Statistiques avec R

Christophe Ambroise et Cyril Dalmasso

École doctorale *« du génome aux organismes »*
Université d'Évry

27-29 janvier 2014

`http://stat.genopole.cnrs.fr/~jchiquet/fr/initiation_R`

Intervenants

Equipe « Statistique & Génome »

`http://stat.genopole.cnrs.fr`

Cyril Dalmasso

Statistique

Christophe Ambroise

Statistique

`prenom.nom@genopole.cnrs.fr`

Agenda (théorique) de la semaine

Lundi Introduction à la programmation en R, rappel de statistiques (tests)

Mardi Analyse exploratoire : analyse en composante principale, clustering et modèle linéaire

Mercredi Modèle linéaire généralisé (régression logistique)

. . . et travaux dirigés !

Agenda (théorique) de la semaine

Lundi	Introduction à la programmation en R, rappel de statistiques (tests)
Mardi	Analyse exploratoire : analyse en composante principale, clustering et modèle linéaire
Mercredi	Modèle linéaire généralisé (régression logistique)

. . . et travaux dirigés !

Agenda (théorique) de la semaine

Lundi Introduction à la programmation en R, rappel de statistiques (tests)

Mardi Analyse exploratoire : analyse en composante principale, clustering et modèle linéaire

Mercredi Modèle linéaire généralisé (régression logistique)

. . . et travaux dirigés !

Première partie I

Structures de données

Plan
Structures de données

Plan
Structures de données

Plan
Structures de données

Vecteurs : définition

Propriétés

- objet le plus élémentaire sous R,
- collection d'entités de même nature,
- mode (ou type) défini par la nature des entités qui le composent.

Les modes possibles

1. numérique (`numeric`),
2. caractère (`caracter`),
3. logique (`boolean`).

Vecteurs : définition

Propriétés

- objet le plus élémentaire sous `R`,
- collection d'entités de même nature,
- mode (ou type) défini par la nature des entités qui le composent.

Les modes possibles

1. numérique (`numeric`),
2. caractère (`caracter`),
3. logique (`boolean`).

Vecteurs : définition

Propriétés

- objet le plus élémentaire sous `R`,
- collection d'entités de même nature,
- mode (ou type) défini par la nature des entités qui le composent.

Les modes possibles

1. numérique (`numeric`),
2. caractère (`caracter`),
3. logique (`boolean`).

Quelques vecteurs typés

1. Numérique

```
> x0 <- 0
> x1 <- c(-1,23,98.7)
> mode(x0)

[1] "numeric"
```

2. Caractère

```
> y0 <- "bonjour"
> y1 <- c("Pomme","Flore","Alexandre")
> mode(y1)

[1] "character"
```

3. Logique

```
> z0 <- TRUE
> z1 <- c(FALSE,TRUE,FALSE,TRUE,TRUE)
> z2 <- c(T,F,F)
> mode(z2)

[1] "logical"
```

Remarques sur l'affectation

Définition (affectation)

C'est l'opération qui consiste à attribuer une valeur à une variable.

En R, plusieurs choix sont possibles :

- l'opérateur usuel est '<-' (signe inférieur suivi du signe moins)

```
> jo <- "l'indien"
> jo
[1] "l'indien"
```

- l'opérateur '=' peut être utilisé la plupart du temps

```
> nb.max.d.annees.pour.faire.une.these = 3
> nb.max.d.annees.pour.faire.une.these
[1] 3
```

- la commande `assign` permet cette opération (d'où l'anglicisme *assignation*)

```
> assign("x", c(8,9,-pi,sqrt(2)))
> x
[1]  8.000000  9.000000 -3.141593  1.414214
```

Valeurs spéciales

Variables réservées par R

- `NA` est le code R pour les valeurs manquantes (absentes des données),
- `NaN` est le code de R pour signifier un résultat numérique aberrant ,
- `Inf` et `-Inf` sont les valeurs réservées pour plus et moins ∞,
- `NULL` est l'objet nul.

```
> c(4,2,NA,5)
[1]  4  2 NA  5
> 0/0
[1] NaN
> 1/0
[1] Inf
> names(1)
NULL
```

Opérations arithmétiques
s'effectuent terme-à-terme

Soient `x,y` tels que

```
> x<-c(1,2,-3,-4)
> y<-c(-5,-6,9,0)
```

'+' addition des éléments de deux vecteurs

```
> x+y
[1] -4 -4  6 -4
```

'-' soustraction des éléments de deux vecteurs

```
> x-y
[1]   6   8 -12  -4
```

'*' multiplication des éléments de deux vecteurs

```
> x*y
[1]  -5 -12 -27   0
```

'/' division des éléments de deux vecteurs

```
> x/y
[1] -0.2000000 -0.3333333 -0.3333333       -Inf
```

Le « recyclage » des éléments du vecteur

Lors d'une opération entre vecteurs, les vecteurs trop courts sont ajustés pour atteindre la taille du plus grand vecteur en recyclant les données.

Exemple

```
> x <- c(10,100,1000,10000)
> y <- c(1,2)
> 2*x + y - 1
```

```
[1]    20   201  2000 20001
```

$\rightsquigarrow$ souvent pratique mais attention aux effets de bords !

Opérateurs mathématiques I

Fonctions numériques élémentaires

`floor,ceiling,round`.

```
> floor(2/3)
[1] 0
> ceiling(2/3)
[1] 1
> round(2/3,3)
[1] 0.667
```

Fonctions arithmétiques élémentaires

`^,%%,%/%,abs,log,exp,log10,sqrt,cos,tan,sin...` s'appliquent toutes terme-à-terme.

```
> log10(c(10,100,1000))
```

Opérateurs mathématiques II

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

```
> cos(c(pi/2,pi))^2 + sin(c(pi/2,pi))^2
```

```
[1] 1 1
```

Fonctions caractérisant un vecteur

`prod, sum, max, min, range, which.min, which.max, length`

```
> x <- c(-8,1.5,3)
> prod(x)
```

```
[1] -36
```

```
> sum(x)
```

```
[1] -3.5
```

```
> length(x)
```

```
[1] 3
```

```
> max(x)
```

Opérateurs mathématiques III

```
[1] 3

> min(x)

[1] -8

> range(x)

[1] -8  3

> which.max(x)

[1] 3

> which.min(x)

[1] 1
```

Pour le minimum / maximum terme-à-terme : `pmin,pmax`.

Fonctions appliquées le long du vecteur

`cumsum, cumprod, cummin, cummax`

Opérateurs mathématiques IV

```
> x <- c(-2,1,-3,2)
> cumprod(x)

[1] -2 -2  6 12

> cumsum(x)

[1] -2 -1 -4 -2

> cummax(x)

[1] -2  1  1  2

> cummin(x)

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

Opérateurs ensemblistes

Fonctionnent pour tous les modes

`unique,intersect,union,setdiff,setequal,is.element`

```
> unique(c("banane","citron","banane"))
[1] "banane" "citron"
> intersect(c("banane","citron"),c("orange","banane"))
[1] "banane"
> union(c("banane","citron"),c("orange","banane"))
[1] "banane" "citron" "orange"
> setequal(c("banane","citron"),c("orange","banane"))
[1] FALSE
> is.element(1,sample(c(1,2,3),2))
[1] TRUE
```

Plan
Structures de données

L'opérateur ' : '

`from:to`

Génère une séquence par pas de un depuis le nombre `from` jusqu'à `to` (si possible).

```
> -5:5
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
> 5:-5
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5
> pi:6
[1] 3.141593 4.141593 5.141593
> 1:6/2
[1] 0.5 1.0 1.5 2.0 2.5 3.0
> 1:(6/2)
[1] 1 2 3
```

La commande seq

Plusieurs schémas possibles

- `seq(from,to)`
- `seq(from,to,by=)`
- `seq(from,to,length.out=)`

```
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(2,10,by=2)
[1]  2  4  6  8 10
> seq(2,10,length.out=6)
[1]  2.0  3.6  5.2  6.8  8.4 10.0
```

La commande `rep`

Fonctionne pour tous les modes

- `rep(x,times)`, où `times` peut être un vecteur,
- `rep(x,each)`.

```
> rep(1,3)
[1] 1 1 1
> rep("Mercy",3)
[1] "Mercy" "Mercy" "Mercy"
> rep(c("A","B","C"),c(3,2,4))
[1] "A" "A" "A" "B" "B" "C" "C" "C" "C"
> rep(c(TRUE,FALSE), each=2)
[1]  TRUE  TRUE FALSE FALSE
```

Génération de vecteurs logiques

Obtenus par conditions avec

- les opérateurs logiques `'<'`, `'<='`, `'>'`, `'>='`, `'=='` `'!='`
- le ET, le OU, NON, OU exclusif : `'&'` (intersection), `'|'` (union), `'!'` (négation), `xor`.

```
> note1 <- c(8,9,14,3,17.5,11)
> note2 <- c("C","B","A","B","E","B")
> admis <- (note1 >= 10) & (note2 == "A" | note2 == "B")
> mention <- (note1 >= 15) & (note2 == "A")
> admis
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE
> sum(admis)
[1] 2
> sum(mention)
[1] 0
```

Par concaténation

Avec 'c()'

L'opérateur 'c()' peut s'appliquer à n'importe quoi pourvu que l'on concatène des vecteurs de même type.

```
> c( c(1,2), c(3,4))

[1] 1 2 3 4

> round(c(seq(-pi,pi,len=4),rep(c(1:3),each=2),0),2)

 [1] -3.14 -1.05  1.05  3.14  1.00  1.00  2.00  2.00  3.00  3.00  0.00
```

Remarque

Dans le second exemple, les entiers composants `c(1:3)` ont été forcés au typage flottant.

Par concaténation II

Avec `paste`

Concaténation de chaînes de caractères. Convertit en caractères les éléments passés en argument avant toute opération.

```
> paste("R","c'est","bien")
[1] "R c'est bien"
> paste(2:4,"ieme")
[1] "2 ieme" "3 ieme" "4 ieme"
> paste("A",1:5, sep="")
[1] "A1" "A2" "A3" "A4" "A5"
> paste("A",1:5, sep="",collapse="")
[1] "A1A2A3A4A5"
```

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;
4. un vecteur de chaînes de caractères, qui spécifie les noms des éléments de `x` à conserver.

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;
4. un vecteur de chaînes de caractères, qui spécifie les noms des éléments de `x` à conserver.

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;
4. un vecteur de chaînes de caractères, qui spécifie les noms des éléments de `x` à conserver.

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;
4. un vecteur de chaînes de caractères, qui spécifie les noms des éléments de `x` à conserver.

Principe

- Permet la sélection d'un sous-ensemble du vecteur `x`.
- Le sous-ensemble est spécifié entre crochets `x[subset]`.

L'objet `subset` peut prendre 4 types différents :

1. un vecteur logique, qui doit être de la même taille que le vecteur `x` ;
2. un vecteur numérique aux composantes positives, qui spécifie les valeurs à inclure ;
3. un vecteur numérique aux composantes négatives, qui spécifie les valeurs à exclure ;
4. un vecteur de chaînes de caractères, qui spécifie les noms des éléments de `x` à conserver.

Indexation des vecteurs : exemples I

Vecteurs logiques

```
> x <- c(3,6,-2,9,NA,sin(-pi/6))
> x[x > 0]

[1]  3  6  9 NA

> x[!is.na(x)]

[1]  3.0  6.0 -2.0  9.0 -0.5

> x[!is.na(x) & x>0]

[1] 3 6 9

> mean(x,na.rm=TRUE)

[1] 3.1

> x[x <= mean(x,na.rm=TRUE)]

[1]  3.0 -2.0   NA -0.5
```

Indexation des vecteurs : exemples II

Vecteurs aux composantes positives ou négatives

```
> x
[1]  3.0  6.0 -2.0  9.0   NA -0.5
> x[2]
[1] 6
> x[1:5]
[1]  3  6 -2  9 NA
> x[-c(1,5)]
[1]  6.0 -2.0  9.0 -0.5
> x[-(1:5)]
[1] -0.5
```

Indexation des vecteurs : exemples III

Vecteurs de chaînes de caractères

```
> names(x) <- c("var1","var2","var3","var4","var5","var6")
> x

var1 var2 var3 var4 var5 var6
 3.0  6.0 -2.0  9.0   NA -0.5

> x[c("var1","var3")]

var1 var3
   3   -2
```

Autres commandes d'indexation et de sélection

1. Classer
   - `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
   - `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,

2. Extraire
   - `which` renvoie les indices de `x` vérifiant une condition ;

3. Échantillonner
   - `sample` échantillonne aléatoirement dans un vecteur `x`, avec ou sans remise.

Autres commandes d'indexation et de sélection

1. Classer
   - `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
   - `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,

2. Extraire
   - `which` renvoie les indices de `x` vérifiant une condition ;

3. Échantillonner
   - `sample` échantillonne aléatoirement dans un vecteur `x`, avec ou sans remise.

Autres commandes d'indexation et de sélection

1. Classer
   - `sort` renvoie le vecteur classé par ordre croissant ou décroissant,
   - `order` renvoie les indices d'ordre des éléments par ordre croissant ou décroissant,

2. Extraire
   - `which` renvoie les indices de `x` vérifiant une condition ;

3. Échantillonner
   - `sample` échantillonne aléatoirement dans un vecteur `x`, avec ou sans remise.

Exemples

```
> x <- -5:5
> y <- sample(x)
> sort(y)

 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5

> order(y)

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

> y[order(y)]

 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5

> y[order(y,decreasing=TRUE)]

 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5

> which(sample(x,4) > 0)

[1] 2
```

Plan
Structures de données

Définition

Définition

Un facteur est un vecteur de variables catégorielles. Les niveaux du facteur peuvent être ordonnés ou pas.

Utilisation

les facteurs s'utilisent pour catégoriser les données d'un vecteur (ce qui s'avère très utile pour la gestion des variables qualitatives).

$\rightsquigarrow$ un facteur est souvent associé à d'autres vecteurs pour en définir une partition.

Définition

Définition

Un facteur est un vecteur de variables catégorielles. Les niveaux du facteur peuvent être ordonnés ou pas.

Utilisation

les facteurs s'utilisent pour catégoriser les données d'un vecteur (ce qui s'avère très utile pour la gestion des variables qualitatives).

$\rightsquigarrow$ un facteur est souvent associé à d'autres vecteurs pour en définir une partition.

Définition

Définition

Un facteur est un vecteur de variables catégorielles. Les niveaux du facteur peuvent être ordonnés ou pas.

Utilisation

les facteurs s'utilisent pour catégoriser les données d'un vecteur (ce qui s'avère très utile pour la gestion des variables qualitatives).

$\rightsquigarrow$ un facteur est souvent associé à d'autres vecteurs pour en définir une partition.

Création : la fonction `factor`

```
> factor(sample(1:3,10,replace=TRUE))

 [1] 2 2 3 2 1 2 2 3 3 3
Levels: 1 2 3

> factor(sample(1:3,10,replace=TRUE),levels=1:5)

 [1] 1 1 3 1 1 3 2 3 2 2
Levels: 1 2 3 4 5
```

Gestion : `nlevels,levels,table`

```
> x <- factor(sample(c("thésard","CR","MdC"),15,replace=TRUE))
> cat(nlevels(x),"niveaux:",levels(x))

3 niveaux: CR MdC thésard

> table(x)

x
     CR     MdC thésard
      5       5       5
```

Un exemple de facteur associé à un vecteur

Données

Chacun me donne son âge et son grade [1]

```
> age <- c(25,35,32,27,32,40,26,25,26,28,30,NA,36,30,30)
> grd <- c("thd","CR","MdC","thd","thd","MdC","MdC","thd","thd","MdC","CR"
```

Question : nombre d'individus par catégorie ?

