LAB 2: INTRODUCTION AU LANGAGE R (2/2)

2 ème année Génie Informatique

Élément: Data mining

Pr : Nabila ZRIRA
2016-2017

Plan

- Opérateurs en R
- Quelques fonctions utiles
- Structures de contrôle
- Concepts avancés
- Générateurs de nombres aléatoires
- Importation et exportation des données
- TP

Opérateurs en R

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

Opérateurs en R

- Les opérations sur les vecteurs sont effectuées *élément par élément*:

```
> c(1, 2, 3) + c(4, 5, 6)
[1] 5 7 9
> 1:3 * 4:6
[1]  4 10 18
```

- Si les vecteurs impliqués dans une expression arithmétique ne sont pas de la même longueur, les plus courts sont *recyclés* de façon à correspondre au plus long vecteur.

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

Opérateurs en R

- Si la longueur du plus long vecteur est un multiple de celle du ou des autres vecteurs, ces derniers sont recyclés un nombre entier de fois :

```
> 1:10 + 1:5 + c(2, 4)  # vecteurs recyclés 2 et 5 fois
 [1]  4  8  8 12 12 11 11 15 15 19
```

- Sinon, le plus court vecteur est recyclé un nombre fractionnaire de fois, mais comme ce résultat est rarement souhaité et provient généralement d'une erreur de programmation, un avertissement est affiché :

```
> 1:10 + c(2, 4, 6)
 [1]  3  6  9  6  9 12  9 12 15 12
Message d'avis :
In 1:10 + c(2, 4, 6) :
la taille d'un objet plus long n'est pas un multiple de la
taille d'un objet plus court
```

Quelques fonctions utiles: manipulation de vecteurs

- seq(): génération de suites de nombres

```
> seq(1, 9, by = 2)
[1] 1 3 5 7 9
```

- seq_len(): version plus rapide de seq pour générer la suite des nombres de 1 à la valeur de l'argument

```
> seq_len(10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

- rep(): répétition de valeurs ou de vecteurs

```
> rep(2, 10)
 [1] 2 2 2 2 2 2 2 2 2 2
```

Quelques fonctions utiles: manipulation de vecteurs

- sort(): tri en ordre croissant ou décroissant

```
> sort(c(4, -1, 2, 6))
[1] -1  2  4  6
```

- rank(): rang des éléments d'un vecteur dans l'ordre croissant ou décroissant

```
> rank(c(4, -1, 2, 6))
[1] 3 1 2 4
```

- rev(): renverser un vecteur

```
> rev(1:10)
 [1] 10  9  8  7  6  5  4  3  2  1
```

Quelques fonctions utiles: manipulation de vecteurs

- head(): extraction des n premiers éléments d'un vecteur ($n > 0$) ou suppression des n derniers ($n < 0$)

```
> head(1:10, 3); head(1:10, -3)
[1] 1 2 3
[1] 1 2 3 4 5 6 7
```

- tail(): extraction des n derniers éléments d'un vecteur ($n > 0$) ou suppression des n premiers ($n < 0$)

```
> tail(1:10, 3); tail(1:10, -3)
[1]  8  9 10
[1]  4  5  6  7  8  9 10
```

Quelques fonctions utiles: manipulation de vecteurs

- unique(): extraction des éléments différents d'un vecteur

```
> unique(c(2, 4, 2, 5, 9, 5, 0))
[1] 2 4 5 9 0
```

- which(): positions des valeurs TRUE dans un vecteur booléen

```
> x

[1]  4 -1  2 -3  6
```

```
> which(x < 0)
[1] 2 4
```

Quelques fonctions utiles: manipulation de vecteurs

- which.min(): position du minimum dans un vecteur

```
> x

[1]  4 -1  2 -3  6
```

```
> which.min(x)
[1] 4
```

- which.max(): position du maximum dans un vecteur

```
> which.max(x)
[1] 5
```

- match(): position de la première occurrence d'un élément dans un vecteur

