

Formation – Module 2

Nicolas Bez, Jérôme Guitton,

Pascal Neveu *, Emmanuel Paradis

Sète, 27–29 novembre 2007

Programme

	Mardi 27	Mercredi 28	Jeudi 29
9:00–12:00	Rappel sur R (EP) Programmation R (PN)	Interface R et bases de données (JG, PN)	Données spatiales et cartographie (NB, JG)
13:30–17:00	Manipulation de chaînes de caractères (EP) Graphiques avancés (EP)	Modèles linéaires (EP)	Modèles mixtes, méthodes multivariées (EP) <i>questions ouvertes</i>

I Rappel sur R

La structure des données dans R

R ne manipule que des données en mémoire vive (RAM).

Les lectures et écritures de fichiers de données doivent être faites explicitement.

La structure de base des données est le **vecteur** (*atomic object*).

Les informations supplémentaires sont stockées dans les **attributs**.

Tous les objets ont deux attributs intrinsèques : le **mode** et la **longueur**.

```
> x <- c("B", "R")
> mode(x); length(x)
[1] "character"
[1] 2
```

Les autres structures de données sont créées soit en ajoutant des attributs (facteur, matrice, 'Date', 'ts', ...), soit en assemblant des objets (tableau, liste).

```
> attributes(x)
```

```
NULL
```

```
> f <- factor(x)
```

```
> f
```

```
[1] B R
```

```
Levels: B R
```

```
> attributes(f)
```

```
$levels
```

```
[1] "B" "R"
```

```
$class
```

```
[1] "factor"
```

```
> mode(f)
```

```
[1] "numeric"
```

```
> str(f)
```

```
Factor w/ 2 levels "B","R": 1 2
```

La **classe** est un attribut qui va affecter l'action des fonctions **génériques** : `print`, `summary`, `plot`, ...

```
> print
function (x, ...)
UseMethod("print")
<environment: namespace:base>
> print.factor
function (x, quote = FALSE, max.levels = NULL,
         width = getOption("width"), ...)
{
  ord <- is.ordered(x)
  if (length(x) <= 0)
    cat(if (ord)
        "ordered"
        ....
```

`print.factor` est une *method* dans le jargon de R.

```
> z <- as.Date(as.character(2001:2005), "%Y")
```

```
> z
```

```
[1] "2001-11-15" "2002-11-15" "2003-11-15" "2004-11-15"  
[5] "2005-11-15"
```

```
> str(z)
```

```
Class 'Date' num [1:5] 11637 12002 12367 12733 13098
```

```
> attributes(z)
```

```
$class
```

```
[1] "Date"
```

```
> str(co2)
```

```
Time-Series [1:468] from 1959 to 1998: 315 316 316 318 318 ...
```

```
> mode(co2)
```

```
[1] "numeric"
```

```
> attributes(co2)
$ts
[1] 1959.000 1997.917 12.000

$class
[1] "ts"

> plot(co2)
```


La plupart des fonctions faisant une analyse dans R retournent un objet de classe du même nom :

```
> class(lm(1 ~ 1))
[1] "lm"
> class(aov(1 ~ 1))
[1] "aov" "lm"
```

Utilisation des fonctions de R

Les arguments sont passés par leurs positions ou par leur noms :

```
> args(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE,
         dimnames = NULL)
NULL
> matrix(1:6, 3) # == matrix(1:6, nrow = 3)
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

 En pratique, le passage de plusieurs arguments par position est à éviter pour les fonctions avec beaucoup d'options (`plot`, `read.table`, `legend`, ...).

Dans certaines fonctions, les arguments ne peuvent pas être passés par position :
save, rm, cat ...

```
> args(cat)
function (... , file = "", sep = " ", fill = FALSE,
          labels = NULL, append = FALSE)
> cat(1:10, "titi.txt")
1 2 3 4 5 6 7 8 9 10 titi.txt>
```

Si le nom de l'argument est abrégé de façon non-ambigüe, une correspondance est faite :

```
> matrix(1:6, nc = 3)
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6, n = 3)
Error in matrix(1:6, n = 3) :
  argument 2 matches multiple formal arguments
```

R recycle largement les arguments :

```
> data.frame(1:3, "a")
```

```
  X1.3 X.a.
```

```
1     1     a
```

```
2     2     a
```

```
3     3     a
```

```
> 1:6 / 10 # == 1:6 / rep(10, 6)
```

```
[1] 0.1 0.2 0.3 0.4 0.5 0.6
```

```
> plot(rnorm(100), pch = 1:2)
```

```
> points(1, rnorm(100)) # idem avec plot
```

```
Error in xy.coords(x, y) : 'x' and 'y' lengths differ
```

```
> segments(50, -2, 0:100, 2)
```

L'indexation logique recycle les indices, mais pas l'indexation numérique :

```
> x[1] # != x[c(1, 1)]  
[1] "B"  
> x[TRUE] # == x[c(TRUE, TRUE)]  
[1] "B" "R"
```

Les données sont souvent converties implicitement (coercion) :

```
> c(1, "a")  
[1] "1" "a"  
> if (0) print("OK")  
>
```

Les « packages »

```
> sessionInfo()
```

```
R version 2.6.0 (2007-10-03)
```

```
i486-pc-linux-gnu
```

```
locale:
```

```
C
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets
```

```
[6] methods   base
```

```
loaded via a namespace (and not attached):
```

```
[1] rcompgen_0.1-15
```

```
> search()
[1] ".GlobalEnv"      "package:stats"
[3] "package:graphics" "package:grDevices"
[5] "package:utils"    "package:datasets"
[7] "package:methods" "Autoloads"
[9] "package:base"

> library()
> library(lattice)
> library(help = lattice) # ou help.start() -> HTML

> install.packages("plotrix")
> update.packages()
```

<http://bg9.imslab.co.jp/Rhelp/>

II Manipulation des chaînes de caractères

- Gestion des 'names', 'dimnames', 'levels', ...
- Annotation de figures : série 1, série 2, ...
- « post-processing » de fichiers non conformes :

1 1 ou 2 2	→ read.table →	"1" "1 ou 2" "2"
------------------	----------------	------------------------

```
as.numeric(gsub("ou.+"," ", x))
```

- Gérer les expressions :

```
> e <- parse(text = "1 + 2")
```

```
> e
```

```
expression(1 + 2)
```

```
attr(,"srcfile")
```

```
<text>
```

```
> eval(e)
```

```
[1] 3
```

Couper et coller des chaînes

```
> args(paste)
function (..., sep = " ", collapse = NULL)
NULL
> paste("site", 1:3)
[1] "site 1" "site 2" "site 3"
> paste("site", 1:3, sep = "")
[1] "site1" "site2" "site3"
> paste(letters[24:26], 1:3, sep = " < ")
[1] "x < 1" "y < 2" "z < 3"
> paste(1:5, "x", letters[1:2], sep = "+")
[1] "1+x+a" "2+x+b" "3+x+a" "4+x+b" "5+x+a"
> paste(LETTERS, collapse = "")
[1] "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
> paste(1:5, collapse = "-")
[1] "1-2-3-4-5"
> paste("x", 1:5, sep = "*", collapse = "; ")
[1] "x*1; x*2; x*3; x*4; x*5"
```

```
> x <- c("1", "1 ou 2", "2")
> strsplit(x, "") # pas de défaut pour 2nd argument
[[1]]
[1] "1"

[[2]]
[1] "1" " " "o" "u" " " "2"

[[3]]
[1] "2"
> strsplit(x, "ou")
[[1]]
[1] "1"

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

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




```
> strsplit(x, " ou ")
[[1]]
[1] "1"

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

[[3]]
[1] "2"


> unlist(strsplit(x, " ou "))
[1] "1" "1" "2" "2"
```

 Le second argument de `strsplit` est interprété, par défaut, comme une ***expression régulière*** (*regexp*).


Les expressions régulières

Kit de survie (tous les détails : `?regex`) :

<code>.</code>	nimporte quel caractère
<code>[azerty]</code>	l'un des caractères entre crochets
<code>[a-h]</code> ou <code>[0-9]</code>	idem que <code>[abcdefgh]</code> ou <code>[0123456789]</code>
<code>a{6}</code>	idem que <code>aaaaaa</code>
<code>a{n,}</code>	le caractère en question n fois ou plus
<code>a+</code>	idem que <code>a{1,}</code>
<code>a{n,m}</code>	le caractère en question n à m fois
<code>^aze</code>	au début de la chaîne
<code>rtty\$</code>	à la fin de la chaîne

 Les caractères `{ }` `[]` `+` `^` `$` `\` font partie de la syntaxe des *regex* et doivent être précédés de `\\`.

Note : `[A-Za-z]` signifie « une lettre (majuscule ou minuscule) », alors que `[A-Z][a-z]` signifie « une majuscule suivie d'une minuscule ».