```
> table(grd)

grd
 CR MdC thd
  3   5   7
```

1. sauf un qui refuse :'(

Utilisation

Applique une fonction sur un vecteur partitionné en groupes.

Question : âge moyen / écart-type par catégorie ?

```
> tapply(age,grd,mean,na.rm=TRUE)

      CR       MdC       thd
33.66667 31.50000 27.85714

> tapply(age,grd,sd,na.rm=TRUE)

      CR       MdC       thd
3.214550 6.191392 2.794553
```

Plan
Structures de données

Plan
Structures de données

Tableau : définition

Définition (objet `array`)

Un tableau est un vecteur muni d'un attribut dimension (`dim`), lui même défini par un vecteur. Il est défini par la commande

`array(data,dim,dimnames=)`

```
> array(1:8,c(2,2,2))

, , 1

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

, , 2

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

Matrice : définition

Définition (objet `matrix`)

Une matrice est un tableau à deux dimensions. Elle est définie par la commande

`matrix(data,nrow=,ncol=,byrow)`

En conséquence

- Un objet `array` à deux dimensions est automatiquement converti en `matrix`
- Un vecteur auquel on ajoute un atribut dimension est automatiquement converti en `matrix`

```
> class(array(1:4,c(2,2)))

[1] "matrix"

> x <- c(1,2,3,4)
> dim(x) <- c(2,2)
> class(x)

[1] "matrix"
```

Remarques importantes

1. R range les éléments d'une matrice par défaut par colonne.

```
> matrix(1:6,nrow=2)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6,nrow=2,byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

2. Lors de la création d'une matrice, R recycle les éléments jusqu'à ce que les contraintes de dimension soient vérifiées.

```
> matrix(1:2,nrow=2,ncol=2)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
```

Matrices : opérateurs élémentaires

Étant donné qu'une matrice est un vecteur pourvu d'une dimension, on a la proposition suivante :

Proposition

La plupart des opérateurs vectorielles s'appliquent (arithmétiques/mathématiques, ensemblistes, d'indexation).

```
> a <- matrix(sample(-4:4,9),3,3)
> cat(max(a),sum(a),prod(a))

4 0 0

> which(a > 0)

[1] 1 3 6 7

> cumsum(a[a > 0])

[1]  4  6  9 10

> order(a)

[1] 2 8 5 4 9 7 3 6 1

> round(exp(a),4)

        [,1]    [,2]   [,3]
[1,] 54.5982  0.3679 2.7183
[2,]  0.0183  0.1353 0.0498
[3,]  7.3891 20.0855 1.0000
```

Manipulation de matrices

Opérateurs matriciels usuels

- `+,/,*,^` sont les opérateurs usuels terme-à-terme,
- `%*%` est le produit matriciel,
- `crossprod()` est le produit scalaire,
- `t()` transpose une matrice,
- `diag()` extrait / spécifie la diagonale.

```
> a <- matrix(sample(-4:4,9),3,3)
> b <- matrix(sample(a),3,3)
> diag(a)

[1]  1  4 -4

> diag(a) <- diag(b) <- 1
> diag(a)

[1] 1 1 1

> a + t(b) %*% b

     [,1] [,2] [,3]
[1,]   19   -7   -9
[2,]  -11   27   -4
[3,]  -12   -8   15
```

Concaténation de matrices

Trois fonctions selon l'effet voulu :

1. `c()` concatène les éléments de plusieurs matrices en un vecteur,
2. `cbind()` empile horizontalement plusieurs matrices,
3. `rbind()` empile verticalement plusieurs matrices.

```
> a <- matrix(1,2,3)
> b <- matrix(2,2,3)
> c(a,b)

 [1] 1 1 1 1 1 1 2 2 2 2 2 2

> cbind(a,b)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    1    2    2    2
[2,]    1    1    1    2    2    2

> rbind(a,b)

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

Résolution de systèmes linéaires, inversion matricielle

La commande `solve` résout

$$\mathbf{A}x = \mathbf{b},$$

```
> A <- matrix(c(4,2,8,-3),2,2)
> b <- c(2,3)
> solve(A,b)
```

```
[1]  1.0714286 -0.2857143
```

ou inverse une matrice :

```
> round(solve(A) %*% A,8)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Commandes avancées d'algèbre linéaire

Utile pour l'analyse numérique

`R` dispose des outils classiques d'algèbre linéaire

- `det` : calcule le déterminant d'une matrice ;
- `chol` : factorisation de Cholesky ($A = C^\intercal C$, avec A symétrique, C triangulaire supérieure) ;
- `qr` : factorisation QR ($A = QR$ avec Q orthogonale, R triangulaire supérieure) ;
- `eigen` : calcule valeurs propres et vecteurs propres d'une matrice ;
- `svd` : calcule la décomposition en valeurs singulières.
- `...`

Plan
Structures de données

Liste : définition

Définition (objet `list`)

Une liste est une collection d'objets hétérogènes. Elle est définie par la commande `list(el1=, el2=, ...)`. Les éléments d'une liste peuvent posséder un nom.

```
> list(c(1,2,3),c("robert","johnson"),matrix(rnorm(4),2,2))

[[1]]
[1] 1 2 3

[[2]]
[1] "robert"  "johnson"

[[3]]
           [,1]       [,2]
[1,] -2.2759227 -0.3960857
[2,] -0.6925659 -0.4085253
```

```
> list(numero = c(1,2,3), noms = c("robert","johnson"), mat = matrix(rnorm(4),2,2))

$numero
[1] 1 2 3

$noms
[1] "robert"  "johnson"

$mat
          [,1]       [,2]
[1,] 0.4636346 -0.4597605
[2,] 1.2458616 -1.0437894
```

Accéder aux éléments

Deux situations

1. Les éléments de la liste ne sont pas nommés : on accède au i^{e} élément par indexation `nom_liste[[i]]` uniquement.
2. Les éléments de la liste sont nommés : on peut y accéder comme ci-dessus ou en utilisant le nom de l'élément `nom_liste$nom_elt`.

```
> maliste <- list(numero = c(1,2,3), noms = c("robert","johnson"), mat = matrix(rnor
> maliste$nom
[1] "robert"  "johnson"
> maliste$nom[2]
[1] "johnson"
> maliste[[2]]
[1] "robert"  "johnson"
> maliste[[2]][2]
[1] "johnson"
```

Manipulation de listes I

Sélectionner des éléments

Fonctionne (presque) comme pour les vecteurs

```
> l1 <- list(1:2,c("a","c","g","t"))
> l1[[-2]]

[1] 1 2
```

Commande `lapply`

Applique une fonction à chaque élément d'une liste

```
> lapply(maliste,length)

$numero
[1] 3

$noms
[1] 2

$mat
[1] 4
```

Manipulation de listes II

Commande `c()`

Permet de concaténer deux listes.

```
> c(list(1:2,c("a","c","g","t")),list(rnorm(3),"yop"))

[[1]]
[1] 1 2

[[2]]
[1] "a" "c" "g" "t"

[[3]]
[1]  1.066570  2.307537 -0.703831

[[4]]
[1] "yop"
```

Définition (objet `data.frame`)

C'est une liste à laquelle on impose certaines contraintes[2]*, afin de rassembler vecteurs et facteurs sous la forme d'un tableau de données.*

- Pratiquement, *un tableau de données est une matrice dont les colonnes sont de mode différent,*
- C'est l'objet idéal pour la manipulation de données (forcez-vous à l'utiliser).

2. que je vous épargne

Définition (objet `data.frame`)

C'est une liste à laquelle on impose certaines contraintes[2]*, afin de rassembler vecteurs et facteurs sous la forme d'un tableau de données.*

- Pratiquement, *un tableau de données est une matrice dont les colonnes sont de mode différent,*
- C'est l'objet idéal pour la manipulation de données (forcez-vous à l'utiliser).

2. que je vous épargne

Création de tableau de données

Syntaxe

On peut spécifier le nom des colonnes par le vecteur `row.names` ou directement comme pour une liste :

`data.frame(e1=,e2=,...,row.names=)`

```
> age <- c(25,35,32,27,32,40,26,25,26,28,30,NA,36,30,30)
> grd <- c("thd","CR","MdC","thd","thd","MdC","MdC","thd","thd","MdC","CR","MdC","CR
> sex <- factor(sample(c(rep("M",3),rep("F",12))))
> donnees <- data.frame(age=age,grade=grd,sexe=sex)
> head(donnees)
```

```
  age grade sexe
1  25   thd    M
2  35    CR    F
3  32   MdC    F
4  27   thd    F
5  32   thd    F
6  40   MdC    M
```

Manipulation des éléments du tableau de données

- Comme une liste !
- les commandes `attach()` / `detach` placent / ôtent les éléments du tableaux de données dans l'itinéraire de recherche.

```
> donnees$age

 [1] 25 35 32 27 32 40 26 25 26 28 30 NA 36 30 30

> attach(donnees,warn.conflicts=FALSE)
> grade

 [1] thd CR  MdC thd thd MdC MdC thd thd MdC CR  MdC CR  thd thd
Levels: CR MdC thd

> detach(donnees)
```

Travailler avec les tableaux de données

- beaucoup de fonctions prédéfinies
- penser aux fonctions `tapply` (ou `by`)

```
> summary(donnees)
      age        grade   sexe
 Min.   :25.00   CR :3   F:12
 1st Qu.:26.25   MdC:5   M: 3
 Median :30.00   thd:7
 Mean   :30.14
 3rd Qu.:32.00
 Max.   :40.00
 NA's   :1
> attach(donnees,warn.conflicts=FALSE)
> by(age,sexe,mean,na.rm=TRUE)
sexe: F
[1] 29.72727
------------------------------------------------------------
sexe: M
[1] 31.66667
> by(age,grade,mean,na.rm=TRUE)
grade: CR
[1] 33.66667
------------------------------------------------------------
grade: MdC
[1] 31.5
------------------------------------------------------------
grade: thd
[1] 27.85714
> detach(donnees)
```

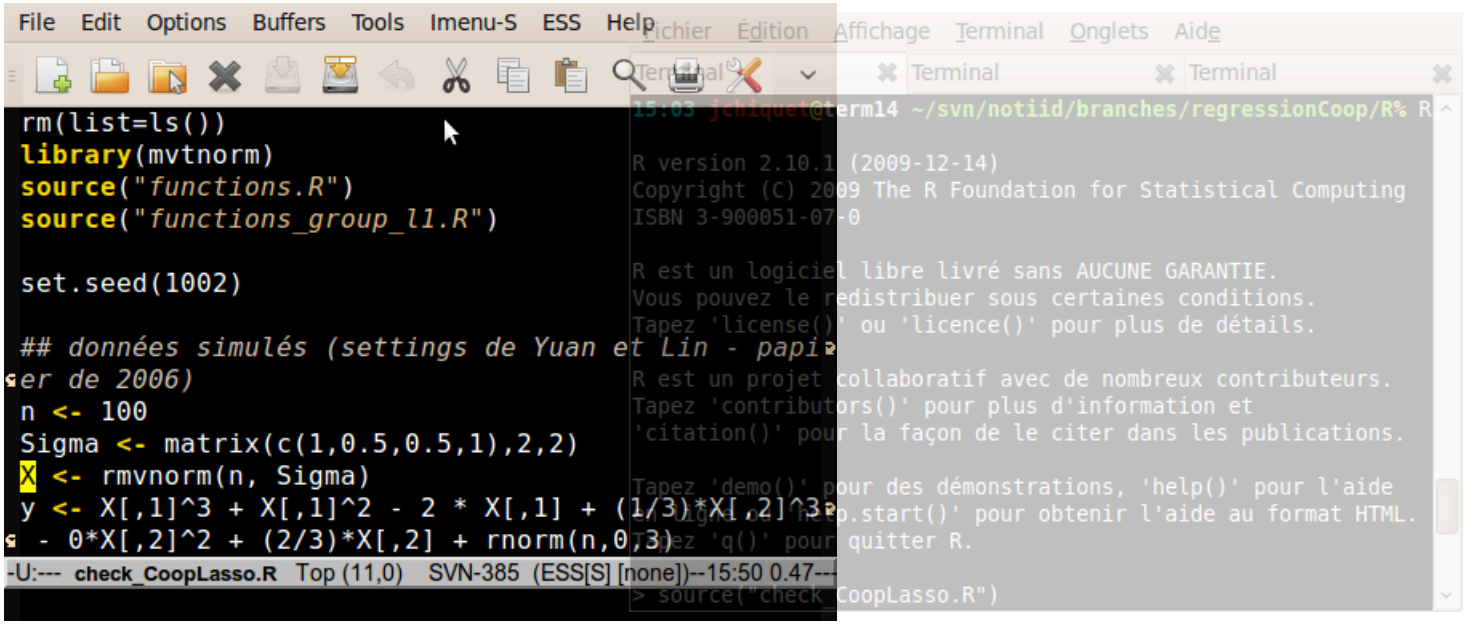
Idéal pour analyse statistique type anova
Pour plus tard...

