

Graphiques de base

Pr Jean R. LOBRY

Université Claude Bernard Lyon I – France

9 septembre 2026

Table des matières

- 1 Introduction
- 2 Variables numériques
- 3 Variables qualitatives
- 4 Croisement de variables
- 5 Courbes
- 6 Les paramètres graphiques

Table des matières

- 1 Introduction
- 2 Variables numériques
- 3 Variables qualitatives
- 4 Croisement de variables
- 5 Courbes
- 6 Les paramètres graphiques

Plan détaillé

1 Introduction

- Le graphique de Charles Minard
- Importance des représentations graphiques
- Les grandes familles de fonctions graphiques

Citation

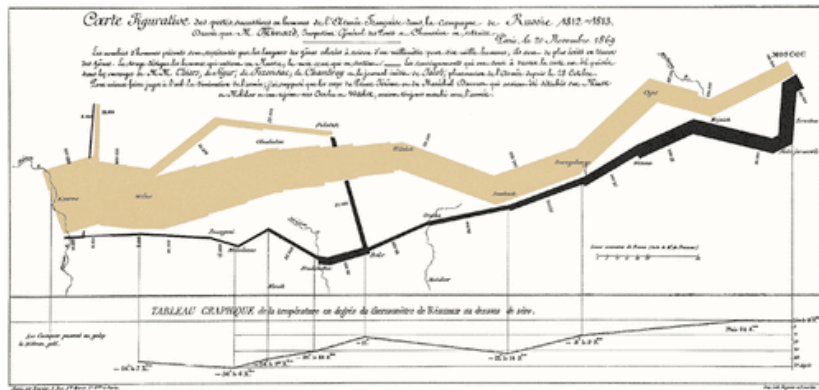


« Un bon croquis vaut mieux qu'un long discours. »

Charles Minard

- Par une ironie de l'histoire, le graphique statistique considéré par beaucoup comme étant le meilleur jamais produit illustre la désastreuse campagne de Russie conduite par Napoléon en 1812.
- Ce graphique est de l'ingénieur français Charles Minard (1781-1870).
- Le graphique représente le nombre de survivants de l'armée par l'épaisseur des bandes sur la carte de la campagne, à l'aller et au retour. La température pendant la retraite est indiquée au bas de la figure.

Le graphique original de Charles Minard



Carte figurative des pertes successives en hommes de l'armée française dans la campagne de Russie 1812-1813.

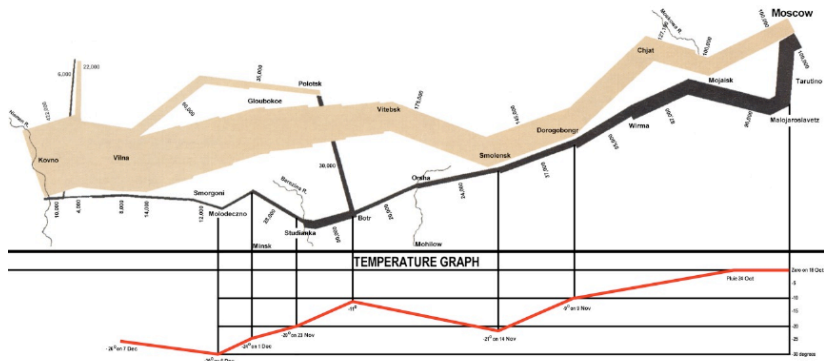
Introduction

Le graphique de Charles Minard

Reprises du graphique de Charles Minard

Voir par exemple

(http://www.math.yorku.ca/SCS/Gallery/re-minard.html).

Source : <http://www.napoleonic-literature.com/index.html>

Plan détaillé

1 Introduction

- Le graphique de Charles Minard
- Importance des représentations graphiques
- Les grandes familles de fonctions graphiques

Explosion de la navette spatiale Challenger

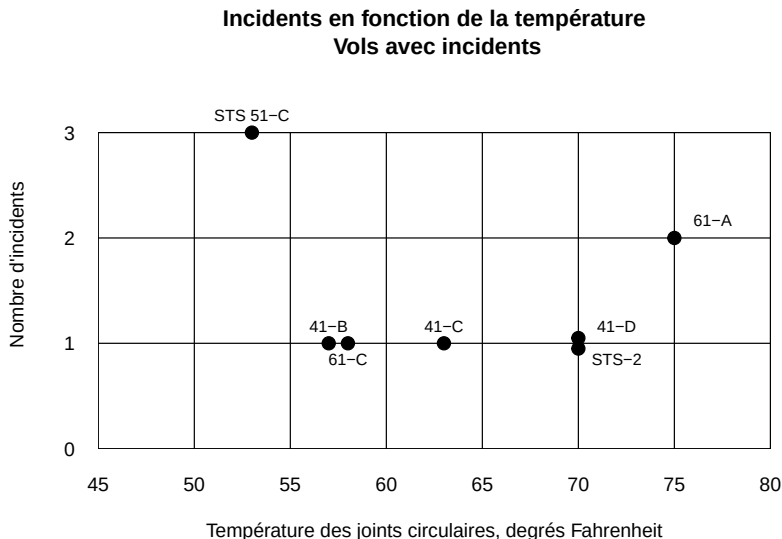
Un mauvais graphique peut avoir des conséquences catastrophiques.



On peut (aussi) faire de mauvais graphiques sous R

```
orf <- read.table("https://esb.univ-lyon1.fr/donnees/ORingFailure.txt", header=
plot(orf[orf$Failure > 0 & orf$Temperature != 70, ], pch = 19, xlim = c(45,80),
ylim = c(0,3.5), cex = 1.5,
las = 1, xlab = "Température des joints circulaires, degrés Fahrenheit", bty =
ylab = "Nombre d'incidents", xaxs = "i", yaxs = "i", xaxt = "n", yaxt = "n",
main = "Incidents en fonction de la température\n Vols avec incidents")
points(c(70,70), c(0.95,1.05), pch = 19, cex = 1.5)
axis(1, at = seq(45,80,by = 5), tick = FALSE)
axis(2, at = 0:3, las = 1, tick = FALSE)
abline(h = 0:3)
for( i in seq(45,80,by = 5)) segments(i,0,i,3)
text(53, 3, "STS 51-C", pos = 3, cex = 0.8)
text(77, 2, "61-A", pos = 3, cex = 0.8)
text(57, 1, "41-B", pos = 3, cex = 0.8)
text(58, 1, "61-C", pos = 1, cex = 0.8)
text(63, 1, "41-C", pos = 3, cex = 0.8)
text(72, 1, "41-D", pos = 3, cex = 0.8)
text(72, 1, "STS-2", pos = 1, cex = 0.8)
```

On peut (aussi) faire de mauvais graphiques sous R



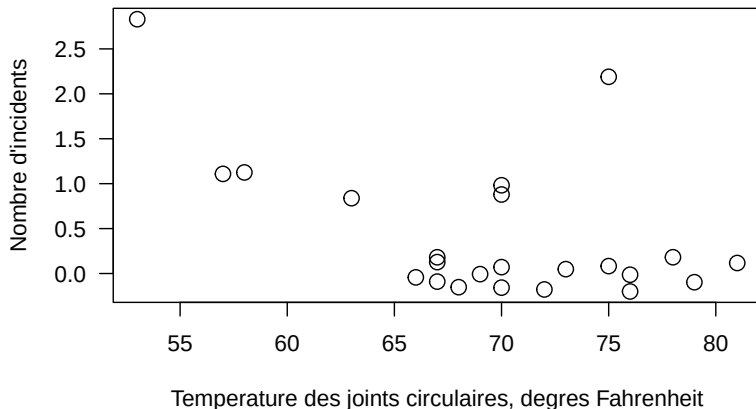
Les options par défaut des graphiques R

Les options par défaut des fonctions graphiques de  sont étudiées pour donner de bons résultats.

```
plot(orf$Temperature, jitter(orf$Failure), cex = 1.5,  
las = 1, xlab = "Temperature des joints circulaires, degres Fahrenheit",  
ylab = "Nombre d'incidents",  
main = "Vols avec et sans incidents")
```

Les options par défaut des graphiques R

Vols avec et sans incidents



Plan détaillé

1 Introduction

- Le graphique de Charles Minard
- Importance des représentations graphiques
- Les grandes familles de fonctions graphiques

Les fonctions graphiques de

 est un très bon environnement pour produire de façon reproductible des graphiques statistiques de haute qualité. On peut classer les fonctions graphiques en plusieurs catégories :

- Les fonctions liées format de sortie des graphiques.
- Les fonction permettant d'interagir avec les graphiques.
- Les fonctions graphiques de bas niveau pour retoucher un graphique existant.
- Les fonctions graphiques de haut niveau.

Les fonctions de format de sortie des graphiques

- Il existe de nombreuses fonctions pour ouvrir un nouveau périphérique graphique (e.g. `pdf()`, `jpeg()`, `postscript()`, `x11()`, `png()`, `gnome()`, `quartz()`, `xfig()`, `bitmap()`, `pictex()`).
- Elles ne sont pas toutes disponibles pour tous les systèmes d'exploitation.
- Pour en savoir plus voir ?Devices.
- Le dispositif utilisé par défaut est donné par `getOption("device")`.

Les fonctions de format de sortie des graphiques

Exemple d'utilisation pour sauvegarder un graphique dans un fichier au format PDF :

```
pdf("monfichier.pdf")  
plot(0)  
dev.off()
```

Les fonctions interactives

Ces fonctions permettent de retoucher « à la main » un graphique, tout en conservant le résultat pour sa reproductibilité ultérieure.

- `locator()` permet de récupérer les coordonnées des points lorsque en cliquant dessus.
- `identify()` permet d'identifier des points. Donne le rang des points dans le jeu de données.

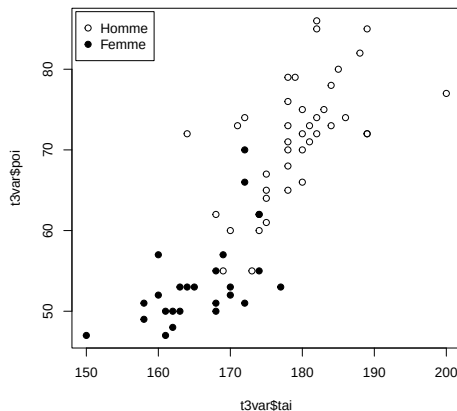
Les fonctions graphiques de bas niveau

Ces fonctions permettent de retoucher un graphique déjà existant (e.g. `points()`, `abline()`, `arrows()`, `lines()`, `segments()`, `polygon()`, `rect()`, `box()`, `axis()`, `title()`, `rug()`, `grid()`, `legend()`, `text()`, `mtext()`).

Par exemple pour ajouter une légende :

```
t3var <- read.table("https://esb.univ-lyon1.fr/donnees/t3var.txt", h = T)
plot(t3var$tai,t3var$poi, pch = ifelse(t3var$sex == "h", 1, 19))
legend("topleft", inset = 0.01, c("Homme","Femme"), pch = c(1, 19))
```

Les fonctions graphiques de bas niveau



Les fonctions graphiques de haut niveau

Ce sont celles que l'on utilise le plus souvent parce qu'elles donnent un graphique complet. Elles sont très nombreuses (e.g. `plot()`, `hist()`, `dotchart()`, `stripchart()`, `pie()`, `barplot()`, `boxplot()`, `curve()`, `sunflowerplot()`, `symbols()`, `pairs()`, `stars()`, `assocplot()`, `mosaicplot()`, `coplot()`, `contour()`, `image()`, `persp()`).