 Le caractère ‘\’ étant lui-même doublé s’il est présent dans une chaîne, il sera quadruplé dans la *regex* correspondante :

```
> strsplit("C:\\Program Files", "C:\\")
Error in strsplit("C:\\Program Files", "C:\\") :
  invalid split pattern 'C:\\'
> strsplit("C:\\Program Files", "C:\\\\")
[[1]]
[1] ""          "Program Files"
```

Note : "\\ ", "\t" ou "\n" est un caractère unique :

```
> strsplit("new\nline", "\n")
[[1]]
[1] "new"  "line"
> nchar("\n")
[1] 1
```

```
> strsplit("abcde", "bd")
[[1]]
[1] "abcde"
> strsplit("abcde", "[bd]")
[[1]]
[1] "a" "c" "e"
> strsplit("abcde", "[b-d]")
[[1]]
[1] "a" "" "" "e"
> strsplit("abcde", c("b", "c", "d")) # == strsplit("abcde", "b")
[[1]]
[1] "a" "cde"
> strsplit("abcde", "bcd")
[[1]]
[1] "a" "e"
> strsplit("abcde", "b-d")
[[1]]
[1] "abcde"
```

```
> strsplit("a[bcd]e", "\\[bcd\\]")
```

```
[[1]]
```

```
[1] "a" "e"
```

```
> strsplit("a[bcd]e", "\\[[b-d]{3}\\]") # 27 combinaisons
```

```
[[1]]
```

```
[1] "a" "e"
```

```
> # utile avec: strsplit(X, "\\[.+\\]")
```

```
> strsplit("a[bcd]e", "[bcd]", fixed = TRUE)
```

```
[[1]]
```

```
[1] "a" "e"
```

Remplacer des caractères

```
> sub("a", "?", c("aaa", "aba"))
[1] "?aa" "?ba"
> gsub("a", "?", c("aaa", "aba"))
[1] "???" "?b?"
```

On notera les options `fixed` et `ignore.case` (FALSE par défaut les deux).

```
> nms_form <- c("Nicolas Bez", "Jérôme Guitton",
+ "Pascal Neveu", "Emmanuel Paradis")
> gsub("[a-zéô]+", ".", nms_form)
[1] "N. Bez"      "J. Guitton"  "P. Neveu"    "E. Paradis"
> sub("[a-zéô]+", ".", nms_form)
[1] "N. Bez"      "J. Guitton"  "P. Neveu"    "E. Paradis"
```

```
> as.Date("15 jan 2001", "%d %b %Y")
[1] "2001-01-15"
> as.Date("15 janvier 2001", "%d %B %Y")
[1] NA
> x <- gsub("janvier", "01", "15 janvier 2001")
> x
[1] "15 01 2001"
> as.Date(x, "%d %m %Y")
[1] "2001-01-15"
```

Des fonctions plus simples à utiliser (sans *regexp*) :

```
> x <- "AAACGGTTCGTA"
> tolower(x)
[1] "aaacggttcgta"
> toupper(tolower(x))
[1] "AAACGGTTCGTA"
> chartr("AGCT", "TCGA", x)
[1] "TTTGCCAAGCAT"
```

Avec `chartr`, les deux premiers arguments doivent avoir le même nombre de caractères.

`substr` sert à extraire ou modifier une sous-chaîne dans un vecteur de mode caractère :

```
> substr(x, 1, 2)
[1] "AA"
> substr(x, 1, 2) <- "NN"
> x
[1] "NNACGGTTCGTA"
```


Recherche de motifs

```
> grep("Emmanuel", nms_form)
[1] 4
> grep("Emmanuel", nms_form, value = TRUE)
[1] "Emmanuel Paradis"
> grep("P", nms_form)
[1] 3 4
> grep("^P", nms_form)
[1] 3
```

Exemple : recherche de 'names' (applicable également aux 'dimnames' bien sûr) :

```
> x <- 1:3
> names(x) <- c("Pan paniscus", "Pan troglodytes",
+              "Homo sapiens")
> x["^Pan "]
<NA>
NA
```

```
> sel <- grep("^Pan ", names(x))
> x[sel]
      Pan paniscus Pan troglodytes
              1              2
```

Note : `apropos` et `ls` supportent aussi les expressions régulières (la seconde via l'option `pattern`).

Exercices II

1. Faites afficher la liste des fonctions dont le nom n'est composé que d'un caractère.
2. À l'aide de la fonction `scan`, lire le fichier 'Mammal_lifehistories_v2.txt' dans un vecteur de mode caractère tel que chaque ligne corresponde à un élément de ce vecteur. Quels sont les n^o des lignes se rapportant aux Mustelidae ? Enregistrer les données de cette famille dans un fichier séparé.
3. Concevoir un code qui changera dans un vecteur de mode caractère les noms de mois dans le n^o correspondant. On procédera progressivement :
 - (a) considérer uniquement "janvier" et "février" ;
 - (b) inclure la possibilité que ces mois soient abrégés ;
 - (c) inclure la possibilité que les mois soient en anglais ;
 - (d) résoudre le problème de la casse (majuscules/minuscules) ;
 - (e) généraliser aux douze mois.

III Graphiques avancés

Distinguer plusieurs séries sur un même graphe à l'aide d'un ou plusieurs facteurs.

```
> x <- iris[, 1]; y <- iris[, 2]; g <- iris$Species
> co <- c("blue", "red", "yellow")
> plot(x, y, col = co[g])
> co <- c("blue", "red", "red")
> psym <- c(1, 1, 19)
> plot(x, y, col = co[g], pch = psym[g])
> legend(6.5, 4.5, levels(g), pch = psym, col = co)
```

Le même principe peut être utilisé avec `font` et/ou `cex`.

Représenter les points avec des noms :

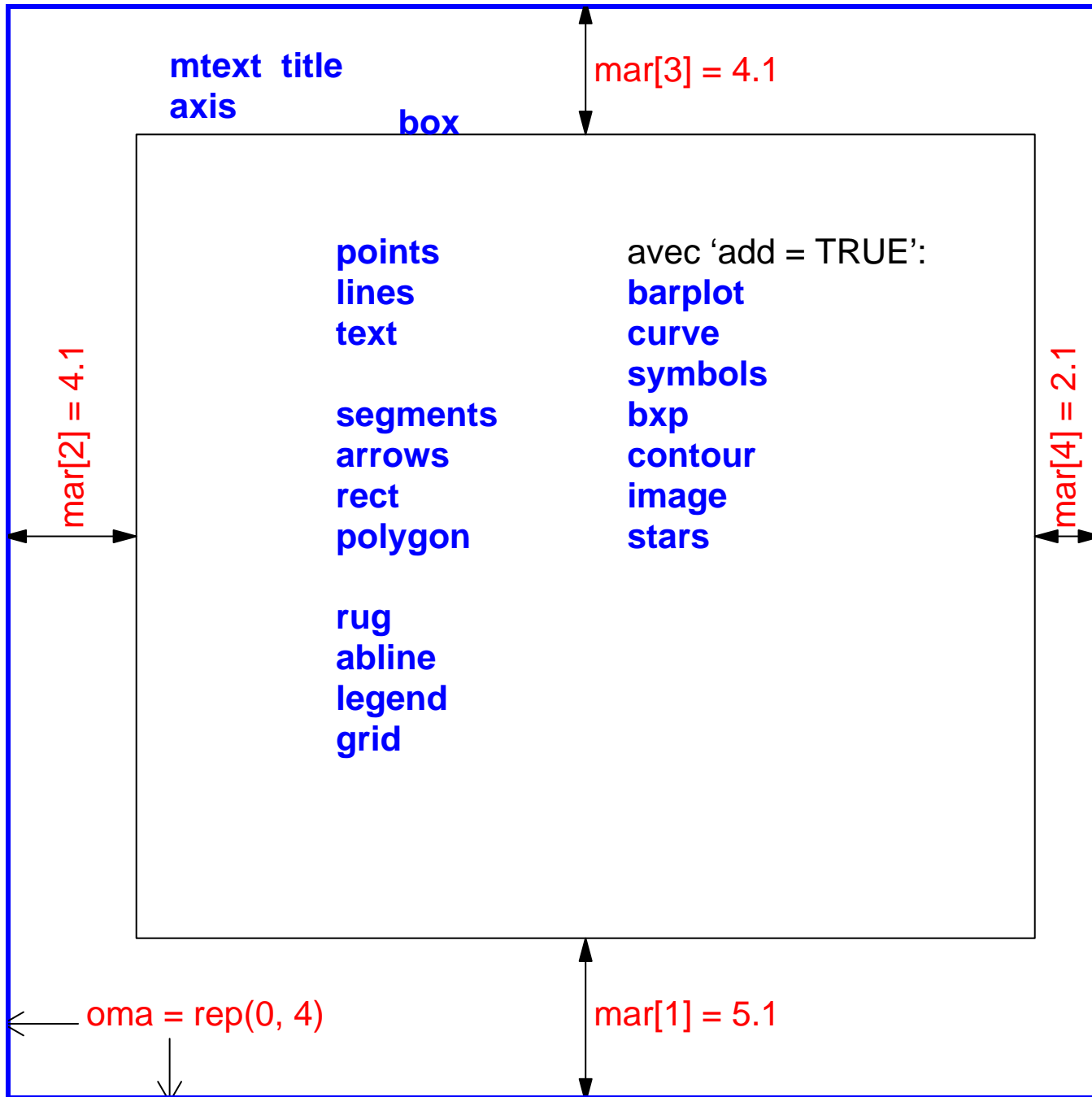
```
> plot(x, y, "n")  
> text(x, y, as.character(g))
```