```
> anova(lm(age ~ sexe*grade,data=donnees))

Analysis of Variance Table

Response: age
           Df  Sum Sq Mean Sq F value  Pr(>F)
sexe        1   8.866   8.866  0.9296 0.36015
grade       2 100.149  50.075  5.2505 0.03082 *
sexe:grade  1  68.866  68.866  7.2209 0.02491 *
Residuals   9  85.833   9.537
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Deuxième partie II

Développer avec R

Programmer en pratique : rappels



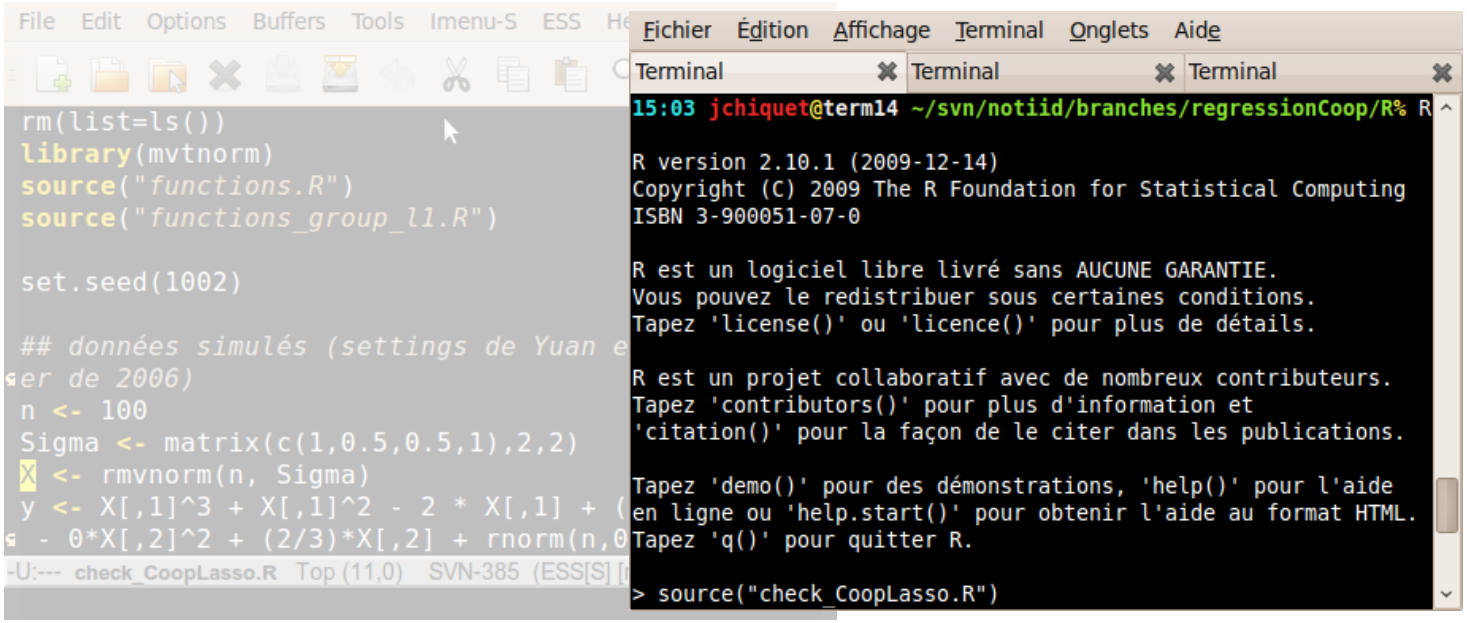
1. un éditeur de texte (vos fonctions / scripts)
2. un terminal avec R (tester, « sourcer »)

« Sourcer »

`source("un_script.R")` : exécute la série de commandes de `mon_script.R`

`source("mes_fonctions.R")` : charge le contenu (les fonctions) de `mes_fonctions.R`

Programmer en pratique : rappels



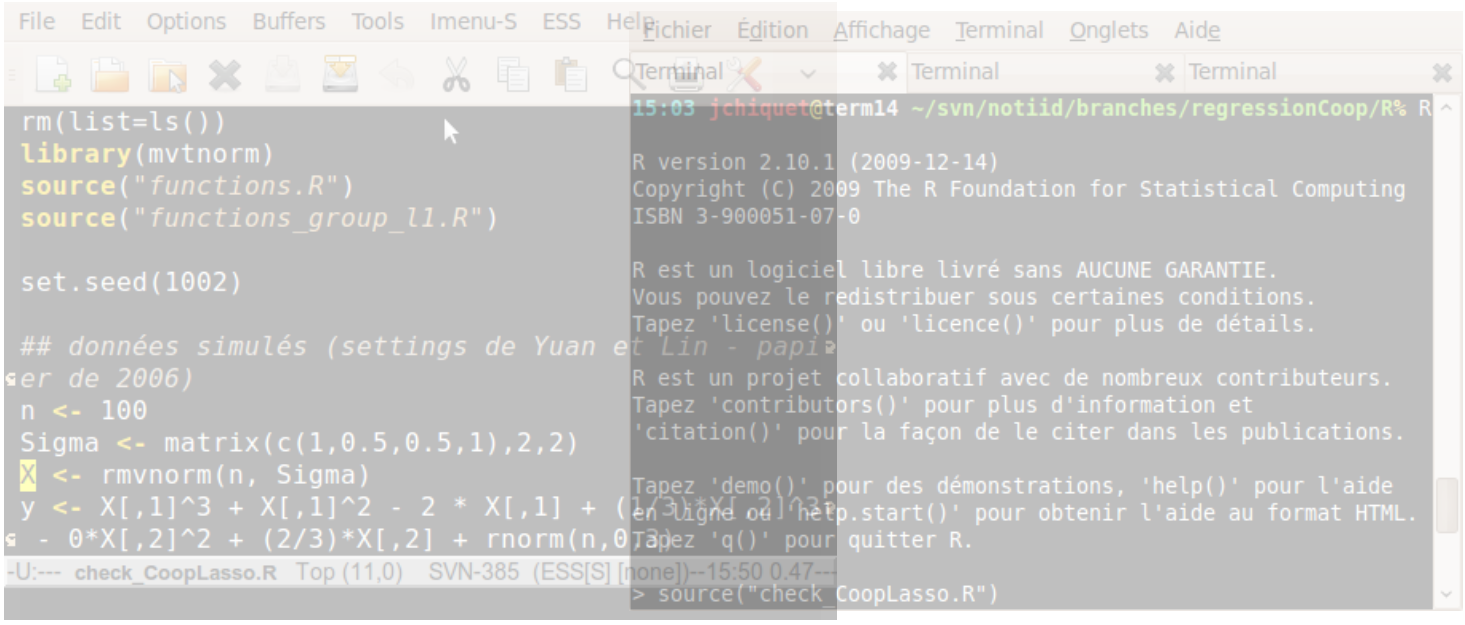
1. un éditeur de texte (vos fonctions / scripts)
2. un terminal avec R (tester, « sourcer »)

« Sourcer »

`source("un_script.R")` : exécute la série de commandes de `mon_script.R`

`source("mes_fonctions.R")` : charge le contenu (les fonctions) de `mes_fonctions.R`

Programmer en pratique : rappels



1. un éditeur de texte (vos fonctions / scripts)
2. un terminal avec `R` (tester, « sourcer »)

« Sourcer »

`source("un_script.R")` : exécute la série de commandes de `mon_script.R`

`source("mes_fonctions.R")` : charge le contenu (les fonctions) de `mes_fonctions.R`

Regrouper les expressions

Syntaxe

```
{expr_1; expr_2; ...; expr_n }
```

ou

```
{
  expr_1
   ...
  expr_n
}
```

Remarques sur les groupes

- La dernière valeur du groupe est retournée ;
- un groupe d'expressions peut être passé à une fonction, réutilisé dans une expression plus grande, etc.

Exécution conditionnelle : `if`, `if/else`, `ifelse`

Syntaxe

```
if (condition) {
  expr_1
} else {
  expr_2
}
```

ou

```
ifelse(condition, a, b)
```

Remarques

- `condition` est une valeur logique : penser à `&`, `|`, `!`, ...
- le `else` est optionnel,
- `elseif` permet d'imbriquer les conditionnements.

Exécution répétée : boucle `for`

Syntaxe

```
for (var in set) {
  expr(var)
}
```

ou

```
for (var in set)
  expr(var)
```

à fuir pour éviter les effets de bords sournois !

Remarques sur la boucle `for`

- `var` est la variable incrémentée,
- `set` est un vecteur définissant les valeurs successives,
- *lente* comparée aux opérateurs matriciels.

Exécution répétée : boucles `while` et `repeat`

Syntaxe

```
while (condition) {
  expr
}
```

ou

```
repeat {
  expr
}
```

Remarque

- Comme pour `for`, éviter les imbrications sources de lenteur.

Contrôle des boucles : `break, next`

Exemples d'utilisation

```
repeat {
  expr
  if (condition) {break}
}
```

ou

```
while (condition1){
  expr_1
  if (condition2) {next}
  expr_2
}
```

Remarque

- `break` est la seule manière d'interrompre une boucle `repeat`.

Définir une fonction

Syntaxe

```
nom_de_la_fonction <- function(arg1,arg2, ...) {
  expression

  return(var)
}
```

Remarques

- `return` peut être omis (à éviter) : dans ce cas la dernière valeur calculée est renvoyée.
- peut être tapée directement dans l'interpréteur ou dans un fichier externe `functions.R`, chargé par `source`.

Un exemple simple

Moyenne empirique d'un vecteur

Avec suppression des valeurs manquantes !

```
> moyenne <- function(x) {
+   ## suppression des valeurs manquantes
+   x.not.na <- x[!is.na(x)]
+   ## moyenne empirique
+   resultat <- sum(x.not.na) / length(x.not.na)
+
+   return(resultat)
+ }
```

Tests

```
> moyenne(rnorm(100))
```

```
[1] -0.008308431
```

```
> moyenne(c(1,-5,3,NA,8.7))
```

```
[1] 1.925
```

Les arguments, leurs valeurs par défaut
Encore un point fort de R

Propriétés

- *les arguments peuvent être passés dans le désordre s'ils sont nommés :* `var=object,`
- *on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction :* `var=10`.
- *en cas de sorties multiples, les sorties doivent être renvoyées sous forme de liste.*

Remarques

- *Les valeurs par défaut rendent la lecture des fonctions beaucoup plus aisée pour l'utilisateur : imposer peu d'arguments obligatoires.*
- *Les noms des éléments de la liste définie dans la fonction sont conservés à l'extérieur de la fonction.*

Les arguments, leurs valeurs par défaut
Encore un point fort de R

Propriétés

- *les arguments peuvent être passés dans le désordre s'ils sont nommés :* `var=object`,
- *on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction :* `var=10`.
- *en cas de sorties multiples, les sorties doivent être renvoyées sous forme de liste.*

Remarques

- *Les valeurs par défaut rendent la lecture des fonctions beaucoup plus aisée pour l'utilisateur : imposer peu d'arguments obligatoires.*
- *Les noms des éléments de la liste définie dans la fonction sont conservés à l'extérieur de la fonction.*

Les arguments, leurs valeurs par défaut
Encore un point fort de R

Propriétés

- *les arguments peuvent être passés dans le désordre s'ils sont nommés :* `var=object`,
- *on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction :* `var=10`.
- *en cas de sorties multiples, les sorties doivent être renvoyées sous forme de liste.*

Remarques

- *Les valeurs par défaut rendent la lecture des fonctions beaucoup plus aisée pour l'utilisateur : imposer peu d'arguments obligatoires.*
- *Les noms des éléments de la liste définie dans la fonction sont conservés à l'extérieur de la fonction.*

Les arguments, leurs valeurs par défaut
Encore un point fort de R

Propriétés

- *les arguments peuvent être passés dans le désordre s'ils sont nommés : `var=object`,*
- *on peut définir une valeur par défaut pour n'importe quel argument lors de la définition de la fonction : `var=10`.*
- *en cas de sorties multiples, les sorties doivent être renvoyées sous forme de liste.*

Remarques

- *Les valeurs par défaut rendent la lecture des fonctions beaucoup plus aisée pour l'utilisateur : imposer peu d'arguments obligatoires.*
- *Les noms des éléments de la liste définie dans la fonction sont conservés à l'extérieur de la fonction.*

Un exemple (un tout petit peu) plus avancé

Résumé numérique d'un vecteur

```
> resume <- function(x,na.rm=TRUE,affiche=FALSE) {
+   mu <- mean(x,na.rm=na.rm)
+   s2 <- var(x,na.rm=na.rm)
+   if (affiche) {
+     cat("\nMoyenne:",mu,"et variance:",s2)
+   }
+   return(list(moyenne = mu, variance = s2))
+ }
```

```
> out <- resume(rnorm(100))
> out$variance
```

```
[1] 0.9764857
```

```
> out <- resume(affiche=TRUE,x=rexp(100,0.5))
```

```
Moyenne: 2.02495 et variance: 3.29611
```

Motivations : reproductibilité de la recherche

Principe de Claerbout (Géophysicien, Stanford)

An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

Démarche

1. Proposer une méthode et exposer dans un article ses propriétés,
2. Écrire et déposer un package sur CRAN,
3. Publier un article dans « journal of statistical software » ou une note dans « Bioinformatics ».

