

Notes de Cours sur le logiciel R

Anne PHILIPPE

22 janvier 2018

Université de Nantes,
Laboratoire de Mathématiques Jean Leray
email : Anne.philippe@math.univ-nantes.fr

1

Plan

1. Objets et Opérations
2. Les fonctions
3. Les graphiques
4. Graphiques avec ggplot2
5. Structures de contrôle et Itérations
6. Autour des lois de probabilités
7. Outils graphiques en statistique
8. Inférence statistique

2

Installation

Le logiciel R est un *freeware* disponible sur le site

<http://cran.r-project.org/>

Il existe des versions

- Windows
- MacOS X
- Linux ...

Outils disponibles :

- un langage de programmation orienté objet
- des fonctions de "base"
- des librairies/packages complémentaires (1800 sur le site CRAN)

3

Documentations

- Documents sur le logiciel R :

http://www.math.sciences.univ-nantes.fr/~philippe/R_freeware.html

- Site consacré aux graphiques
addictedtor.free.fr/graphiques/
- Collection spécifique UseR chez Springer
- Plus de 80 livres,
par exemple
 - Introductory Statistics With R
 - Bayesian Computation With R
 - Applied Statistical Genetics With R :
 - Generalized Additive Models : An Introduction with R
 - Extending the Linear Model With R
 - Time Series Analysis And Its Applications : With R Examples

4

Librairies

- Toutes les librairies ne sont pas chargées au lancement du logiciel
- `library()` retourne la liste des librairies installées.
- `library(LIB)` charge la librairie `LIB`
- `library(help = LIB)` retourne la liste des fonctions de la librairie `LIB`
- `search()`, `searchpaths()` retourne la liste des librairies chargées.

9

Objets et Opérations

Opérations élémentaires

1. Opérations élémentaires sur les scalaires : *, -, +, /, ^

```
>2+4  
6
```

2. Opérations avec affectation (avec ou sans affichage)

```
x=2+4  
x  
6  
(x=2+4)           # avec affichage du résultat  
6
```

3. Les principaux types sont

- entier , réel, complexe
- caractère
- logique : TRUE, FALSE, NA (not available)

10

Objets

Les objets de base sont

- vecteurs, matrices
- data.frames, listes

Quelques fonctions génériques :

- `ls()` retourne la liste des objets de la session.
- `rm(a)` supprime l'objet `a`

11

Fonctions `is/as`

- `is.xxx(obj)` teste si `obj` est un objet de type `xxx`
- `as.xxx(obj)` contraint si possible `obj` au type d'objet `xxx` où `xxx` représente un type d'objet (complex, real, vector matrix etc...)

```
> x=3
> is.real(x)
[1] TRUE
> is.complex(x)
[1] FALSE
```

```
> as.complex(x)
[1] 3+0i
> as.character(x)
[1] "3"
```

Remarque :

Conversion de TRUE / FALSE en valeur numérique :

```
> as.integer(T)
[1] 1
> as.integer(F)
[1] 0
```

12

Créer des vecteurs

- la fonction `c()` concatène des scalaires ou des vecteurs :

```
> x=c(1,4,9)
> y=c(x,2,3)
> y
[1] 1 4 9 2 3
```

- Suites arithmétiques de raison 1 ou -1 : `c(a:b)`.

```
> c(1:4)           > c(4:1)
# a<b  raison 1   # a>b  raison -1
[1] 1 2 3 4       [1] 4 3 2 1

# a-b n'est pas un entier
> c(1.4:7)
[1] 1.4 2.4 3.4 4.4 5.4 6.4
```

13

- Généralisation : `seq(a,b,t)` où `a` est premier terme, le dernier $\leq b$ et la raison `t`

```
seq(from, to)           la raison est 1
seq(from, to, by= )     on fixe la raison
seq(from, to, length.out= ) on fixe le nb de termes
```

par exemple

```
> seq(1,4,by =0.1)
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 ...
[26] 3.5 3.6 3.7 3.8 3.9 4.0
```

- `x=rep(y ,n)` pour créer un vecteur constitué de l'élément `y` répété `n` fois. (`y` peut être un scalaire ou un vecteur) par exemple

```
> rep(1,4)
[1] 1 1 1 1
```

14

Créer des matrices

Les matrices sont créées avec la fonction `matrix()` à partir d'un vecteur. On doit fixer le nombre de colonnes `ncol` et/ou le nombre de lignes `nrow`.

```
> x = matrix(c(2,3,5,7,11,13),ncol=2)
```

Par défaut la matrice est remplie colonne par colonne. Pour remplir ligne par ligne, on ajoute l'argument `byrow=T`

```
> y = matrix(c(2,3,5,7,11,13),ncol=2, byrow=T)
> x           > y
[ ,1] [ ,2]           [ ,1] [ ,2]
[1,]  2   7           [1,]  2   3
[2,]  3  11           [2,]  5   7
[3,]  5  13           [3,] 11  13
```

15

Attention : si la dimension du vecteur n'est pas égale au produit (`ncol` × `nrow`) alors l'opération effectuée est la suivante :

```
> matrix(c(1:3), ncol=2,nrow=3)
     [,1] [,2]
[1,]  1   1
[2,]  2   2
[3,]  3   3
```

```
> matrix(c(1:3), ncol=2)
     [,1] [,2]
[1,]  1   3
[2,]  2   1
```

16

Quelques matrices particulières : diagonale, Toeplitz

```
> diag(1:4)
     [,1] [,2] [,3] [,4]
[1,]  1   0   0   0
[2,]  0   2   0   0
[3,]  0   0   3   0
[4,]  0   0   0   4
```

```
> toeplitz(1:4)
     [,1] [,2] [,3] [,4]
[1,]  1   2   3   4
[2,]  2   1   2   3
[3,]  3   2   1   2
[4,]  4   3   2   1
```

diag

La fonction `diag` retourne une matrice diagonale lorsque le paramètre d'entrée est un vecteur.

Si le paramètre d'entrée est une matrice, alors elle retourne un vecteur constitué de la diagonale de la matrice

17

Concaténer des vecteurs/matrices

- `rbind`



- `cbind`



```
> x=1:10
> y=x^2
> rbind(x,y)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
x      1    2    3    4    5    6    7    8    9    10
y      1    4    9   16   25   36   49   64   81   100
> cbind(x,y)
     x    y
[1,]  1    1
[2,]  2    4
[3,]  3    9
[4,]  4   16
[5,]  5   25
[6,]  6   36
etc
```

18

Extraire des éléments d'un vecteur ou d'une matrice

```
> vect=c(1.5:9.5)
> vect
[1] 1.5 2.5 3.5 4.5 5.5
6.5 7.5 8.5 9.5
```

```
> mat=matrix(vect, ncol=3,nrow=3)
     [,1] [,2] [,3]
[1,]  1.5  4.5  7.5
[2,]  2.5  5.5  8.5
[3,]  3.5  6.5  9.5
```

Extraire un élément

```
> vect[1]
[1] 1.5
> mat[2,1]
[1] 2.5
```

Colonne/ligne d'une matrice

```
> mat[,1]
[1] 1.5 2.5 3.5
> mat[3,]
[1] 3.5 6.5 9.5
```

Extraire un bloc ou plusieurs coordonnées

```
> mat[2:3,1:2]
     [,1] [,2]
[2,]  2.5  5.5
[3,]  3.5  6.5
> vect[c(1,3,7)]
[1] 1.5 3.5 7.5
```

Attention : `vect[-j]` retourne le vecteur `vect` sans la `j` ème coordonnée

```
> vect[-c(1,3,7)]
retourne 2.5 4.5 5.5 6.5 8.5 9.5
```

19

Opérations sur les Matrices/Vecteurs

- Les opérations + * - / entre 2 vecteurs ou matrices de même dimension sont des **opérations terme à terme**.

```
> x
[1] 1 2 3 4 5
> y
[1] 0 0 0 1 1
> x*y
[1] 0 0 0 4 5

> A
  [,1] [,2]
[1,]  2   1
[2,]  4   9
> B
  [,1] [,2]
[1,]  0   2
[2,]  1   1
> A*B
  [,1] [,2]
[1,]  0   2
[2,]  4   9
>
```

20

Attention

Si les vecteurs ne sont pas de même longueur, le plus court est complété automatiquement.

```
> x =c(1:5)
> x
[1] 1 2 3 4 5
> y =c(1,2)
> y
[1] 1 2
> x + y
[1] 2 4 4 6 6

x : 1 2 3 4 5
y : 1 2 1 2 1
-----
x+y : 2 4 4 6 6
```

21

Quelques opérations particulières sur les matrices

```
> a=matrix(1, ncol=2, nrow=2)
> a
  [,1] [,2]
[1,]  1   1
[2,]  1   1
> a+3 #matrice +scalaire
  [,1] [,2]
[1,]  4   4
[2,]  4   4
> a+c(1:2) #matrice + vecteur
  [,1] [,2]
[1,]  2   2
[2,]  3   3
```

22

Action d'une fonction sur un vecteur ou une matrice

Soit FUN une fonction définie sur les scalaires qui retourne un scalaire.

Par exemple

sqrt	square root
abs	absolute value
sin cos tan	trigonometric functions (radians)
exp log	exponential and natural logarithm
log10	common logarithm
gamma lgamma	gamma function and its natural log

Lorsque le paramètre d'entrée est un vecteur (respectivement une matrice), la fonction FUN est appliquée sur chacune des composantes. L'objet retourné est un vecteur (respectivement une matrice).

s

Exemple

Si $A = (a_{i,j})$ est une matrice, alors `exp(A)` retourne une matrice constituée des éléments $e^{a_{i,j}}$.

23

Quelques fonctions sur les matrices

- Le produit matriciel est obtenu avec `% * %`
- Calcul des valeurs/vecteurs propres : `eigen`
- Calcul du déterminant : `det`
- `t(A)` retourne la transposée de la matrice A
- décomposition de Choleski : `chol(X)` retourne R telle que $X = R'R$ où R est une matrice triangulaire supérieure et R' est la transposée de R .
- décomposition svd : `svd(X)` retourne (U,D,V) telles que $X = UDV'$ où U et V sont orthogonales et D est diagonale.

`solve`

- `solve(A)` retourne l'inverse de la matrice A
- `solve(A,b)` retourne x tel que $Ax = b$

24

Objets booléens et instructions logiques

- Les opérations logiques : `<` , `>` , `<=` , `>=` , `!=` [différent] , `==` [égal] retournent `TRUE` ou `FALSE`.
- La comparaison entre deux vecteurs est une comparaison terme à terme.
- Si les vecteurs ne sont pas de même longueur, le plus court est complété automatiquement.

```
> a= 1:5 ; b=2.5
> a<b
[1] TRUE TRUE FALSE FALSE FALSE
```

Il est possible de définir plusieurs conditions à remplir avec les opérateurs

- ET : `&`
- OU : `|`

25

Pour extraire les éléments d'un vecteur `vect`, on peut utiliser des instructions logiques.

- Soit I un vecteur de booléens de même longueur que `vect` : `vect[I]` retourne les coordonnées `vect[j]` telles que $I[j] = TRUE$.

Applications

- extraire les composantes >8
`vect[vect>8]` : `vect>8` est un vecteur de `TRUE` et `FALSE`, on extrait les composantes affectées à `TRUE`.
- extraire les composantes >8 ou <2
`vect[(vect>8) | (vect<2)]`
- extraire les composantes >8 et <10
`vect[(vect>8) & (vect<10)]`

26

Effet de la précision sur la comparaison de réels

Est ce que $\sqrt{2}^2 = 2$?

```
> (sqrt(2)^2 == 2)
[1] FALSE
```

Une solution

```
> all.equal(sqrt(2)^2,2)
[1] TRUE
#ou
> isTRUE(all.equal(sqrt(2)^2,2))
```

27

Fonction `which`

Soit `vec` un vecteur logique. La fonction `which(vec)` retourne les indices des coordonnées du vecteur `vec` qui prennent la valeur `TRUE`

```
> x=(1:10)^2
> x
 [1]  1  4  9 16 25 36 49 64 81 100
> which(x== 25)
 [1] 5
> which(x > 21)
 [1] 5 6 7 8 9 10
```

Exemple

Les commandes `x[x>1]` et `x[which(x>1)]` retournent le même vecteur.

Cas particulier

- `which.max(x)` retourne `which(x==max(x))`

28

Ces fonctions retournent un scalaire :

- `sum()` (somme $\sum_i x_i$), `prod()` (produit $\prod_i x_i$), `mean()` (moyenne $\frac{1}{n} \sum_{i=1}^n x_i$)
- `max()`, `min()`
- `length()` (longueur du vecteur),
- `dim()`, `ncol()`, `nrow()` (dimension de la matrice/nombre de lignes / nombre de colonnes.)