Application typique :

```
plot(DF$V1, DF$V2, "n")  
text(DF$V1, DF$V2, rownames(DF))
```

Annotations de graphes

Vue d'ensemble des fonctions graphiques secondaires :



Exemple : polygon

```
> args(polygon)
function (x, y = NULL, density = NULL, angle = 45,
  border = NULL, col = NA, lty = par("lty"), ...)
> plot(rnorm(100))
> a <- c(1, 50, 100, 50)
> b <- c(0, -2, 0, 2)
> polygon(a, b)
> polygon(a, b, col = "white")
> polygon(a, b, col = "yellow", border = NA) # border="yellow")
```

mtext (*marginal text*) :

```
> par(oma = rep(1, 4))
> plot(1)
> mtext("Texte marginal")
> mtext("Texte marginal \"externe\"", outer = TRUE)
```

Le second appel à `mtext` n'a aucun effet si `oma` a ses valeurs par défaut.

oma (*outer margin*) est utile en conjonction avec layout :

```
> par(oma = c(3, 3, 0, 0))
> layout(matrix(1:4, 2, 2))
> layout.show()
> par(mar = rep(2, 4))
> for (i in 1:4) plot(runif(50), xlab="", ylab="")
> mtext("index", 1, outer = TRUE, line = 1)
> mtext("runif(50)", 2, outer = TRUE, line = 1)
```

Le paramètre `mar` peut-être fixé individuellement pour chaque graphe.

Produire une zone de « dessin » 100 × 100 :

```
par(mar = rep(0, 4))
plot(0, type = "n", xlim = c(0, 100), ylim = c(0, 100),
     xlab = "", ylab = "", xaxt = "n", yaxt = "n", xaxs = "i",
     yaxs = "i", bty = "n")
```


« **clipping** » avec `par(xpd = TRUE)` :

```
> x11()  
> plot(0)  
> points(0.8, 1.25)  
> par(xpd = TRUE)  
> points(0.8, 1.25)
```

Utile pour placer une légende (avec les données `iris`) :

```
> x11()  
> plot(x, y, col = co[g], pch = psym[g])  
> par(xpd = TRUE)  
> legend(5, 4.7, levels(g), pch = psym, col = co,  
+ bty = "n", horiz = TRUE)
```

Un script typique avec iris :

```
## png("iris.png") # pour page Web
## postscript("iris.eps", width = 8, height = 6)
x <- iris[, 1]; y <- iris[, 2]; g <- iris$Species
co <- c("blue", "red", "yellow")
## co <- rep("black", 3)
psym <- c(19, 19, 19)
## psym <- c(1, 2, 3)
par(bg = "lightslategrey")
plot(x, y, col = co[g], pch = psym[g], xlab = "Sepal.Length",
      ylab = "Sepal.Width")
par(xpd = TRUE)
legend(5, 4.7, levels(g), pch = psym, col = co,
      bty = "n", horiz = TRUE)
## dev.copy2eps(file = "iris.eps") # respecte dim du "device"
## dev.off()
```

... plus sophistiqué :

.....

```
COLOR <- TRUE # FALSE (ou 1 # 0)
if (COLOR) {
  par(bg = "lightslategrey")
  co <- c("blue", "red", "yellow")
  psym <- c(19, 19, 19)
} else {
  par(bg = "transparent") # au cas où...
  co <- rep("black", 3)
  psym <- c(1, 2, 3)
}
```

.....

plotrix (par Jim Lemon) est un package spécialisé dans les fonctions graphiques de haut niveau (avec de nombreux exemples) :

<code>axis.break</code>	<code>axis.mult</code>
<code>barp</code>	<code>color2D.matplot</code>
<code>count.overplot</code>	<code>gantt.chart</code>
<code>pie3D</code>	<code>plotCI</code>
<code>polar.plot</code>	<code>radial.plot</code>
<code>triax.plot</code>	

La plupart des packages spécialisés fournissent des fonctions graphiques appropriées (`ade4`, `ape`, `maps`, ...) utilisant notamment la fonction générique `plot`.

Lattice et Grid

```
> library(lattice)
> xyplot(y ~ x | g)
> xyplot(Sepal.Width ~ Sepal.Length | Species, data = iris)
> histogram(~ Sepal.Length | Species, data = iris)

> m <- sample(1:5, size = 1e3, replace = TRUE)
> v <- sample(1:5, size = 1e3, replace = TRUE)
> x <- rnorm(1e3, m, v)
> table(m, v)
> hist(x) # != histogram(~x)
> histogram(~x | m * v)
> histogram(~x | v * m)
> m <- factor(m, labels = paste("mean =", 1:5))
> v <- factor(v, labels = paste("var =", 1:5))
> histogram(~x | m * v)
```

Principales caractéristiques de lattice :

- les fonctions graphiques « traditionnelles » ne peuvent pas être utilisées ;
- l'argument principal est une formule ;
- il est possible d'utiliser un argument `data` pour localiser les variables ;
- les objets graphiques sont modifiables.

☺ Les graphes conditionnés sont des outils puissants pour l'exploration des données.

Les fonctionnalités de lattice sont vastes : histogrammes, graphes 3D, ...

☹ Les fonctions ayant de nombreuses options il est difficile de s'y retrouver.

Les paramètres graphiques sont difficiles à trouver et à modifier.

Les fonctions `panel*` sont difficiles à utiliser, bien que très puissantes.

```
> trellis.par.get()
> show.settings()
> trellis.par.set(list(strip.background=list(alpha=1,
+                               col="grey")))
```

Les fonctions graphiques secondaires sont dans le package `grid` :

```
> library(grid)
> apropos("^grid\\.")
```

Pour démarrer, il est préférable d'utiliser `lattice` avec les paramètres par défaut. De plus le package est encore à un stade de développement (ver. 0.17 avec R 2.6.0).

rgl

Le package rgl est un « port » d'OpenGL à R :

```
> data(volcano)
> nr <- nrow(volcano)
> nc <- ncol(volcano)
> library(rgl)
> surface3d(1:nr, 1:nc, volcano, col = "green")
> clear3d()
> surface3d(1:nr*10, 1:nc*10, volcano, col = "green")
> surface3d(1:nr, 1:nc, volcano, col = "blue")
> # à comparer avec:
> persp(1:nr, 1:nc, volcano, col = "blue")
```


Exercices III

1. Écrire un script qui ajoutera une “zone” colorée à partir de trois vecteurs de même longueur (il n’est pas interdit de faire un brouillon avant d’écrire le code). On testera ce code avec :

```
x <- 1:100  
tmin <- runif(100, 12, 15)  
tmax <- runif(100, 22, 35)
```

2. Créer une fonction graphique secondaire à partir du code écrit ci-dessus.

IV Les Modèles Linéaires

Le modèle : $E(y) = \beta_1 x_1 + \beta_2 x_2 + \dots + \alpha$

y : réponse

x_1, x_2, \dots : prédicteurs

$\beta_1, \beta_2, \dots, \alpha$: paramètres

Les observations : $y_i = \beta_1 x_{1i} + \beta_2 x_{2i} + \dots + \alpha + \epsilon_i$

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

Une régression « simple » :

```
> x <- 1:50  
> y <- x + 3 + rnorm(50, 0, 10)  
> mod <- lm(y ~ x)  
> summary(mod)
```

Call:

```
lm(formula = y ~ x)
```

Residuals:

Min	1Q	Median	3Q	Max
-17.1684	-4.6348	0.8496	6.7277	16.4951

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	3.81285	2.46452	1.547	0.128
x	0.93191	0.08411	11.079	7.95e-15

Residual standard error: 8.583 on 48 degrees of freedom

Multiple R-Squared: 0.7189, Adjusted R-squared: 0.713

F-statistic: 122.8 on 1 and 48 DF, p-value: 7.952e-15

```
> plot(x, y)
```

```
> abline(mod)
```

```
> abline(b = 1, a = 3, lty = 2)
```

```
> legend("topleft", legend = c("Modèle estimé",  
+ "Modèle simulé"), lty = 1:2)
```

Les variables qualitatives sont codées avec les **contrastes**.

Cas avec deux classes : $z = \{R, B\}$, z est substituée par x_z :

$$z = R \rightarrow x_z = 0$$

$$z = B \rightarrow x_z = 1$$

Cas avec trois classes : $z = \{R, B, V\}$, z est substituée par x_{z_1} et x_{z_2} :

	x_{z_1}	x_{z_2}
R	0	0
B	1	0
V	0	1

Cas général : pour une variable avec n catégories, $n - 1$ variables 0/1 sont créées ; il y a donc $n - 1$ paramètres supplémentaires associés à l'effet de cette variable.