```
> match(2, x)
[1] 3
```

Quelques fonctions utiles: statistiques descriptives

- sum() et prod(): somme et produit des éléments d'un vecteur

```
> x

 [1] 14 17  7  9  3  4 25 21 24 11
```

```
> sum(x); prod(x)
[1] 135
[1] 24938020800
```

- diff(): différences entre les éléments d'un vecteur (opérateur mathématique ∇)

```
> diff(x)
[1]   3 -10   2  -6   1  21  -4   3 -13
```

Quelques fonctions utiles: statistiques descriptives

- mean(): moyenne arithmétique

```
> mean(x)
[1] 13.5
```

- var() et sd(): variance et écart type (versions sans biais)

```
> var(x)
[1] 64.5
```

- min() et max(): minimum et maximum d'un vecteur

```
> min(x); max(x)
[1] 3
[1] 25
```

Quelques fonctions utiles: statistiques descriptives

- range(): vecteur contenant le minimum et le maximum d'un vecteur

```
> range(x)
[1]  3 25
```

- median(): médiane empirique

```
> median(x)
[1] 12.5
```

- quantile(): quantiles empiriques

```
> quantile(x)
  0%  25%  50%  75% 100%
 3.0  7.5 12.5 20.0 25.0
```

Quelques fonctions utiles: statistiques descriptives

- summary(): statistiques descriptives d'un échantillon

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     7.5    12.5    13.5    20.0    25.0
```

Quelques fonctions utiles: sommaires cumulatifs

```
> x

[1] 14 17  7  9  3
```

- cumsum() et cumprod(): somme et produit cumulatif d'un vecteur

```
> cumsum(x); cumprod(x)
[1] 14 31 38 47 50
[1]    14   238  1666 14994 44982
```

- cummin() et cummax():

```
> cummin(x); cummax(x)
[1] 14 14  7  7  3
[1] 14 17 17 17 17
```

Quelques fonctions utiles: manipulation des matrices

```
> x

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

- nrow() et ncol(): nombre de lignes et de colonnes d'une matrice

```
> nrow(x); ncol(x)
[1] 2
[1] 2
```

Quelques fonctions utiles: manipulation des matrices

- rowSums() et colSums(): sommes par ligne et par colonne, respectivement, des éléments d'une matrice

```
> rowSums(x)
[1] 6 4
```

- rowMeans() et colMeans(): moyennes par ligne et par colonne, respectivement, des éléments d'une matrice

```
> colMeans(x)
[1] 1.5 3.5
```

Quelques fonctions utiles: manipulation des matrices

- t(): transposée

```
> t(x)
     [,1] [,2]
[1,]    2    1
[2,]    4    3
```

- det(): déterminant

```
> det(x)
[1] 2
```

Quelques fonctions utiles: manipulation des matrices

- solve():

  1) avec un seul argument (une matrice carrée) : inverse d'une matrice;

  2) avec deux arguments (une matrice carrée et un vecteur) : solution du système d'équations linéaires ***Ax = b***

```
> solve(x)
     [,1] [,2]
[1,]  1.5   -2
[2,] -0.5    1
> solve(x, c(1, 2))
[1] -2.5  1.5
```

Quelques fonctions utiles: manipulation des matrices

- diag():
  1) *avec une matrice en argument : diagonale de la matrice ;*
  2) *avec un vecteur en argument : matrice diagonale formée avec le vecteur ;*
  3) *avec un scalaire p en argument : matrice identité $p \times p$*

```
> diag(x)
[1] 2 3
```

Structures de contrôle: exécution conditionnelle

- if (*condition*) *branche.vrai* else *branche.faux*

  if (x>0) y=x*log(x) else y=0

- ifelse(*condition*, *expression.vrai*, *expression.faux*)

  y=ifelse(x>0,x*log(x),0)

- switch(*test*, *cas.1 = action.1*, *cas.2 = action.2*, ...)

Structures de contrôle: boucles

- for (*variable* in *suite*) *expression*
- while (*condition*) *expression*
- repeat *expression*
- *break*
- *next*