Nous allons envisager ci-après quelques fonctions graphiques de haut niveau très utilisés en analyse exploratoire des données.

Table des matières

- 1 Introduction
- 2 Variables numériques**
- 3 Variables qualitatives
- 4 Croisement de variables
- 5 Courbes
- 6 Les paramètres graphiques

Plan détaillé

2 Variables numériques

- Variables discrètes et variables continues
- Variables discrètes
- Variables continues

Variables numériques

On parle également de variables **quantitatives**, elles sont représentées par une valeur numérique (`numeric()`).

On distingue parfois :

- les variables quantitatives **discrètes**, ne pouvant prendre qu'un nombre fini de valeurs (par exemple le nombre de jambes d'un individu).
- les variables quantitatives **continues**, pouvant prendre un nombre infini de valeurs (par exemple la taille d'un individu).

Cette distinction est un peu artificielle puisque les variables continues *stricto sensu* n'existent pas à cause de la précision limitée des instruments de mesure. Illustrons ce point.

Taille de 237 étudiants

Intéressons nous à la taille de 237 étudiants disponibles dans le jeu de données survey de la bibliothèque MASS. Utilisons un **histogramme** pour représenter ces données.

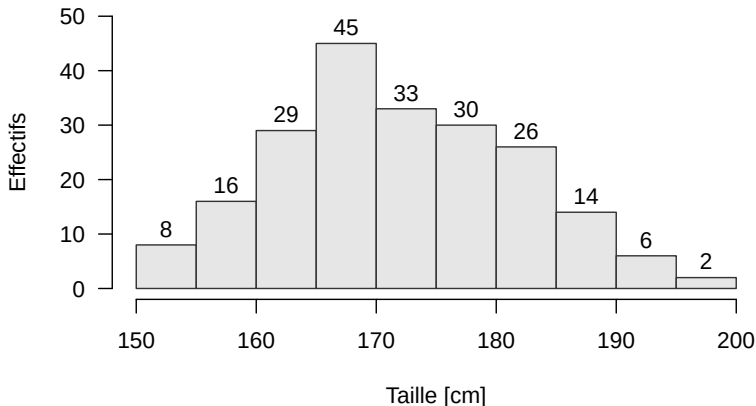
```
library(MASS)
data(survey)
names(survey)

[1] "Sex"      "Wr.Hnd"  "NW.Hnd"  "W.Hnd"   "Fold"    "Pulse"   "Clap"
[8] "Exer"     "Smoke"   "Height"  "M.I"     "Age"

hist( survey$Height, col = grey(0.9), border = grey(0.2),
      main = paste("Taille de", nrow(survey), "étudiants"),
      xlab = "Taille [cm]",
      ylab = "Effectifs",
      labels = TRUE, las = 1, ylim = c(0, 50))
```

Taille de 237 étudiants

Taille de 237 étudiants

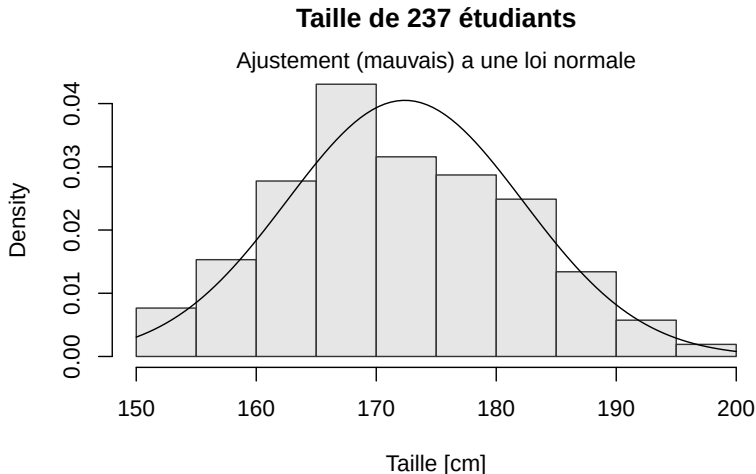


Taille de 237 étudiants

Nous avons utilisé ici des fréquences **absolues**, on préfère généralement utiliser des fréquences **relatives** (`proba = TRUE`) pour pouvoir superposer facilement des distributions de référence, par exemple :

```
hist( survey$Height, col = grey(0.9), border = grey(0.2),  
      main = paste("Taille de", nrow(survey), "étudiants"),  
      xlab = "Taille [cm]",  
      proba = TRUE)  
x <- seq(from = min(survey$Height, na.rm=T), to = max(survey$Height, na.rm=T),  
         length = 100)  
lines(x, dnorm(x, mean(survey$Height, na.rm = TRUE), sd(survey$Height,  
                                                         na.rm = TRUE)))  
mtext("Ajustement (mauvais) a une loi normale")
```

Taille de 237 étudiants



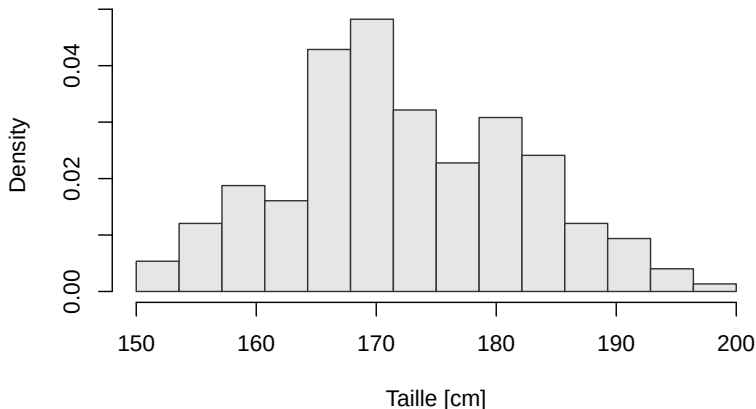
Taille de 237 étudiants

Le problème des histogrammes est que le choix du découpage en intervalles est assez arbitraire. On peut le contrôler avec le paramètre `break` de la fonction `hist()`, par exemple :

```
hist( survey$Height, col = grey(0.9), border = grey(0.2),  
      main = paste("Taille de", nrow(survey), "étudiants"),  
      xlab = "Taille [cm]",  
      proba = TRUE, breaks = seq(from = 150, to = 200, length = 15))
```

Taille de 237 étudiants

Taille de 237 étudiants



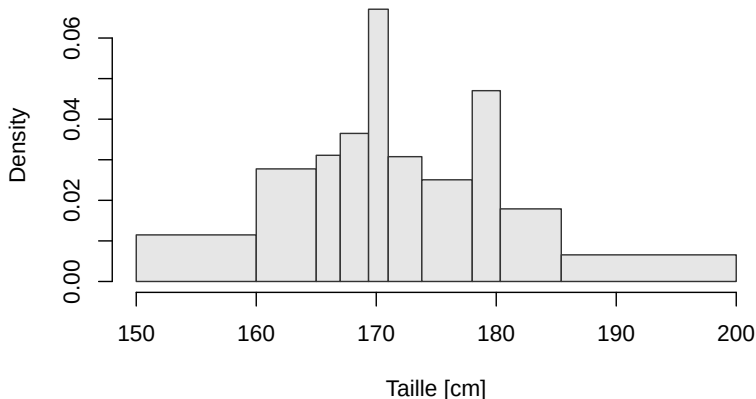
Taille de 237 étudiants

On a plus l'impression dans ce dernier cas que la distribution est bimodale. Le choix du découpage en intervalle est un problème délicat qui risque de biaiser fortement notre perception des données. Avec des intervalles de même effectifs on aurait :

```
isohist <- function(x, nclass, ...){  
  breaks <- quantile(x, seq(from = 0, to = 1, length = nclass + 1),  
                     na.rm = TRUE)  
  invisible(hist(x, breaks = breaks, ...))  
}  
isohist(survey$Height, 10, col = grey(0.9), border = grey(0.2),  
        main = paste("Taille de", nrow(survey), "étudiants"),  
        xlab = "Taille [cm]",  
        proba = TRUE)
```

Taille de 237 étudiants

Taille de 237 étudiants



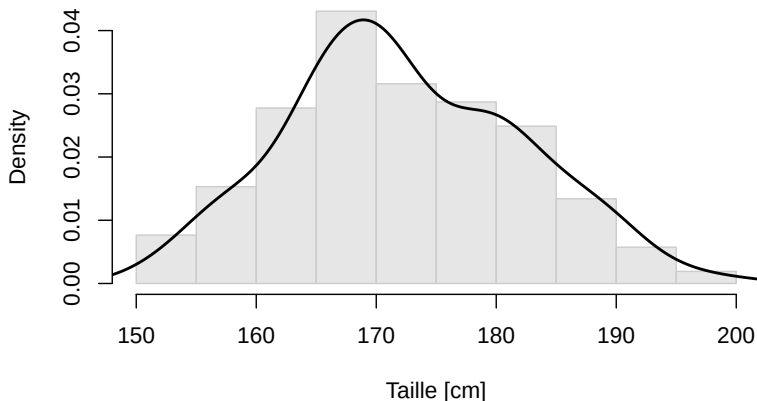
Taille de 237 étudiants

De nos jours, on préfère utiliser des estimateurs locaux de la densité des points et explorer différentes échelles. Par exemple :

```
hist( survey$Height, col = grey(0.9), border = grey(0.8),  
      main = paste("Taille de", nrow(survey), "étudiants"),  
      xlab = "Taille [cm]",  
      proba = TRUE)  
lines(density(survey$Height, na.rm = TRUE), lwd = 2)
```

Taille de 237 étudiants

Taille de 237 étudiants

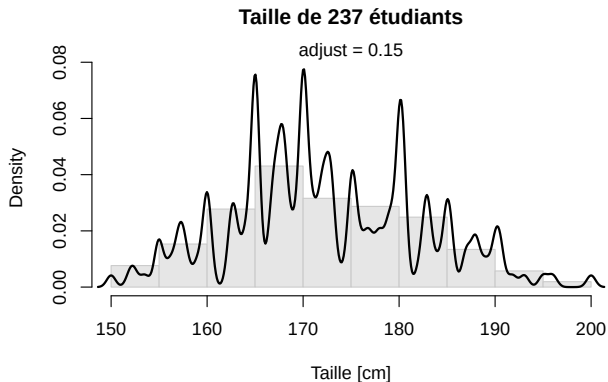


Taille de 237 étudiants

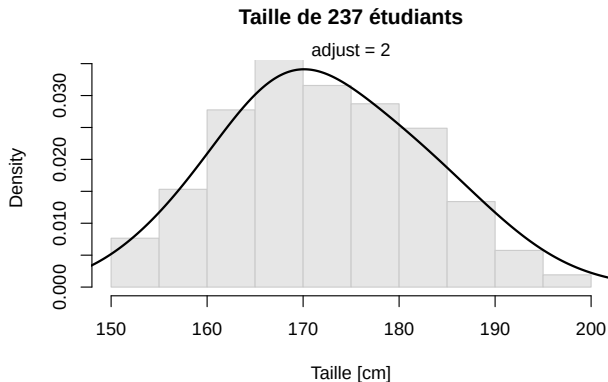
Le paramètre important de la fonction `density()` est le paramètre `adjust` :

- `adjust = 1` C'est la valeur par défaut, celle que nous avons utilisée dans le graphe précédent.
- `adjust < 1` On regarde les choses de près, on va vers la nature discrète de la variable.
- `adjust > 1` On regarde les choses de loin, on veut lisser le signal pour voir la variable comme étant continue.