Ces fonctions retournent un vecteur :

- `cumsum()` (sommés cumulés ($x_1, x_1 + x_2, \dots, \sum_{i=1}^n x_i$), `cumprod()` (produits cumulés),
- `sort (tri)`, `order`, `unique`
remarque : `sort(x) = x[order(x)]`
- `fft()` (transformé de Fourier)

29

Définition des `data.frames`

C'est une matrice dont toutes les colonnes ne sont pas nécessairement du même type : scalaire, booléen, caractère. Par exemple

```
> data1= data.frame(x1=1,x2=1:10,a=letters [1:10])
  x1 x2 a
1  1  1 a
2  1  2 b
3  1  3 c
4  1  4 d
5  1  5 e
6  1  6 f
7  1  7 g
8  1  8 h
9  1  9 i
10 1 10 j
```

Par défaut les lignes sont numérotées 1,2 etc.

30

Opérations sur les `dataframes`

1. Pour visualiser les premières lignes `head()`
2. Pour définir ou visualiser le nom des lignes `row.names`
3. Pour définir ou visualiser le nom des colonnes `names`
4. La dimension de l'objet est donnée par `dim`

```
> names(data1)
 [1] "x1" "x2" "a"
> names(data1)<- c("c1","c2","c3")
> head(data1,3)
  c1 c2 c3
1  1  1  a
2  1  2  b
3  1  3  c
> dim(data1)
 [1] 10  3
> row.names(data1) <- letters[1:10]
#le vecteurs letters contient les lettres de l'alphabet
> head(data1,2)
  c1 c2 c3
a  1  1  a
b  1  2  b
```

31

Opérations sur les dataframes

Les opérations entre des dataframes sont opérations terme à terme comme pour les matrices.

```
A = data.frame(x=1:3,y=2:4)
B = data.frame(xx=1,yy=1:3)
C = data.frame(x=1:3,y=rep("a",3))
```

```
> A      > B      > A+B      > C
  x y      xx yy      x y      x y
1 1 2      1  1  1      1 2 3      1 1 a
2 2 3      2  1  2      2 3 5      2 2 a
3 3 4      3  1  3      3 4 7      3 3 a
```

```
> A+C
  x y
1 2 NA
2 4 NA
3 6 NA
```

Warning message:
In Ops.factor(left, right):
ceci n'est pas pertinent pour des
variables facteurs

32

Opérations sur les dataframes

Pour extraire un élément ou un bloc, la syntaxe est la même que pour les matrices.

Pour extraire une colonne les deux syntaxes suivantes peuvent être utilisées

```
> A$x
[1] 1 2 3
> A[,1]
[1] 1 2 3
```

Pour concaténer des dataframes ayant le même nombre de lignes

```
data.frame(A,B)
  x y xx yy
1 1 2  1  1
2 2 3  1  2
3 3 4  1  3
```

33

Définition d'une liste

C'est une structure qui regroupe des objets (pas nécessairement de même type). On crée les listes avec la fonction `list`

Exemple

On construit une liste appelée `rdn` qui contient 3 objets :

- un vecteur dans `serie`
- un scalaire dans `taille`
- une chaîne de caractères dans `type`

La syntaxe est la suivante

```
>rdn=list(serie=c(1:100),taille=100,type="arithm")
```

attention

Une liste peut être créée sans donner des noms aux variables c'est à dire `rdn=list(c(1:100),100,"arithm")`.

34

Opérations sur les listes

- Pour visualiser la liste des composantes d'une liste

```
>names(rdn)
[1] "serie" "taille" "type"
> summary(rdn)
      Length Class  Mode
serie   100  -none- numeric
taille    1  -none- numeric
type     1  -none- character
```

- Pour atteindre les composantes d'une liste

```
>rdn$taille OU >rdn[[2]]
[1] 100          [1] 100
```

attention

Si la liste a été créée sans spécifier de noms aux variables, il n'y a pas de nom par défaut et la seule la première syntaxe est utilisable.

35

Importer/exporter des données

- Pour extraire les objets d'une liste

```
>attach(rdn)
"serie" "taille" "type"
```

- supprimer les objets créés à la ligne précédente :

```
>detach(rdn)
```

36

1. Importer une suite : `x=scan("data.dat")` : pour créer un vecteur à partir de données stockées dans un fichier, ici `data.dat`.

2. Importer un tableau :

```
x=read.table("data.dat")
```

```
x=read.table("data.dat", header=TRUE)
```

L'instruction `header=TRUE` permet de préciser que la première ligne du fichier contient le nom des colonnes du tableau.

3. Exporter : `write`, `write.table`

37

Importer/exporter des objets R

1. Sauvegarder des objets R dans un fichier

```
> x=1:10
> y=list(a=1, b=TRUE, c="exemple")
> save(x,y, file='sav.rda')
```

2. Lire un fichier qui contient des objets R

```
> load("sav.rda")
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

Attention si un objet R appelé `x` (ou `y`) existait avant l'appel de la fonction `load`, il a été remplacé par celui contenu dans le fichier `sav.rda`

3. `saveRDS` peut aussi être utilisé si on sauvegarde une unique liste. Le fichier est lu avec `readRDS`. On peut changer le nom de la liste à la lecture du fichier

```
> a= list(x=1,y=3)
> saveRDS(a, 'sav.rds')
```

```
> b = readRDS('sav.rds')
> b
$x
[1] 1
```

38

Les fonctions

Structure générale pour créer des fonctions

- La structure générale d'une fonction est

```
>FUN=function(liste_des_paramètres)
{
  commandes
  return(objets_retournés)
}
```

- Les accolades { et } définissent le début et la fin de la fonction.
- La dernière instruction `return` contient le ou les objets retournés par la fonction.
- Exécuter la fonction : `FUN(...)`

39

Exemple

La fonction suivante retourne le résultat de n lancers d'une pièce.

```
PF = function(n ,proba.pile)
{
  #nb aléatoire suivant Unif(0,1)
  u=runif(n)
  pf=(u<proba.pile)
  pf = as.integer(pf)
  return(pf)
}
```

La sortie de la fonction est

```
PF(10,1/2)
[1] 0 0 1 0 1 0 1 1 0 0
# avec affectation de la sortie dans le vecteur x
x = PF(10,1/2)
x
[1] 0 1 1 1 0 0 0 0 0 0
```

40

Renvois multi-arguments

La fonction `return` interdit les sorties avec plusieurs arguments : il faut les regrouper dans un seul objet sous la forme d'une liste.

```
PF = function(n ,proba.pile)
{
  u=runif(n) #nb aléatoire suivant Unif(0,1)
  pf=(u<proba.pile)
  pf = as.integer(pf)
  f =mean(pf)
  return(list(echantillon = pf,frequence = f))
}
```

Exécution de la fonction :

```
PF(4,1/2)
$echantillon
[1] 0 0 1 0
$frequence
[1] 0.25
```

```
l= PF(4,1/2)
l$echantillon
[1] 0 0 1 1
l$f
[1] 0.5
```

41

Paramètres par défaut

On peut affecter des valeurs par défaut aux paramètres d'entrée d'une fonction.

Modification de la fonction PF

par défaut on suppose que la pièce est équilibrée.

```
PF = function(n ,proba.pile=1/2)
{
  u=runif(n) #nb aléatoire suivant Unif(0,1)
  pf=(u<proba.pile)
  pf = as.integer(pf)
  return(pf)
}
```

Les commandes

```
PF(10) OU PF(10,1/2)
```

retournent le même résultat

42

Paramètres d'entrée

Il y a trois façons de spécifier les paramètres d'entrée d'une fonction

- par la position : les paramètres d'entrée sont affectés aux premiers arguments de la fonction. `PF(3,1/2)` : les paramètres d'entrée sont `n=3` et `proba.pile=1/2`
- par le nom : il s'agit du moyen le plus sûr, les noms des arguments sont précisés de manière explicite. On peut alors écrire `PF(proba.pile=1/2 ,n=3)`, l'ordre n'est plus prioritaire
- avec des valeurs par défaut : ces valeurs par défaut seront utilisées si les paramètres d'entrée ne sont pas spécifiés. On peut alors écrire `PF(3)` ou `PF(n=3)` : les paramètres d'entrée sont `n=3` et la valeur par défaut pour `proba.pile` c'est à dire `1/2`.

43

Remarque sur les valeurs par défaut

Modification de la fonction PF :

on inverse l'ordre des paramètres d'entrée :

```
PF = function(proba.pile=1/2, n)
{
  ...
}

PF(10)
Erreur dans .Internal(runif(n, min, max)) : 'n' est manquant
PF(n=10)
[1] 0 0 1 0 1 0 1 0 1 0
```

IL est donc préférable de placer les paramètres sans valeur par défaut en premier dans la déclaration des variables d'entrée.

44

Pour les fonctions de deux variables

- f une fonction de deux variables $f : (x, y) \mapsto f(x, y)$
- x et y deux vecteurs de même dimension.

La commande `f(x,y)` retourne le vecteur constitué des éléments $f(x_i, y_i)$

Si x et y ne sont pas de même dimension, celui de plus petite dimension est répété.

Tableaux croisés

La fonction `outer` retourne une matrice de la forme

$$M(i,j) = fun(x_i, y_j)$$

```
x=1:5
y=1:5
M=outer(x,y,'fun')
```

`fun` peut aussi être une opération élémentaire `+/-*`

45

la fonction Vectorize

Soit f une fonction dont le paramètre d'entrée x est un scalaire.

`Vectorize` transforme la fonction f en une fonction vectorielle c'est à dire une fonction qui évalue la fonction f en chaque point d'un vecteur d'entrée.

Soit $x = (x_1, \dots, x_n)$, on veut évaluer f aux points x_i .

```
# on transforme la fonction {f} en une fonction vectorielle df.
> df = Vectorize (f, 'x')
> y=df(x)
```

Autres programmations possibles

1. avec une boucle `for`

```
> y = rep(0,n)
> for ( i in 1:n) y[i] = f(x[i])
```

2. avec la fonction `sapply`

```
> y = sapply(x, 'f')
```

46

Illustration sur des fonctions de 2 variables

```
> f=function(x,y) sin(x+y^2)
> f(1,1)
[1] 0.9092974
> #
> x = 1:3
> y= 1:3
> # on calcule f(x[i],y[i])
> f(x,y)
[1] 0.9092974 -0.2794155 -0.5365729
> z =1:2
> f(x,z)
[1] 0.9092974 -0.2794155 -0.7568025
Message d'avis :
In x + y^2 :
 la taille d'un objet plus long n'est pas multiple de la taille d
> # le calcul effectué est
> #f(x[1],z[1]) f(x[2],z[2]) f(x[3],z[1])
> #identique à
> f(x,c(z,z[1]))
[1] 0.9092974 -0.2794155 -0.7568025
```

47

suite