```
> for ( i in 1:10)
+ {
+ print(i)
+ }
```

```
while (i< 11)
+ {
+ print(i)
+ i=i+1
+}
```

```
> repeat
+ {
+   print(i)
+ if (i <10 )
+   {i=i+1} else
+   { break}
+ }
```

Concepts avancés: fonction apply

La fonction apply sert à appliquer une fonction quelconque sur une partie d'une matrice ou, plus généralement, d'un tableau. La syntaxe de la fonction est la suivante :

apply(X, MARGIN, FUN, ...)

avec

- X est une matrice ou un tableau ;
- MARGIN est un vecteur d'entiers contenant la ou les dimensions de la matrice ou du tableau sur lesquelles la fonction doit s'appliquer ;
- FUN est la fonction à appliquer ;
- '...' est un ensemble d'arguments supplémentaires, séparés par des virgules, à passer à la fonction FUN.

Concepts avancés: fonction apply

```
> (x <- matrix(sample(1:100, 20, rep = TRUE), 5, 4))

     [,1] [,2] [,3] [,4]
[1,]   27   90   21   50
[2,]   38   95   18   72
[3,]   58   67   69  100
[4,]   91   63   39   39
[5,]   21    7   77   78

> apply(x, 1, var)                # variance par ligne
[1]  978.000 1181.583  335.000  612.000 1376.917

> apply(x, 2, min)                # minimum par colonne

[1] 21  7 18 39
```

Concepts avancés: fonctions lapply et sapply

- Les fonctions lapply et sapply sont similaires à la fonction apply en ce qu'elles permettent d'appliquer une fonction aux éléments d'une structure— le vecteur ou la liste en l'occurrence. Leur syntaxe est similaire :

  lapply(X, FUN, ...)

  sapply(X, FUN, ...)

- La fonction lapply applique une fonction FUN à tous les éléments d'un vecteur ou d'une liste X et retourne le résultat sous forme de liste.
- La fonction sapply est similaire à lapply, sauf que le résultat est retourné sous forme de vecteur, si possible. Le résultat est donc *simplifié* par rapport à celui de lapply, d'où le nom de la fonction.

Concepts avancés: fonctions lapply et sapply

```
> (x <- lapply(5:8, sample, x = 1:10))

[[1]]
[1]  7  4  3 10  9

[[2]]
[1] 3 2 4 7 8 5

[[3]]
[1]  8  4  9  2  1 10  6

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

```
> lapply(x, mean)

[[1]]
[1] 6.6

[[2]]
[1] 4.833333

[[3]]
[1] 5.714286

[[4]]
[1] 4.5
```

```
> sapply(x, mean)

[1] 6.600000 4.833333 5.714286 4.500000
```

Générateurs de nombres aléatoires: uniformes

On obtient des nombres uniformes sur un intervalle quelconque avec la fonction runif() dans R.

```
> runif(3, 1.2, 5.8)
[1] 5.777474 4.995633 2.492278
```

Générateurs de nombres aléatoires: non uniformes

Loi de probabilité	Racine dans R	Noms des paramètres
Bêta	`beta`	`shape1`, `shape2`
Binomiale	`binom`	`size`, `prob`
Binomiale négative	`nbinom`	`size`, `prob` ou `mu`
Cauchy	`cauchy`	`location`, `scale`
Exponentielle	`exp`	`rate`
F (Fisher)	`f`	`df1`, `df2`
Gamma	`gamma`	`shape`, `rate` ou `scale`
Géométrique	`geom`	`prob`
Hypergéométrique	`hyper`	`m`, `n`, `k`
Khi carré	`chisq`	`df`
Logistique	`logis`	`location`, `scale`
Log-normale	`lnorm`	`meanlog`, `sdlog`
Normale	`norm`	`mean`, `sd`
Poisson	`pois`	`lambda`
t (Student)	`t`	`df`
Uniforme	`unif`	`min`, `max`
Weibull	`weibull`	`shape`, `scale`
Wilcoxon	`wilcox`	`m`, `n`