Taille de 237 étudiants



Taille de 237 étudiants



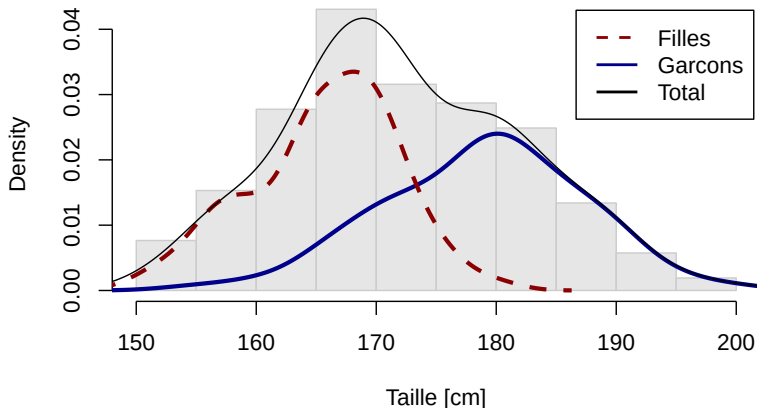
Taille de 237 étudiants

Un autre avantage des estimateurs locaux de la densité par rapport aux histogrammes est qu'il permettent de superposer facilement plusieurs distributions :

```
par(lend="butt")
ng <- sum(survey$Sex == "Male", na.rm = TRUE)
nf <- sum(survey$Sex == "Female", na.rm = TRUE)
n <- ng + nf
dst <- density(survey$Height, na.rm = TRUE)
dstg <- density(survey$Height[survey$Sex == "Male"], na.rm = TRUE)
dstf <- density(survey$Height[survey$Sex == "Female"], na.rm = TRUE)
hist( survey$Height, col = grey(0.9), border = grey(0.8),
      main = paste("Taille de", nrow(survey), "étudiants"),
      xlab = "Taille [cm]",
      proba = TRUE, ylim = c(0, max(dst$y)))
lines(dstg$x, ng/n*dstg$y, lwd = 3, col = "darkblue")
lines(dstf$x, nf/n*dstf$y, lwd = 3, lty = 2, col = "darkred")
lines(dst$x, dst$y)
legend("topright", inset = 0.01, legend = c("Filles", "Garçons", "Total"),
      col = c("darkred", "darkblue", "black"),
      lty = c(2, 1, 1), lwd = 2, pt.cex = 2)
```

Taille de 237 étudiants

Taille de 237 étudiants



Plan détaillé

2 Variables numériques

- Variables discrètes et variables continues
- Variables discrètes
- Variables continues

Diagramme en bâtons

Quand la nature discrète de la variable étudiée ne fait pas de doute, on utilise en général un diagramme en bâtons :

```
library(ade4)
data(deug)
plot(table(deug$tab$Option1), main = paste("Notes de", nrow(deug$tab),
                                           "étudiants"),
     las = 1, xlab = "note (Option 1)", ylab = "Nombre d'étudiants")
```

Diagramme en bâtons

Notes de 104 étudiants

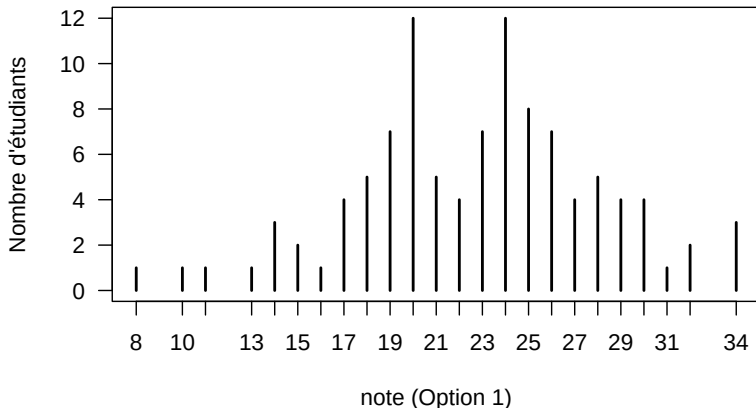


Diagramme en bâtons

On peut considérer les diagramme en bâtons comme une forme dégénérée des estimateurs locaux de densité quand le paramètre `adjust` est très petit :

```
plot(table(deug$tab$Option1), main = paste("Notes de", nrow(deug$tab),  
                                           "étudiants"),  
     las = 1, xlab = "note (Option 1)", ylab = "Nombre d'étudiants")  
dst <- density(deug$tab$Option1, adjust = 0.1)  
lines(dst$x, max(table(deug$tab$Option1))*dst$y/max(dst$y), col = "red")
```

Diagramme en bâtons

Notes de 104 étudiants

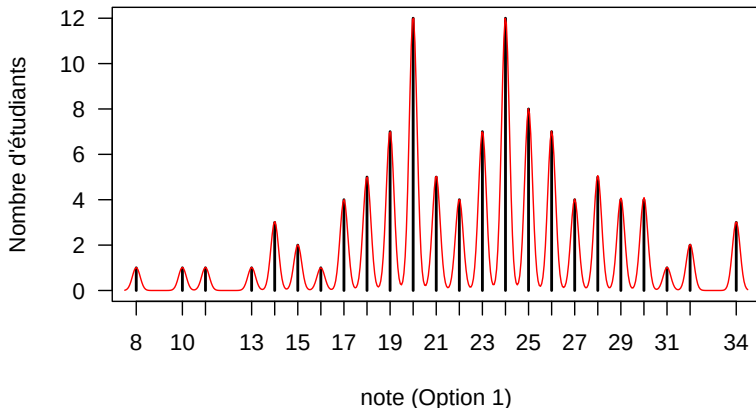


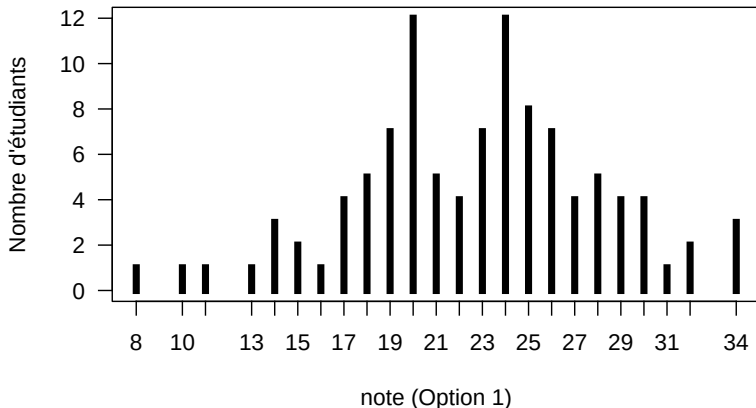
Diagramme en bâtons

Les paramètres graphiques **lend** (line end : fin des lignes) et **lwd** (line width : épaisseur des lignes) permettent de contrôler l'aspect terminal des bâtons et l'épaisseur des bâtons :

```
plot(table(deug$tab$Option1), main = paste("Notes de", nrow(deug$tab),  
                                           "étudiants"),  
     las = 1, xlab = "note (Option 1)", ylab = "Nombre d'étudiants",  
     lwd = 5, lend = "square")
```

Diagramme en bâtons

Notes de 104 étudiants



Plan détaillé

2 Variables numériques

- Variables discrètes et variables continues
- Variables discrètes
- Variables continues

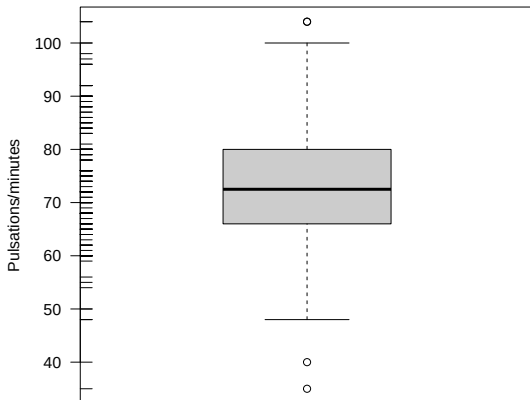
Boîte à moustaches

Nous avons déjà vu l'utilisation pour les variables continues des histogrammes et des estimateurs de la densité locale. On peut également utiliser une représentation en **boîte à moustaches** :

```
boxplot( survey$Pulse, col = grey(0.8),  
        main = paste("Rythme cardiaque de", nrow(survey), "étudiants"),  
        ylab = "Pulsations/minutes", las = 1)  
rug(survey$Pulse, side = 2)
```

Boîte à moustaches

Rythme cardiaque de 237 étudiants



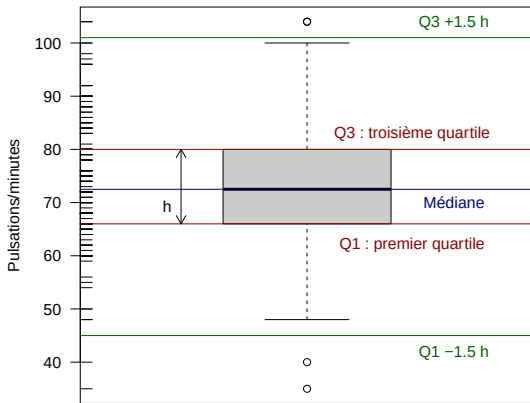
Boîte à moustaches

La signification est explicitée dans le graphique suivant :

```
boxplot( survey$Pulse, col = grey(0.8),
        main = paste("Rythme cardiaque de", nrow(survey), "étudiants"),
        ylab = "Pulsations/minutes", las = 1)
rug(survey$Pulse, side = 2)
abline( h = median(survey$Pulse, na.rm = TRUE), col = "navy")
text(1.35, 70, "Médiane", col = "navy")
Q1 <- quantile(survey$Pulse, probs = 0.25, na.rm = TRUE)
abline( h = Q1, col = "darkred")
text(1.25, 62, "Q1 : premier quartile", col = "darkred")
Q3 <- quantile(survey$Pulse, probs = 0.75, na.rm = TRUE)
abline( h = Q3, col = "darkred")
text(1.25, 83, "Q3 : troisième quartile", col = "darkred")
arrows(x0 = 0.7, y0 = quantile(survey$Pulse, probs = 0.75, na.rm = TRUE), x1 =
y1 = quantile(survey$Pulse, probs = 0.25, na.rm = TRUE), length = 0.1, code =
text(0.7, 69, "h", pos = 2)
mtext("L'écart inter-quartile h contient 50 % des individus", side = 1)
abline( h = Q1-1.5*(Q3-Q1), col = "darkgreen")
text(1.35, 42, "Q1 -1.5 h", col = "darkgreen")
abline( h = Q3+1.5*(Q3-Q1), col = "darkgreen")
text(1.35, 104, "Q3 +1.5 h", col = "darkgreen")
```

Boîte à moustaches

Rythme cardiaque de 237 étudiants



L'écart inter-quartile h contient 50 % des individus

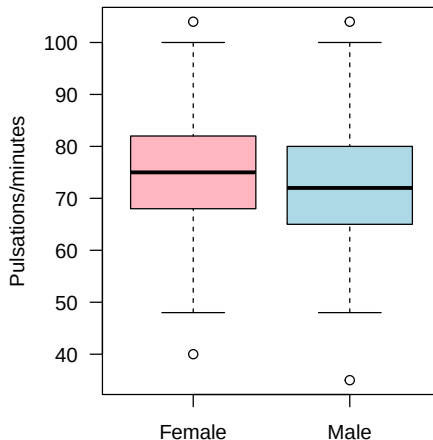
Boîte à moustaches

Les boîtes à moustaches permettent de comparer facilement des groupes d'individus, par exemple ici les garçons et les filles :

```
boxplot(survey$Pulse~survey$Sex, col = c("lightpink","lightblue"),  
        main = paste("Rythme cardiaque de", nrow(survey), "étudiants"),  
        ylab = "Pulsations/minutes", las = 1)
```