Une analyse de variance à un facteur à trois niveaux :

```
> yb <- rnorm(60, 1:3)
> z <- gl(3, 1, 60)
> mod.b <- lm(yb ~ z)
> summary(mod.b)
```

Call:

```
lm(formula = yb ~ z)
```

Residuals:

Min	1Q	Median	3Q	Max
-2.00118	-0.52484	0.07318	0.78672	1.93827

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.3646	0.2210	6.174	7.44e-08
z2	0.4141	0.3126	1.325	0.190
z3	2.1182	0.3126	6.777	7.48e-09

Residual standard error: 0.9884 on 57 degrees of freedom
Multiple R-Squared: 0.4752, Adjusted R-squared: 0.4567
F-statistic: 25.8 on 2 and 57 DF, p-value: 1.049e-08

```
> plot(z, yb)
> plot.default(z, yb)
> library(lattice)
> histogram(~ yb | z, layout = c(1, 3))
```

Note : il y a plusieurs type de contrastes :

```
> apropos("^contr\\.")
[1] "contr.SAS"          "contr.helmert"      "contr.poly"
[4] "contr.sum"          "contr.treatment"
```

Les tests statistiques (des effets) ne sont pas affectés par le choix du type de contrastes, mais les paramètres estimés seront différents :

```
> contr.treatment(g1(3, 1)) # défaut dans R, pas dans S-PLUS
  2 3
1 0 0
2 1 0
3 0 1
> contr.SAS(g1(3, 1))
  1 2
1 1 0
2 0 1
3 0 0
```

Cela peut avoir des conséquences pour comparer les résultats de différents logiciels.

Formulation générale des modèles linéaires

$E(y) = \beta x$	Régression linéaire
$E(y) = \beta z$	Analyse de variance (ANOVA)
$E(y) = \beta_1 x + \beta_2 z$	Analyse de covariance (ANCOVA)

Dans tous les cas les erreurs sont normalement distribuées autour de la moyenne : ces modèles sont ajustés par la méthode des moindres carrés (minimiser $\sum (y_i - \hat{y}_i)^2$).

Dans R, `aov` et `lm` produisent le même ajustement ; la différence est dans l'affichage des résultats.

```
> summary(aov(yb ~ z))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
z	2	50.412	25.206	25.802	1.049e-08
Residuals	57	55.683	0.977		

Certaines fonctions nécessitent un objet de classe "aov", comme TukeyHSD (*Tukey's honest significant difference*) :

```
> TukeyHSD(aov(yb ~ z))
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = yb ~ z)

$z
      diff      lwr      upr      p adj
2-1 0.7669532 0.0253120 1.508594 0.0411461
3-1 1.9325080 1.1908668 2.674149 0.0000002
3-2 1.1655548 0.4239136 1.907196 0.0010780

> plot(TukeyHSD(aov(yb ~ z)))
```

Les interactions

Pour coder une interaction, de nouvelles variables sont créées par le produit des variables ou de leurs codages numériques. Si une variable qualitative a plus de deux niveaux, de nouvelles variables sont créées avec le produit de toutes les combinaisons 2 à 2 possibles entre les deux variables.

Pour deux variables avec n_1 et n_2 catégories, respectivement, $(n_1 - 1)(n_2 - 1)$ nouvelles variables 0/1 sont créées.

Pour les interactions d'ordre supérieur (entre trois variables ou plus), les combinaisons 3 à 3, 4 à 4, etc, sont utilisées.

`model.matrix` permet de visualiser le modèle numérique créé par une formule (notez la première colonne de 1) :

```
> model.matrix(yb ~ z)
> data.frame(model.matrix(yb ~ z), z)
```

Les diagnostics de régression

```
> plot(y, mod$fitted.values)
> abline(a = 0, b = 1, lty = 3)
> hist(mod$residuals)

> par(mfcol = c(2, 2))
> plot(mod)
```

1. Valeurs prédites par le modèle \hat{y}_i (en x) et résidus r_i (en y); idem que `plot(mod$fitted.values, mod$residuals)`.
2. Valeurs prédites (en x) et racine carrée des résidus standardisés, pour la $i^{\text{ème}}$ observation :

$$e_i = r_i / \left(\hat{\sigma} \sqrt{1 - h_{ii}} \right)$$

Les résidus r_i ne sont en fait pas indépendants et de variance homogène. La matrice de variance-covariance H est calculée avec $H = X(X^T X)^{-1} X^T$

(dans R : `H <- x %*% solve(t(x) %*% x) %*% t(x)`, en prenant éventuellement `x <- cbind(1, x)`, ou `x <- model.matrix(mod)`); les h_{ii} sont les éléments sur la diagonale de cette matrice (`H[i, i]`) qui peuvent être aussi calculés par `hatvalues(mod)`.

3. Vu que $e_i \sim \mathcal{N}(0, 1)$, le graphe des valeurs de distribution prédites par cette loi et celles observées doit donner $x = y$.
4. *leverage* = h_{ii} , mesure de l'influence de chaque observation sur la régression.

Pour apprécier la différence entre résidus et influence :

```
> mod.new <- lm(c(y, 75) ~ c(x, 100))  
> plot(mod.new)
```

Les tests d'hypothèse

```
> LIFEHIST <- read.table("Mammal_lifehistories_v2.txt",  
+                       header = TRUE, sep = "\t",  
+                       na.strings = c("-999", "-999.00"))  
> LIFEHIST <- LIFEHIST[1:1440, ]  
> m1 <- lm(litter.size ~ mass.g., data = LIFEHIST)  
> summary(m1)
```

Call:

```
lm(formula = litter.size ~ mass.g., data = LIFEHIST)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.8053	-1.7253	-0.3053	1.1947	11.3747

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	2.805e+00	4.949e-02	56.68	< 2e-16

```
mass.g.      -2.605e-08  9.238e-09   -2.82  0.00488
```

```
Residual standard error: 1.769 on 1284 degrees of freedom  
(154 observations deleted due to missingness)
```

```
Multiple R-Squared: 0.006156,   Adjusted R-squared: 0.005382
```

```
F-statistic: 7.953 on 1 and 1284 DF,   p-value: 0.004875
```

```
> m2 <- lm(litter.size ~ order, data = LIFEHIST)
```

```
> summary(m2)
```

```
Call:
```

```
lm(formula = litter.size ~ order, data = LIFEHIST)
```

```
Residuals:
```

```
      Min       1Q   Median       3Q      Max  
-3.0338 -0.7268 -0.1844  0.5615 10.5390
```

```
Coefficients:
```

```
              Estimate Std. Error t value Pr(>|t|)
```

(Intercept)	1.27192	0.11261	11.295	<2e-16
orderCarnivora	1.49249	0.15108	9.879	<2e-16
orderCetacea	-0.25953	0.23598	-1.100	0.2716
[...]				

Residual standard error: 1.406 on 1339 degrees of freedom
 (84 observations deleted due to missingness)

Multiple R-Squared: 0.3791, Adjusted R-squared: 0.3716

F-statistic: 51.09 on 16 and 1339 DF, p-value: < 2.2e-16

```
> anova(m2) # == summary(aov(litter.size ~ order ...
```

Analysis of Variance Table

Response: litter.size

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
order	16	1616.98	101.06	51.087	< 2.2e-16
Residuals	1339	2648.85	1.98		

```
> m3 <- lm(litter.size ~ mass.g.*order, data = LIFEHIST)
```

```
> summary(m3)
```

Call:

```
lm(formula = litter.size ~ mass.g. * order, data = LIFEHIST)
```

Residuals:

Min	1Q	Median	3Q	Max
-3.1418	-0.6929	-0.1807	0.5588	10.4874

Coefficients: (2 not defined because of singularities)

[...]

Residual standard error: 1.381 on 1254 degrees of freedom
(154 observations deleted due to missingness)

Multiple R-Squared: 0.4086, Adjusted R-squared: 0.394

F-statistic: 27.95 on 31 and 1254 DF, p-value: < 2.2e-16

```
> anova(m3)
```

Analysis of Variance Table

Response: litter.size

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
mass.g.	1	24.90	24.90	13.0521	0.0003149
order	16	1548.57	96.79	50.7338	< 2.2e-16
mass.g.:order	14	79.29	5.66	2.9688	0.0001725
Residuals	1254	2392.27	1.91		

```
> anova(lm(litter.size ~ order*mass.g., data = LIFEHIST))
```

Analysis of Variance Table

Response: litter.size

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
order	16	1573.46	98.34	51.5493	< 2.2e-16
mass.g.	1	0.01	0.01	0.0044	0.9470630
order:mass.g.	14	79.29	5.66	2.9688	0.0001725
Residuals	1254	2392.27	1.91		

```
> drop1(m3, test = "F")
```

Single term deletions

Model:

```
litter.size ~ mass.g. * order
              Df Sum of Sq      RSS      AIC F value      Pr(F)
<none>                2392.27  862.23
mass.g.:order 14      79.29 2471.56  876.16  2.9688 0.0001725
> drop1(lm(litter.size~order+mass.g., data=LIFEHIST), test="F")
Single term deletions
```

Model:

```
litter.size ~ order + mass.g.
              Df Sum of Sq      RSS      AIC F value      Pr(F)
<none>                2471.6  876.2
order    16      1548.6 4020.1 1469.8 49.6544 <2e-16
mass.g.   1    0.008413 2471.6  874.2  0.0043 0.9476
```

```
> library(lattice)
> xyplot(litter.size ~ log(mass.g.) | order, data = LIFEHIST)
```

Les modèles ne peuvent être comparés que s'ils ont été ajustés au même vecteur de réponses :

- $y \sim x$ et $\log(y) \sim x$ ne peuvent **pas** être comparés !
- $y \sim x$ et $y \sim x + z$ ne seront pas ajustés aux mêmes données si x et z ont des NA à des observations différentes.

```
res.lm <- lm(...  
res.aov <- aov(...
```

1. `summary(res.aov)` : tableau d'ANOVA (= tests sur les effets $\sim F$)
`summary(res.lm)` : tests sur paramètres ($\sim t$)

2. `anova`

Si un modèle : tableau d'ANOVA en incluant les effets dans l'ordre de la formule.