```
> f(1,y)
[1] 0.9092974 -0.9589243 -0.5440211
> # identique à
> f(rep(3,1),y)
[1] -0.7568025 0.6569866 -0.5365729
>
> #calcul du tableau croisé f(x[i],y[j])
> df = Vectorize(f,'x')
> df(x,z)
      [,1]      [,2]      [,3]
[1,] 0.9092974 0.1411200 -0.7568025
[2,] -0.9589243 -0.2794155 0.6569866
> # les vecteurs colonnes sont f(1,1:2) f(2,1:2) f(3,1:2)
```

48

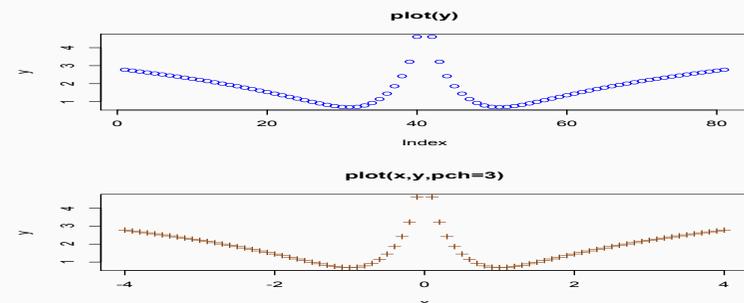
Les graphiques

Les fonctions usuelles plot(), lines(), points()

- plot est la fonction centrale
- Les fonctions points ou lines sont utilisées pour superposer des courbes ou des nuages de points.

Premier exemple : représenter des vecteurs plot(y) / plot(x,y)

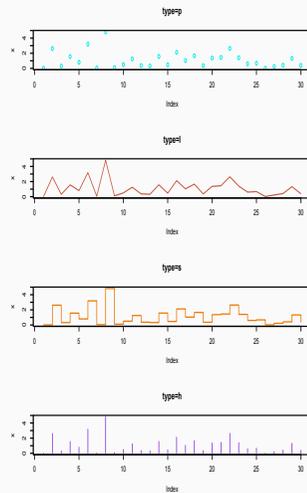
```
x= seq(-4,4,.1) y=log(x^2+1/x^2)
```



49

Quelques arguments de la fonction `plot`

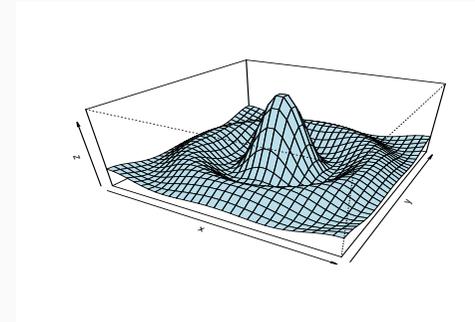
- pour fixer les limites des axes
 - `ylim=c(ay,by)` et `xlim=c(ax,bx)`
 - par défaut les bornes sont optimisées sur la première courbe tracée
- `type="p"` (points) ou `"l"` (ligne) : pour tracer une ligne ou un nuage de points.
- `pch` : type de points
- `lty` : type de lignes.
- `col` : couleur



50

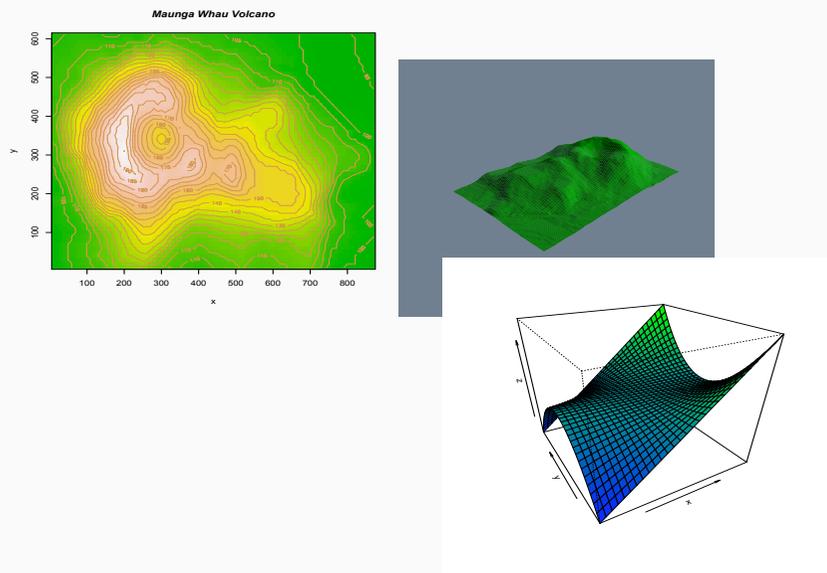
Graphique en 3D

```
x <- seq(-10, 10, length= 30)
y <- x
f <- function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightbl
```



51

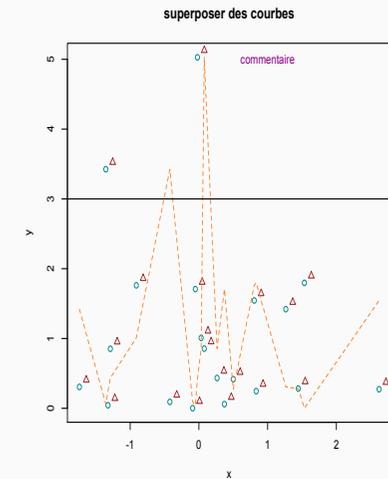
Représentation graphique d'une matrice



52

Superposition de courbes

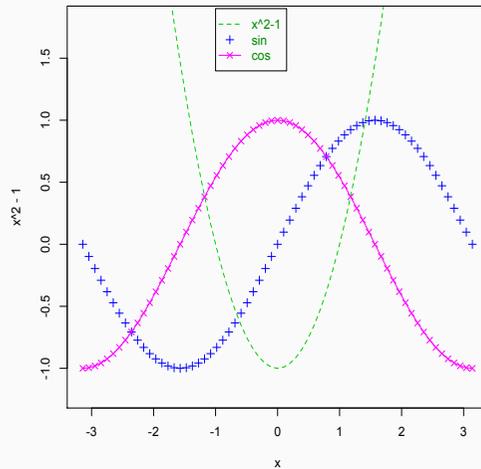
```
x=rnorm(20)
y=exp(20)
#nuage de points
plot(x,y)
#ajouter un nuage de points
points(x+.1,y+.1, pch=2)
#ajouter une ligne
lines(sort(x),y, lty=2)
#ajouter une ligne horizontale
abline(h=3)
#texte + frametitle
text(1,5,"commentaire")
title("superposer des courbes")
```



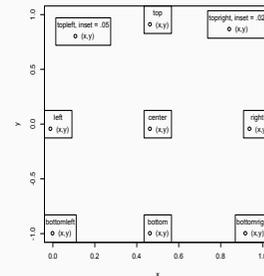
53

légende legend

```
legend(-1, 1.9, c("x^2-1", "sin", "cos"), col = c(3,4,6),  
      lty = c(2, -1, 1), pch = c(-1, 3, 4))
```



les emplacements
prédéfinis :



54

Autour de la fonction plot

Le graphique produit par la fonction `plot(x)` dépend de la classe de l'objet `x`.

```
methods(plot)  
[1] "plot.data.frame" "plot.default"  
[3] "plot.density"    "plot.factor"  
[5] "plot.formula"    "plot.function"  
[7] "plot.histogram"  "plot.lm"  
[9] "plot.mlm"        "plot.mts"  
[11] "plot.new"        "plot.POSIXct"  
[13] "plot.POSIXlt"   "plot.table"  
[15] "plot.ts"         "plot.window"  
[17] "plot.xy"
```

55

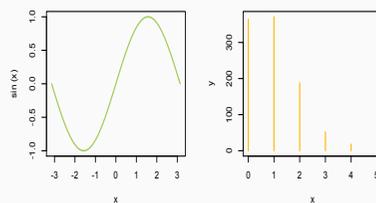
Illustrations

- sur une fonction (par ex `sin`)

```
plot(sin, xlim=c(-pi, pi))
```

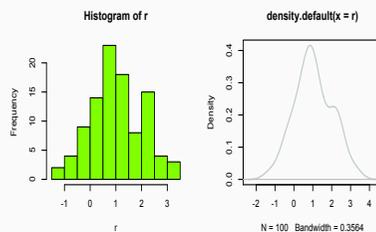
- sur un tableau

```
x=rpois(1000,1)  
y=table(x)  
y  
x 0 1 2 3 4 6  
374 372 162 71 20 1  
plot(y)
```



- sur un histogramme ou une densité

```
r=rnorm(100,1)  
z=hist(r, plot=F)  
plot(z)  
w=density(r)  
plot(w)
```



56

Représentation graphique d'une matrice ou dataframe

Les outils graphiques `matplot` et `pairs` sont adaptés aux matrices dont les colonnes correspondent à des variables.

Exemple

```
>data(iris)  
>iris
```

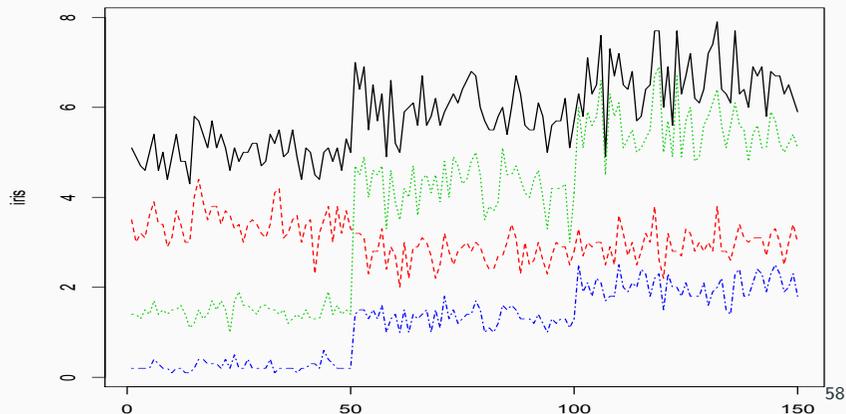
	Sepal.Length	Sepal.Height	Petal.Length	Petal.Height	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4					
...					
54	5.5	2.3	4.0	1.3	versicolor
55	6.5	2.8	4.6	1.5	versicolor
...					
147	6.3	2.5	5.0	1.9	virginica
148	6.5	3.0	5.2	2.0	virginica

57

Fonction `matplot(matrice ...)`

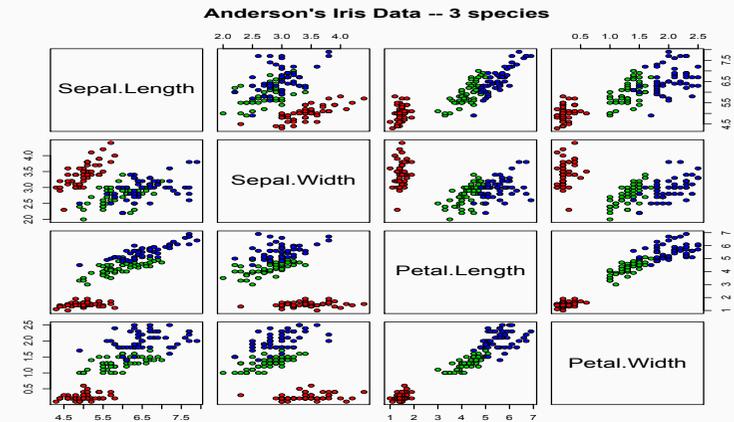
Cette fonction représente sur un même graphique les colonnes d'une matrice ou d'une data.frame.

Anderson's Iris Data – 3 species



Utilisation des biplots , `pairs(matrice , ...)`

Cette fonction représente tous les nuages de points possibles entre les différentes colonnes.



59

Construction d'un polygone

1. On commence par fixer les axes des abscisses et des ordonnées à l'aide d'un graphique vide.

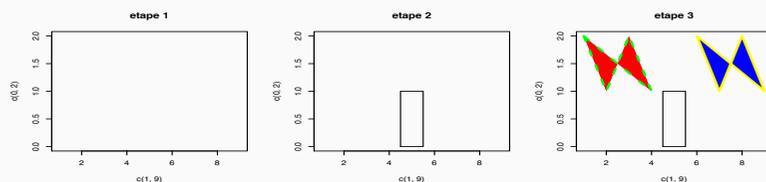
```
plot(c(1,9), c(0,2), type="n")
```

2. Avec la fonction `polygon`, on trace le polygone défini par ses sommets

```
polygon(c(4.5,5.5,5.5,4.5), c(0,0,1,1))
```

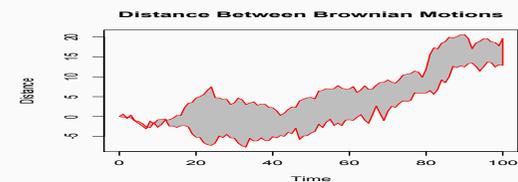
3. Arguments supplémentaires

```
polygon(1:9, c(2,1,2,1,NA,2,1,2,1), col=c("red", "blue"),  
border=c("green", "yellow", lwd=3, lty=c("dashed", "solid"))
```



60

autre exemple