Boîte à moustaches

Rythme cardiaque de 237 étudiants



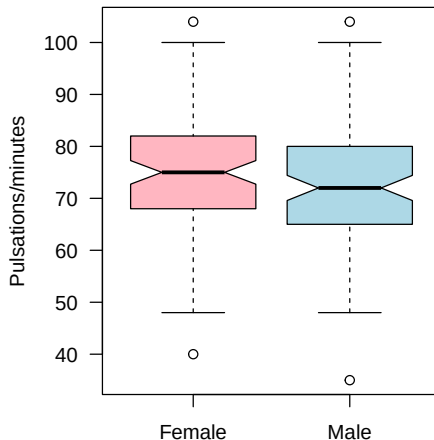
Boîte à moustaches

Une extension intéressante des boîtes à moustaches consiste à ajouter des encoches pour représenter un intervalle de confiance pour la médiane de chaque groupe :

```
boxplot( survey$Pulse~survey$Sex, col = c("lightpink","lightblue"),  
        main = paste("Rythme cardiaque de", nrow(survey), "étudiants"),  
        ylab = "Pulsations/minutes", las = 1, notch = TRUE)
```

Boîte à moustaches

Rythme cardiaque de 237 étudiants

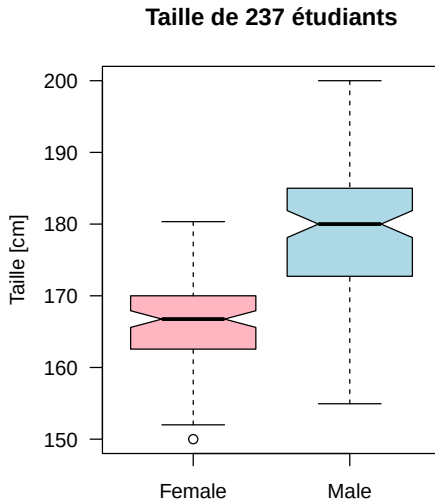


Boîte à moustaches

Ainsi, nous n'avons pas d'indication forte qu'il y ait une différence du rythme cardiaque entre les filles et les garçons. Pour ce qui est de la taille on trouve que les garçons sont significativement plus grands que les filles.

```
boxplot( survey$Height~survey$Sex, col = c("lightpink","lightblue"),  
  main = paste("Taille de", nrow(survey), "étudiants"),  
  ylab = "Taille [cm]", las = 1, notch = TRUE)
```

Boîte à moustaches



Diagrammes en violon

Les diagrammes en violon essayent de combiner les avantages des boîtes à moustaches et des estimateurs de la densité locale, par exemple :

```
library(sm)
library(vioplot)
par(las = 1)
vioplot(survey$Height[!is.na(survey$Height)], h = 1.5, names = "",
        col = "lightblue")
title(main = paste("Taille de", nrow(survey), "étudiants"))
```

Diagrammes en violon

Taille de 237 étudiants

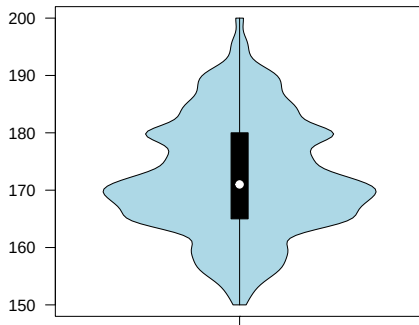


Table des matières

- 1 Introduction
- 2 Variables numériques
- 3 Variables qualitatives**
- 4 Croisement de variables
- 5 Courbes
- 6 Les paramètres graphiques

Plan détaillé

- 3 Variables qualitatives
 - Variables nominales et ordonnées
 - Variables qualitatives nominales
 - Variables qualitatives ordinales

Variables qualitatives nominales et ordonnées

Les variables **qualitatives** sont toutes les variables à valeur non numérique (e.g. bleu, blanc, rouge) et codées par les `factor()`. Les variables qualitatives peuvent être :

- **nominales** sans ordre particulier : un simple nom.
- **ordonnées** avec un ordre : un peu, beaucoup, passionnément, à la folie.

La règle est simple : s'il y a un ordre, vos graphiques doivent **impérativement** le respecter.

Plan détaillé

- 3 Variables qualitatives
 - Variables nominales et ordonnées
 - Variables qualitatives nominales
 - Variables qualitatives ordinales

Variables qualitatives

Variables qualitatives nominales

592 étudiants

Nous allons illustrer le cas des variables qualitatives non ordonnées avec un jeu de données réelles portant sur 592 étudiants (extrait de Snee, R. D. (1974) Graphical display of two-way contingency tables. *The American Statistician*, **28** :9-12). Pour chaque étudiant on a observé 3 variables qualitatives : la couleur des cheveux, la couleur des yeux et le sexe.

```
genet <- read.table("https://esb.univ-lyon1.fr/donnees/qualitatif.txt",
                    header=TRUE)
```

```
summary(genet)
```

cheveux		yeux		sexe	
Length	:592	Length	:592	Length	:592
N.unique	: 4	N.unique	: 4	N.unique	: 2
N.blank	: 0	N.blank	: 0	N.blank	: 0
Min.nchar	: 4	Min.nchar	: 4	Min.nchar	: 4
Max.nchar	: 6	Max.nchar	: 8	Max.nchar	: 7

Diagrammes en secteurs

Intéressons nous à la couleur des cheveux. On peut représenter les données sous la forme d'un diagramme en secteurs avec la fonction `pie()` :

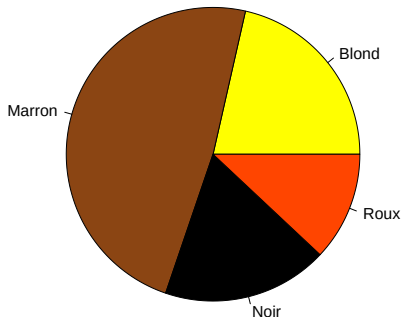
```
pie(table(genet$cheveux), col = c("yellow", "chocolate4", "black", "orangered")  
     main = "Couleur des cheveux de 592 étudiants")
```

Variables qualitatives

Variables qualitatives nominales

Diagrammes en secteurs

Couleur des cheveux de 592 étudiants



Diagrammes en secteurs

Mais c'est une mauvaise idée. En effet, la documentation de la fonction `pie()` nous dit :

Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.

On peut s'en convaincre facilement à l'aide de l'exemple suivant :

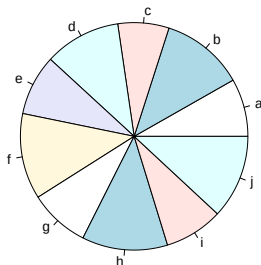
```
set.seed(01071966)
data <- rep(10,10) + rep( 2*runif(5), rep(2,5)) + rep(c(-2,2),5)
data <- 100*data/sum(data) # as percentage
names(data) <- letters[1:10]
par(mfrow = c(1, 2))
pie(data, main = "Diagramme en secteur")
dotchart(data, xlim = c(0, 14), pch = 19, main = "Graphe de Cleveland")
par(mfrow = c(1, 1))
```

Variables qualitatives

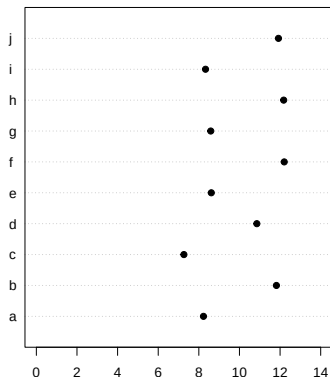
Variables qualitatives nominales

Diagrammes en secteurs - graphe de Cleveland

Diagramme en secteur



Graphe de Cleveland



Diagrammes en secteurs - graphe de Cleveland

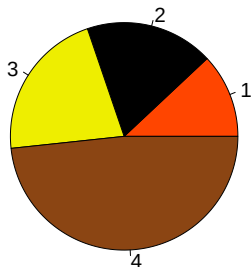
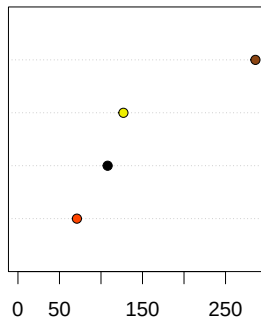
Il faut donc éviter d'utiliser des diagrammes en secteurs. Avec notre exemple sur la couleur des cheveux de 592 étudiants, On voit par exemple que l'écart entre les Noir et les Roux est plus important que l'écart entre les Blond et les Noir. On ne le voit pas avec un diagramme en secteurs.

```
data <- sort(as.numeric(table(genet$cheveux)))
par(mfrow = c(1, 2))
pie(data, col = c("orangered", "black", "yellow2", "chocolate4"), cex = 1.5)
dotchart(data, xlim = c(0, max(data)), pch = 21, bg = c("orangered",
  "black", "yellow2", "chocolate4"), cex = 1.5)
par(mfrow=c(1,1))
```

Variables qualitatives

Variables qualitatives nominales

Diagrammes en secteurs - graphe de Cleveland



Graphe de Cleveland

Quand le nombre de modalités est important, on a tout intérêt à les classer en fonction de leur fréquence pour faciliter les comparaisons. Illustrons ceci avec un jeu de données consistant en un échantillon de 93 voitures sur lesquelles on a observé une variable qualitative non-ordonnée : le nom du constructeur.

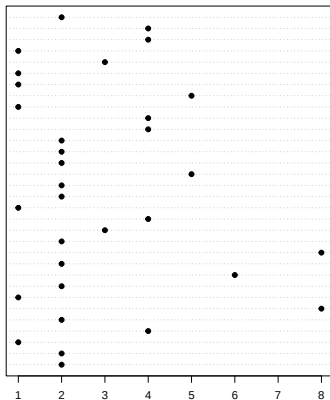
```
library(MASS)
data(Cars93)
par(mfrow=c(1,2))
dotchart(as.numeric(table(Cars93$Manufacturer)), pch=19, cex = 0.8,
          main="Dans le desordre")
dotchart(sort(as.numeric(table(Cars93$Manufacturer))), pch=19, cex = 0.8,
          main = "Dans l'ordre")
par(mfrow=c(1,1))
```

Variables qualitatives

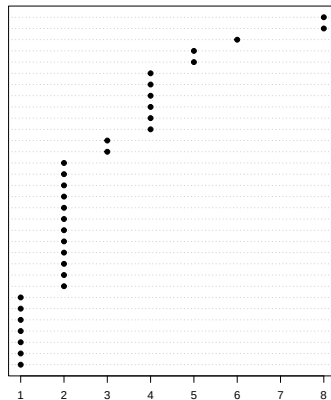
Variables qualitatives nominales

Graphe de Cleveland

Dans le desordre



Dans l'ordre



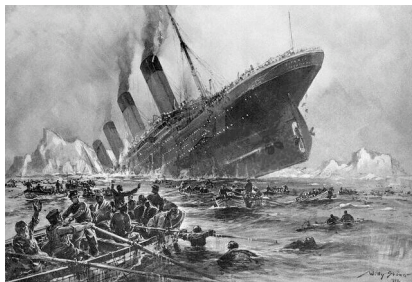
Plan détaillé

- 3 Variables qualitatives
 - Variables nominales et ordonnées
 - Variables qualitatives nominales
 - Variables qualitatives ordinales

Variables qualitatives

Variables qualitatives ordinales

Variables qualitatives ordinales



Les passagers du Titanic pouvaient voyager en première classe (1st), en seconde classe (2nd) ou en troisième classe (3rd). La variable classe est donc une variable qualitative **ordinales** dont les trois **modalités** sont 1st, 2nd et 3rd.