Si plusieurs modèles : tableau d'ANOVA entre les modèles.

- (a) `anova(res.lm)` et `summary(res.aov)` sont identiques.
- (b) L'ordre des termes dans la formule est important s'il y a plusieurs prédictors qualitatifs et que les effectifs retournés par `table` sont inégaux (*unbalanced design*).

3. `drop1` : teste les effets individuels vs. le modèle complet.

(a) Le principe de marginalité est respecté.

(b) `drop1(res.lm)` et `summary(res.lm)` sont identiques si tous les prédicteurs sont continus ou avec deux catégories (car chaque effet a 1 ddl) et qu'il n'y a pas d'interaction dans le modèle.

4. `add1` teste l'ajout d'un ou plusieurs effets.

Ex. : si le modèle initial n'inclut pas d'interaction : `add1(res, ~.^2)` testera l'addition de chaque interaction individuellement.

5. `predict` calcule les valeurs prédites par le modèle. L'option `newdata` sert à spécifier de nouvelles valeurs pour les prédicteurs ; ceux-ci doivent être nommés de la même façon que dans la formule :

```
> predict(m1, newdata = 1e9)
```

```
Error in eval(predvars, data, env) :
```

```
not that many frames on the stack
```


```
> predict(m1, newdata = data.frame(mass.g. = 1e9))
```

```
[1] -23.24696
```

```
> ndf <- data.frame(mass.g. = 1e9)
```

```
> predict(m1, newdata = ndf)
[1] -23.24696
```

anova, drop1 add1 et predict sont des fonctions génériques.

 Les 'méthodes' sont documentées séparément : ?anova donne peu d'information ; c'est ?anova.lm (ou ?anova.glm, ...) qu'il faut généralement consulter.

V Les modèles linéaires généralisés

Deux caractéristiques du modèle linéaire classique sont limitantes : (i) la moyenne prédite est distribuée entre $-\infty$ et $+\infty$, (ii) l'hypothèse de normalité des résidus.

Les transformations de la réponse font perdre la nature des données analysées (fécondité, effectifs, durée de vie, ...).

L'idée des modèles linéaires généralisés (MLG ou *GLM* pour *generalized linear models*) est de reprendre le modèle linéaire $E(y) = \beta x$ et de l'étendre à d'autres distributions. Pour prendre en compte que la moyenne peut être distribuée sur un intervalle fini, on considérera une fonction de celle-ci :

$$g[E(y)] = \beta x$$

telle que $-\infty < g[E(y)] < +\infty$. g est appelée une fonction de **lien** (*link*).

 Les données originales ne sont pas transformées $g[E(y)] \neq g(y)$.

Quatre distributions sont couramment utilisées :

Gaussienne : réponse continue entre $-\infty$ et $+\infty$

Poisson : réponse entière entre 0 et $+\infty$

Binomiale : réponse entière entre 0 et n (nb de cas)

Gamma : réponse continue entre 0 et $+\infty$

La distribution des données pouvant être asymétrique, la méthode des moindres carrés n'est pas applicable : les MLG sont ajustés par la méthode du maximum de vraisemblance (ML pour *maximum likelihood*).

```
> gm1 <- glm(litter.size ~ mass.g., data = LIFEHIST)
> summary(gm1)
```

Call:

```
glm(formula = litter.size ~ mass.g., data = LIFEHIST)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.8053	-1.7253	-0.3053	1.1947	11.3747

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	2.805e+00	4.949e-02	56.68	< 2e-16
mass.g.	-2.605e-08	9.238e-09	-2.82	0.00488

(Dispersion parameter for gaussian family taken to be 3.130944)

Null deviance: 4045.0 on 1285 degrees of freedom
Residual deviance: 4020.1 on 1284 degrees of freedom
(154 observations deleted due to missingness)
AIC: 5121.3

Number of Fisher Scoring iterations: 2

La déviance est $-2 \ln L$ (L vraisemblance à son maximum).

AIC : *Akaike Information Criterion*

$AIC = -2 \ln L + 2k$ (k : nombre de paramètres estimés)

Pour une distribution gaussienne, moindres carrés et maximum de vraisemblance donnent les mêmes résultats.

Avec la méthode du maximum de vraisemblance, la significativité des effets est testée par les quotients de vraisemblance (*likelihood ratio tests, LRT*). Sous H_0 (pas d'effet), ce test suit une loi du χ^2 avec un nombre de ddl égal au nombre de paramètres associés à l'effet :

```
> anova(gml, test = "Chisq")
```

```
Analysis of Deviance Table
```

```
Model: gaussian, link: identity
```

```
Response: litter.size
```

```
Terms added sequentially (first to last)
```

```

          Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                1285      4045.0
mass.g.             1      24.9      1284      4020.1  0.004801
> 1 - pchisq(24.9/3.14, 1)
[1] 0.004862351

```

AIC peut être utilisé à la place du LRT :

1. modèles non-emboîtés ;
2. grand nombre de modèles considérés a priori (évite tests 2 à 2) ;
3. l'objectif est de sélectionner un modèle à but prédictif (car AIC est moins conservateur que LRT).

Note : LRT et AIC sont applicables à tout type de modèles ajustés par maximum de vraisemblance (pas seulement les MLG).

Le second argument de `glm` est `family` qui est une fonction et par défaut `family = gaussian` :

```
> args(gaussian)
function (link = "identity")
```

Par défaut `glm` ajuste le même modèle que `lm`.

Trois liens sont possibles pour `gaussian` : "identity", "log" et "inverse".

Essayons le lien inverse :

```
> gmlinv <- glm(litter.size ~ mass.g., gaussian("inverse"),
+             data = LIFEHIST)
> summary(gmlinv)
```

Call:

```
glm(formula = litter.size ~ mass.g., family = gaussian("inverse",
      data = LIFEHIST))
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.5995	-1.0753	0.1331	0.9656	10.5914

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	2.774e-01	4.676e-03	59.315	<2e-16
mass.g.	2.882e-05	3.218e-06	8.957	<2e-16

(Dispersion parameter for gaussian family taken to be 2.418827)

Null deviance: 4045.0 on 1285 degrees of freedom
Residual deviance: 3105.3 on 1284 degrees of freedom
(154 observations deleted due to missingness)
AIC: 4789.2

Number of Fisher Scoring iterations: 23

Les deux modèles sont :

$$\text{gm1} : E(y) = -2.60 \times 10^{-8} \text{ mass} + 2.80 \quad \hat{\sigma}^2 = 3.13$$

$$\text{gm1inv} : \frac{1}{E(y)} = 2.88 \times 10^{-5} \text{ mass} + 0.28 \quad \hat{\sigma}^2 = 2.42$$

L'ajustement est sensiblement amélioré par le lien inverse (mais n'oublions pas les données sont hétérogènes).

Les diagnostics de régression sont identiques à ceux du modèle linéaire, avec la différence que, par défaut, les résidus de déviance sont calculés.

Distribution de Poisson

Les liens permis sont "log" (le default), "identity" et "sqrt".

```
> x <- runif(50, 1, 50)
> y <- rpois(50, x)
> plot(x, y)
> gm4 <- glm(y ~ x)
> gm5 <- glm(y ~ x, poisson)
> summary(gm4)
```

Call:

```
glm(formula = y ~ x)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-7.8011	-2.0088	-0.5986	1.8911	11.3623

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-1.70545	0.95811	-1.78	0.0814
x	1.11443	0.03506	31.79	<2e-16

(Dispersion parameter for gaussian family taken to be 13.40717)

Null deviance: 14191.68 on 49 degrees of freedom
Residual deviance: 643.54 on 48 degrees of freedom
AIC: 275.64

Number of Fisher Scoring iterations: 2

```
> summary(gm5)
```

Call:

```
glm(formula = y ~ x, family = poisson)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-----	----	--------	----	-----

-3.73525 -0.79706 -0.02801 0.79571 1.84332

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	1.830827	0.075914	24.12	<2e-16
x	0.047641	0.002117	22.51	<2e-16

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 630.755 on 49 degrees of freedom
Residual deviance: 66.642 on 48 degrees of freedom
AIC: 302.63

Number of Fisher Scoring iterations: 4

La prise en compte de la nature de la distribution de la réponse améliore très nettement la dispersion des résidus.

On notera le paramètre de dispersion égal à 1 (car la variance est contrainte à être égale à la moyenne dans la loi de Poisson).

