicName <- "nhanes.csv"
#-------------------------------------------------------------------------------------

# Importation de mon fichier de données
MyData <- read.csv2(paste(PathIn, FicName, sep=""))
...
# Sauvegarde du graphique des corrélations
jpeg(paste(PathOut, "correl1.jpeg", sep=""))
...
```

Source

- Introduction à la programmation en R. Vincent Goulet

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