Motivations : reproductibilité de la recherche

Principe de Claerbout (Géophysicien, Stanford)

An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

Démarche

1. Proposer une méthode et exposer dans un article ses propriétés,
2. Écrire et déposer un package sur CRAN,
3. Publier un article dans « journal of statistical software » ou une note dans « Bioinformatics ».

Simplicité de la création d'un package

Définir un objectif

Par exemple, `SIMoNe` : construire un graphe des interactions significatives entre gènes à partir de données du transcriptome.

Organisation type

1. Fichier DESCRIPTION
2. Répertoire R : fonctions R (fonction `inferGraph(data)`)
3. Répertoire man : documentation des fonctions
4. Répertoire data : données
5. (Répertoire src : pour les fichiers à compiler, header etc.)

Motivations

Produire un document scientifique, c'est

- expérimenter

  (pipette + éprouvette + blouse) ou ordi $\rightsquigarrow$ données

- analyser les résultats

  méthode + logiciel + données $\rightsquigarrow$ graphes

- rédiger des observations

  idée + (bloc-note ou traitement de texte) $\Rightarrow$ texte

- mettre en forme

  texte + graphes + traitement de texte $\Rightarrow$ article ☺

Motivations

Produire un document scientifique, c'est

- expérimenter

  (pipette + éprouvette + blouse) ou ordi $\rightsquigarrow$ données

- analyser les résultats

  méthode + logiciel + données $\rightsquigarrow$ graphes

- rédiger des observations

  idée + (bloc-note ou traitement de texte) $\Rightarrow$ texte

- mettre en forme

  texte + graphes + traitement de texte $\Rightarrow$ article ☺

Motivations

Produire un document scientifique, c'est

- expérimenter

  (pipette + éprouvette + blouse) ou ordi $\rightsquigarrow$ données

- analyser les résultats

  méthode + logiciel + données $\rightsquigarrow$ graphes

- rédiger des observations

  idée + (bloc-note ou traitement de texte) $\Rightarrow$ texte

- mettre en forme

  texte + graphes + traitement de texte $\Rightarrow$ article

Motivations

Produire un document scientifique, c'est

- expérimenter

  (pipette + éprouvette + blouse) ou ordi $\rightsquigarrow$ données

- analyser les résultats

  méthode + logiciel + données $\rightsquigarrow$ graphes

- rédiger des observations

  idée + (bloc-note ou traitement de texte) $\Rightarrow$ texte

- mettre en forme

  texte + graphes + traitement de texte $\Rightarrow$ article

Motivations

Produire un document scientifique, c'est

- expérimenter

  (pipette + éprouvette + blouse) ou ordi $\rightsquigarrow$ données

- analyser les résultats

  méthode + logiciel + données $\rightsquigarrow$ graphes

- rédiger des observations

  idée + (bloc-note ou traitement de texte) $\Rightarrow$ texte

- mettre en forme

  texte + graphes + traitement de texte $\Rightarrow$ article 

Solutions

1.
   - analyser : MS Excel
   - rédiger : MS Word
   - mettre en forme : MS Word
2.
   - analyser : MatLab
   - rédiger : OpenOffice
   - mettre en forme : OpenOffice
3.
   - analyser : R
   - rédiger : Emacs
   - mettre en forme : LaTeX

mon opinion [a] : 

a. de geek *très* subjective

Le package `sweave` de LaTeX permet de faire appel à du code R dans le document

Solutions

1.
   - analyser : MS Excel
   - rédiger : MS Word
   - mettre en forme : MS Word
2.
   - analyser : MatLab
   - rédiger : OpenOffice
   - mettre en forme : OpenOffice
3.
   - analyser : R
   - rédiger : Emacs
   - mettre en forme : LaTeX

mon opinion [a] : ☹

---

a. de geek *très* subjective

Le package `sweave` de LaTeX permet de faire appel à du code R dans le document

Solutions

1.
   - analyser : MS Excel
   - rédiger : MS Word
   - mettre en forme : MS Word
2.
   - analyser : MatLab
   - rédiger : OpenOffice
   - mettre en forme : OpenOffice
3.
   - analyser : R
   - rédiger : Emacs
   - mettre en forme : LaTeX

mon opinion [a] : ☺

a. de geek *très* subjective

Le package `sweave` de LaTeX permet de faire appel à du code R dans le document

Solutions

1.
   - analyser : MS Excel
   - rédiger : MS Word
   - mettre en forme : MS Word
2.
   - analyser : MatLab
   - rédiger : OpenOffice
   - mettre en forme : OpenOffice
3.
   - analyser : R
   - rédiger : Emacs
   - mettre en forme : LaTeX

mon opinion [a] : ☺

a. de geek *très* subjective

Le package `sweave` de LaTeX permet de faire appel à du code R dans le document

Code latex

```
\documentclass[a4paper]{article}
\begin{document}
In this example we embed parts of the examples
\texttt{kruskal.test} help page into a \LaTeX{} document: ...
\end{document}
```

Sortie pdf

In this example we embed parts of the examples from the `kruskal.test` help page into a LaTeX document : ...

Le module Sweave I

Fichier mélange ".rnw"

```
\documentclass[a4paper]{article}
\begin{document}
In this example we embed parts of the examples from the
\texttt{kruskal.test} help page into a \LaTeX\ document:

<<>>=

data(airquality)
kruskal.test(Ozone  ~ Month, data = airquality)

@

which shows that the location parameter of the Ozone
distribution varies significantly from month to month.
Finally we include a boxplot of the data:

<<fig=TRUE,echo=FALSE>>=

boxplot(Ozone  ~ Month, data = airquality)

@

\end{document}
```

Le module Sweave II

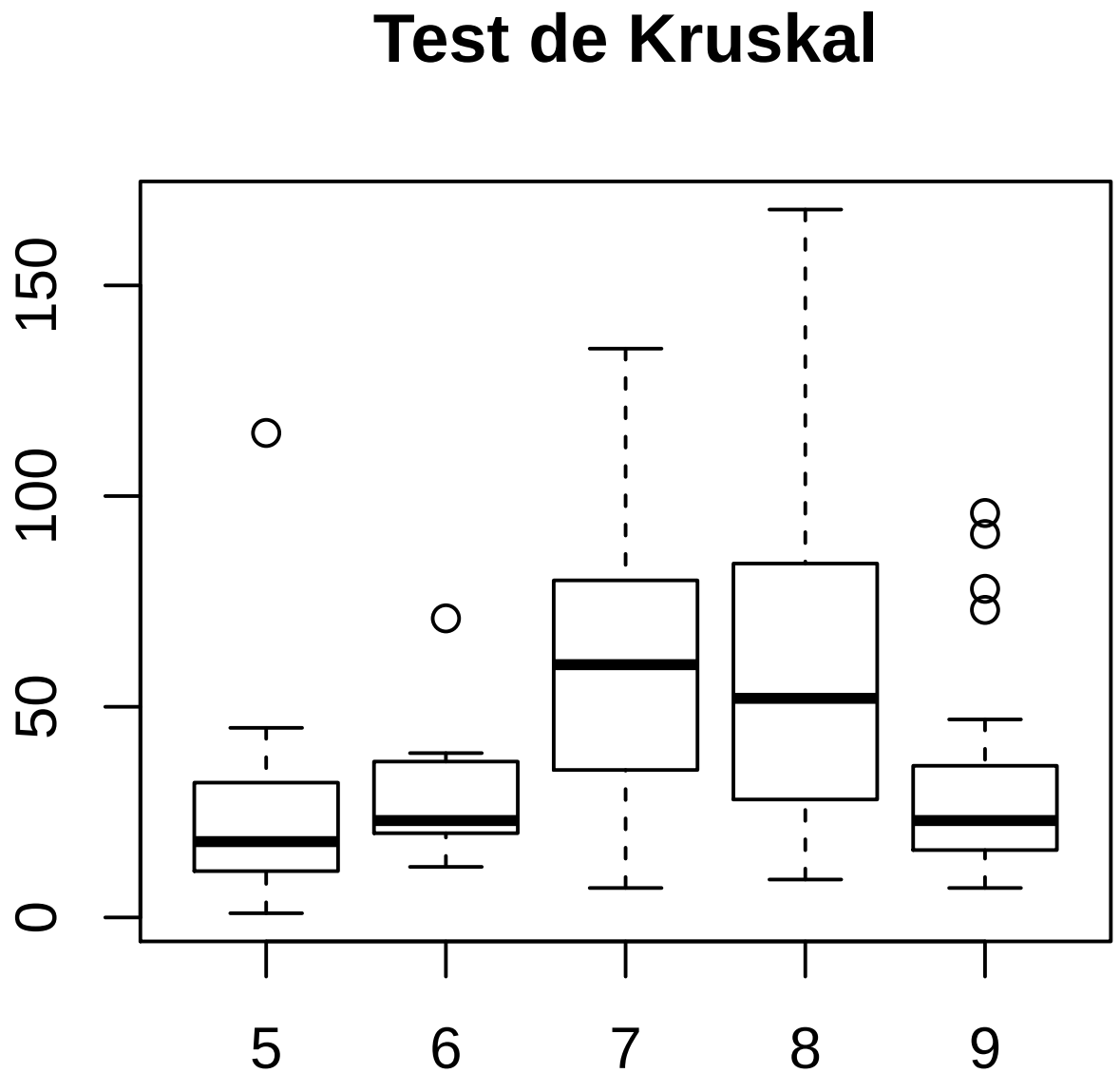
Fichier LaTeX ".tex"

```
\documentclass[a4paper]{article}
\usepackage{Sweave}
\begin{document}
In this example we embed parts of the examples from the
\texttt{kruskal.test} help page into a \LaTeX\ document:
\begin{Sinput}
  R> data(airquality)
  R> kruskal.test(Ozone ~ Month, data = airquality)
\end{Sinput}
\begin{Soutput}
  Kruskal-Wallis rank sum test data: Ozone by Month Kruskal-Wallis
  chi-squared = 29.2666, df = 4, p-value = 6.901e-06
\end{Soutput}
which shows that the location parameter of the Ozone
distribution varies significantly from month to month.
Finally we include a boxplot of the data:
\includegraphics{example-1-002}
\end{document}
```

In this example we embed parts of the examples from the `kruskal.test` help page into a LaTeX document :

```
> data(airquality)
> kruskal.test(Ozone ~ Month, data = airquality)

        Kruskal-Wallis rank sum test

data:  Ozone by Month
Kruskal-Wallis chi-squared = 29.2666, df = 4, p-value = 6.901
```

which shows that the location parameter of the Ozone distribution varies significantly from month to month. Finally we include a boxplot of the data :


Test de Kruskal
0
50
100
150
5
6
7
8
9

Troisième partie III

Entrées / Sorties

Charger des données

Les graphiques sous R

commande `scan`

Une utilisation élémentaire de `scan` permet une saisie plus agréable que la saisie directe des éléments d'un vecteur.

```
> x<-scan()
1: 1
2: 2
3: 3
4: 4
5: 5
6:
Read 5 items
>
> x
[1] 1 2 3 4 5
```

$\rightsquigarrow$ valable pour les jeux de données d'au plus quelques dizaines d'éléments...

Éditer des données

commande `edit`

Permet d'éditer des données existantes à l'aide d'un mini-tableur. Utile pour faire de petites modifications.

```
> new.data <- edit(old.data)
```

FIGURE : Éditeur Mac OS 10.6 / R 2.10

Fichiers binaires

commandes `save` et `load`

La commande `save` permet de sauvegarder un sous ensemble des données de l'espace de travail dans un fichier binaire ; `load` permet de les recharger.

```
> x <- rnorm(125)
> y <- 1 + x + x^2
> save(file="mes_simus",x,y)
> rm(list=ls())
> objects()

character(0)

> load(file="mes_simus")
> objects()

[1] "x" "y"
```

Jeux de données prédéfinies

commande `data`

R dispose d'une collection de données prédéfinies directement utilisables. La commande `data()` permet de les lister puis de les charger.

```
> data(iris3)
> head(iris3)
```

```
[1] 5.1 4.9 4.7 4.6 5.0 5.4
```

- La description d'un jeu de données est accessible dans l'aide.
- L'installation d'un nouveau package rend souvent disponibles de nouveaux jeux de données accessibles par `data`.

D'abord, un bon éditeur . . . il vous permet de constater le formatage d'un fichier texte et comment en « attaquer » l'importation.