Variables qualitatives ordinales

Récupérons les données historiques nous donnant le nombre de voyageur de chaque classe :

```
data(Titanic)
classe <- apply(Titanic, 1, sum)[1:3]
classe
```

1st	2nd	3rd
325	285	706

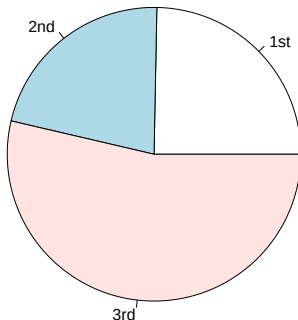
Il y avait donc 323 passagers en première classe, 285 en seconde classe et 706 en troisième classe.

Variables qualitatives

Variables qualitatives ordinales

Horreur ! La première classe jouxte la troisième classe !

Don't try this at home kids!



Variables qualitatives ordinales

Il faut préserver l'ordre des modalités, c'est facile avec un diagramme de Cleveland :

```
dotchart(rev(classe), main="Classe des passagers du Titanic", pch = 19,  
xlim = c(0,max(classe)), cex = 1.5)
```

Graphe de Cleveland

Classe des passagers du Titanic

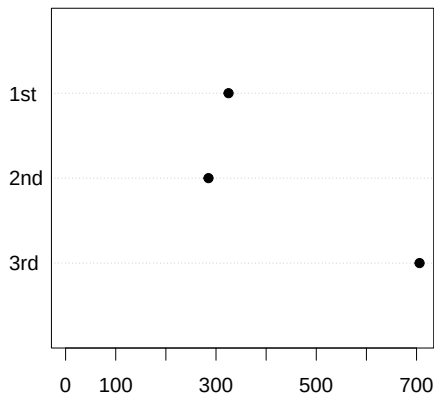


Table des matières

- 1 Introduction
- 2 Variables numériques
- 3 Variables qualitatives
- 4 Croisement de variables**
- 5 Courbes
- 6 Les paramètres graphiques

Plan détaillé

- 4 Croisement de variables
 - Croisement qualitatif qualitatif
 - Croisement numérique qualitatif
 - Croisement numérique numérique

Tables de contingence

Quand on croise deux variables qualitatives on obtient une table de contingence, par exemple :

```
(tc <- table(genet[,1:2]))
```

cheveux	yeux			
	Bleu	Marron	Noisette	Vert
Blond	94	7	10	16
Marron	84	119	54	29
Noir	20	68	15	5
Roux	17	26	14	14

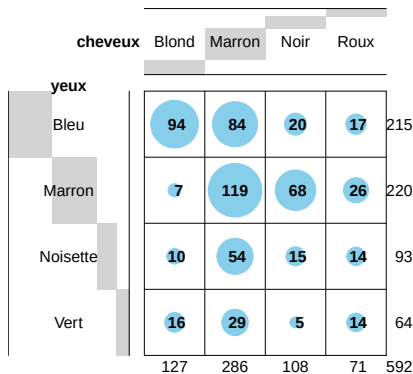
Tables de contingence

On peut utiliser la fonction `balloonplot()` pour faire une représentation directe des données :

```
library(gdata)
library(gtools)
library(gplots)
balloonplot(tc, dotsize=10)
```

Tables de contingence : représentation directe

Balloon Plot for x by y.
Area is proportional to Freq.



Tables de contingence : hypothèse d'indépendance

Sous l'hypothèse d'indépendance entre les deux variables, on déduit facilement les effectifs attendus :

$$P(A \wedge B) = P(A)P(B) \iff P(A \wedge B) = \frac{n_A}{n} \frac{n_B}{n} \iff n_{A \wedge B} = \frac{n_A n_B}{n}$$

Graphiquement :

```
par(mfrow = c(1,2))  
balloonplot(tc, dotsize=8, main = "Observé")  
balloonplot(as.table(chisq.test(tc)$expected), dotsize=8, main = "Attendu")
```

Tables de contingence : hypothèse d'indépendance

Observé

		cheveux				
		Blond	Marron	Noir	Roux	
yeux	Bleu	94	84	20	17	215
	Marron	7	119	68	26	220
	Noisette	10	54	15	14	93
	Vert	16	29	5	14	64
		127	286	108	71	592

Attendu

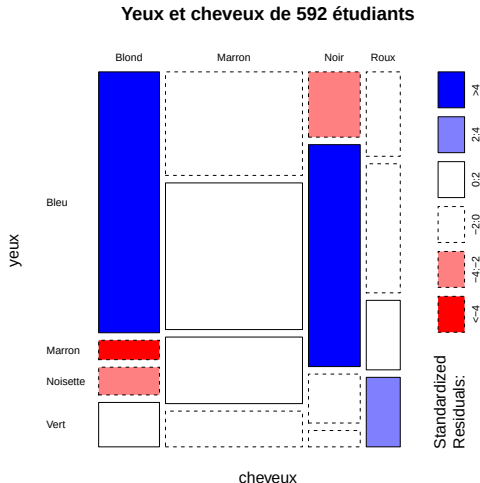
		cheveux				
		Blond	Marron	Noir	Roux	
yeux	Bleu	46.1	103.9	39.2	25.8	215
	Marron	47.2	106.3	40.1	26.4	220
	Noisette	20.0	44.9	17.0	11.2	93
	Vert	13.7	30.9	11.7	7.7	64
		127	286	108	71	592

Tables de contingence : hypothèse d'indépendance

Les représentations directes ne permettent pas de faire facilement des comparaisons. Pour visualiser les **écarts** par rapport à l'hypothèse d'indépendance, on peut utiliser la fonction `mosaicplot()` :

```
mosaicplot(tc, shade = TRUE, las =1,  
main = paste("Yeux et cheveux de", nrow(genet),"étudiants"))
```

Tables de contingence : mosaicplot



Plan détaillé

- 4 Croisement de variables
 - Croisement qualitatif qualitatif
 - Croisement numérique qualitatif
 - Croisement numérique numérique

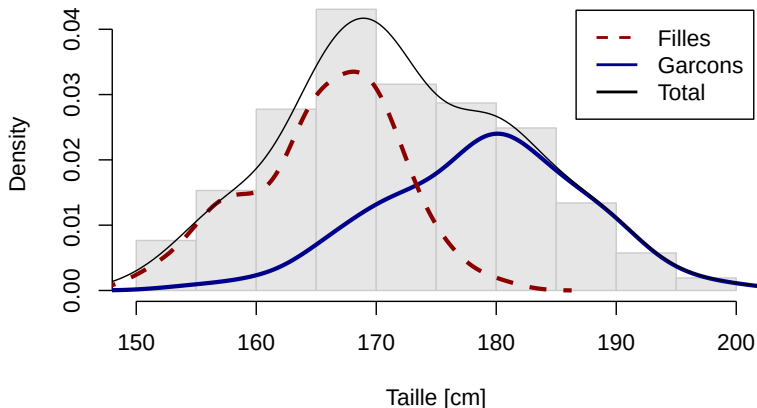
Croisement numérique qualitatif

Nous avons vu deux façon de représenter la distribution d'une variable numérique en fonction d'une variable qualitative :

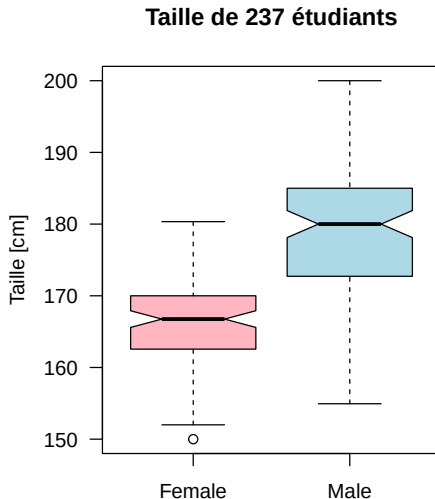
- ➊ Avec des estimateurs de la densité locale.
- ➋ Avec des boîtes à moustache.

Croisement numérique qualitatif : density()

Taille de 237 étudiants



Croisement numérique qualitatif : boxplot()



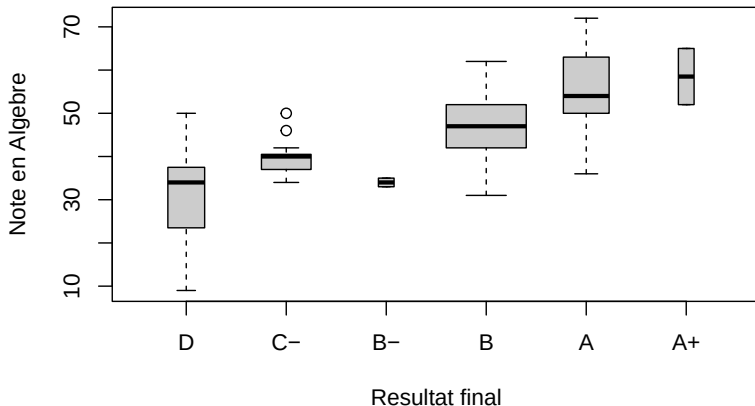
l'option varwidth des boîtes à moustaches

A noter l'option `varwidth = TRUE` qui permet de conserver graphiquement l'information sur les effectifs des groupes dans le cas d'une représentation avec des boîtes à moustaches.

```
boxplot(deug$tab$Algebra~deug$result, at = c(1, 5, 6, 2, 3, 4),  
        col = grey(0.8), xlab = "Resultat final", ylab = "Note en Algebre",  
        varwidth = TRUE,  
        main = paste("Notes de", nrow(deug$tab), "étudiants"))
```

Croisement numérique qualitatif : boxplot()

Notes de 104 étudiants



Plan détaillé

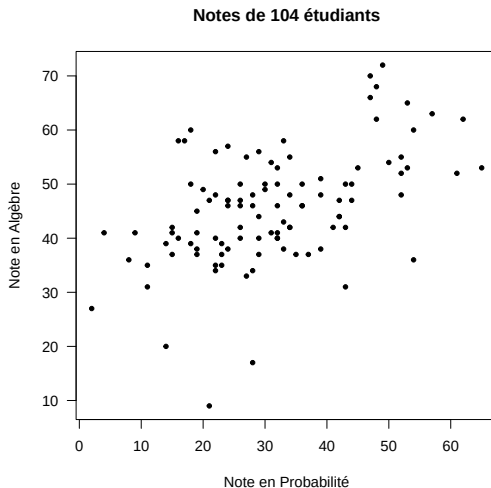
- 4 Croisement de variables
 - Croisement qualitatif qualitatif
 - Croisement numérique qualitatif
 - Croisement numérique numérique

Les nuages de points

On utilise classiquement un nuage de points, où chaque point représente un individu. Exemple :

```
plot(x = deug$tab$Proba, y = deug$tab$Algebra, pch = 20,  
main = paste("Notes de", nrow(deug$tab), "étudiants"),  
xlab = "Note en Probabilité", ylab = "Note en Algèbre", las = 1)
```