Distribution binomiale

La réponse est spécifiée dans `glm` avec :

1. une matrice à deux colonnes où la première est le nombre de « succès » et la seconde le nombre d'« échecs » ;
2. un vecteur avec des 0 et des 1 (ou `FALSE` et `TRUE`) ;
3. un facteur où le premier niveau (codé 1) est pris comme « échec » et les autres comme « succès ».

```
> y <- sample(0:1, size = 20, replace = TRUE)
```

```
> x <- 1:20
```

```
> gmb1 <- glm(y ~ x, binomial)
```

```
> gmb2 <- glm(cbind(y, 1 - y) ~ x, binomial)
```

```
> gmb3 <- glm(factor(y) ~ x, binomial)
```

```
> AIC(gmb1); AIC(gmb2); AIC(gmb3)
```

```
[1] 23.67469
```

```
[1] 23.67469
```

```
[1] 23.67469
```

Dans les deux premières situations il est aisé d'inverser les succès et les échecs :

```
> glm(!y ~ x, binomial)
> glm(cbind(1 - y, y) ~ x, binomial)
```


Les coefficients seront de signe opposé. Si on a préparé une matrice au préalable :

```
> Y <- cbind(y, 1 - y)
> glm(Y ~ x, binomial)
> glm(Y[, 2:1] ~ x, binomial)
```

Par défaut le lien est la fonction logit. Les choix possibles sont :

"logit"	$\ln \frac{p}{1-p}$, le défaut
"probit"	$F_{\mathcal{N}}^{-1}(p)$
"cauchit"	$F_{\mathcal{C}}^{-1}(p)$
"log"	$\ln p$
"cloglog"	$\ln(-\ln(1-p))$

$p = E(y)$; F^{-1} est l'inverse de la fonction de densité de probabilité cumulée.

 Le choix d'un lien n'est pas trivial. Il est recommandé, pour débiter, d'utiliser le lien par défaut.

Données Aids2

```
> library(MASS)
> data(Aids2)
> ?Aids2
```

Le genre a-t-il un effet sur la « survie » ?

```
> aids.m1 <- glm(status ~ sex, binomial, data = Aids2)
> anova(aids.m1, test = "Chisq")
```

Analysis of Deviance Table

Model: binomial, link: logit

Response: status

Terms added sequentially (first to last)

	Df	Deviance	Resid. Df	Resid. Dev	P(> Chi)
NULL			2842	3777.5	
sex	1	0.2	2841	3777.3	0.6

L'âge a-t-il un effet sur la « survie » ?

```
> aids.m2 <- glm(status ~ age, binomial, data = Aids2)
```

```
> anova(aids.m2, test = "Chisq")
```

Analysis of Deviance Table

Model: binomial, link: logit

```
Response: status
```

```
Terms added sequentially (first to last)
```

	Df	Deviance	Resid. Df	Resid. Dev	P(> Chi)
NULL			2842	3777.5	
age	1	6.8	2841	3770.7	0.009191

Une fois l'effet de l'âge pris en compte, le genre a-t-il un effet ? On prend soin de mettre `sex` après `age` dans la formule, sinon, il faudra tester les effets avec `drop1` (mais sans interaction).

```
> aids.m3 <- glm(status ~ age*sex, binomial, data = Aids2)
```

```
> anova(aids.m3, test = "Chisq")
```

```
Analysis of Deviance Table
```

```
Model: binomial, link: logit
```

Response: status

Terms added sequentially (first to last)

	Df	Deviance	Resid. Df	Resid. Dev	P(> Chi)
NULL			2842	3777.5	
age	1	6.8	2841	3770.7	0.009191
sex	1	0.2	2840	3770.5	0.6
age:sex	1	3.8	2839	3766.6	0.1

```
> summary(aids.m2)
```

Call:

```
glm(formula = status ~ age, family = binomial, data = Aids2)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.5696	-1.3663	0.9450	0.9917	1.1303

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	0.111819	0.149300	0.749	0.4539
age	0.010065	0.003881	2.593	0.0095

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 3777.5 on 2842 degrees of freedom
Residual deviance: 3770.7 on 2841 degrees of freedom
AIC: 3774.7

Number of Fisher Scoring iterations: 4


Le modèle sélectionné est donc (p : probabilité de décès) :

$$\ln\left(\frac{p}{1-p}\right) = 0.01 \text{ age} + 0.11$$

```
> range(Aids2$age)
[1] 0 82
> a <- seq(0, 82, 0.1)
> risk <- predict(aids.m2, newdata = data.frame(age = a),
+               type = "response")
> plot(a, risk, type = "l", xlab = "Âge (années)",
+      ylab = "Probabilité de décès", ylim = 0:1)
> title("Risque de mortalité prédit par le modèle \"aids.m2\"")
```

La page d'aide de `predict.glm` inclut des informations très intéressantes sur comment calculer un intervalle de prédiction.

Loi gamma

 Gamma () donne la loi gamma pour un MLG ;
gamma (x) calcule la fonction $\Gamma(x) = 1 \times 2 \times \dots \times (x - 1)$ pour x entier.

Les liens permis sont "inverse" (le défaut), "identity" et "log".

```
> Aids2$surv <- Aids2$death - Aids2$diag
> surv.m1 <- glm(surv ~ age, Gamma, data = Aids2,
+               subset = surv > 0)
> anova(surv.m1, test = "Chisq")
```

Analysis of Deviance Table

Model: Gamma, link: inverse

Response: surv

Terms added sequentially (first to last)

```
      Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL          2813      3219.5
age          1      27.5      2812      3192.0 3.945e-09
> summary(surv.m1)
```

Call:

```
glm(formula = surv ~ age, family = Gamma, data = Aids2, subset =
      0)
```

Deviance Residuals:

```
      Min       1Q   Median       3Q      Max
-3.2565  -0.9426  -0.2280   0.3740   3.0442
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.539e-03  1.543e-04  9.969  < 2e-16
age          2.471e-05  4.199e-06  5.884  4.47e-09
```

(Dispersion parameter for Gamma family taken to be 0.7935563)

Null deviance: 3219.5 on 2813 degrees of freedom
Residual deviance: 3192.0 on 2812 degrees of freedom
AIC: 39502

Number of Fisher Scoring iterations: 6

Formulation générale des MLG

$$g(E[y]) = \beta X \quad \text{Var}(y_i) = \phi \mathcal{V}(E[y_i])$$

ϕ : paramètre de dispersion ; \mathcal{V} : fonction de variance

Réponse	g^*	ϕ	\mathcal{V}	$\text{Var}(y_i)$
normale	identité	σ^2	.	σ^2
gamma	inverse	$1/\nu$	μ^2	μ^2/ν
Poisson	log	1	μ	μ
binomiale	logit	1	$\mu(1 - \mu)$	$\mu(1 - \mu)n_i$

*fonction de lien par défaut dans R

L'utilisateur choisit la distribution de la réponse et la fonction de lien.
Le reste découle de ces choix.

Sur- (sous-) dispersion avec les lois binomiale et de Poisson

Avec ces deux lois de distribution, la variance est contrainte par la moyenne. La méthode de quasivraisemblance permet de lever cette contrainte.

Seules la moyenne et la variance des observations sont considérées : il n'est donc pas possible de calculer la vraisemblance (pas d'AIC). La comparaison avec un MLG standard par LRT n'est donc pas possible.

En pratique, on considère que si $\phi > 4$ (ou $\phi < 0.25$), il y a sur- (sous) dispersion. Ces phénomènes sont généralement dûs à une dépendance entre observations.

```
> summary(glm(status ~ age, quasibinomial, data = Aids2))
```

Call:

```
glm(formula = status ~ age, family=quasibinomial, data=Aids2)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-----	----	--------	----	-----

-1.5696 -1.3663 0.9450 0.9917 1.1303

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.111819	0.149342	0.749	0.45407
age	0.010065	0.003882	2.593	0.00957

(Dispersion parameter for quasibinomial family taken to be 1.00)

Null deviance: 3777.5 on 2842 degrees of freedom
Residual deviance: 3770.7 on 2841 degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4

La fonction `quasi` permet de construire une famille avec lien et fonction de variance spécifiques (`?quasi` pour tous les détails : toutes les familles de distributions sont documentées sur cette page).

Exercices V

1. Reprendre le modèle `aids.m2` et tracer l'intervalle de confiance à 95% du risque prédit. Le graphe sera légendé.
2. Charger les données `eagles` du package MASS.
 - (a) Reprendre le modèle `eagles.glm` dans les exemples et analyser le avec `anova` et `summary`. Commenter.
 - (b) Ré-ajuster ces modèles avec `family = quasibinomial`. Que constatez-vous? (Comparer notamment les résultats avec ou sans la variable `v`.) Commenter sur la récolte des données.