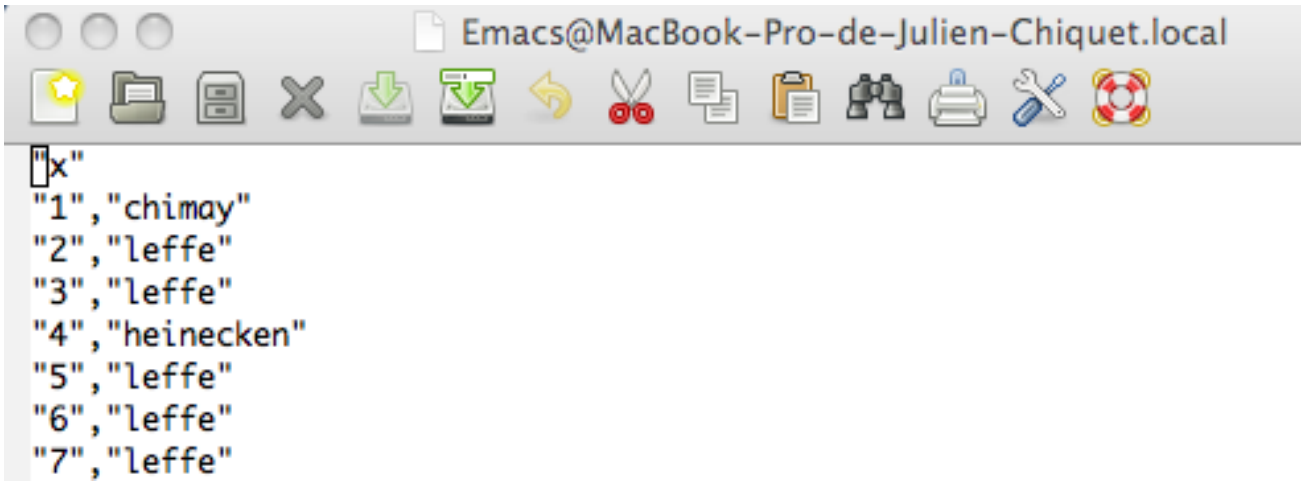


FIGURE : Fichier au formatage "csv"

commande `read.table`

Elle permet de lire un fichier formaté sous forme de table.
`read.table` stocke les données sous forme d'objet `data.frame`.

```
> mes_donnees<-read.table("mesures_baie_raisin_2008-2009.txt",header=TRUE,sep="\t")
> head(mes_donnees)
  Population variete nbre.pepin.baie.2008 poids.pulpe.baie..g..2008
1         CE    1784                   1.0                       0.89
2         CE     124                   1.0                       1.14
3         CE     210                   1.2                       1.26
4         CE    1805                   1.2                       0.66
5         CE    1303                   1.2                       0.83
6         CE     284                   1.3                       0.54
  volume.baie..cm3..2008 nbre.pepin.baie.2009 poids.pulpe.baie..g..2009
1                   7.70                    NA                       NA
2                   8.82                    NA                       NA
3                  10.20                    NA                       NA
4                     NA                    NA                       NA
5                     NA                    NA                       NA
6                   4.61                    NA                       NA
```

Lecture de fichiers III

`read.csv,read.delim`

commandes `read.csv` et `read.delim`

Ce sont des raccourcis pour la fonction `read.table`, spécialisés dans l'importation des données « .csv » (*comma-separated value*) ou tabulées (le séparateur est la tabulation).

commandes `write.table`, `write.csv` et `write.delim`

La fonction `write.table` permet d'imprimer les données issues d'une `data.frame` dans un fichier texte externe. `write.csv` et `write.delim` sont des raccourcis pour les données `csv` ou tabulée.

Pour aller plus loin. . .

Beaucoup de choses sur l'importation des données dans

R Data Import /Export.
`http://cran.r-project.org/doc/manuals/R-data.pdf`

- Exemples avancés avec `read.table`,
- communication avec les bases de données (SQL),
- importation de données Excel,
- . . .

Paramètres récurrents

Forme générique

La plupart des fonctions graphique s'utilisent par un appel du type

1. `nom.function(object, options)`,
2. `nom.function(x, y , options)`.

Parmi les options les plus courantes, on trouve :

- type="p" ; spécifie le type de tracé : "p" pour points, "l" pour lignes, "b" pour points liés par des lignes, "o" pour lignes superposées aux points. . .
- xlim= ; ylim=, spécifie les limites de axes x et y
- xlab= ; ylab=, annotation des axes x et y
- main= ; titre du graphe en cours
- sub= ; sous-titre du graphe en cours
- add=FALSE ; si TRUE superpose le graphe au précédent
- axes=TRUE ; si FALSE ne trace pas d'axes

Paramètres récurrents

Forme générique

La plupart des fonctions graphique s'utilisent par un appel du type

1. `nom.function(object, options)`,
2. `nom.function(x, y , options)`.

Parmi les options les plus courantes, on trouve :

- type="p" ; spécifie le type de tracé : "p" pour points, "l" pour lignes, "b" pour points liés par des lignes, "o" pour lignes superposées aux points...
- xlim= ; ylim=, spécifie les limites de axes x et y
- xlab= ; ylab=, annotation des axes x et y
- main= ; titre du graphe en cours
- sub= ; sous-titre du graphe en cours
- add=FALSE ; si TRUE superpose le graphe au précédent
- axes=TRUE ; si FALSE ne trace pas d'axes

Tracer une fonction numérique I

commande `plot`

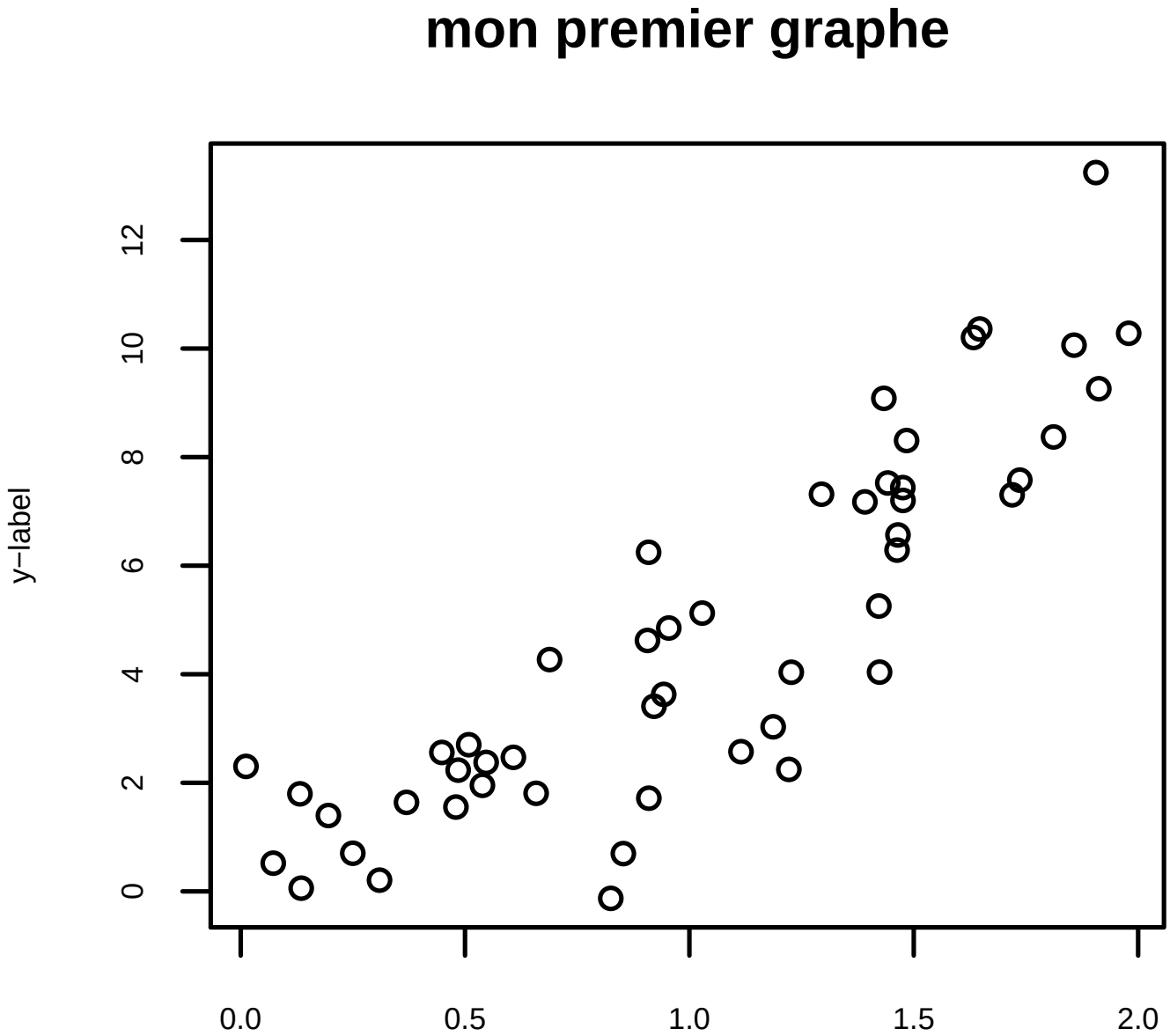
Fonction élémentaire de réprésentation graphique.

- `plot(vect)` représente le graphe des valeurs de `vect` sur l'axe des y.
- `plot(vect1,vect1)` représente le graphe des valeurs de `vect2` en fonction de `vect1`.

Par exemple, avec deux vecteurs :

```
> x <- runif(50,0,2)
> y <- 3 * x + 2 * x^2 + 1 + rnorm(50,sd=1.5)
> plot(x, y, xlab="x-label",ylab="y-label",main="mon premier graphe")
```

Tracer une fonction numérique II

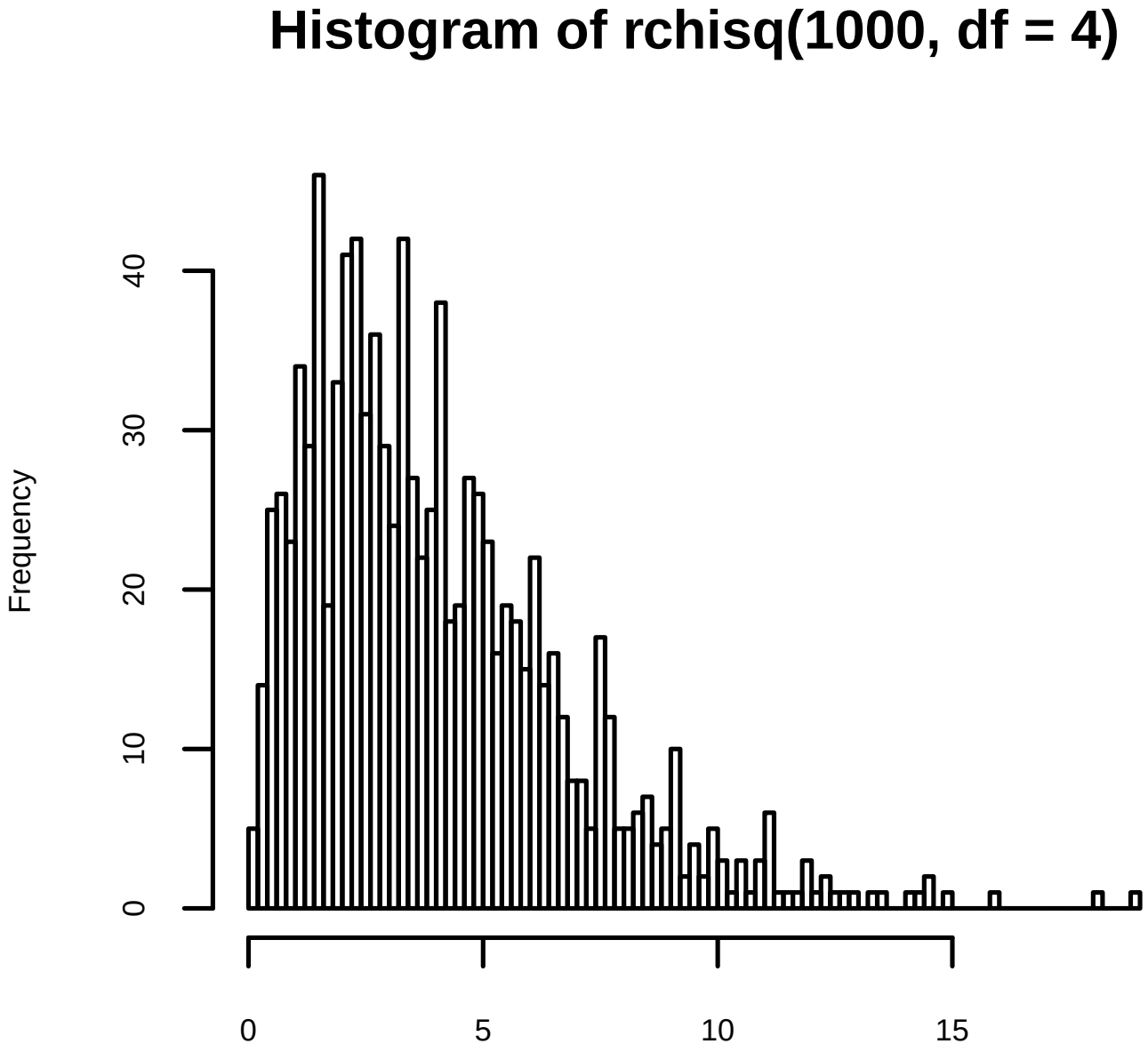


Tracer un objet R I
Toujours avec la commande plot

Beaucoup d'objet R accepte la commande plot ! En particulier, les histogrammes :

```
> mon_histo <- hist(rchisq(1000,df=4),nclass=75)
> plot(mon_histo,main="distribution empirique du Khi-2")
```