Les nuages de points

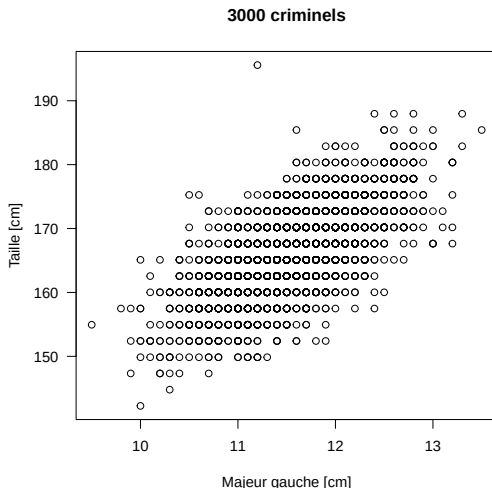


Le problème de la superposition des points

La principale difficulté des nuages de points vient de la gestion de la superposition des points. Nous allons illustrer ce point avec un exemple simple : la taille et la longueur du majeur de 3000 criminels (les données du test *t* de Student).

```
data(crimtab)
crimtab.dft <- as.data.frame(crimtab)
expand.dft <- function(x, na.strings = "NA", as.is = FALSE, dec = ".") {
  DF <- sapply(1:nrow(x), function(i) x[rep(i, each = x$Freq[i]), ],
              simplify = FALSE)
  DF <- subset(do.call("rbind", DF), select = -Freq)
  for (i in 1:ncol(DF))
  {
    DF[[i]] <- type.convert(as.character(DF[[i]]),
                           na.strings = na.strings,
                           as.is = as.is, dec = dec)
  }
  DF
}
crimtab.raw <- expand.dft(crimtab.dft)
x <- crimtab.raw[,1]
y <- crimtab.raw[,2]
plot(x, y, las = 1, main = "3000 criminels", ylab = "Taille [cm]",
     xlab = "Majeur gauche [cm])")
```

Un nuage de points trompeur



Tournesol (sunflower)

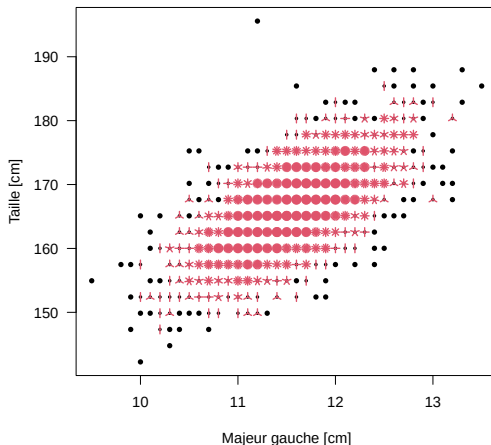


Premier palliatif possible : utiliser la représentation donnée par `sunflowerplot()` : en cas de superposition on trace autant de rayons qu'il y a de superpositions, d'où le nom de tournesol.

```
sunflowerplot(crimtab.raw, las = 1, main = "3000 criminels",  
ylab = "Taille [cm]", xlab = "Majeur gauche [cm]", size = 1/20)
```

Tournesol

3000 criminels



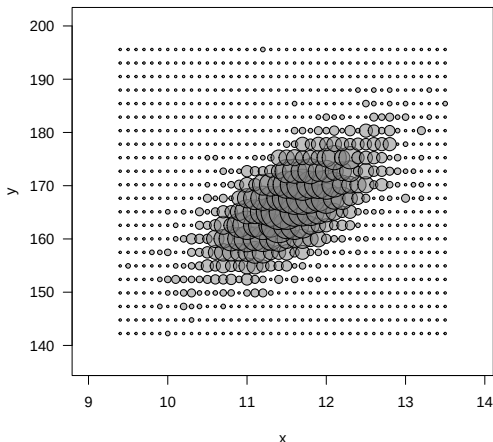
Le problème de la superposition des points

Deuxième palliatif possible : utiliser des symboles dont la **surface** est proportionnelle au nombre de points superposés.

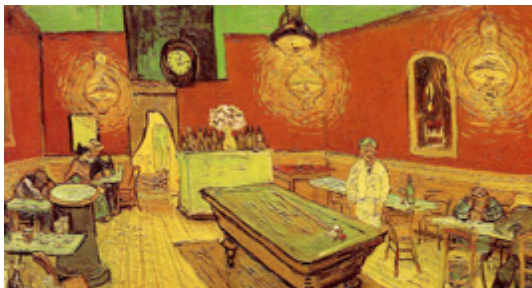
```
xyg <- expand.grid(as.numeric(rownames(crimtab)),  
                  as.numeric(colnames(crimtab)))  
symbols(x = xyg[,1], y = xyg[,2], circles = sqrt(as.vector(crimtab)),  
        inches = 0.2, bg = rgb(0.5,0.5,0.5,0.5), xlab = "x", ylab = "y", las=1,  
        main = "Avec des symboles de taille variable")
```

Le problème de la superposition des points

Avec des symboles de taille variable



Le problème de la superposition des points

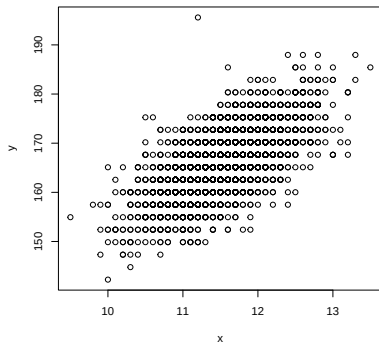


Troisième palliatif possible : **bruit** les données ! C'est une solution qui semble paradoxale à première vue puisque l'on dégrade l'information mais qui donne souvent de bons résultats du point de vue de la perception globale de l'information.

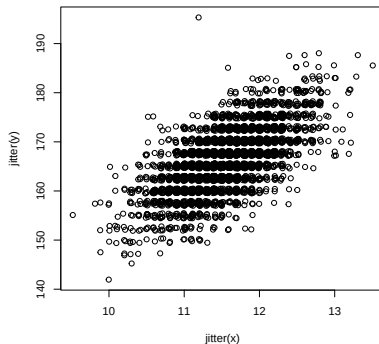
```
par(mfrow = c(1,2))  
plot(x, y, main = "Sans bruitage")  
plot(jitter(x), jitter(y), main = "Avec bruitage")
```

De l'intérêt de bruitez les données : jitter()

Sans bruitage



Avec bruitage



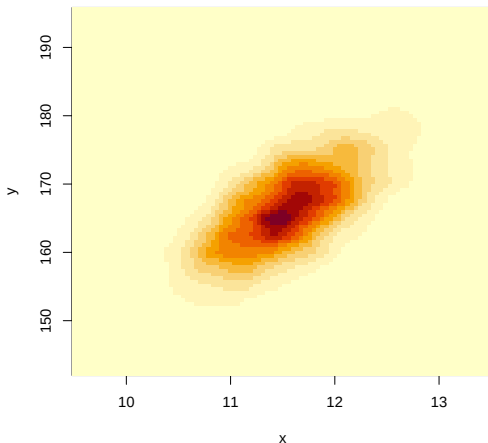
Le problème de la superposition des points

Quatrième palliatif possible : utiliser un estimateur de la densité locale des points (comme `density()` mais en 2 dimensions). On peut utiliser la fonction `kde2d()` de la bibliothèque MASS.

```
ed1 <- kde2d(x,y, n = 100)
image(ed1, main = "Avec un estimateur de la densite locale",
      xlab = "x", ylab = "y")
```

Avec image()

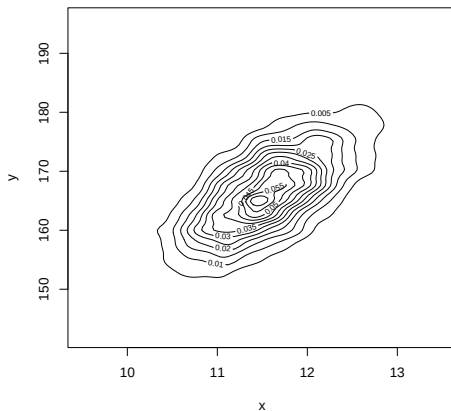
Avec un estimateur de la densité locale



Avec contour()

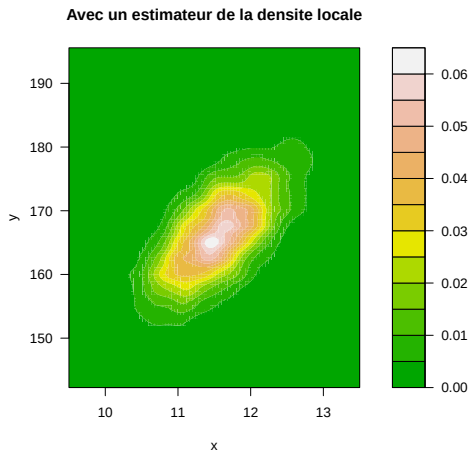
```
contour(edl, main = "Avec un estimateur de la densite locale", xlab = "x",  
        ylab = "y")
```

Avec un estimateur de la densite locale



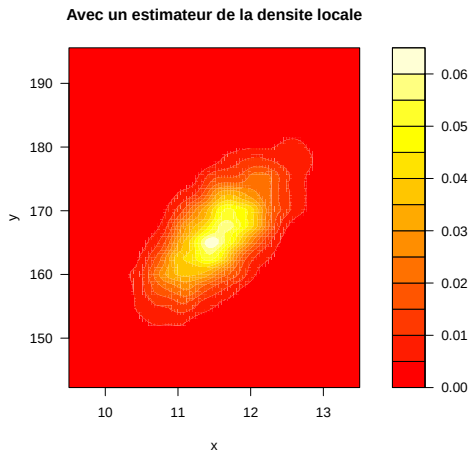
Avec `filled.contour()` et `terrain.colors`

```
filled.contour(edl, main = "Avec un estimateur de la densite locale",  
              xlab = "x", ylab = "y", color = terrain.colors)
```



Avec filled.contour() et heat.colors

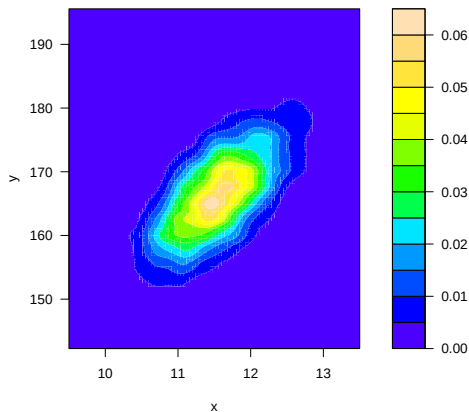
```
filled.contour(edl, main = "Avec un estimateur de la densite locale",  
              xlab = "x", ylab = "y", color = heat.colors)
```



Avec filled.contour() et topo.colors

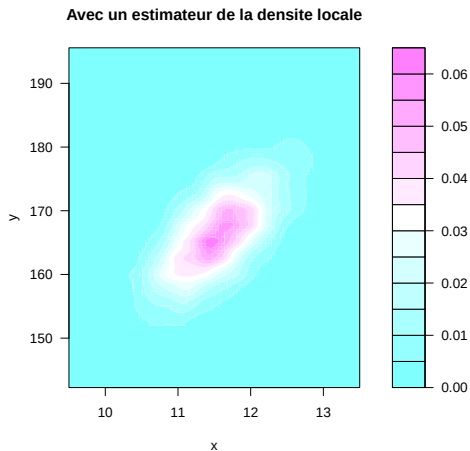
```
filled.contour(edl, main = "Avec un estimateur de la densite locale",  
              xlab = "x", ylab = "y", color = topo.colors)
```

Avec un estimateur de la densite locale



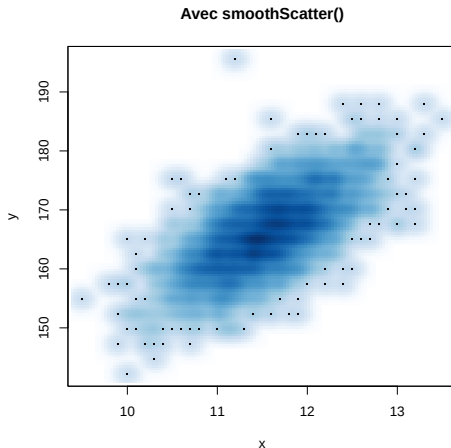
Avec filled.contour() et cm.colors

```
filled.contour(edl, main = "Avec un estimateur de la densite locale",  
              xlab = "x", ylab = "y", color = cm.colors)
```



Avec smoothScatter()

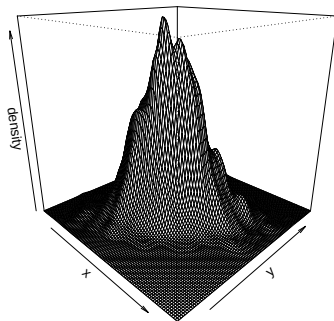
```
smoothScatter(x, y, main = "Avec smoothScatter()")
```



Avec persp()

```
persp(edl, xlab = "x", ylab = "y", zlab = "density", theta = 45, phi = 20,  
main = "Avec un estimateur de la densite locale")
```

Avec un estimateur de la densite locale



Avec persp3d()

Pour manipuler des graphiques 3D on peut utiliser les fonctions de la bibliothèque `rgl`.