```
n=100  
z.haut=c(0,cumsum(rnorm(n)))  
z.bas= c(0,cumsum(rnorm(n)))  
  
xx <- c(0:n, n:0)  
yy <- c(z.bas, rev(z.haut))  
#graphique vide pour fixer les dimensions  
plot (xx, yy, type="n", xlab="Time", ylab="Distance")  
#tracer le polygone  
polygon(xx, yy, col="gray", border = "red")  
title("Distance Between Brownian Motions")
```

61

- Pour sauvegarder un graphique :
 - utilisation de la fonction `dev.print`.
 - pour obtenir un fichier postscript :
`dev.print(postscript, file="essai.ps")`
 - pour obtenir un fichier pdf :
`dev.print(pdf, file="essai.pdf")`
 - utilisation des menus (sous widows ou mac seulement)
- La fenêtre graphique peut être fractionnée en utilisant
 - `par(mfrow=c(n,m))`, on obtient alors $n \times m$ graphiques sur une même page organisés sur n lignes et m colonnes
 - `split.screen(m,n)`
 - `screen(i)`, `screen(i,FALSE)` pour sélectionner la sous fenêtre
 - `erase.screen()`
 - `close.screen(all = TRUE)`

62

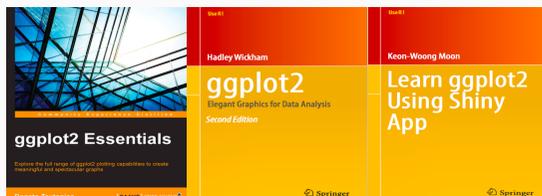
Graphiques avec ggplot2

Description

La librairie `ggplot2` fournit un outil de visualisation des données à l'aide de jolis graphiques

1. La fonction pour produire rapidement des graphiques simples est `qplot` (elle remplace la fonction `plot`)
2. Pour des graphiques plus complexes la fonction a utilisée est `ggplot`
 - Toutes les données doivent être au format `data.frame`
 - les variables sont organisées en colonne.

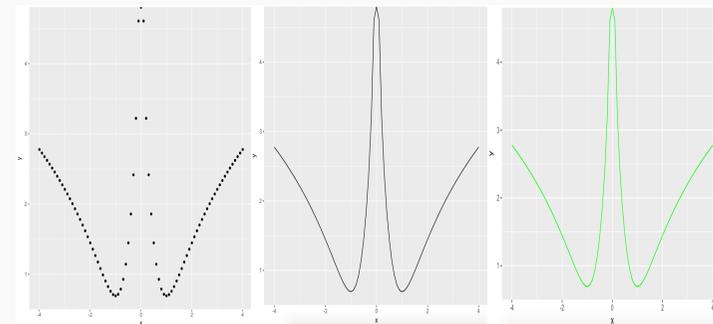
Référence :



63

La fonction `qplot` ou `quickplot`

```
x= seq(-4,4,.1)
y=log(x^2+1/x^2)
qplot(x,y)
qplot(x,y geom="line")
qplot(x,y ,geom="line",colour=l("green"))
```



Représentation par des points (par défaut si deux vecteurs en entrée), par une ligne et par une ligne en modifiant la couleur noire par défaut.

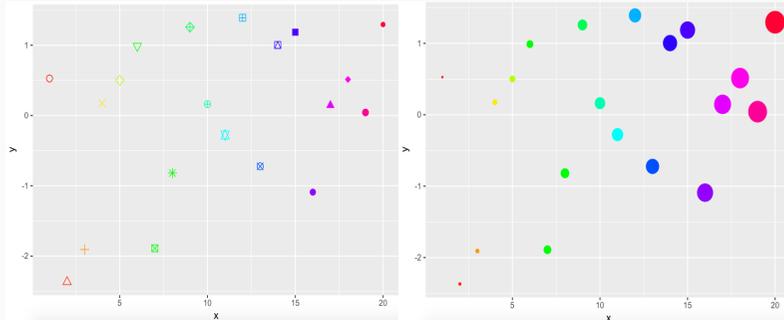
64

Option de la fonction `qplot`

On conserve les options "xlab", "ylab", "xlim" et "ylim".

Pour changer la couleur ou le type de point

```
y = rnorm(20)
x = 1:20
qplot(x,y, shape = I(1:20), colour=I(rainbow(20)))
qplot(x,y, colour=I(rainbow(20)), size=I((1:20)/2))
```



65

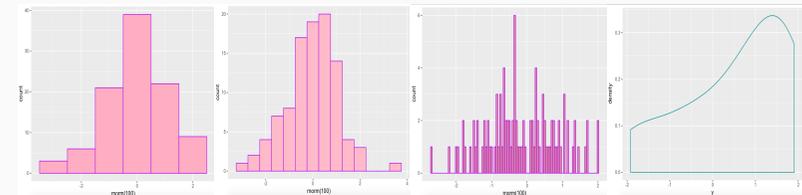
histogramme avec `qplot`

S'il y a un seul vecteur en entrée le graphique par défaut est un histogramme. On contrôle la largeur des classes avec `binwidth`. Par défaut, l'historgramme est dessiné avec 30 classes.

```
x=rnorm(100)
qplot(x, colour=I("purple"), fill =I("pink3"), binwidth=.05)
qplot(x, colour=I("purple"), fill =I("pink"), binwidth=.5)
qplot(x, colour=I("purple"), fill =I("pink1"), binwidth=1)
```

Estimateur à noyau de la densité

```
qplot(y,geom = c("density"), col=I("cyan4"))
```



66

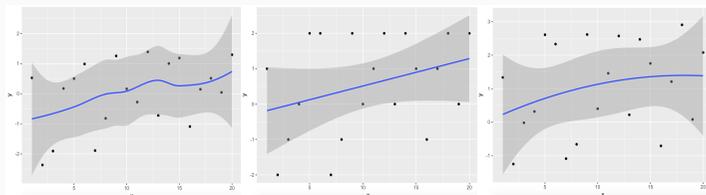
Couplage avec une fonction statistique de lissage

1. Le lissage : par défaut `loess`

```
qplot(x,y,geom = c("point", "smooth"))
```

2. regression linéaire ou polynomiale

```
qplot(x,y, geom = c("point", "smooth"), method="lm")
qplot(x,y, geom = c("point", "smooth"), method="lm",
      formula=y~poly(x,2))
```



67

La fonction `ggplot`

- La fonction `ggplot` est utilisée pour construire l'objet plot initial.
- elle est presque toujours suivi de `+` pour ajouter un composant au tracé.
- La fonction `aes` liste les variables utilisées dans la construction du graphique `x` et `y` ainsi que les variables qui contrôlent l'esthétique du graphique `colour` (`col` dans `plot`) / `shape` (`pch` dans `plot`) / `size` (`cex` dans `plot`)

Il existe différentes façons d'appeler la fonction `ggplot`

1. si toutes les calques utilisent les mêmes données dans l'objet `data.frame df` et les mêmes arguments de style. graphique

```
ggplot(df, aes(x, y, <autres variables>))
```

2. si toutes les tracés utilisent les mêmes données

```
ggplot(df)
```

3. si les calculs utilisent différents jeux de données

```
ggplot()
```

68

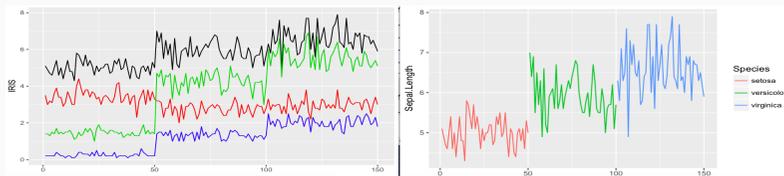
ggplot sur les données data(iris)

- Equivalent du graphique avec `matplot`

```
data(iris)
iris = data.frame(x=1:150, iris)
ggplot(iris) +
  geom_line(aes(x, Sepal.Length), colour = I(1)) +
  geom_line(aes(x, Sepal.Width), colour = I(2)) +
  geom_line(aes(x, Petal.Length), colour = I(3)) +
  geom_line(aes(x, Petal.Width), colour = I(4)) +
  ylab("IRIS") +
  xlab("")
```

- la couleur dépend de la variable `Species`

```
ggplot(iris) +
  geom_line(aes(x, Sepal.Length, colour = Species))
```



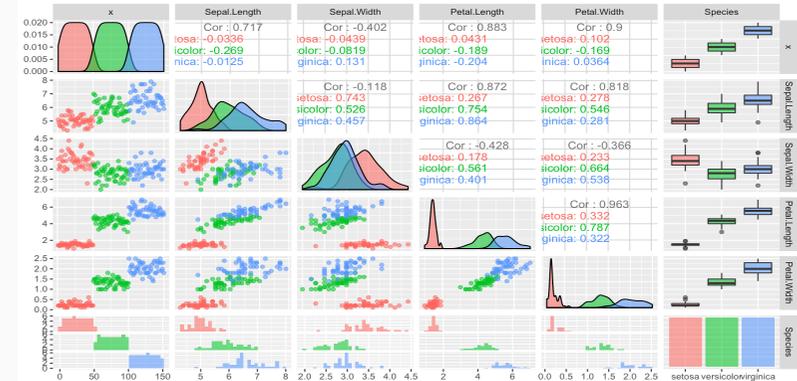
69

Structures de contrôle et Itérations

Autre librairie : (GGally)

Une version sophistiquée de `pairs`

```
ggpairs(iris, aes(colour = Species, alpha = 0.4))
```



70

Instructions conditionnelles

La syntaxe

`if (condition) {instructions}` permet de calculer les instructions uniquement si la condition est vraie.

`if (condition) { A } else {B}` calcule les instructions A si la condition est vraie et les instructions B sinon.

Par exemple,

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

Remarque : Si les instructions se limitent à un seul calcul comme dans cet exemple on peut utiliser la fonction `ifelse`

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

71

Opération non vectorielle

1. Avec la commande `if (condition) ... else ...` la condition ne peut pas être vectorielle

Par exemple

```
> x= 1:7
> if(x>2) print("A") else print("B")
[1] "B"
```

Dans la condition, `x` a une longueur supérieure à 1. Seul le **premier élément** est utilisé :

(`x>2`) correspond à (`x[1] >2`)

2. `ifelse` permet d'appliquer une instruction conditionnelle sur chacune des coordonnées d'un vecteur.

```
> ifelse(x<2,"A","B")
[1] "A" "B" "B" "B" "B" "B" "B"
```

Alternative :

```
> g = function(x) if(x>2) print("A") else print("B")
> vg = Vectorize(g, 'x')
> vg(1:10)
```

72

Itérations

On utilise les boucles pour exécuter plusieurs fois une instruction ou un bloc d'instructions

Les trois types de boucle sont

- `for` :
`for(var in seq) {commandes}`
- `while` :
`while(cond) { commandes}`
- `repeat` :
`repeat { commandes; if (cond) break }`

1. Dans une boucle `for`, le nombre d'itérations est fixé.
2. La durée d'exécution des boucles `while/repeat` peut être infinie!

73

Exemple

Les 3 programmations suivantes retournent le même résultat.

1. Avec l'instruction `for`

```
for ( i in 1:10)
{
  commandes
}
```

2. Avec l'instruction `while`.

```
i= 1
while ( i <= 10 )
{
  commandes
  i = i+1
}
```

3. Avec l'instruction `repeat`

```
i= 1
repeat
{
  commandes
  i = i+1
  if (i>10) break
}
```

74

Exemple

On veut simuler n variables aléatoires suivant la loi de $X_1 + \dots + X_p$ où les X_i sont iid suivant la loi uniforme sur $[0, 1]$.

On stocke l'échantillon dans `Y`.

```
# initialisation de Y avec des 0
Y = rep(0,n)
for ( i in 0:n )
{ Y[i] = sum(runif(p,0,1)) }
```

Une autre programmation :

```
# initialisation de Y avec des 0
Y = rep(0,n)
for( i in 1:p)
{ Y = Y + runif(n,0,1) }
```

Une autre façon de remplir `Y` par concaténation :

```
# Y est un vecteur vide
Y = NULL
for ( i in 0:n )
{
  Y = c(Y, sum(runif(p,0,1)))
}
```

75

Exemple

On simule des variables aléatoires suivant la loi de Bernoulli $B(1/2)$ jusqu'à l'obtention du premier 1.

Le nombre de variables (noté N dans le code ci dessous) simulées suivant la loi de Bernoulli suit une loi géométrique de paramètre $1/2$.

En utilisant `while`

```
x=rbinom(1,1,.5)
N=1
while ( x != 1)
{
x=rbinom(1,1,.5)
N=N+1
}
```

En utilisant `repeat`

```
N=0
repeat {
x=rbinom(1,1,.5)
N=N+1
if (x==1) break
}
```

76

Exemple

On dispose des températures moyennes mensuelles relevées à Nottingham pendant 10 ans de 1920 à 1939.

```
> nottem
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov
Dec
1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
etc
> nottem=matrix(nottem, ncol=12,byrow=T)
```

On souhaite calculer un profil moyen annuel et le stocker dans le vecteur `temp`.