Tracer un objet R II
Toujours avec la commande `plot`



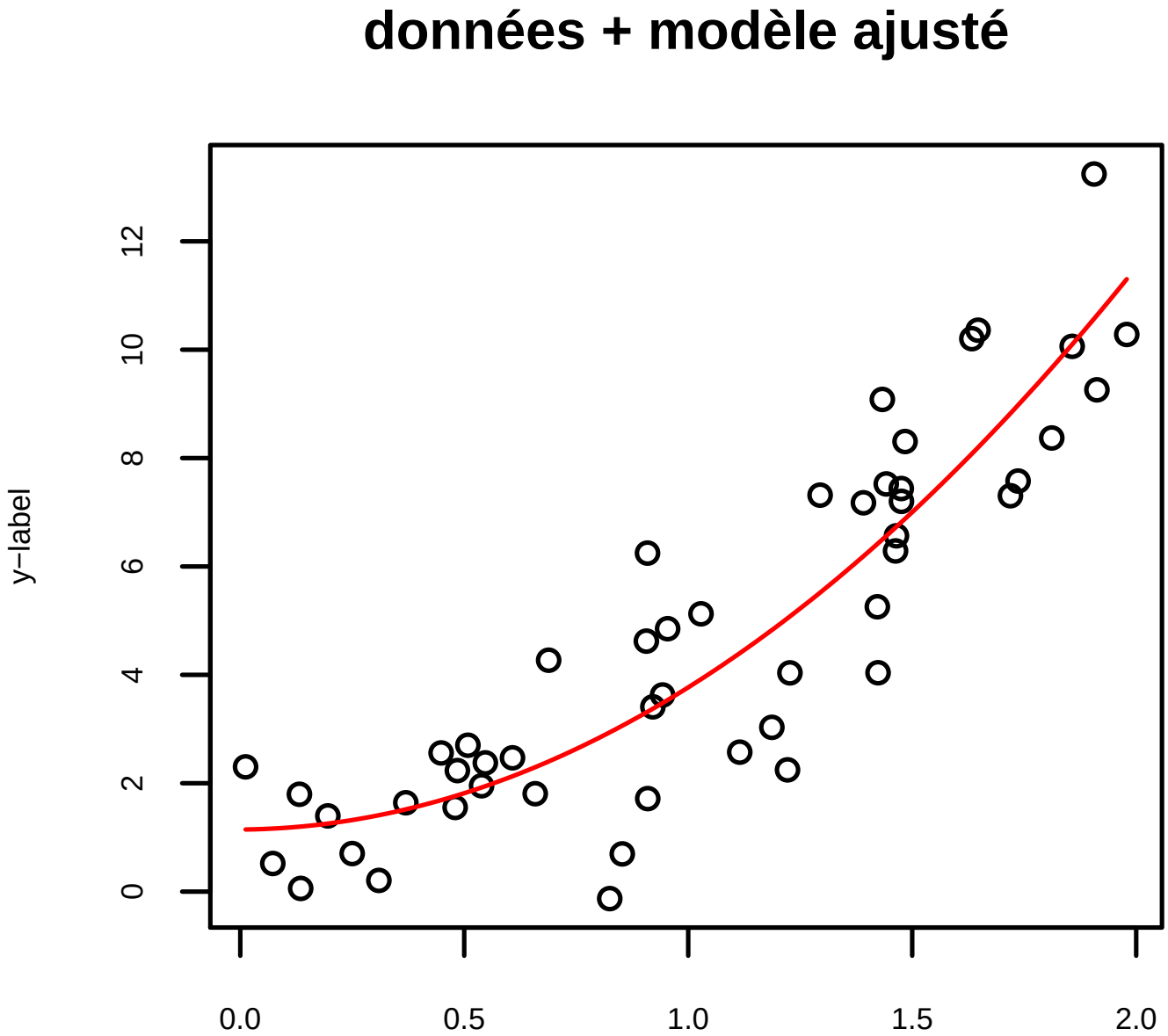
Tracer une fonction symbolique I

commande `curve`

Elle permet de tracer une fonction définie par une expression de `x`.

```
> plot(x, y, main="données + modèle ajusté",
+      xlab="x-label", ylab="y-label")
> a <- coefficients(lm(y~1+x+I(x^2)))
> curve(a[1] + a[2]*x + a[3]*x^2,add=TRUE,col="red")
```

Tracer une fonction symbolique II



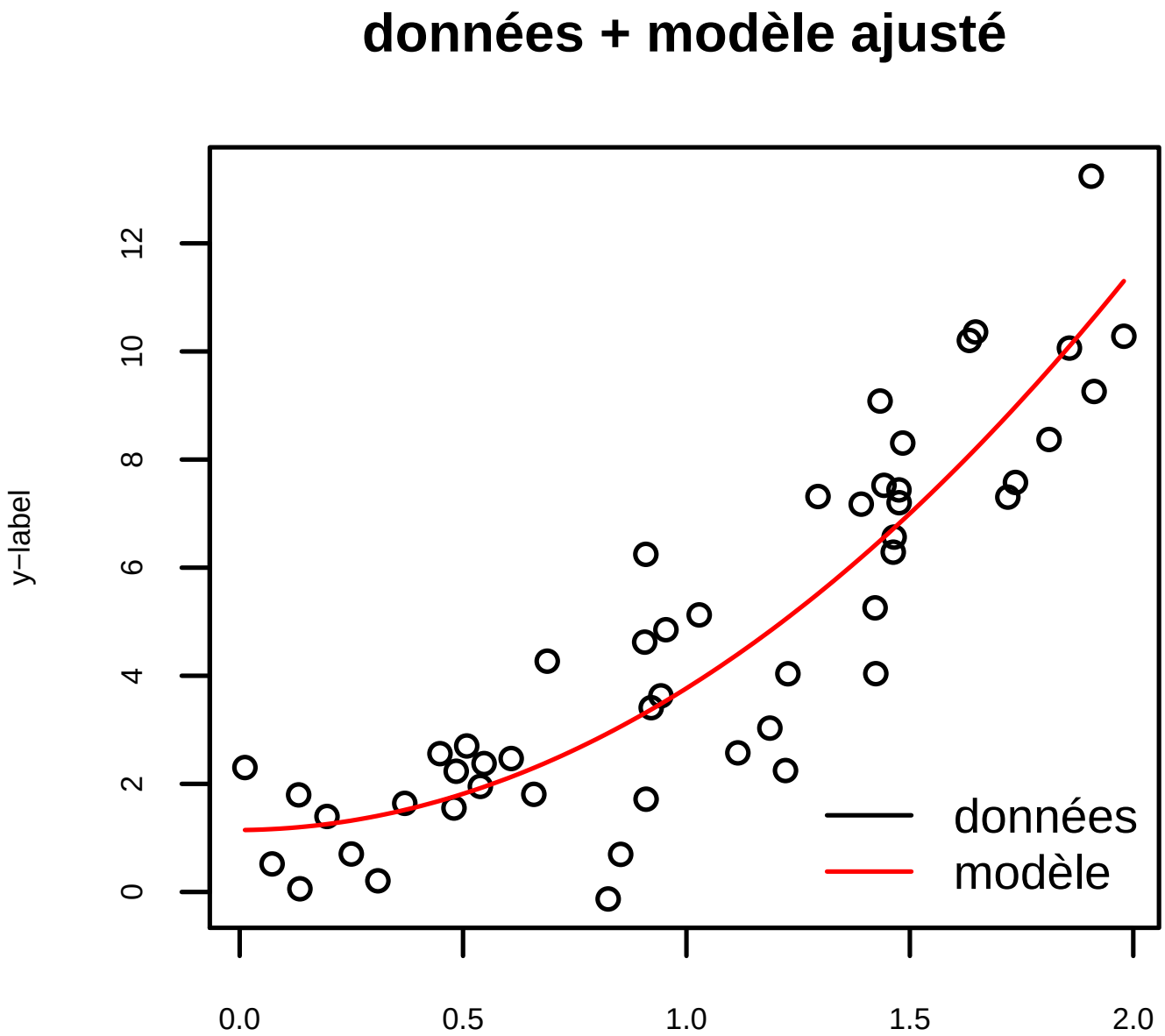
Ajouter une légende I

commande `legend`

Pour ajouter une légende. Attention aux options, assez nombreuses !

```
> plot(x, y, main="données + modèle ajusté",
+      xlab="x-label", ylab="y-label")
> a <- coefficients(lm(y~1+x+I(x^2)))
> curve(a[1] + a[2]*x + a[3]*x^2,add=TRUE,col="red")
>  legend("bottomright",c("données","modèle"),lty=c(1,1),col=c("black","red"),bty="n
```

Ajouter une légende II



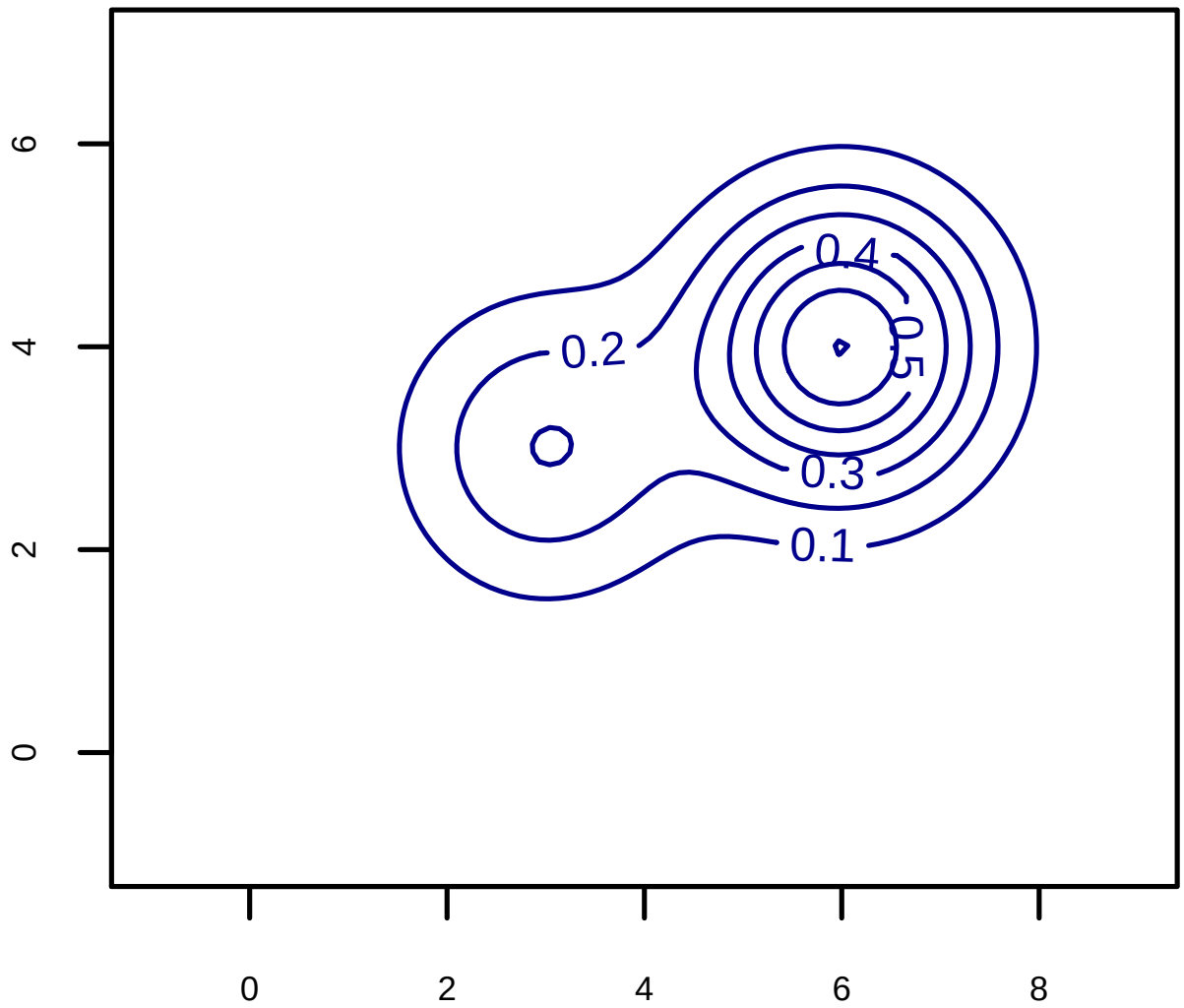
Représentation 3D (courbe de niveaux) I

commande `contour`

`contour(x,y,z)` permet de tracer des courbes de niveaux : `x` et `y` sont des vecteurs et `z` une matrice telle que les dimensions de `z` soient `length(x),length(y)`.

```
> x<-seq(-1,9,length=100)
> y<-seq(-1,7,length=100)
> z<-outer(x,y,function(x,y) 0.3*exp(-0.5*((x-3)^2 +(y -3)^2)) +
+          0.7*exp(-0.5*((x-6)^2 +(y -4)^2)))
> contour(x,y,z,col="blue4")
```

Représentation 3D (courbe de niveaux) II



Ajout de droites I

commande `abline`

`abline` permet d'ajouter à un graphe courant

- des droites de décalage `a` et de coefficient directeur `b` avec `abline(a,b)`,
- des droites verticales avec `abline(v=)`,
- des droites horizontales avec `abline(h=)`.

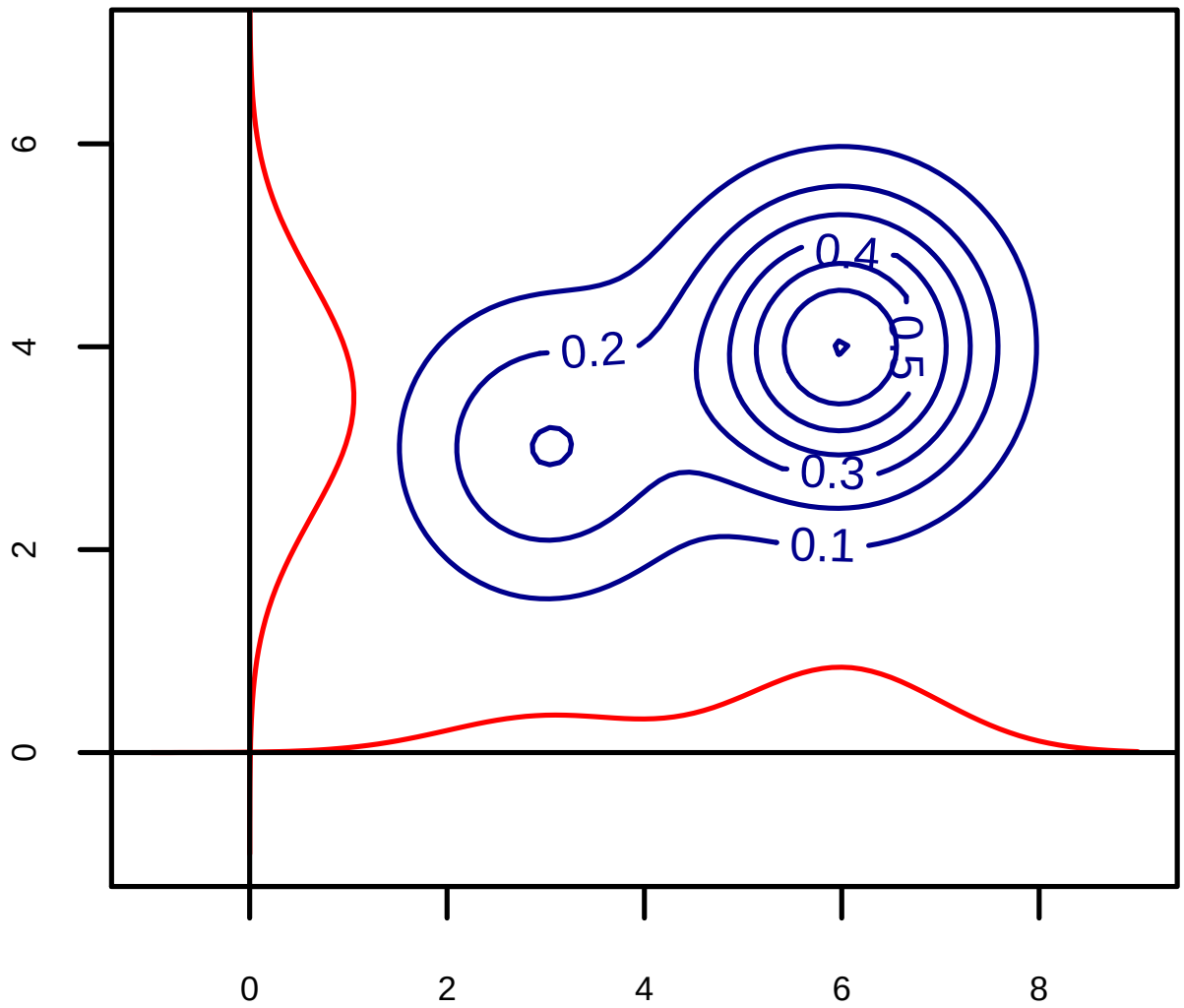
commandes `lines` et `points`

Pour ajouter une courbe ou des points : s'utilisent de manière similaire à `plot`.