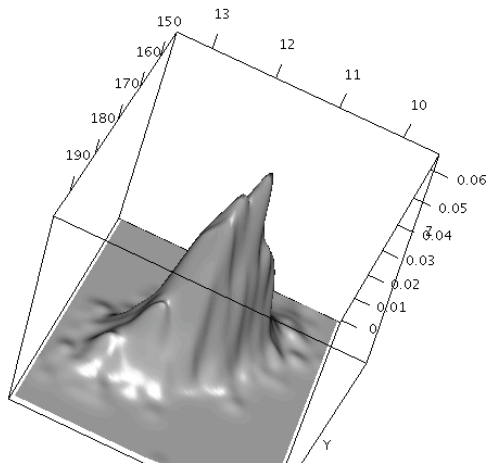


Table des matières

- 1 Introduction
- 2 Variables numériques
- 3 Variables qualitatives
- 4 Croisement de variables
- 5 Courbes**
- 6 Les paramètres graphiques

Plan détaillé

5 Courbes

- Tracé d'une fonction
- Série "Temporelles"

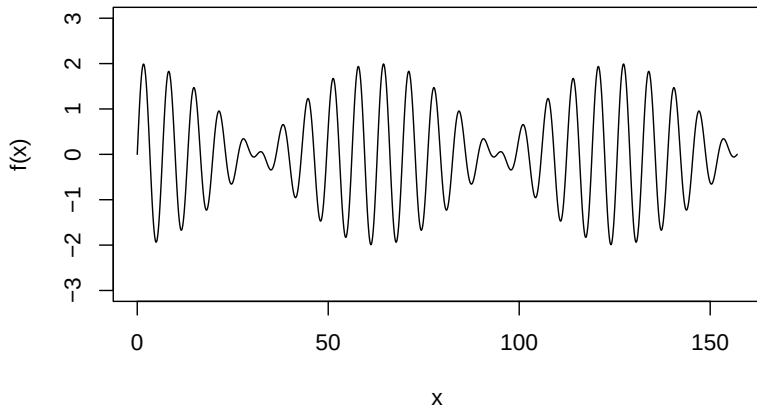
Tracé d'une fonction

On peut utiliser `curve()` pour représenter des fonctions :

```
f <- function(x) { sin(x) + sin(0.9*x) }  
curve(f, from = 0, to = 50*pi, n = 1000, main = "Battements", ylim = c(-3,3))
```

Tracé d'une fonction

Battements



Plan détaillé

5 Courbes

- Tracé d'une fonction
- Série "Temporelles"

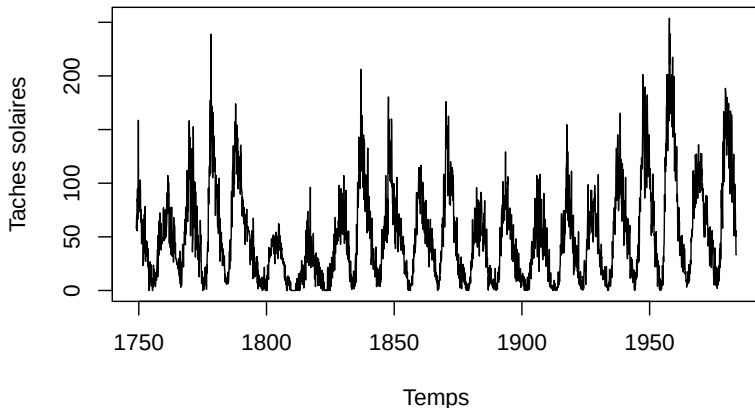
Série "Temporelles"

On peut utiliser des courbes pour représenter l'évolution d'une variable au cours du temps (où d'une autre variable) :

```
data(sunspots)
plot(sunspots, main = "Évolution de la densité de taches solaires",
     xlab = "Temps", ylab = "Taches solaires")
```

Série "Temporelles"

Évolution de la densité de taches solaires

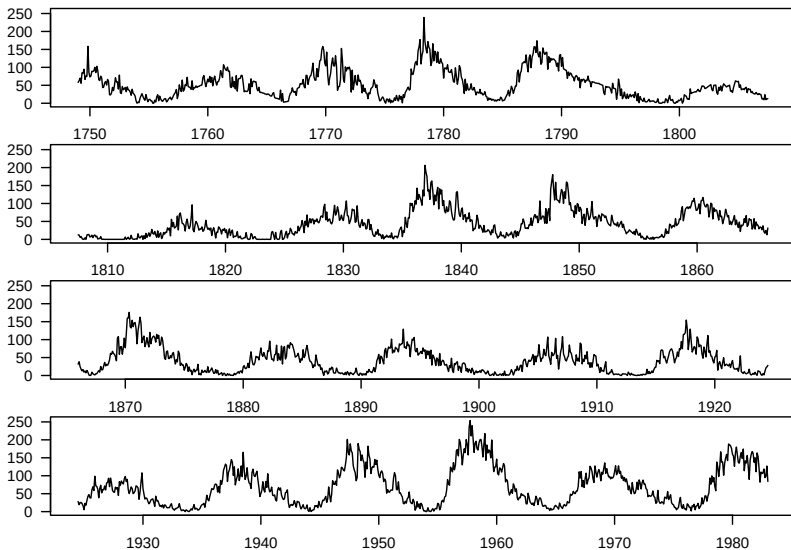


Série "Temporelles"

La représentation graphique précédente n'est pas bonne. Il faut que les courbes suivent au plus près la direction de la première ou de la deuxième bissectrice.

```
n <- 4
wd <- seq( start(sunspots)[1], end(sunspots)[1], length = n+1)
opar <- par(no.readonly = TRUE)
par(mfrow = c(n,1), mar = c(2,3,0.1,1))
for( i in 1:(length(wd)-1))
{
  plot( window(sunspots, wd[i], wd[i+1]), ylab = "", xlab = "", las = 1,
        ylim = c(0,max(sunspots)))
}
```

Série "Temporelles"



Série "Temporelles"

On voit maintenant que les montées et descentes ne sont pas symétriques. On peut aussi lisser le signal avec divers outils, par exemple :

```
opar <- par(no.readonly = TRUE)
par(mfrow = c(n,1), mar = c(2,3,0.1,1))
for( i in 1:(length(wd)-1))
{
  plot( window(sunspots, wd[i], wd[i+1]), ylab = "", xlab = "", las = 1, type =
  ylim = c(0,max(sunspots)))
  lines(lowess(window(sunspots, wd[i], wd[i+1]), f = 0.025), lwd = 2)
}
```

Série "Temporelles"

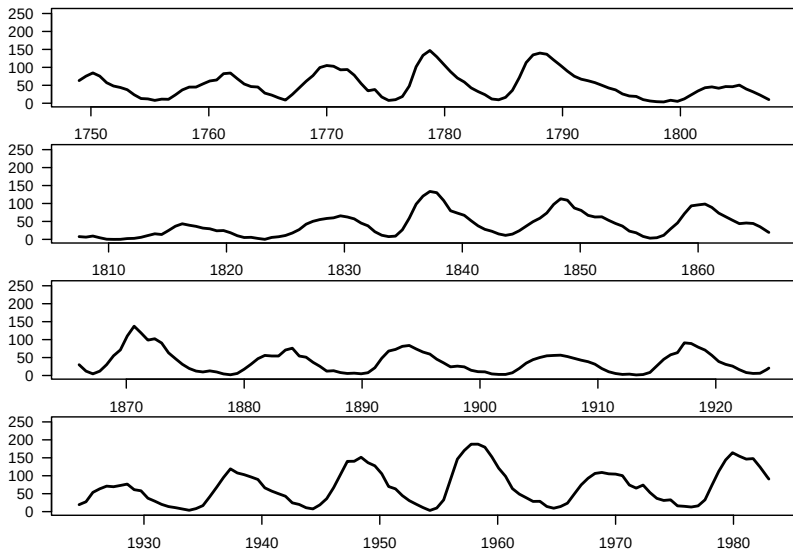


Table des matières

- 1 Introduction
- 2 Variables numériques
- 3 Variables qualitatives
- 4 Croisement de variables
- 5 Courbes
- 6 Les paramètres graphiques

Sauvegarde des paramètres graphiques

Les options des graphiques sont consultables et contrôlables avec la fonction `par()` que l'on utilise typiquement de la façon suivante :

```
# On sauvegarde les options graphiques modifiables :  
old.par <- par(no.readonly = TRUE)  
# ...  
# ... plein de modifications des options graphiques,  
# ... par exemple par(mfrow = c(1, 2))  
# ...  
# On restaure les options graphiques précédentes :  
par(old.par)
```

Les paramètres graphiques

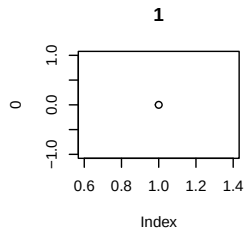
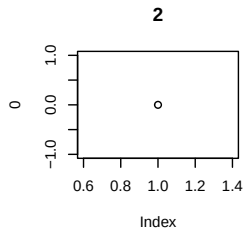
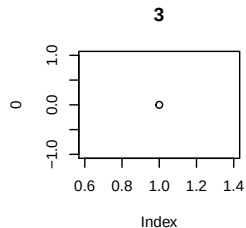
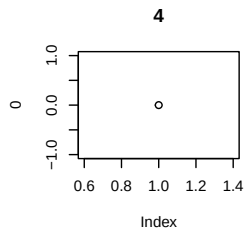
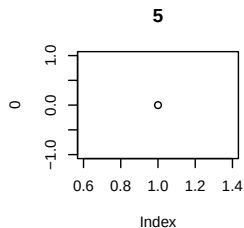
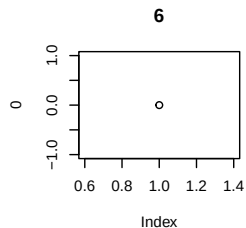
Pour une exploration systématique des paramètres graphiques voir la fiche de TD <https://esb.univ-lyon1.fr/pdf/tdr75.pdf>.
On n'envisage ici que les paramètres graphiques les plus courants.