```
temp = rep(0,12)
for (i in 1:12) temp[i] = mean(nottem[,i])
```

On peut aussi initialiser le vecteur `temp` comme le vecteur vide, puis on le remplit en concaténant les résultats

```
temp = NULL
for (i in 1:12) temp = c(temp, mean(nottem[,i]))
```

77

La fonction `apply`

```
apply(matrice, MARGIN, FUN, ARG.COMMUN)
```

La fonction `apply()` permet d'appliquer la même fonction `FUN` sur toutes les lignes (`MARGIN=1`) ou les colonnes (`MARGIN=2`) d'une matrice `MAT`

- Chaque ligne (`MARGIN=1`) ou les colonnes (`MARGIN=2`) est affectée au premier paramètre de la fonction `FUN`
- La syntaxe de `FUN` est `FUN(x, ARG.COMMUN)` où
 - `x` est un vecteur
 - `ARG.COMMUN` représente éventuellement des paramètres supplémentaires qui sont communs à toutes les exécutions.

Cette fonction remplace une boucle sur le nombre de colonnes ou de lignes

78

Exemple d'utilisation de la fonction `apply`

Retour à l'exemple des températures

Calcul du profil annuel :

```
> temp = apply(nottem,2,mean) # moyenne sur les colonnes
[1] 39.695 39.190 42.195 46.290 52.560 ...
```

Calcul des moyennes annuelles

```
> temp.annuelle = apply(nottem,1,mean) # moyenne sur les lignes
```

79

Supprimer une boucle en utilisant `apply`

Objectif : Estimer la distribution de la médiane empirique

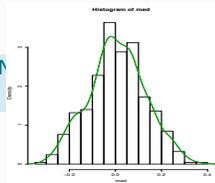
- On simule N échantillons de taille n iid suivant la loi gaussienne standard.
- Pour chaque échantillon, on calcule et on stocke la valeur de la médiane empirique dans le vecteur `med`.

1. avec boucle

```
n =100
N=500
med = 1:N
for (i in 1:N) med[i] = quantile(rnorm(n,0,1), probs=1/2)
```

2. Sans boucle

```
Alea = matrix(rnorm(n*N,0,1), ncol=N)
med = apply(Alea, 2, quantile, probs=1/2)
```



80

`replicate`

La fonction `replicate` évalue n fois une même expression

[utile l'expression implique des nombres aléatoires!!].

La syntaxe : `replicate(n, expression)`

Exemples

```
> replicate(10,1+1)
[1] 2 2 2 2 2 2 2 2 2 2

> x=rnorm(10)
> replicate(5,mean(x))
[1] -0.5073058 -0.5073058 -0.5073058 -0.5073058 -0.5073058

> replicate(5,mean(rnorm(10)))
[1] 0.22137912 0.09330663 -0.12511439 -0.02613061 -0.19182371
```

81

Suite de l'exemple sur la médiane empirique

La fonction `replicate` permet de programmer sans boucle le code suivant

```
n =100
N=500
med = 1:N
for (i in 1:N) med[i] = quantile(rnorm(n,0,1), probs=1/2)
```

La syntaxe est la suivante

```
med = replicate(500, quantile(rnorm(100,0,1), probs=1/2))
```

82

`sapply,lapply`

Ces fonctions calculent la même fonction sur tous les éléments d'un vecteur ou d'une liste.

La syntaxe :

`lapply(X,FUN, ARG.COMMUN)`

- La fonction `lapply` applique une fonction `FUN` à tous les éléments du vecteur ou de la liste `X`.
- Les valeurs de `X` sont affectées au premier argument de la fonction `FUN`.
- Si la fonction `FUN` a plusieurs paramètres d'entrée, ils sont spécifiés après le nom de la fonction : `ARG.COMMUN`
- Cette fonction retourne le résultat sous la forme de listes.
- `sapply` est une fonction similaire à `lapply` mais le résultat est retourné sous forme de vecteurs, si possible.

83

Exemple d'utilisation de lapply

`cars` est une liste constituée de deux vecteurs `speed` et `dist`

On calcule la moyenne et les quantiles des deux composantes de la liste.

```
> cars
  speed dist
1     4    2
2     4   10
3     7    4
> lapply(cars, mean)
$speed      $dist
[1] 15.4      [1] 42.98
> lapply(cars, quantile, probs = (0:4)/4)
$speed
 0% 25% 50% 75% 100%
 4  12  15  19  25
$dist
 0% 25% 50% 75% 100%
 2  26  36  56 120
```

84

Supprimer une boucle à l'aide de la fonction `sapply`

Il est très souvent possible de supprimer les boucles `for` en utilisant `lapply` ou `sapply`.

Soit $x = (x_1, \dots, x_t)$ une série représentant l'évolution du prix d'une action sur une durée de t jours.

On veut calculer M_i le prix maximum sur la période 1 à i pour $i = 1, \dots, t$.

- avec une boucle

```
M = 1:t
for ( i in 1:t )
{
M[i] = max(x[1:i])
}
```

- sans boucle

```
FUN.max = fonction (n,y) max(y[1:n ])
```

85

La fonction `tapply`

Le type `factor` est un objet vectoriel qui permet de spécifier une classification discrète (nombre fini de groupes).

La fonction `tapply` applique une fonction `FUN` sur les sous groupes d'un vecteur `X` définis par une variable de type `factor` `GRP`

`tapply(X,GRP,FUN,...)`

```
> note
[1] 10.83676 11.63757 11.07312 13.79699 10.84186
10.72562 13.58680 14.85070 13.15659 14.36744
> xgr
[1] "a" "b" "a" "b" "a" "a" "b" "a" "b" "b"
> gr=factor(xgr)
> gr
[1] a b a b a a b a b b
Levels: a b
> tapply(note, gr, mean)
  a      b
11.66561 13.30908
```

86

Evaluer le temps de calcul

`proc.time` évalue combien de temps réel et CPU (en secondes) le processus en cours d'exécution a déjà pris.

```
> ptm <- proc.time()
> for ( i in 1:50) mean(runif(1000))
> proc.time() - ptm
utilisateur      système      écoulé
 0.016          0.001      0.027
```

87

Comparaison de trois programmations for/replicate/apply

```
#boucle
> x=1:M
> for (i in 1:M)
x[i] = mean(runif(n))
#replicate
> x=replicate(M,mean(runif(n)))
#apply
> Z=matrix(runif(n*M) , ncol=M)
> x=apply(Z , 2, mean)
```

```
(n= 10^3 , M= 10^4)
      user  system elapsed
wfor  0.680   0.051   0.760
wrep  0.618   0.018   0.632
wapp  1.132   0.157   1.286

(n= 10^2 , M= 10^5)
      user  system elapsed
wfor  3.160   0.239   3.451
wrep  2.522   0.066   2.611
wapp  3.031   0.253   3.456

(n= 10 , M= 10^6)
      user  system elapsed
wfor 32.077   0.419  43.655
wrep 37.219   0.481  41.147
wapp 29.911   0.392  30.357
```

88

Autour des lois de probabilités

Généralités

Soit X une variable aléatoire de loi P_X

$$P_X(A) = P(X \in A) = \begin{cases} \sum_{x \in A} P(X = x) & \text{loi discrète} \\ \int_A f(x) dx & \text{loi continue} \end{cases}$$

Pour les lois classiques, des fonctions existent pour

- calculer
 - la densité $\begin{cases} P(X = x) & \text{pour les lois discrètes} \\ f(x) & \text{pour les lois continues} \end{cases}$
 - la fonction de répartition $F(x) = P(X \leq x)$
 - les quantiles $F^{-1}(u) = \inf\{x : F(x) \geq u\}^1$
- simuler des nombres aléatoires suivant la même loi que X .

1. Si F est une bijection alors $F^{-1} = F^{-1}$

89

Quelques lois disponibles

1. Lois discrètes

- Loi binomiale (n, p) binom
- Loi hypergéométrique (N, n, k) hyper
- Loi de Poisson (a) pois
- Loi géométrique (p) geom
- Loi à support fini $\{(a_i, p_i), i = 1 \dots m\}$ sample

2. Lois continues

- Loi Gaussienne (m, σ^2) norm
- Loi uniforme sur $[a, b]$ unif
- Loi de Student à ν degrés de liberté t
- Loi du χ^2 à ν degrés de liberté chisq

90

Si `www` représente le nom d'une des lois alors

- `dwww(x,...)` calcule la densité de la loi `www` au point `x`
- `pwww(x,...)` : calcule la fonction de répartition au point `x`
- `qwww(α,...)` : calcule le quantile d'ordre `α`
- `rwww(n,...)` retourne un échantillon de taille `n`

les ... représentent les paramètres spécifiques à chaque loi.

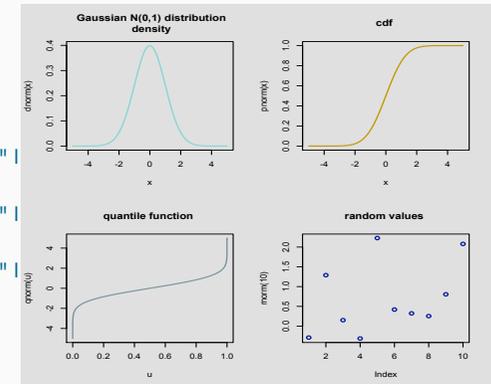
91

Exemple : loi gaussienne de moyenne 0 et de variance 1

- `dnorm(x,0,1)` : densité au point `x`
- `pnorm(x,0,1)` : fonction de répartition au point `x`
- `qnorm(α,0,1)` : quantile d'ordre `α`
- `rnorm(n,0,1)` : échantillon de taille `n`

```
x=seq(-5,5,.01)
u=seq(0,1,.01)

plot(x,dnorm(x,0,1), type="l")
plot(x,pnorm(x,0,1), type="l")
plot(u,qnorm(u,0,1), type="l")
plot(rnorm(10,0,1))
```



92

Simulation d'une loi discrète finie

Soit Z une variable aléatoire à valeurs dans $\{x_1, \dots, x_k\}$ telle que

$$p(Z = x_i) = prob_i$$

pour tout $i = 1 \dots k$

- La fonction `sample` est un générateur de nombres aléatoires pour les lois discrètes à support fini.
- La syntaxe est `sample(x, n, replace=TRUE, prob)` où
 - `x` contient les valeurs prises par Z
 - `prob` contient les valeurs des probabilités.lorsque l'option `prob` n'est pas spécifié, par défaut c'est la loi uniforme. Lorsque `replace=TRUE` indique que le tirage est avec remise. Autrement dit on simule une réalisation de n variables aléatoires indépendantes et de même loi

93

Simuler le résultat d'un dé à 6 faces

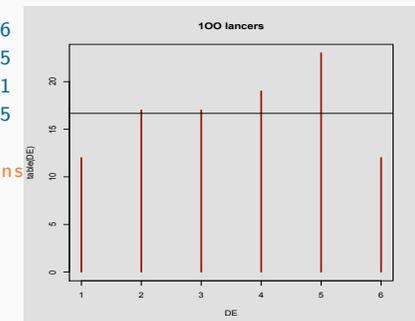
On simule suivant la loi uniforme sur l'ensemble $\{1, \dots, 6\}$.

On simule $n = 100$ réalisations de façon indépendante

```
# Les réalisations
> DE
[1] 3 2 5 3 4 5 4 4 2 4 2 5 2 6
[24] 3 5 4 4 5 5 4 4 4 3 5 5 2 5
[47] 2 5 6 2 1 5 1 6 2 1 2 3 5 1
[70] 6 4 1 3 4 5 6 1 6 3 5 4 1 5
[93] 6 4 2 5 5 1 5 4

# calcul du nombre de 1,...,6 dans
# l'échantillon simulé :
> table(DE)
DE
1 2 3 4 5 6
12 17 17 19 23 12

# représentation de la répartition
> plot(table(DE))
```



94

Construire une fonction càdlàg constante par morceaux

La fonction `stepfun` retourne un objet de type `function` qui définit une fonction constante par morceaux et continue à droite.

La syntaxe est

```
F=stepfun(x, z)
```

où

- le vecteur `x` contient les points de discontinuité de la fonction,
- le vecteur `z` est de la forme `c(a, y)` où `a` est la valeur prise par `F` avant le premier point de discontinuité et `y` les valeurs de la fonction aux points de discontinuités `x`.

Pour représenter graphique la fonction `F` on utilise `plot` ou `lines` (pour superposer) : avec l'argument `vertical=FALSE`

```
> plot(F, vertical=FALSE) ou lines(F, vertical=FALSE)
```

Application : les fonctions de répartition des lois discrètes

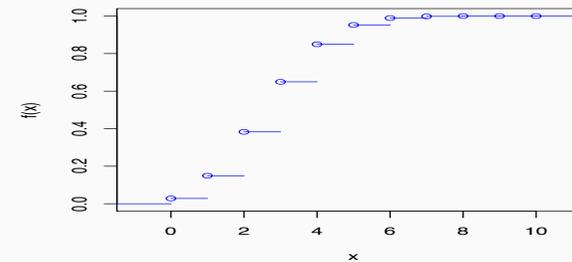
95

Exemple : loi binomiale

On trace la fonction de répartition de la loi binomiale $B(10, 1/2)$