VI Modèles mixtes linéaires

```
> X <- read.table("Donnees_sevrage.txt", header=TRUE, dec=",")
> X$Bac <- factor(X$Bac)
> summary(X)
```

	Bac	Colonne	Sevrage	Longueur
5	: 82	d:180	D:128	Min. : 9.205
11	: 81	g:173	L:233	1st Qu.:14.080
15	: 70	m:206	R:198	Median :15.463
16	: 70			Mean :15.912
9	: 69			3rd Qu.:17.009
4	: 59			Max. :35.102
	(Other):128			

```
> table(X$Bac)
```

3	4	5	9	10	11	15	16	17
39	59	82	69	56	81	70	70	33

```
> densityplot(~ Longueur | Sevrage, data = X, groups = Bac)
```


Pour avoir une (autre) vue de la variabilité au sein de chaque traitement :

```
> histogram(~ Longueur | Bac, data = X, subset = Sevrage=="D")  
> histogram(~ Longueur | Bac, data = X, subset = Sevrage=="L")  
> histogram(~ Longueur | Bac, data = X, subset = Sevrage=="R")
```

Il n'y a pas de variation systématique liée aux bacs ; l'effet « Bac » est une nuisance.

```
> library(nlme)  
> X.lme <- lme(Longueur ~ Sevrage, random = ~ 1 | Bac, data=X)  
> summary(X.lme)
```

Linear mixed-effects model fit by REML

Data: X

AIC	BIC	logLik
2836.525	2858.129	-1413.263

Random effects:

Formula: ~1 | Bac
(Intercept) Residual
StdDev: 0.7111282 3.007008

Fixed effects: Longueur ~ Sevrage

	Value	Std.Error	DF	t-value	p-value
(Intercept)	14.847844	0.4916370	550	30.200825	0.0000
SevrageL	2.302461	0.6702563	6	3.435194	0.0139
SevrageR	0.330652	0.6753925	6	0.489571	0.6418

Correlation:

	(Intr)	SevrgL
SevrageL	-0.734	
SevrageR	-0.728	0.534

Standardized Within-Group Residuals:

Min	Q1	Med	Q3	Max
-2.0219390	-0.5700525	-0.1843231	0.3517755	5.7724699

Number of Observations: 559

Number of Groups: 9

```
> X.lm <- lm(Longueur ~ Sevrage, data = X)
> summary(X.lm)
```

Call:

```
lm(formula = Longueur ~ Sevrage, data = X)
```

Residuals:

Min	1Q	Median	3Q	Max
-5.6337	-1.7641	-0.5778	1.0163	17.9830

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	14.8383	0.2701	54.932	<2e-16
SevrageL	2.2810	0.3362	6.784	3e-11
SevrageR	0.3459	0.3466	0.998	0.319

Residual standard error: 3.056 on 556 degrees of freedom

Multiple R-Squared: 0.1023, Adjusted R-squared: 0.09911

F-statistic: 31.69 on 2 and 556 DF, p-value: 9.22e-14

La prise en compte de l'effet aléatoire associé aux bacs améliore nettement la distribution des résidus.

REML (*residual maximum likelihood*) corrige le biais dans l'estimation des composants de variance dû au fait que des effets fixes sont estimés (on retrouve cette idée dans l'estimation de la variance d'un échantillon sur $n - 1$ et non n).

La significativité des effets aléatoires peut être testée en comparant avec le même modèle ajusté par `lm`, mais le modèle mixte doit être ajusté par ML et non REML.

```
> X.lme.ML <- lme(Longueur ~ Sevrage, random = ~ 1 | Bac,  
+               data = X, method = "ML")  
> anova(X.lme.ML, X.lm)
```

	Model	df	AIC	BIC	logLik	Test
X.lme.ML	1	5	2836.780	2858.411	-1413.390	
X.lm	2	4	2840.311	2857.616	-1416.156	1 vs 2

	L.Ratio	p-value
X.lme.ML		
X.lm	5.530973	0.0187

Le modèle statistique derrière X.lme est :

$$\text{Longueur}_i = \beta \text{Sevrage}_i + \zeta_j + \epsilon_i$$

où j est l'indice du bac, $\zeta_j \sim \mathcal{N}(0, \sigma_B^2)$, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

Les composants de variance estimés sont $\hat{\sigma}_B = 0,71$ et $\hat{\sigma} = 3,01$.

Considérer l'effet bac comme fixe est problématique car l'effet sevrage est emboîté dans celui-ci :

```
> X.lm2 <- lm(Longueur ~ Sevrage + Bac, data = X)
> summary(X.lm2)
```

```
.....  
Coefficients: (2 not defined because of singularities)
```

```
.....  
> drop1(X.lm2, test = "F")
```

```
Single term deletions
```

```
Model:
```

```
Longueur ~ Sevrage + Bac
```

	Df	Sum of Sq	RSS	AIC	F value	Pr (F)
<none>			4970.2	1239.5		
Sevrage	0	0.0	4970.2	1239.5		
Bac	6	222.5	5192.8	1251.9	4.1044	0.0004843

L'estimation des effets fixes (le vecteur β) n'est pas biaisée par la présence d'effets aléatoires, mais les erreurs-standards sont sous-estimées si ces derniers sont ignorés. Cela se comprend dans le sens où la variation aléatoire entre les bacs est « captée » par le modèle à effet fixe.

Une analyse de variance à effets mixtes se contente de tester l'effet fixe au sein de chaque niveau (= strate) de Bac :

```
> summary(aov(Longueur ~ Sevrage + Error(Bac), data = X))
```

Error: Bac

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Sevrage	2	592.03	296.01	7.9809	0.02039
Residuals	6	222.54	37.09		

Error: Within

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Residuals	550	4970.2	9.0		

Dépendance entre observations

`lme` a l'option `correlation` qui prend pour argument une fonction qui spécifie la forme de la dépendance entre observations ; celles fournies avec `nlme` sont documentées à `?corClasses`.

La plus simple est `corCompSymm` qui spécifie que toutes les observations sont corrélées de façon identique : il n'y a donc qu'un paramètre à estimer. L'option `form` permet de spécifier, à l'aide d'une formule, les éventuelles strates de cette dépendance.

```
> X.lme.cor <- lme(Longueur ~ Sevrage, random = ~ 1 | Bac,  
+ data = X, correlation = corCompSymm(form = ~ 1 | Bac))  
> summary(X.lme.cor)
```

.....

Correlation Structure: Compound symmetry

Formula: ~1 | Bac

Parameter estimate(s):

Rho

-8.468964e-08

....

Il est possible de spécifier que les données sont corrélées sans stratification avec `corCompSymm(form = ~ 1)` : les résultats ne sont pas changés.

La fonction `gls` (*generalized least squares*) dans `nlme` permet d'ajuster un modèle avec données corrélées sans effets aléatoires (`lme` en requiert au moins un) :

```
> X.gls.cor <- gls(Longueur ~ Sevrage, data = X,  
+ correlation = corCompSymm(form = ~ 1 | Bac))  
> X.gls.cor
```

```
Generalized least squares fit by REML  
Model: Longueur ~ Sevrage  
Data: X  
Log-restricted-likelihood: -1413.263
```

Coefficients:

(Intercept)	SevrageL	SevrageR
14.8478443	2.3024604	0.3306525

Correlation Structure: Compound symmetry

Formula: $\sim 1 \mid \text{Bac}$

Parameter estimate(s):

Rho

0.05296477

Degrees of freedom: 559 total; 556 residual

Residual standard error: 3.089951

Il est intéressant que d'ignorer la stratification entre bacs rend la corrélation entre observations nulle :

```
> gls(Longueur ~ Sevrage, data = X,  
+      correlation = corCompSymm(form = ~ 1))  
Generalized least squares fit by REML
```

Model: Longueur ~ Sevrage

Data: X

Log-restricted-likelihood: -1417.847

Coefficients:

(Intercept)	SevrageL	SevrageR
14.8383493	2.2809583	0.3459145

Correlation Structure: Compound symmetry

Formula: ~1

Parameter estimate(s):

Rho

2.164525e-19

Degrees of freedom: 559 total; 556 residual

Residual standard error: 3.056057

Le même modèle mais avec observations indépendantes :

```
> X.gls <- gls(Longueur ~ Sevrage, data = X)
```

```
> X.gls
```

```
Generalized least squares fit by REML
```

```
Model: Longueur ~ Sevrage
```

```
Data: X
```

```
Log-restricted-likelihood: -1417.847
```

```
Coefficients:
```

```
(Intercept)      SevrageL      SevrageR
```

```
14.8383493      2.2809583      0.3459145
```

```
Degrees of freedom: 559 total; 556 residual
```

```
Residual standard error: 3.056057
```

```
> AIC(X.gls.cor); AIC(X.gls)
```

```
[1] 2836.780
```

```
[1] 2843.694
```

Le modèle avec corrélation intra-bac semble préférable. On réajuste les modèles par ML pour faire un LRT :

```
> X.gls.cor.ML <- gls(Longueur ~ Sevrage, data = X,  
+ correlation = corCompSymm(form = ~ 1 | Bac), method = "ML")  
> X.gls.ML <- gls(Longueur ~ Sevrage, data = X, method = "ML")  
> anova(X.gls.cor.ML, X.gls.ML)
```

	Model	df	AIC	BIC	logLik	Test
X.gls.cor.ML	1	5	2836.780	2858.411	-1413.390	
X.gls.ML	2	4	2840.311	2857.616	-1416.156	1 vs 2
			L.Ratio	p-value		
X.gls.cor.ML						
X.gls.ML			5.530973	0.0187		

Les résultats de `gls` sont cohérents avec ceux de `lme`, mais `lme` est ici plus facile à interpréter.

VII Méthodes multivariées

Les méthodes multivariées considèrent un ensemble de variables sur le même plan, par opposition aux modèles de régression où les variables ont des rôles différents.

Réduction du nombre de dimensions

Analyse en composantes principales (ACP)

Toutes les variables sont continues. Le but est de chercher une combinaison linéaire qui maximise la dispersion (variance) des points.