mfrow

Le paramètre `mfrow` permet de disposer simultanément plusieurs graphiques.

```
par(mfrow = c(2,3))  
for(i in 1:6) plot(0, main = i)
```

mfrow

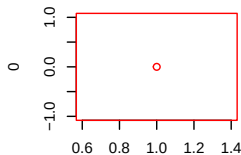
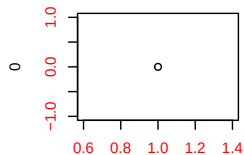
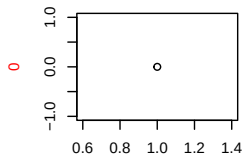
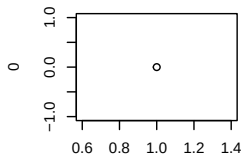
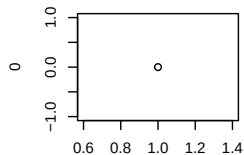
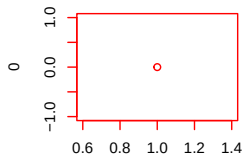


col*

Les paramètres `col*` contrôlent la couleur des éléments du graphique.

```
par(mfrow = c(2,3))
par(col = "red")
plot(0, main = "col = \"red\"", sub = "sous-titre")
par(col = "black", col.axis = "red")
plot(0, main = "col.axis = \"red\"", sub = "sous-titre")
par(col.axis = "black", col.lab = "red")
plot(0, main = "col.lab = \"red\"", sub = "sous-titre")
par(col.lab = "black", col.main = "red")
plot(0, main = "col.main = \"red\"", sub = "sous-titre")
par(col.main = "black", col.sub = "red")
plot(0, main = "col.sub = \"red\"", sub = "sous-titre")
par(col.sub = "black", fg = "red")
plot(0, main = "fg = \"red\"", sub = "sous-titre")
```

col*

col = "red"**col.axis = "red"****col.lab = "red"****col.main = "red"****col.sub = "red"****fg = "red"**

mar

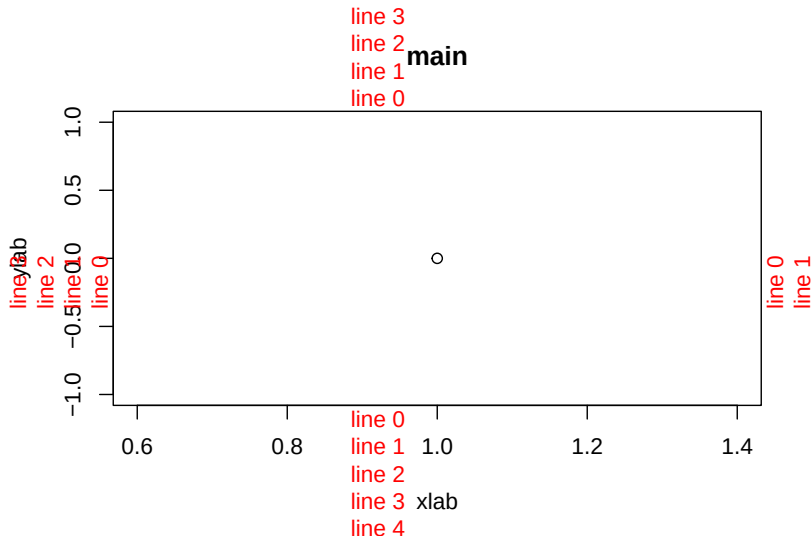
Ce paramètre contrôle les marges, exprimées en lignes de texte, du graphique qui par défaut sont :

- ❶ En bas : 5 lignes : une ligne pour les graduations (*ticks*), une ligne pour les étiquettes des graduations, une ligne vide, une ligne de légende, une ligne vide.
- ❷ À gauche 4 lignes : une ligne pour les graduations, une ligne pour les labels des graduations, une ligne vide, une ligne de légende.
- ❸ En haut : 4 lignes pour le titre.
- ❹ À droite : 2 lignes vides.

plus d'une bordure vide d'un dixième de ligne tout autour.

Graphiquement :

mar

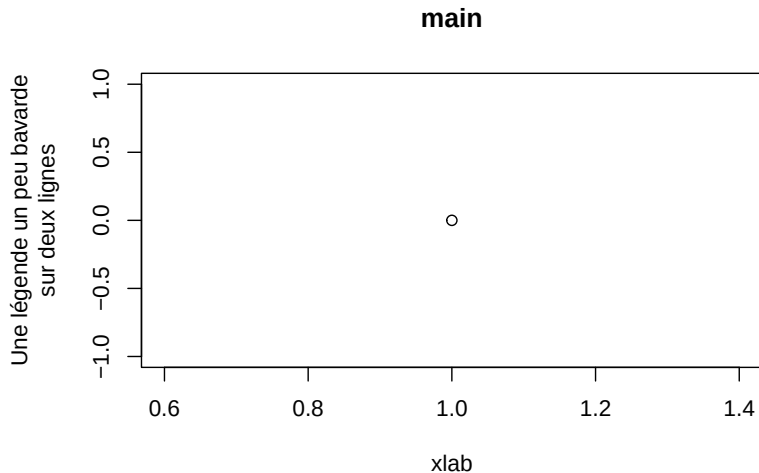


mar

Pour ajouter une ligne à la marge de gauche, on utiliserait :

```
par(mar = par("mar") + c(0, 1, 0, 0))  
plot(0, xlab = "xlab", ylab = "Une légende un peu bavarde\nsur deux lignes",  
main = "main")
```

mar

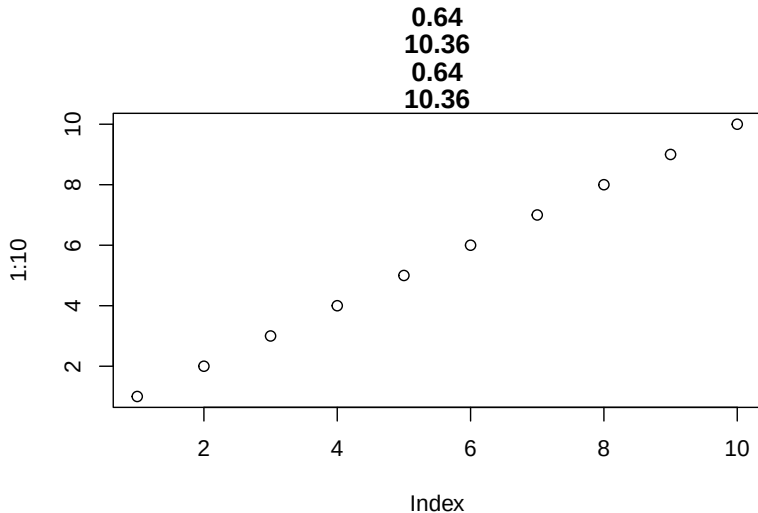


usr

Ce paramètre donne les coordonnées extrêmes de la région utile du graphique, par exemple :

```
plot(1:10,main = par("usr"))
```

usr

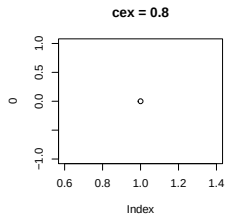
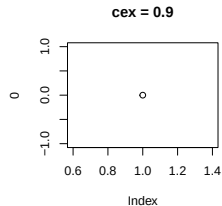
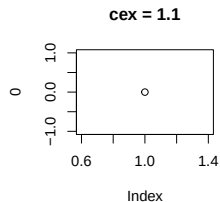
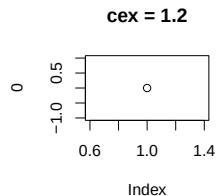


cex*

Ces paramètres contrôlent la taille relative des chaînes de caractères, notez que les marges s'adaptent automatiquement à la taille des caractères :

```
par(mfrow=c(2,2))  
for(cex in seq(from = 0.8, to = 1.2, length = 4))  
{  
  par(cex=cex)  
  plot(0, main = paste("cex =", round(cex,1)))  
}
```

cex*

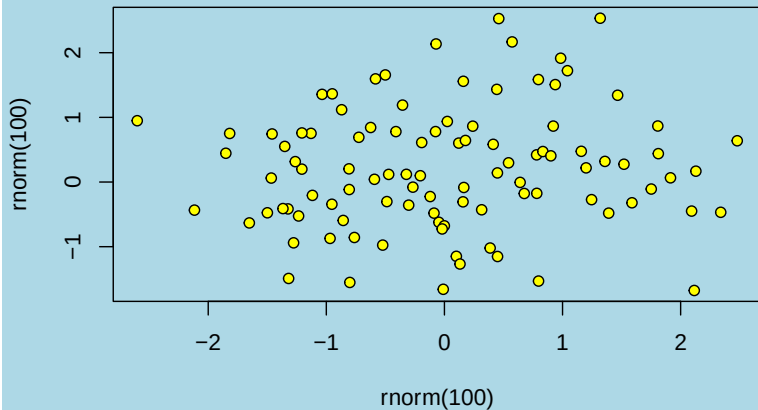


bg

La couleur de fond du graphique, attention les fonctions graphiques de haut niveau comme `plot()` ou `points()` ont un **argument** de même nom pour donner la couleur de remplissage des points :

```
par(bg = "lightblue")  
plot(rnorm(100), rnorm(100), pch = 21, bg = "yellow")
```

bg



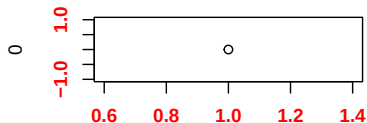
las

Le paramètre `las` contrôle le style des étiquettes des axes. À noter l'option `las = 1` qui permet de lire tous les étiquettes facilement car elles sont alors horizontales.

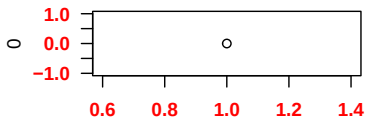
```
par(mfrow=c(2,2))  
for(i in 0:3) plot(0,las=i,main=paste("las =",i), font.axis = 2,  
                  col.axis = "red")
```

las

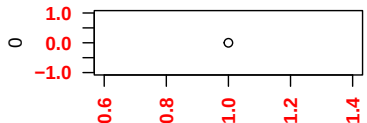
las = 0



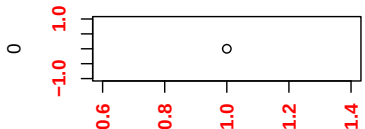
las = 1



las = 2



las = 3

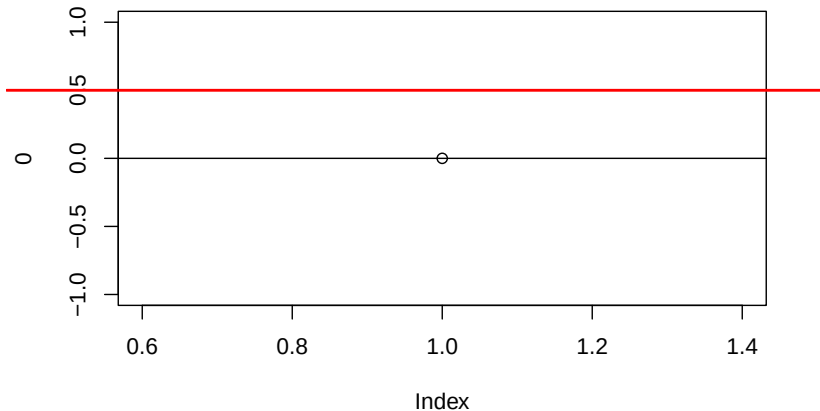


xpd

Le paramètre `xpd` permet de déborder de la région utile du graphique, par exemple :

```
plot(0)
abline(h=0)
par(xpd=NA)
abline(h=0.5,col="red",lwd=2)
```

xpd



Graphiques de base

Pr Jean R. LOBRY

Université Claude Bernard Lyon I – France

9 septembre 2026