```
points.discont = 0:10
F.points.discont = pbinom(points.discont, 10, 0.3)
```

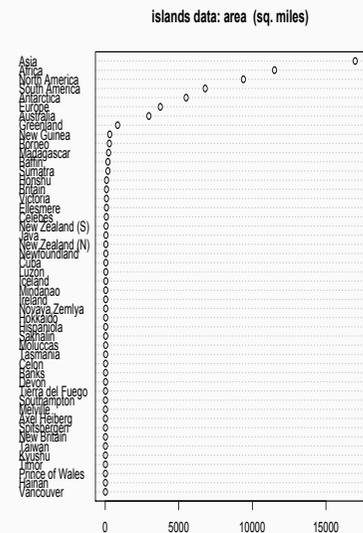
```
F = stepfun( points.discont , c(0,F.points.discont) )
plot(F, vertical =FALSE ,col=4,main="fct rep loi binomiale" ,ylab="
```



96

Outils graphiques en statistique

Les données Island

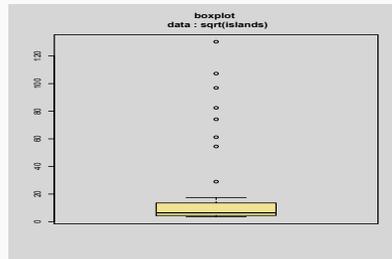


Africa	Antarctica	Asia
11506	5500	16988
Australia	Axel Heiberg	Baffin
2968	16	184
Banks	Borneo	Britain
23	280	84
Celebes	Celon	Cuba
73	25	43
Devon	Ellesmere	Europe
21	82	3745
Greenland	Hainan	Hispaniola
840	13	30
Hokkaido	Honshu	Iceland
30	89	40
Ireland	Java	Kyushu
33	49	14
Luzon	Madagascar	Melville
42	227	16
Mindanao	Moluccas	New Britain
36	29	15
New Guinea	New Zealand (N)	New Zealand (S)
306	44	58
Newfoundland	North America	Novaya Zemlya
43	9390	32
Prince of Wales	Sakhalin	South America
13	29	6795
Southampton	Spitsbergen	Sumatra
16	15	183
Taiwan	Tasmania	Tierra del Fuego
14	26	19
Timor	Vancouver	Victoria
13	12	82

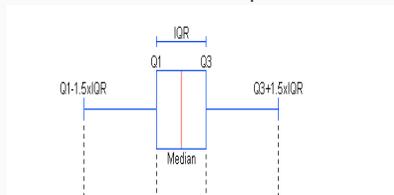
97

Statistique descriptive sur les données `islands`

```
> mean(x)
[1] 3.734162
> var(x)
[1] 9.134685
> quantile(x, c(.25, .5, .75))
 25%    50%    75%
1.794741 2.990066 4.326417
  Q1      mediane    Q3
> boxplot(x)
```



Construction du boxplot :



Les points représentent les valeurs aberrantes. Elles sont définies comme les observations en dehors de l'intervalle

$$I = [Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)].$$

98

Questions :

Comment représenter graphiquement l'échantillon d'une variable

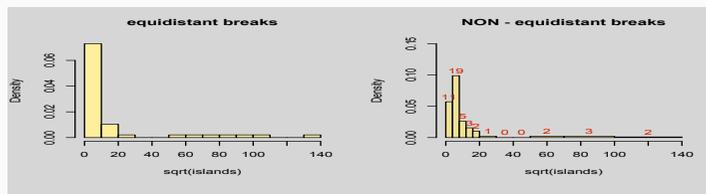
- continue
- discrète
- qualitative ?

Comment représenter un tableau de données numériques ?

99

Histogramme

```
hist(sqrt(islands), breaks = 12)
hist(sqrt(islands), breaks = c(4*0:5, 10*3:5, 70, 100, 140))
```



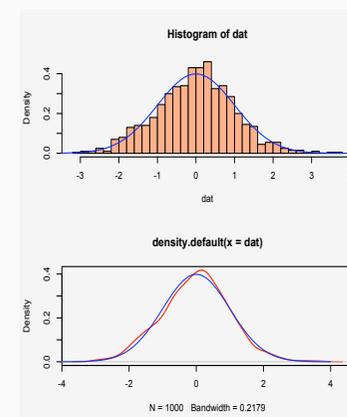
les options

L'argument `breaks` fixe le nombre ou les bornes des classes. On peut aussi utiliser `nclass` pour fixer le nombre de classes.

100

Estimation de la densité pour des lois continues

On compare sur des données simulées suivant la loi gaussienne deux estimateurs de la densité (`hist` / `density`) avec la densité théorique de la loi



```
par(mfrow=c(1,2))
x=rnorm(10000)
y=density(x)
plot(y)
hist(x, proba=T)
lines(y)
z=seq(min(x), max(x), 0.01)
#superpose densité théorique
lines(z, dnorm(z, 0, 1), lty=2)
```

Remarque sur la fonction `hist`
`proba =T` : l'aire en dessous de la courbe est égale à 1. Par défaut `proba =F` : l'aire est égale au nombre d'observations.

101

Variables qualitatives

On relève pendant un mois la météo à Cental Park. La variable est à valeurs dans {clear, partly.cloudy, cloudy }.

```
> central.park.cloud
 [1] partly.cloudy partly.cloudy partly.cloudy clear
 [5] partly.cloudy partly.cloudy clear         cloudy
 [9] partly.cloudy clear         cloudy         partly.cloudy
[13] cloudy         cloudy         clear         partly.cloudy
[17] partly.cloudy clear         clear         clear
```

Pour résumer l'information : on calcule la répartition (ou la fréquence) des modalités dans l'échantillon.

```
> table(central.park.cloud)
central.park.cloud
      clear partly.cloudy      cloudy
         11          11          9
```

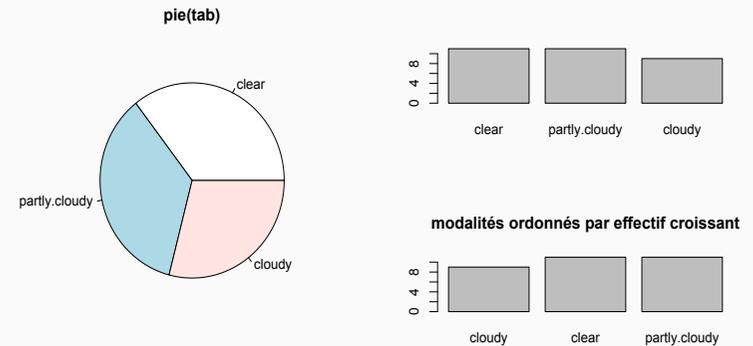
102

Représentation graphique d'une variable quantitative

```
tab=table(central.park.cloud)
pie(tab)
barplot(tab)
```

Comment ordonner les modalités?
en général, on les classe suivant l'ordre croissant des effectifs.

```
barplot(sort(tab))
```



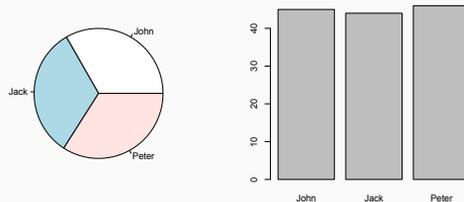
103

Remarque sur le choix de la plage des ordonnées (ylim)

On relève le salaire de
3 personnes

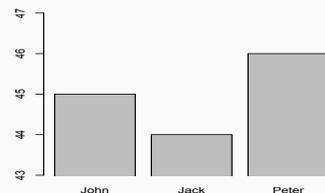
```
> sales
John Jack Peter
 45  44  46
```

Représentation graphique
pie(sales)
barplot(sales)



Modification de l'axe des ordonnées pour améliorer la lisibilité du graphique :

```
barplot(sales, ylim=c(40,47), xpd=F)
```



104

Variables discrètes

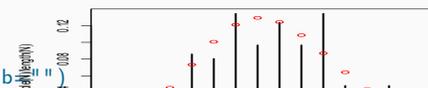
On relève N le nombre d'appels téléphoniques reçus par un central pendant un mois.

```
> N [1] 6 16 3 6 8 10 9 9 10
etc
> t = table(N)
N
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16 17
 2  4  3
 7 16 15 25 18 23 18 25
 9  6  9  4  2
>
```



Représentation de la distribution et comparaison avec la densité de la loi de Poisson de paramètre la moyenne empirique de l'échantillon (points en rouge).

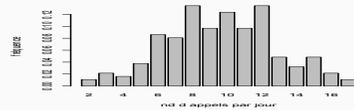
```
J= 2:17
plot(table(N)/length(N), ylab="densité")
```



105

La fonction `barplot` pour des variables quantitatives

```
> barplot(t/length(N))
```



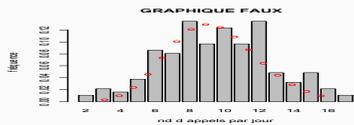
Attention : Les labels des barres ne coïncident pas avec la coordonnée sur l'axe des abscisses.

Illustration : on veut ajouter sur graphique précédent la densité de la loi de Poisson de paramètre égal à la moyenne empirique de l'échantillon (points en rouge), on obtient

```
J= 2:17
```

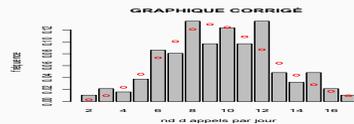
```
# GRAPHIQUE FAUX
```

```
barplot(table(N)/length(N))
points(J, dpois(J, mean(N)))
```



```
# GRAPHIQUE CORRIGÉ
```

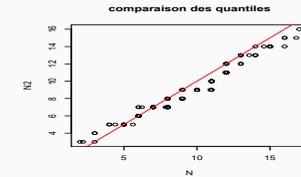
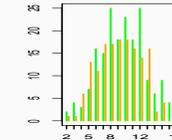
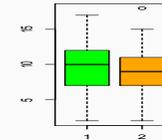
```
br = barplot(table(N)/length(N))
points(br, dpois(J, mean(N)))
```



106

Comparaison de 2 variables quantitatives indépendantes

On compare le nombre d'appels par mois de deux standards



```
boxplot(N,N2, col=c("green", "orange"))
```

```
t=table(N2)
```

```
xt = as.integer(names(t))+.3
```

```
plot(table(N), col="green", lwd=2, xlab="", ylab="")
```

```
lines(xt, t, col="orange", type="h", lwd=2)
```

```
qqplot(N,N2)
```

```
abline(0,1)
```

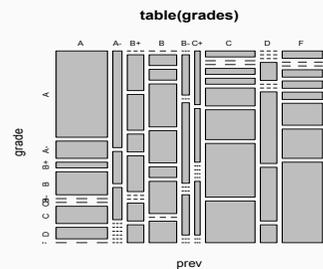
Pour les variables aléatoires continues, on compare les estimations de la densité : histogrammes, estimateur à noyau.

107

Lien entre des variables qualitatives dépendantes

Données : pour 122 étudiants, on dispose de la note obtenue cette année (A+ ...F) et celle de l'année précédente

prev	grade
1	B+ B+
2	A- A-
3	B+ A-
4	F F
5	F F
6	A B
7	A A
8	C D



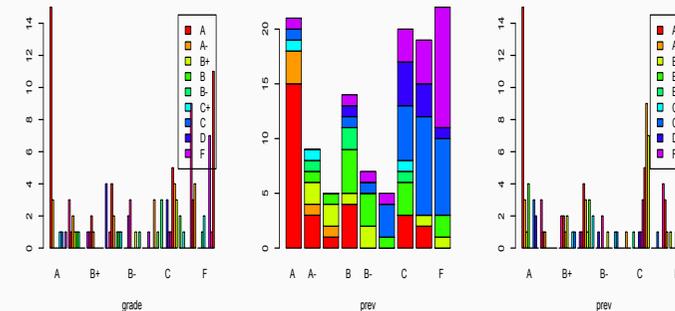
```
> table(grades)
```

prev	grade	A	A-	B+	B	B-	C+	C	D	F
A		15	3	1	4	0	0	3	2	0
A-		3	1	1	0	0	0	0	0	0
B+		0	2	2	1	2	0	0	1	1
B		0	1	1	4	3	1	3	0	2
B-		0	1	0	2	0	0	1	0	0
C+		1	1	0	0	0	0	1	0	0
C		1	0	0	1	1	3	5	9	7
D		0	0	0	1	0	0	4	3	1
F		1	0	0	1	1	1	3	4	11