Ajout de droites II

```
> contour(x,y,z,col="blue4")
> curve((0.3*dnorm(x,mean=3) + 0.7*dnorm(x,mean=6))*3,-1,9,col="red",ylim=
> x<-seq(-1,9,length=100)
> lines((0.5*dnorm(x,mean=3) + 0.5*dnorm(x,mean=4))*3,x,col="red")
> abline(h=0)
> abline(v=0)
```

Ajout de droites III



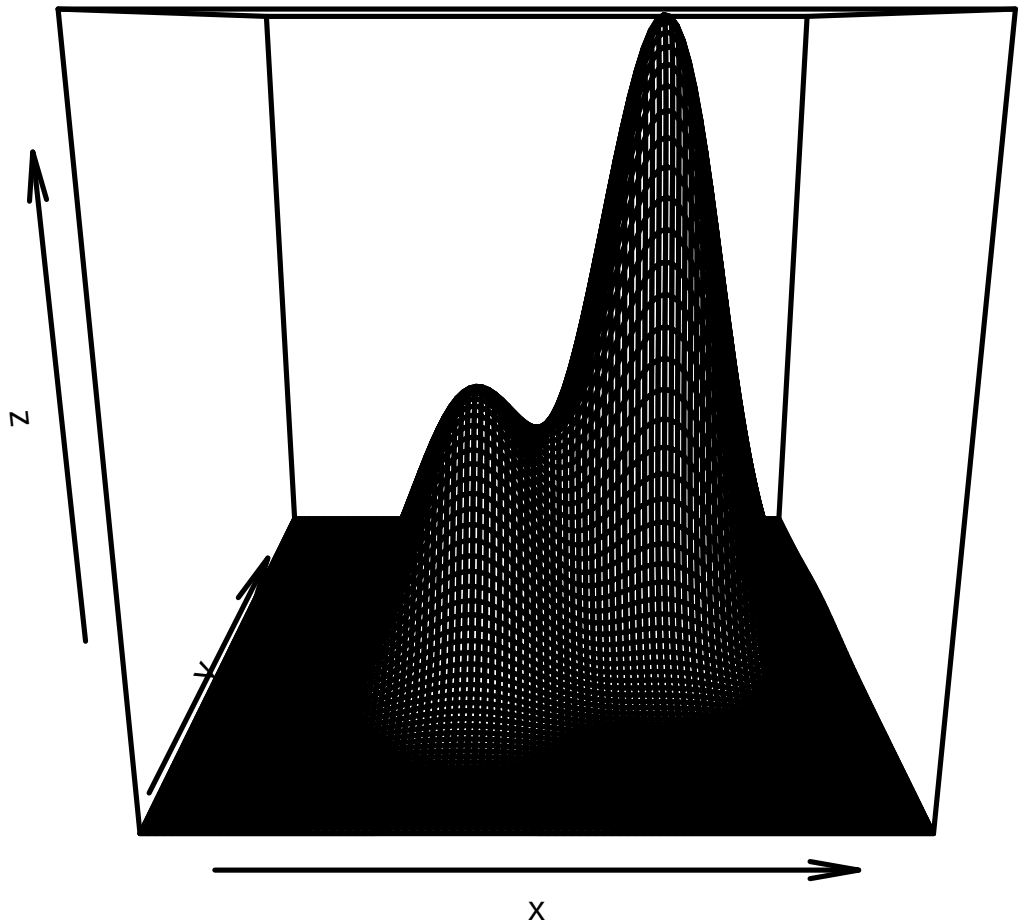
Graphe en 3D I

commande `persp`

Fonctionne comme la fonction `contour` en proposant une représentation en perpective.

```
> persp(x,y,z)
```

Graphe en 3D II



Rediriger la sortie graphique

Par défaut, R envoie les graphiques sur la sortie *écran*. De nombreuses

Exportation de graphes

Se réalise en encadrant les fonctions graphiques par les commandes `format_export(file="nom_fichier")` et `dev.off()`, où `format_fichier` peut prendre les valeurs `pdf`,`postscrip`,`png`,....

```
pdf(file="ma\_sortie.pdf")
plot(runif(20),runif(20))
dev.off()
```

Graphes multiples I

Ouverture d'une nouvelle fenêtre graphique

Se fait, selon les plateformes, avec les commandes

- `x11()` pour Linux,
- `quartz()` ou `x11()` pour Mac OS,
- `windows()`.

Découpage d'une fenêtre

Plusieurs possibilités :

- `layout(mat,width=,height=)`, qui s'utilise en découpant l'écran via la matrice `mat`.
- `par(mfrow=vect)` ou `par(mfcol=vect)` qui découpent en n lignes et m colonne spécifiées par le vecteur `vect`. Le remplissage se fait par ligne ou par colonne selon la fonction choisie.

Pour aller plus loin

- D'autres fonctions de haut niveau dans la partie dédiée aux statistiques
- Utiliser la liste des commandes usuelles pour les options et fonctions secondaires,
- La commande `par` gère les options graphiques,
- Consulter le package `lattice`, *extrêmement* puissant.

Lattice : Multivariate Data Visualization with R
Deepayan Sarkar
http://lmdvr.r-forge.r-project.org/

$\rightsquigarrow$ Cette page web propose toutes les figures et tous les codes R correspondant à leur génération !

Quatrième partie IV

Statistiques et outils connexes sous R

Les distributions disponibles

Distribution	R	Paramètres
beta	`beta`	
binomiale	`binom`	size, prob
binomiale négative	`nbinom`	
Cauchy	`cauchy`	
Chi-deux	`chiqsq`	df
Exponentielle	`exp`	rate
Fisher	`f`	df1, df2
Gamma	`gamma`	
géométrique	`geom`	
hypergéométrique	`hyper`	
log-normal	`lnorm`	
logistique	`logis`	
normale	`norm`	mean, sd
Poisson	`pois`	
Student	`t`	df
uniforme	`unif`	min, max
Weibull	`weibull`	
Wilcoxon	`wilcox`	

TABLE : Principales distributions

Les distributions disponibles

Distribution	R	Paramètres
beta	`beta`	
binomiale	`binom`	size, prob
binomiale négative	`nbinom`	
Cauchy	`cauchy`	
Chi-deux	`chiqsq`	df
Exponentielle	`exp`	rate
Fisher	`f`	df1, df2
Gamma	`gamma`	
géométrique	`geom`	
hypergéométrique	`hyper`	
log-normal	`lnorm`	
logistique	`logis`	
normale	`norm`	mean, sd
Poisson	`pois`	
Student	`t`	df
uniforme	`unif`	min, max
Weibull	`weibull`	
Wilcoxon	`wilcox`	

TABLE : Principales distributions

Les distributions disponibles

Distribution	R	Paramètres
beta	`beta`	
binomiale	`binom`	size, prob
binomiale négative	`nbinom`	
Cauchy	`cauchy`	
Chi-deux	`chiqsq`	df
Exponentielle	`exp`	rate
Fisher	`f`	df1, df2
Gamma	`gamma`	
géométrique	`geom`	
hypergéométrique	`hyper`	
log-normal	`lnorm`	
logistique	`logis`	
normale	`norm`	mean, sd
Poisson	`pois`	
Student	`t`	df
uniforme	`unif`	min, max
Weibull	`weibull`	
Wilcoxon	`wilcox`	

TABLE : Principales distributions

Tirage aléatoire

Forme générique : `r+distrib(n,...)`

`r` pour « random » : `n` donne la taille de l'échantillon et `...` sont les paramètres requis selon la forme de `distrib`.

```
> rexp(10,rate=1/5)

 [1]  1.187797  1.727852 16.687529  2.533265  2.015744  8.079484  1.700084
 [8]  1.532579 10.141084 11.772947

> rchisq(10,df=5)

 [1]  1.625728 11.349976  8.052654  4.616132  9.143822  6.426493  5.917400
 [8]  1.742962  5.332878  3.630132

> runif(10,min=-2,max=2)

 [1] -0.4308736 -1.4752795  1.4340376  0.4006754  1.0750366  0.5067289
 [7] -0.3247492  0.5917762  0.0247791  0.2960174

> mean(rbinom(1000,10,prob=1/2))

[1] 4.99

> var(rnorm(1000,mean=5,sd=2))

[1] 4.220447
```

La loi des grands nombres

Théorème

Soit $(X_i)_{i=1,\dots,n}$ une suite de variables aléatoires indépendantes de même loi et soit S_n la somme de ces variables. Alors,

$$\frac{S_n}{n} \xrightarrow[p.s]{n\to\infty} \mathbb{E}[X].$$

Conséquence théorique

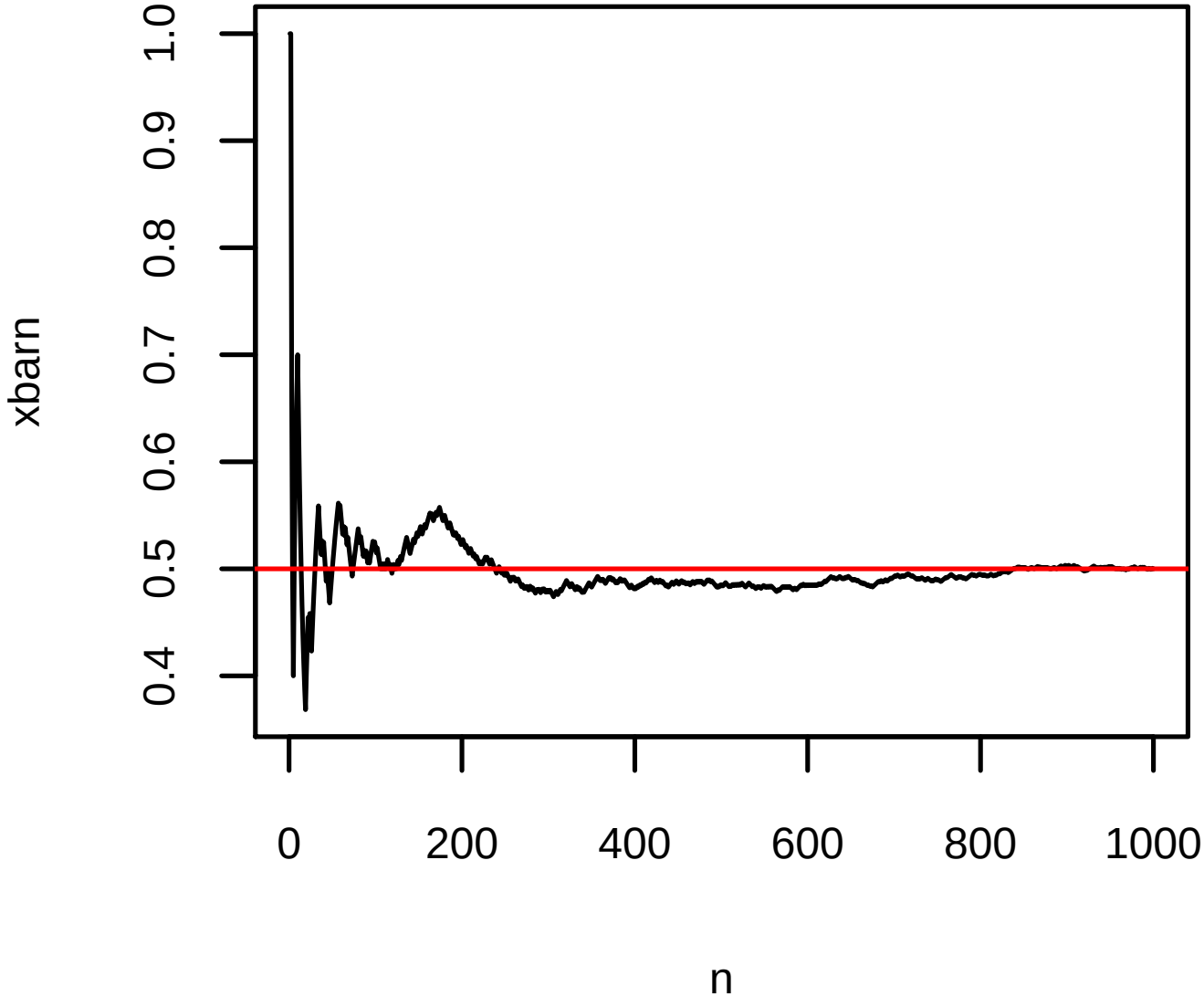
La moyenne empririque $\bar{X} = S_n/n$ est un estimateur fortement convergent de l'espérance d'une loi.

Conséquence pratique

C'est le fondement de la plupart des simulations numériques en statistiques.

Illustration

L'exemple passe-partout (mais parlant) : simuler l'issue d'un tirage de pile ou face, et observer l'évolution.