Deux fonctions dans le package stats :

`prcomp` : par décomposition en valeurs singulières (recommandée pour précision numérique) ;

`princomp` : décomposition en valeurs et vecteurs propres (compatible avec S-PLUS)

```
> data(iris)
> cov(iris[, c(1, 3)])
              Sepal.Length Petal.Length
Sepal.Length  0.6856935      1.274315
Petal.Length  1.2743154      3.116278
> plot(iris[, c(1, 3)], asp = 1)
> prcomp(iris[, c(1, 3)])
Standard deviations:
[1] 1.9136088 0.3742627
```

Rotation:

```
              PC1          PC2
Sepal.Length 0.3936059 -0.9192793
Petal.Length 0.9192793  0.3936059
```

```
> princomp(iris[, c(1, 3)])
```

Call:

```
princomp(x = iris[, c(1, 3)])
```

Standard deviations:

```
      Comp.1      Comp.2
1.9072195 0.3730131
```

```
2 variables and 150 observations.
```

L'option `scale = TRUE` réalise l'ACP sur la matrice de corrélation.

```
> prcomp(iris[, c(1, 3)], scale = TRUE)
```

```
Standard deviations:
```

```
[1] 1.3681205 0.3581148
```

```
Rotation:
```

```
              PC1              PC2
Sepal.Length 0.7071068 -0.7071068
Petal.Length 0.7071068  0.7071068
```

```
> X <- as.matrix(iris[, c(1, 3)])
```

```
> acp.iris <- prcomp(X)
```



```
> summary(acp.iris)
```

Importance of components:

	PC1	PC2
Standard deviation	1.914	0.3743
Proportion of Variance	0.963	0.0368
Cumulative Proportion	0.963	1.0000

L'élément nommé `rotation` de `acp.iris` contient les coefficients ρ de l'ACP tels que :

$$\begin{aligned}y^{(1)} &= \rho_1^{(1)} x_1 + \rho_2^{(1)} x_2 \\y^{(2)} &= \rho_1^{(2)} x_1 + \rho_2^{(2)} x_2\end{aligned}$$

Ce qui peut être calculé dans R par :

```
acp.iris$rotation[1,1]*X[,1] + acp.iris$rotation[2,1]*X[,2]  
acp.iris$rotation[1,2]*X[,1] + acp.iris$rotation[2,2]*X[,2]
```

ou plus simplement par le produit matriciel (qui ne prend pas les 'data frame') :

```
> Y <- as.matrix(X) %*% acp.iris$rotation  
> plot(Y, asp = 1)
```

La fonction `predict` fait tous ces calculs :

```
> X11() # pour comparaison  
> plot(predict(acp.iris))
```

La première composante principale peut être tracée, de façon approximative, sur les données originales avec :

```
> dev.set(2)  
> a <- acp.iris$center + 10*acp.iris$rotation[, 1]  
> b <- acp.iris$center - 10*acp.iris$rotation[, 1]  
> lines(rbind(a, b), col = "red")
```

La fonction `biplot` représente simultanément les observations et les variables. Le second argument `choices = 1:2` sélectionne les composantes principales :

```
> biplot(acp.iris)
> biplot(prcomp(iris[, 1:4]), 2:3)
```

Rappel : si toutes les variables sont positivement corrélées, elles auront toutes une contribution positive au premier axe principal qui est nommé « axe de taille », ce qui peut se vérifier avec :

```
> all(cor(iris[, 1:4]) > 0)
[1] FALSE
```

Analyse des correspondances

Les variables sont des effectifs, typiquement un tableau espèces \times localités. La fonction `corresp` se trouve dans le package MASS.

```
> library(MASS)
> ?corresp
```


Une A(F)C est en fait une double ACP sur les lignes et les colonnes des fréquences calculées à partir des effectifs.

Un tableau de contingences peut être interprété comme une compilation des effectifs de deux facteurs :

```
> sp <- paste("S", 1:5, sep = "")
> SP <- sample(sp, size = 50, replace = TRUE)
> SP <- factor(SP)
> hab <- paste("H", 1:3, sep = "")
> HAB <- sample(hab, size = 50, replace = TRUE)
> HAB <- factor(HAB)
> table(SP, HAB)
```

	HAB		
SP	H1	H2	H3
S1	2	6	4
S2	2	1	8
S3	4	5	1
S4	4	2	3
S5	2	3	3

L'AFC peut être réalisée soit à partir des facteurs originaux, soit à partir du tableau de contingence.

 Le tableau retourné par `table` n'est pas accepté par `corresp`, et la conversion avec `as.matrix` ne suffit pas. Il faut supprimer la classe avec `unclass` ou calculer ce tableau avec `xtabs`.

```
> corresp(SP, HAB)
```

```
First canonical correlation(s): 0.4344339
```

```
x scores:
```

```
          S1          S2          S3          S4          S5
0.37781925 -1.68252230  1.30761935  0.04473031  0.06189349
```

```
y scores:
```

```
          H1          H2          H3
0.4807207  1.0016592 -1.2504367
```

```
> corresp(unclass(table(SP, HAB)))
```

```
First canonical correlation(s): 0.4344339
```

SP scores:

S1	S2	S3	S4	S5
0.37781925	-1.68252230	1.30761935	0.04473031	0.06189349

HAB scores:

H1	H2	H3
0.4807207	1.0016592	-1.2504367

```
> corresp(xtabs(~ SP + HAB))
```

First canonical correlation(s): 0.4344339

SP scores:

S1	S2	S3	S4	S5
0.37781925	-1.68252230	1.30761935	0.04473031	0.06189349

HAB scores:

H1	H2	H3
0.4807207	1.0016592	-1.2504367

Analyse des correspondances multiples

L'ACM est une généralisation de l'AC pour plusieurs facteurs. La fonction `mca` dans `MASS` prend pour argument une série de facteurs dans un tableau 'data frame'.

Une matrice est construite où chaque colonne correspond à un niveau des facteurs, et 1 si ce niveau a été observé, 0 sinon. Exemple :

	SP.S1	SP.S2	SP.S3	SP.S4	SP.S5	HAB.H1	HAB.H2	HAB.H3
[1,]	1	0	0	0	0	0	0	1
[2,]	1	0	0	0	0	0	1	0
[3,]	0	0	1	0	0	0	1	0
[4,]	0	0	0	1	0	1	0	0
[5,]	0	1	0	0	0	0	0	1
[...]								

Les résultats seront donc différents que ceux d'une AC même avec les mêmes données :

```
> mca(data.frame(SP, HAB))
```

```
Call:
```

```
mca(df = data.frame(SP, HAB))
```

```
Multiple correspondence analysis of 50 cases of 2 factors
```

```
Correlations 0.847 0.784 cumulative % explained 84.69 163.10
```

Méthodes plus sophistiquées : `ade4` sur CRAN.

Méthodes discriminantes

Dans MASS : `lda` (*linear discriminant analysis*) et `qda` (*quadratic . . .*); toutes les deux ont une option `data` et méthode `predict` correspondante.

Ces deux fonctions cherchent des combinaisons, linéaires ou quadratiques, des variables originales qui maximisent la variance inter-groupes.

```
> iris.ad <- lda(Species ~ ., data = iris)
> #idem que: iris.ad <- lda(iris[, 1:4], iris$Species)
> plot(iris.ad)
> co <- c("blue", "red", "yellow")
> myf <- function(x, y)
+   plot(x, y, col = co[iris$Species], pch = 19)
> plot(iris.ad, panel = myf)
```

Méthodes plus sophistiquées : `mda` (*mixture discriminant analysis*) sur CRAN.

Classification hiérarchique

Calcul de distances

`dist(x, method = "euclidean")`, six méthodes disponibles (retourne un objet de classe "dist").

Package `cluster` : `daisy(x, metric = "euclidean")`, plus appropriée pour données non-continues.

Distances génétiques : `ade4` à partir de fréquences alléliques, `ape` (sur CRAN) à partir de séquences d'ADN.

Classification

`hclust(d, method = "complete")`, sept méthodes disponibles ; retourne un objet de classe "hclust" pour laquelle il y a des méthodes `print` et `plot` :

```
> x <- matrix(rnorm(500), 50, 10)
> d <- dist(x)
```

```
> str(d)
Class 'dist'  atomic [1:1225] 5.55 5.26 4.80 3.34 4.95 ...
 ..- attr(*, "Size")= int 50
 ..- attr(*, "Diag")= logi FALSE
 ..- attr(*, "Upper")= logi FALSE
 ..- attr(*, "method")= chr "euclidean"
 ..- attr(*, "call")= language dist(x = x)
> hist(d)
> hc <- hclust(d)
> hc # == print(hc)
```

Call:

```
hclust(d = d)
```

```
Cluster method      : complete
Distance            : euclidean
Number of objects: 50
```

```
> plot(hc)
```

Méthodes alternatives : `cluster`, `ade4`.

Arbres évolutifs : `ape`.

Exercices VII

1. Faire une analyse des correspondances avec les données du fichier 'santa-unix.txt' (ou 'santa-DOS.txt' suivant votre système d'exploitation). On fera attention à l'option `nf` de la fonction `corresp`.