```
> plot(table(grades))
```

108

Autres graphiques pour résumer la table de contingence



```
> barplot(tab, xlab="grade", legend.text=T, col=rainbow(10), beside
```

```
> barplot(t(tab), xlab="prev", col=rainbow(10))
```

```
> barplot(t(tab), xlab="prev", legend.text=T, col=rainbow(10), beside
```

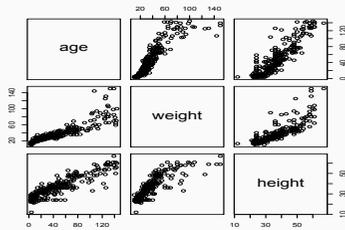
109

Utilisations des couleurs

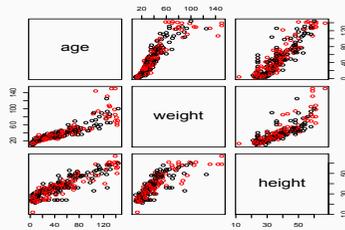
Données : pour 150 enfants, on relève les informations suivantes : age / poids / taille /sexe

```
> kid.weights
  age weight height gender
1  58   38   38      M
2 103   87   43      M
3  87   50   48      M
4 138   98   61      M
5  82   47   47      F
```

Pour visualiser le lien entre les variables : on utilise la fonction `pairs` sur les 3 premières colonnes. On peut ajouter en option la couleur des points définies à partir de la variable `gender`



```
pairs(X[,1:3])
```



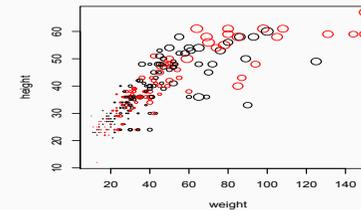
```
pairs(X[,1:3], col=X[,4])
```

110

Inférence statistique

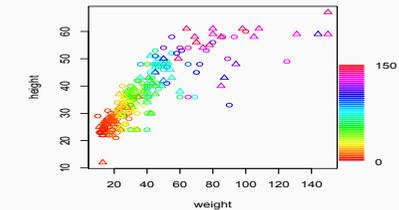
Utilisation des couleurs, de la taille et du type de marques

On représente la taille en fonction du poids.



La couleur des points est définie par la variable `gender`

La taille des points est proportionnelle à l'âge des individus.



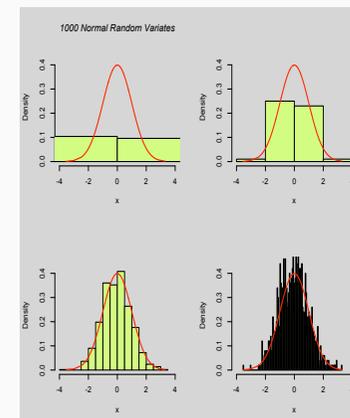
Le type de points est définie par la variable `gender`

Le dégradé de couleurs est défini à partir de la variable `age`.

```
plot(weight,height, cex=age/max(age)*2, col=as.integer(gender))
plot(weight,height, col=rainbow(150)[age], pch=as.integer(gender))
```

111

histogramme : le choix du nombre de classes



1. On simule un échantillon suivant la loi gaussienne de taille 1000
2. On trace l'histogramme pour différentes valeurs du nombre de classes.
3. On compare l'histogramme avec la densité théorique de loi gaussienne (*courbe en rouge*)

`hist(x, nclass=***, proba=T)` ou `hist(x, proba = T)` par défaut le nombre de classes est optimisé pour des échantillons gaussiens

112

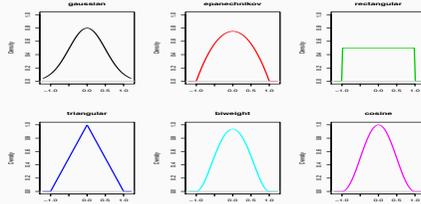
Précision sur la fonction density

Soit X_1, \dots, X_n n variables aléatoires i.i.d. suivant la loi de densité f .

La fonction density calcule l'estimateur de f suivant

$$\hat{f}_n(x) = \frac{1}{n} \sum_{k=1}^n \frac{1}{bw_n} \text{kern} \left(\frac{x - X_k}{bw_n} \right)$$

- le choix du noyau kern



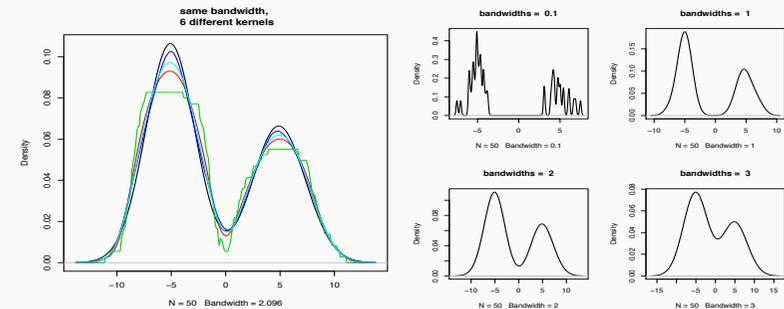
- la dimension de la fenêtre bw. Par défaut le paramètre est optimisé pour un échantillon gaussien

113

Illustration sur un mélange gaussien

On teste la fonction density sur des données simulées² suivant un mélange de lois gaussiennes :

$$f(x) = \frac{1}{4} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-5)^2} + \frac{3}{4} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x+5)^2}$$



2. Simulation des données $w=rbinom(50,1,1/4)$
 $sample = w*rnorm(50,5) + (1-w) * rnorm(50,-5)$

114

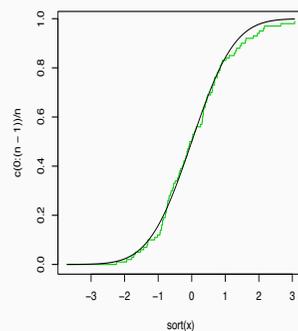
Fonction de répartition empirique (ecdf)

La fonction de répartition empirique est définie par

$$\hat{F}_n(t) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{]-\infty, t]}(X_i) \xrightarrow[p.s.]{n \rightarrow \infty} F(t)$$

C'est un estimateur de la fonction de répartition.

```
> x= rnorm(100)
> Fn=ecdf(x)
> plot(Fn, col="green")
> z=seq(min(x), max(x), 0.01)
> lines(z, pnorm(z), col=1, lwd=2)
```



115

Estimation de la moyenne : comportement asymptotique.

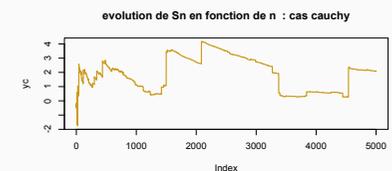
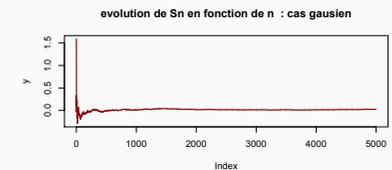
Soit X_1, \dots, X_n une suite de variables aléatoires iid, si $E|X_1| < \infty$ alors

$$S_n = \frac{1}{n} \sum_{i=1}^n X_i \rightarrow E(X_1)$$

n=5000

```
#loi gaussienne
x=rnorm(n,0,1)
y=cumsum(x)/(1:n)
plot(y, type='l')
```

```
# loi de
Cauchy ou Student(1ddl)
xc=rt(n,1)
yc=cumsum(xc)/(1:n)
plot(yc, type='l')
```



116

Les fonctions pour les tests statistiques classiques :

```
t.test(x,mu=5,alt="two.sided") #student
t.test(x,y,alt="less",conf=.95)
```

```
var.test(x,y) #comparaison variance
cor.test(x,y) #non correlation
chisq.test(x,y) #indépendance
Box.test(z,lag=1) #non correlation
```

```
shapiro.test(x) #normalité
ks.test(x,"pnorm") #normalité K-S
ks.test(x,y) #même distribution
```

117

Exemple : Test de Student `t.test()`

X_1, \dots, X_n iid $\mathcal{N}(1,1)$ et Y_1, \dots, Y_m iid $\mathcal{E}(1)$
Test $H_0 : E(X) = E(Y)$ vs $H_1 : E(X) \neq E(Y)$

```
> x = rnorm(100,1,1)
> y = rexp(200,1)
> t.test(x,y)

Welch Two Sample t-test
data: x and y
t = -0.2178, df = 178.446, p-value = 0.8278
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.2648092 0.2121608
sample estimates: mean of x : 0.9544127 mean of y : 0.9807369
```

118

Test d'ajustement

On construit un générateur de nombres aléatoires suivant la loi normale en utilisant le Théorème Central Limite appliqué à des variables aléatoires iid suivant la loi uniforme sur $(0,1)$.

Méthode de simulation non exacte

$U = 1, \dots, U_n$ iid suivant la loi uniforme

$$\sqrt{\frac{n}{12}}(\bar{U}_n - \frac{1}{2}) \Rightarrow X \sim N(0,1) \quad \bar{U}_n = \frac{1}{n} \sum_{i=1}^n U_i$$

La générateur s'écrit

```
simU<-function(taille,size)
{
y = matrix(runif(taille*size),ncol=size)
(apply(y,1,mean)-1/2)*sqrt(taille/12)
}
```

119

Validité de l'algorithme

Pour différentes valeurs de $n = 1, 5, 10$

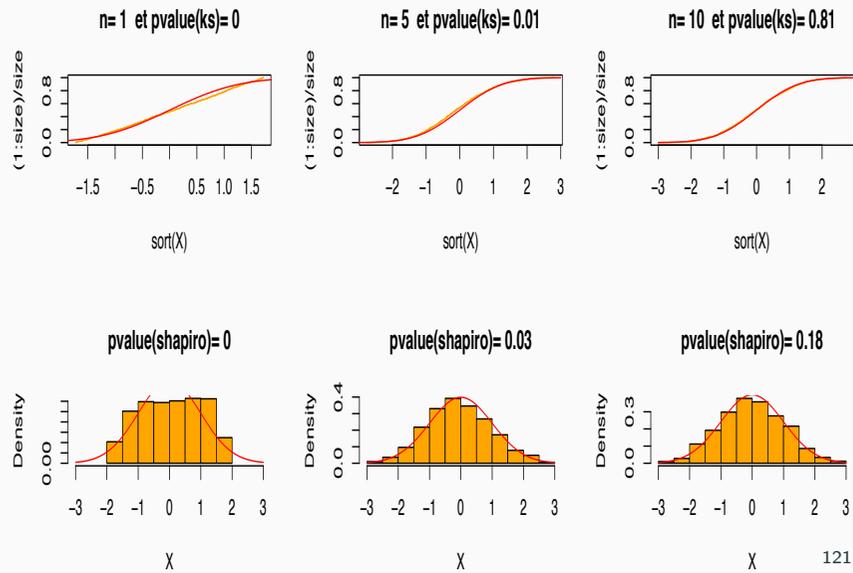
1. On teste la normalité à l'aide de deux procédures de test, le test de Kolmogorov et celui de Shapiro.
2. On compare la fonction de répartition empirique et la fonction de répartition théorique.
3. On compare l'estimation de la densité par un histogramme et la densité théorique.