Le théorème de la limite centrale

Théorème

Soit $(X_i)_{i=1,\dots,n}$ une suite de variable aléatoire indépendante de même loi, d'espérance μ et de variance σ^2. Alors,

$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow[\mathcal{L}]{n\to\infty} \mathcal{N}(0,1).$$

Conséquences

- L'écart à la moyenne suit une loi normale lorsque l'on observe beaucoup d'individus, quelque soit la loi de la variable observée.
- C'est un résultat portant sur un indice global de la popualtion pas sur les individus ! ! !

Illustration

Protocole

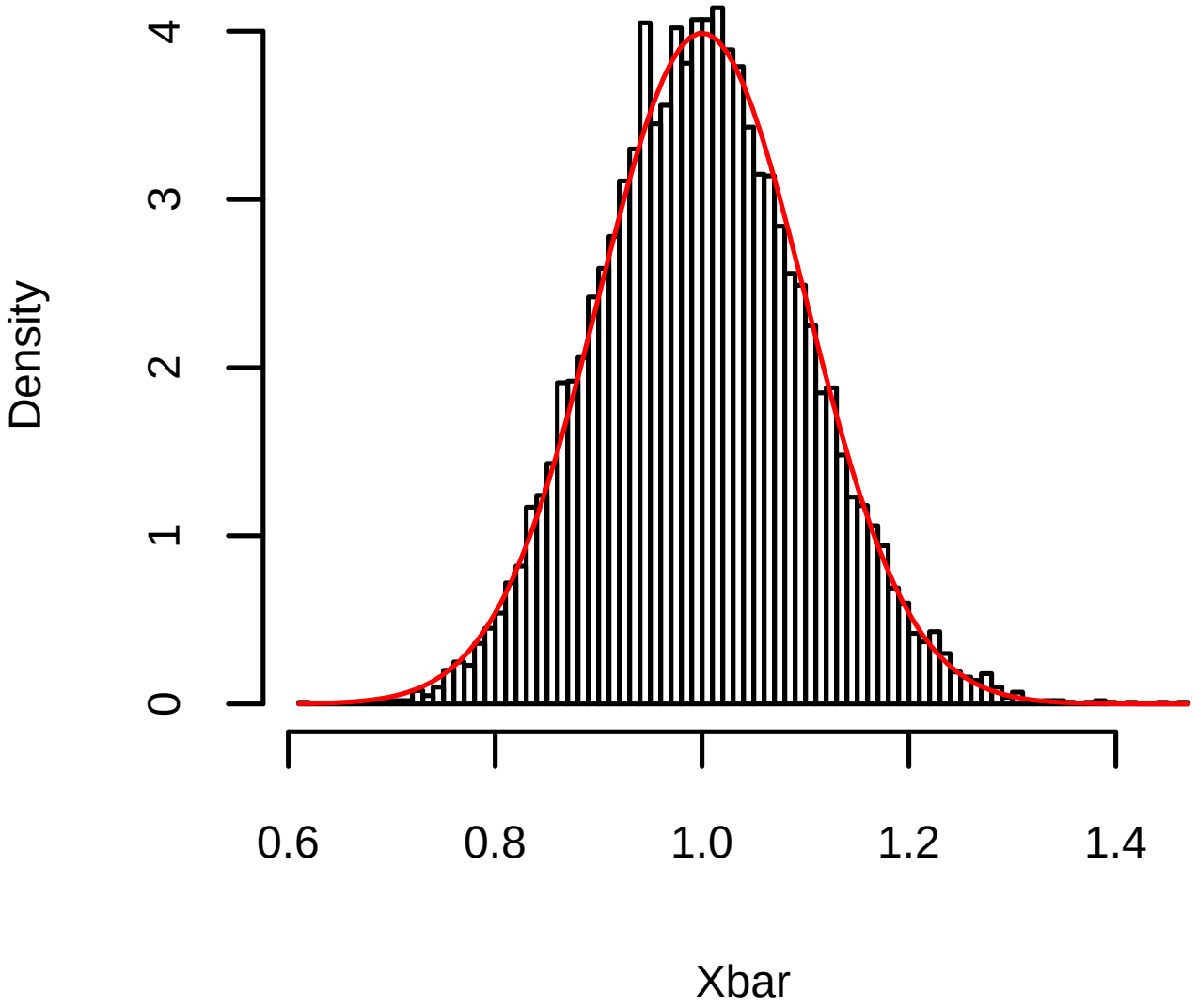
- On simule n réalisations d'une variable exponetielle,
- On calcule la valeur observés de la moyenne empirique,
- On répète m fois cette expérience,
- On trace l'histogramme des m observations de $\bar{X}_n$.

En R (compact !), ca donne :

```
> n <- 100
> m <- 10000
> Xi <- matrix(rexp(n*m),n,m)
> Xbar <- colMeans(Xi)
> hist(Xbar,nclass=100)
```

Illustration : graphique

```
> hist(Xbar,nclass=100,prob=TRUE,main="")
> curve(dnorm(x,mean=1,sd=1/sqrt(n)),add=TRUE,col="red")
```



Le théorème de Cochran

Théorème

Soit $(X_i)_{i=1,\ldots,n}$ une série de variables aléatoires indépendantes, de même loi, d'espérance μ et de variance σ^2. Alors, lorsque $n \to \infty$,

$$\sum_{i=1}^{n}(X_i - \bar{X}_n)^2 \sim \sigma^2 \chi^2_{n-1},$$

ce qu'on écrit également

$$n\hat{\sigma}^2 \sim \sigma^2 \chi^2_{n-1},$$

où $\hat{\sigma}^2$ est la variance empirique.

Démarche

1. Choisir les hypothèses à tester (H0 et H1)
2. Fixer le niveau du test α
3. Choisir une statistique de test
4. Déterminer la règle de décision (région de rejet Γ)
5. Calculer la statistique (et la p-valeur)
6. Conclure

Typologie

- Tests paramétriques versus tests non paramètriques
- Tests de comparaison versus tests d'adéquation

Quelques test en R

- Test de Student (T test) : `t.test()`
- Test de Fisher (F test) : `var.test()`
- Test sur une proportion : `binom.test()`
- Test du Chi2 : `chisq.test()`
- Wilcoxon/Mann-Whitney : `wilcox.test()`
- Kolmogorov-Smirnov : `ks.test()` item Analyse de la variance : `lm()` où `anova()`
- Permutation test (comparaison)

Test de l'espérance d'un caractère (1)

Problème

Soit X un caractère d'intérêt d'une population $\mathcal{P}$ de loi, d'espérance μ et de variance σ^2 inconnue. On propose une valeur μ_0 de l'espérance, et on teste sa validité en observant un échantillon i.i.d $(X_i)_{i=1,\dots,n}$:

$$\begin{cases} H_0 : & \mu = \mu_0, \\ H_1 : & \mu = \mu_1, \text{ avec } \mu_1 \neq \mu_0. \end{cases}$$

Par le théorème central limite et le théorème de Cochran, on montre que

$$\frac{\bar{X}_n - \mu}{\hat{\sigma}/\sqrt{n}} \xrightarrow[n\to\infty]{\mathcal{L}} \mathcal{T}(n-1),$$

où $\mathcal{T}(n-1)$ est une loi de Student à $n-1$ degrés de liberté.

Test de l'espérance d'un caractère (2)

Si on teste

$$\begin{cases} H_0 : & \mu = \mu_0, \\ H_1 : & \mu = \mu_1, \text{ avec } \mu_1 > \mu_0, \end{cases}$$

alors, au niveau de confiance α,

$$\text{on rejette } H_0 \ \alpha \text{ si } \bar{x}_n > \mu_0 + \frac{\hat{\sigma}}{\sqrt{n}} u_{1-\alpha}.$$

Définition

La p–valeur ou degré de significativité donne le risque que l'on prend à choisir H_1 plutôt que H_0. Si elle est supérieure à α, on conserve H_0 et on rejette sinon.

Test du paramètre d'espérance avec R

Exemple

(Sirop contre la toux) La notice d'un sirop contre la toux indique comme valeur de référence pour la moyenne m_0 de l'agent actif $40g$/litre. Le contrôleur de la fabrication décidera d'arrêter provisoirement la production si la moyenne m inconnue est strictement inférieure à cette valeur de référence.
Le contrôleur de la fabrication prélève de manière indépendantes 9 bouteilles au hasard dans la production et mesure la quantité d'agent actif.
Conclusion ?

```
> x <- c(38.7, 39.6, 37.9, 40.6, 40.5, 37.7, 41.2, 37.5, 39.1)
> t.test(x,mu=40,alternative="less")

        One Sample t-test

data:  x
t = -1.7586, df = 8, p-value = 0.05835
alternative hypothesis: true mean is less than 40
95 percent confidence interval:
     -Inf 40.04593
sample estimates:
mean of x
     39.2
```

Test de Student d'égalité des espérance

Problème

Soit X et Y deux caractères d'intérêt d'une population $\mathcal{P}$ de loi, d'espérance μ_1 et μ_2 et de variance commune σ^2, toutes inconnues. On propose de tester l'égalité des espérances en observant deux échantillons i.i.d $(X_i)_{i=1,\dots,n}$ et $(Y_i)_{i=1,\dots,m}$:

$$\begin{cases} H_0 : & \mu_1 = \mu_2, \\ H_1 : & \mu_1 \neq \mu_2, \end{cases}$$

Par le théorème central limite et le théorème de Cochran, on montre que

$$\frac{\bar{X}_n + \bar{Y}_m - (\mu_1 + \mu_2)}{s^* \sqrt{n^{-1} + m^{-1}}} \xrightarrow[n\to\infty]{\mathcal{L}} \mathcal{T}(n+m-2),$$

où $S^{*2} = \frac{(n-1){S_X^*}^2 + (m-1){S_Y^*}^2}{n+m-2}$

Test de Student d'égalité des espérances sous R

Testons l'égalité des poids entre espèces (CE,CO) et TE chez la vigne :

```
> vigne <- read.delim("mesures_baie_raisin_2008-2009.txt",hea
> vigne <- vigne[-c(2,6,7)]
> colnames(vigne) <- c("pop","pepin.08","poids.08","volcm3.08
> attach(vigne)
Les objets suivants sont masqués from vigne (position 3):

    pepin.08, poids.08, pop, volcm3.08
Les objets suivants sont masqués from vigne (position 4):

    pepin.08, poids.08, pop, volcm3.08
Les objets suivants sont masqués from vigne (position 5):

    pepin.08, poids.08, pop, volcm3.08
Les objets suivants sont masqués from vigne (position 6):

    pepin.08, poids.08, pop, volcm3.08
```

Test de permutation pour comparer deux échantillons

Considérons deux groupes de variables aléatoires $(X_1, \ldots, X_m)$ et $(Y_1, \ldots, Y_n)$ i.i.d. Le test vise à trancher entre

H_0 : les échantillons ont même distribution
H_1 : les échantillons ont des distributions différentes.

Il existe $B = C_{n+m}^m$ façons différentes de répartir les $n + m$ observations en deux groupes de tailles respectives n et m (en fait c'est le nombre de groupe de taille m que l'on peut former avec $m + n$ observations, car une fois le premier groupe formé, le second est automatiquement déterminé).

Procédure

1. Calculer la statistique t_b pour chaque permutation b de 1 à B. Si C_{n+m}^{m} le nombre de permutation total est trop important, un échantillonnage aléatoire est réalisé pour obtenir un nombre raisonnable de B échantillons permutés.
2. Ordonner les statistiques obtenues sur les échantillon permutés
$$t_{(1)}, t_{(2)}, \ldots, t_{(B)}$$
3. Calculer t la statistique de décision sur les échantillons réellement observés.
4. Pour une erreur données α de type I, l'hypothèse nulle est rejetée si la statistique observée est parmi les $100\alpha\%$ les plus extrêmes valeurs de la statistique de test.

p-value de permutation

Une façon équivalente de présenter cet algorithme utilise le concept de p-value de permutation. Une définition possible de la p-value de permutation est

$$\frac{j}{B-1}$$

où j est le nombre de valeurs de la statistique $t_{(i)}$ plus extrêmes que la valeur observée t ($B-1$ et non B car la valeur t observée fait forcement partie des valeur calculée par permutation). Cette valeur varie forcement entre 0 et 1 et suit une distribution uniforme conditionnellement à l'hypothèse nulle, car sous H_0 chaque valeur de la statistique a une probabilité $\frac{1}{B}$.

Problème

Soit X et Y deux caractères d'intérêt d'une population $\mathcal{P}$ de lois et de variances σ_1^2 et σ_2^2 inconnues. On propose de tester l'égalité des variances en observant deux échantillons i.i.d $(X_i)_{i=1,\dots,n}$ et $(Y_i)_{i=1,\dots,m}$:

$$\begin{cases} H_0 : & \sigma_1 = \sigma_2, \\ H_1 : & \sigma_1 \neq \sigma_1, \end{cases}$$

Par le théorème central limite et le théorème de Cochran, on montre que

$$\frac{{S_X^*}^2/\sigma_1^2}{{S_Y^*}^2/\sigma_2^2} \xrightarrow[n\to\infty]{\mathcal{L}} \mathcal{F}(n-1, m-1).$$

Le test d'égalité des variances sous R

Testons l'égalité des variances des poids entre espèces (CE,CO) et TE chez la vigne :

```
> var.test(poids.08[pop!="TE"],poids.08[pop=="TE"])

        F test to compare two variances

data:  poids.08[pop != "TE"] and poids.08[pop == "TE"]
F = 0.1846, num df = 150, denom df = 65, p-value < 2.2e-16
alternative hypothesis: true ratio of variances is not equal
95 percent confidence interval:
 0.1198802 0.2746481
sample estimates:
ratio of variances
         0.1845923
```

Exercice

1. Simuler deux populations de tailles 1000, suivant des loi normales de variance unité et de moyenne respective 0 et 0.5,
2. Tester si les moyennes des deux populations sont différentes par
   - un test de Student
   - un test de Wilcoxon
   - un test de permutation