Générateurs de nombres aléatoires: distribution discrète quelconque

- La fonction sample() permet de simuler des nombres d'une distribution discrète quelconque. Sa syntaxe est

sample(x, size, replace = FALSE, prob = NULL),

où

x est un vecteur des valeurs possibles de l'échantillon à simuler (le support de la distribution), size est la quantité de nombres à simuler et prob est un vecteur de probabilités associées à chaque valeur de x (1/length(x) par défaut). Enfin, si replace est TRUE, l'échantillonnage se fait avec remise.

Importation et exportation des données

Lire des données dans un système statistique pour les analyser et ensuite exporter les résultats dans un autre système pour pouvoir les commenter peut être une tâche plus longue et plus pénible que l'analyse statistique elle-même.

Importation et exportation des données: importation

1. La fonction read.table()

- Elle permet de lire des fichiers de données au format texte (ASCII). Cette fonction est destinée à lire les tableaux de données, les données issues d'un tableur qui auraient été préalablement transformées en fichier texte.
- La syntaxe de cette fonction avec la valeur par défaut de ses arguments est :

```
read.table(file, header = FALSE, sep = "", quote = "\"", dec = ".",row.names, col.names, as.is = FALSE, na.strings = "NA",colClasses = NA, nrows = -1,skip = 0, check.names = TRUE, fill = !blank.lines.skip,strip.white = FALSE, blank.lines.skip = TRUE,comment.char = "#")
```

Importation et exportation des données: importation

2. Il existe plusieurs variantes de cette fonction pour lesquelles la valeur par défaut des arguments change.

   read.csv(), read.csv2(), read.delim(),read.delim2() et read.xlsx()

3. La fonction read.fwf()

- Pour cette fonction, les attributs sont similaires à ceux de read.table() avec en plus l'attribut widths qui spécifie la largeur des champs.
- La syntaxe de cette fonction avec la valeur par défaut de ses arguments est :

```
read.fwf(file, widths, sep = "\t", as.is = FALSE,skip = 0, row.names, col.names, n = -1, ...)
```

Importation et exportation des données: importation

4. La fonction scan()

- Elle est plus souple que read.table() et comporte plus d'arguments. L'argument what permet de décrire le mode des variables qui vont être lues dans le fichier. Ces modes peuvent être numériques, caractères ou complexes et sont respectivement spécifiés par 0, "", 0i.
- La syntaxe de cette fonction avec la valeur par défaut de ses arguments est :

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",quote = if (sep=="\n") "" else
"'\"", dec = ".",skip = 0, nlines = 0, na.strings = "NA",flush = FALSE, fill = FALSE,
strip.white = FALSE, quiet = FALSE,blank.lines.skip = TRUE, multi.line = TRUE,
comment.char = "")
```

Importation et exportation des données: importation

5. Le module foreign

Le module foreign fournit des moyens pour importer des données de fichiers produits par des systèmes statistiques tel que Minitab, S, SAS, SPSS et Stata.

- read.mtp() importe un «Minitab Portable Worksheet»
- read.xport() lie les fichiers au format «SAS Transport»
- read.spss() permet de lire les fichiers créés par les commandes save et export de SPSS
- read.S() peut lire les objets produits par S-Plus

Importation et exportation des données: exportation

- Les fonctions write() et write.table() permettent d'exporter des objets R en fichiers textes. La fonction write() sert pour les vecteurs et les matrices, la fonction write.table(), elle, exporte les «data frames» avec les noms de ligne et de colonne.
- La syntaxe de ces fonctions avec la valeur par défaut de leurs arguments est :

```
write(x, file = "data",ncolumns = if(is.character(x)) 1 else 5,append = FALSE)
```

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",eol = "\n", na = "NA", dec
      = ".", row.names = TRUE,col.names = TRUE, qmethod = c("escape", "double"))
```

Suite séance prochaine!