```
X = simU(1000,n)
test1=ks.test(X,"pnorm",0,1)$p.value
test2 = shapiro.test(X)$p.value

plot(sort(X),(1:size)/size,type="s",col="orange")
lines(seq(-3,3,.1),pnorm(seq(-3,3,.1)),col="red")
title(paste("n=",n," et pvalue(ks)=",floor(c(test1)*100)/100))
hist(X,col="orange",xlim=c(-3,3),proba=T,main="")
lines(seq(-3,3,.1),dnorm(seq(-3,3,.1)),col="red")
title(paste("pvalue(shapiro)=",floor(c(test2)*100)/100))
```

120

Les résultats



121

Régression linéaire

Le modèle le plus simple

$$y = ax + b + \epsilon$$

Pour réaliser une régression linéaire, par la méthode des moindres carrés, on utilise la fonction `lm`.

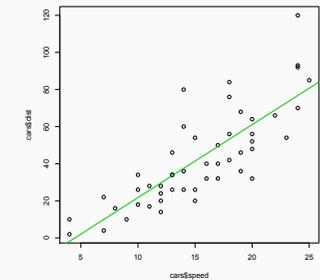
Sur les données `cars`

```

> lm(cars$dist ~ cars$speed)
Call:
lm(formula = cars$dist ~ cars$speed)

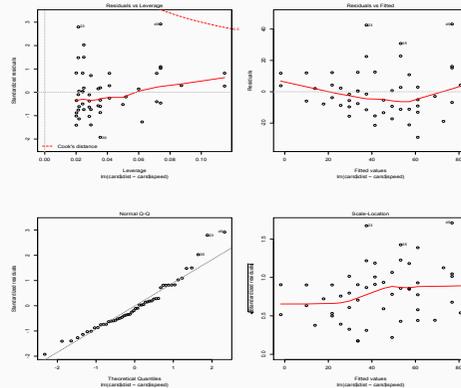
Coefficients:
(Intercept)  cars$speed
 -17.579      3.932

> plot(cars$speed, cars$dist)
> abline(reg, col=3, lwd = 2)
  
```



122

Visualisation : plot.lm



Remarque : Si les données sont disponibles sous la forme d'une liste on peut utiliser la syntaxe

```
> fit = lm(dist ~ speed, cars)
```

123

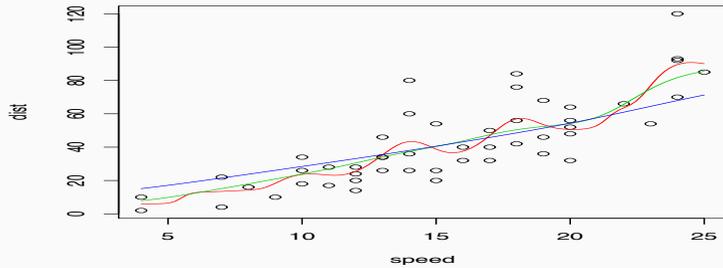
Extension

- Régression multiple `lm(v ~ v1 + v2 + v3)`
- Régression linéaire généralisée `glm`
- etc
- Méthodes non paramétriques

124

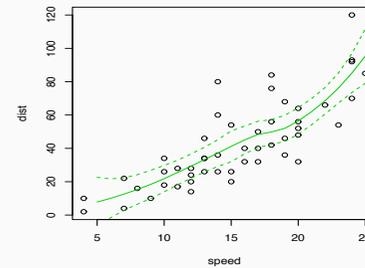
Régression non paramétrique

```
>data(cars)
>attach(cars)
>plot(speed, dist)
>lines(ksmooth(speed, dist, "normal", bandwidth=2), col=2)
>lines(ksmooth(speed, dist, "normal", bandwidth=5), col=3)
>lines(ksmooth(speed, dist, "normal", bandwidth=10), col=4)
```



125

Polynômes Locaux



```
>data(cars)
>cars.lo = loess(dist ~ speed, cars)
>p = predict(cars.lo)
>plot(cars)
>lines(seq(5, 30, 1), p$fit, col=3)
```

126

Séries Chronologiques

L'objet "série chronologique" est une liste qui contient

- les valeurs observées,
- la fréquence des observations,
- la date de la première observation
- la date de la dernière, etc...

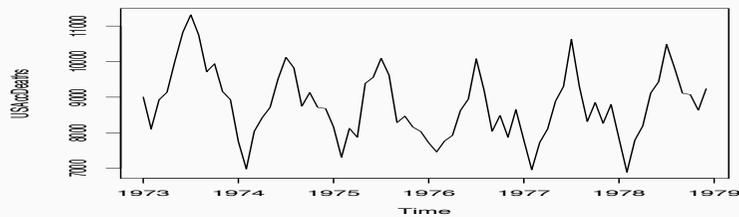
127

Exemple

```
> data(USAccDeaths)
> USAccDeaths
```

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1973	9007	8106	8928	9137	10017	10826	11317	10744	9713	9938	9161	8927
1974	7750	6981	8038	8422	8714	9512	10120	9823	8743	9129	8710	8680
1975	8162	7306	8124	7870	9387	9556	10093	9620	8285	8466	8160	8034
1976	7717	7461	7767	7925	8623	8945	10078	9179	8037	8488	7874	8647
1977	7792	6957	7726	8106	8890	9299	10625	9302	8314	8850	8265	8796
1978	7836	6892	7791	8192	9115	9434	10484	9827	9110	9070	8633	9240

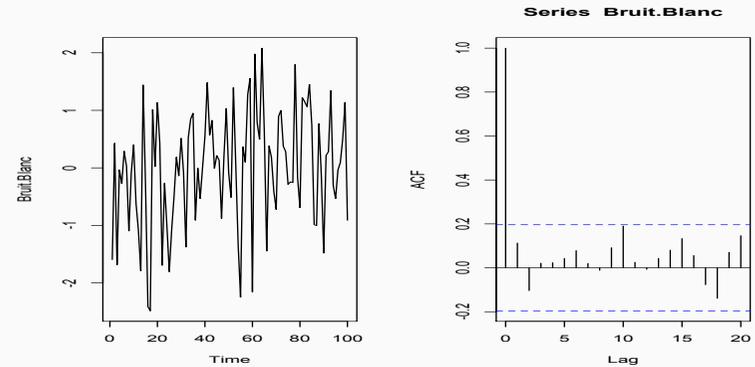
```
> plot(USAccDeaths)
```



128

Pour créer une série chronologique, on utilise la fonction `ts`

```
> bruit.blanc=ts(rnorm(100), frequency = 1, start = c(1), end=c(10))
> plot(bruit.blanc)
> acf(bruit.blanc, type="correlation")
#ou type="covariance" ou type="partial"
```



129

Filtre

- Premier exemple de filtre : $(I - L^p)^d$, on utilise la fonction `diff`

```
> diff(x, lag=p, differences=d)
```

- La fonction `filter` permet d'appliquer des filtres linéaires :

```
y=filter(x, sides= 1, method = "convolution", filter=c(2,3))
# y[i] = 2*x[i] + 3*x[i-1]
y=filter(x, sides= 2, method = "convolution", filter=c(2,3,4))
# y[i] = 2*x[i-1] + 3*x[i] + 4*x[i+1]
y=filter(x, method = "recursive", filter=c(2,3))
#y[i] = x[i] + 2*y[i-1] + 3*y[i-2]
```

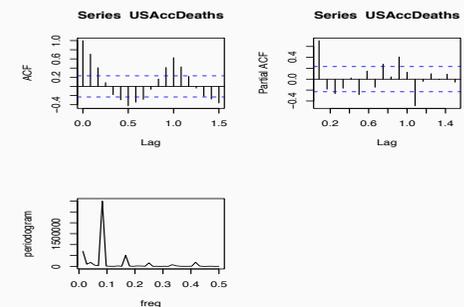
Ces fonctions permettent en particulier de créer un générateur de processus ARMA.

130

Étude préliminaire de la série USAccDeaths

```
periodogram = fonction(traj)
{
  n = length(traj)
  freq = 1:(n %/% 2)/(n)
  periodogram = Mod(fft(traj))[2:(n %/% 2 + 1)]^
    2/(2 * pi * n)
  plot(freq, periodogram, type = "l")
}
```

```
> acf(USAccDeaths)
> pacf(USAccDeaths)
> periodogram(USAccDeaths)
```



131

modélisation et prévision AR

La fonction `ar` permet d'estimer les paramètres d'un modèle AR

$$A_p(L)X[t] = e[t]$$

Si l'ordre p n'est pas précisé, le meilleur modèle AR pour le critère AIC est sélectionné.

```
>data(lh)
>ar(lh)
Call: ar(x = lh)
Coefficients:
      1      2      3
0.6534 -0.0636 -0.2269
Order selected 3  sigma^2 estimated as 0.1959
```

132

```
>ar(lh, aic = FALSE, order.max = 4) # on fixe p=4
Call:
ar(x = lh, aic = FALSE, order.max = 4)
Coefficients:
      1      2      3      4
0.6767 -0.0571 -0.2941  0.1028
Order selected 4  sigma^2 estimated as 0.1983
```

ATTENTION x doit être une série chronologique ($x=ts(x)$)

133

Autour des modèles ARMA

- `ARMAacf` pour le calcul des covariances théorique d'un modèle ARMA
- `ARMAtoMA` pour le développement en MA infinie d'un modèle ARMA
- `arima.sim` pour simuler des trajectoires d'un modèle ARMA ou ARIMA

134

Modèle ARMA : Estimation

- La fonction `ar` permet d'estimer les paramètres d'un processus AR.
- Pour les modèles ARMA d'ordre (p,q)

$$X[t] = a[1]X[t-1] + \dots + a[p]X[t-p] + e[t] + b[1]e[t-1] + \dots + b[q]e[t-q]$$

on utilise la fonction `arima`, la syntaxe est

```
out=arima(x, order=c(p,0,q))
```

la sortie `out` est une liste contenant :

`out$coef` : estimation des coefficients,

`out$resid` : estimation des résidus $e[t]$

ATTENTION x doit être une série chronologique ($x=ts(x)$)

135

modélisation et prévision SARIMA

Plus généralement, la fonction `arima` permet d'estimer les paramètres d'un modèle SARIMA

$$A_p(L)\alpha_P(L^s)Y[t] = \beta_Q(L^s)B_q(L)e[t] \text{ avec } Y[t] = (I - L)^d(I - L^s)^D X[t]$$

la syntaxe est la suivante :

```
out=arima(x, order=c(p,d,q),
          seasonal=list(order=c(P,D,Q), period=s))
```

la sortie est une liste contenant :

`out$coef` : estimation des coefficients,
`out$aic` : critère AIC,
`out$resid` : estimation des résidus $e[t]$
option : `include.mean=F` ou `T`

136

Exemple

```
data(USAccDeaths)
a = c(USAccDeaths)
USAcc= ts(a[1:60], frequency=12, start=c(1973,1))
```

```
fit = arima(USAcc, order=c(0,1,1), seasonal=list(order=c(0,1,1)))
```

Call:

```
arima(x = USAcc, order = c(0, 1, 1),
      seasonal = list(order = c(0,1, 1)))
```

Coefficients:

```
      ma1      sma1
-0.4343  -0.4419
```

Approx standard errors:

```
      ma1      sma1
0.1368   0.0122
```

```
sigma^2 estimated 114276: log likelihood = -341.73, aic = 687.46
```

137

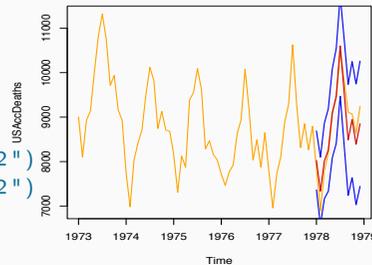
Prévision pour des modèles SARIMA

Pour prévoir à l'horizon h , on utilise la fonction `predict`

```
>out=arima0(...)
> p=predict(out,h)
p$pred #contient les prévisions
p$se #erreurs de prévision ( écart type )
```

Exemple :

```
p = predict(fit, n.ahead=12)
plot(USAccDeaths, col="orange")
lines(p$pred, col="red3")
lines(p$pred+1.96*p$se, col="blue2")
lines(p$pred-1.96*p$se, col="blue2")
```



138

Plus généralement : la fonction `predict`

```
> methods(predict)
[1] "predict.ar"           "predict.arima"
[3] "predict.loess"        "predict.ppr"
[5] "predict.smooth.spline" "predict.smooth.spline.fit"
[7] "predict.glm"          "predict.lm"
[9] "predict.mlm"
```

139

Lissage exponentiel

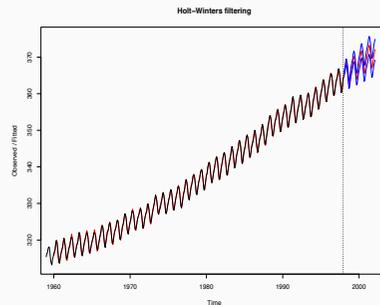
La méthode de lissage de Holt & Winter est disponible sous R. Les fonctions s'appelle `HoltWinters` et `predict.HoltWinters`.

Exemple : sur la série `data(co2)`

```
data(co2)
m = HoltWinters(co2) #lissage
p = predict(m, 50, prediction.interval = TRUE) # prevision
plot(m, p)
```

cette fonction peut aussi être utilisée pour

- des modèles multiplicatifs
- le lissage simple si l'on impose les coefficients.



140

La librairie `forecast`

Cette librairie développée par Rob J Hyndman contient des méthodes pour l'analyse et la prévision de séries temporelles univariées

1. le lissage exponentiel, Holt Winter, ...
2. Modèles BATS (Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components)
3. modélisation automatique ARIMA.
4. modélisation automatique SARIMA .

Par défaut cette librairie utilise le traitement parallèle pour accélérer les calculs dans la sélection automatique des modèles .

141