

IV. Les graphiques

`R` propose de nombreux outils graphiques pour l'analyse et la visualisation des données. On peut avoir un aperçu des possibilités graphiques de `R` grâce à la commande `demo(graphics)`. Le tableau 5 fournit une liste non exhaustive des fonctions graphiques disponibles sous `R`. Par l'intermédiaire d'exemple, nous allons illustrer l'utilisation de certaines de ces fonctions.

IV.1 Gestion des fenêtres graphiques

Lorsqu'une fonction graphique est exécutée, `R` ouvrira une fenêtre graphique et y affichera le graphe. On peut spécifier le dispositif de gestion des fenêtres. La liste des dispositifs graphiques disponibles dépend du système d'exploitation.

Sous Mac, pour créer une nouvelle fenêtre graphique, on utilise la commande `get("quartz")()`.

Sous PC, pour créer une nouvelle fenêtre graphique, on utilise la commande `X11()`.

Tant sur Mac que sur PC, pour sélectionner une des fenêtres, on utilise la commande `dev.set()`. Par exemple, pour sélectionner la fenêtre graphique numéro i , on tape la commande `dev.set(i)`. Si on souhaite connaître le numéro de la fenêtre active, on tape la commande `dev.cur()`.

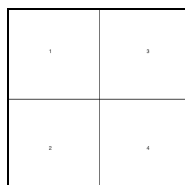
IV.2 Partitionner un graphique

`R` propose deux manières de partitionner les graphiques.

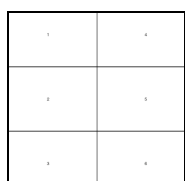
1. La commande `split.screen(c(1, 2))` va diviser le graphique en deux parties qu'on sélectionnera avec les commandes `screen(1)` et `screen(2)`.

2. La fonction `layout()` partitionne le graphique actif en plusieurs parties sur lesquelles sont affichés les graphes successivement. Cette fonction a pour argument une matrice de valeurs entières qui indiquent le numéro des sous-fenêtres. Pour visualiser la partition créée, on utilise la fonction `layout.show` avec, en argument, le nombre de sous-fenêtres. Par exemple, si on veut diviser la fenêtre en quatre parties égales, on tapera la commande suivante `layout(matrix(1:4, 2, 2))` et pour visualiser la partition créée, on utilisera la commande `layout.show(4)`.

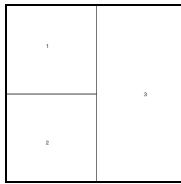
```
layout(matrix(1:4, 2, 2))  
layout.show(4)
```



```
layout(matrix(1:6, 3, 2))  
layout.show(6)
```





```
mat = matrix(c(1:3, 3), 2, 2)
layout(mat)
layout.show(3)
```



IV.3. Les fonctions graphiques principales □ High-level plotting commands




<code>plot(x)</code>	Graphe des valeurs de x (sur l'axe des y) ordonnées sur l'axe des x
<code>plot(x, y)</code>	Graphe bivarié de x (sur l'axe des x) et y (sur l'axe des y)
<code>sunflowerplot(x, y)</code>	Idem que <code>plot()</code> mais les points superposés sont dessinés sous forme de fleurs dont le nombre de pétales représente le nombre de points
<code>pie(x)</code>	Graphe en « <code>camembert</code> »
<code>boxplot(x)</code>	Graphe « <code>boîtes à moustaches</code> »
<code>coplot(x~y z)</code>	Graphe bivarié de x et y pour chaque valeur ou intervalle de valeurs de z
<code>interaction.plot(f1, f2, y)</code>	Si $f1$ et $f2$ sont des facteurs, graphe des moyennes de y (sur l'axe des y) en fonction des valeurs de $f1$ (sur l'axe des x) et de $f2$ (différentes courbes) ; l'option <code>fun</code> permet de choisir la statistique résumée de y (par défaut <code>fun = mean</code>)
<code>matplot(x,y)</code>	Graphe bivarié de la 1ère colonne de x contre la 1ère de y , la 2ème de x contre la 2ème de y , etc.
<code>fourfoldplot(x)</code>	Visualise, avec des quarts de cercles, l'association entre deux variables dichotomiques pour différentes populations (x doit être un array avec <code>dim = c(2, 2, k)</code> ou une matrice avec <code>dim = c(2, 2)</code> si $k = 1$)
<code>assocplot(x)</code>	Graphe de Cohen-Friendly indiquant les déviations de l'hypothèse d'indépendance des lignes et des colonnes dans un tableau de contingence `a deux dimensions
<code>mosaicplot(x)</code>	Si x est une matrice ou un <code>data.frame</code> , graphe en mosaïque des résidus d'une régression log-linéaire sur une table de contingence
<code>Pairs(x)</code>	Dessine tous les graphes bivariés entre les colonnes de x
<code>hist(x)</code>	Histogramme des fréquences de x
<code>barplot(x)</code>	Histogramme des valeurs de x
<code>qqnorm(x)</code>	Quantiles de x en fonction des valeurs attendues selon une loi normale
<code>qqplot(x, y)</code>	Quantiles de y en fonction des quantiles de x
<code>contour(x, y, z)</code>	Courbes de niveau (les données sont interpolées pour tracer les courbes), x et y doivent être des vecteurs et z une matrice telle que <code>dim(z)=c(length(x), length(y))</code> (x et y peuvent être omis)
<code>filled.contour(x, y, z)</code>	Idem mais les aires entre les contours sont colorées, et une légende des couleurs est également dessinée
<code>image(x, y, z)</code>	Idem mais en couleur (les données sont tracées)
<code>persp(x, y, z)</code>	Idem mais en 3-D (les données sont tracées)
<code>stars(x)</code>	si x est une matrice ou un <code>data.frame</code> , dessine un graphe en segments ou en étoile où chaque ligne de x est représentée par une étoile et les colonnes par les longueurs des branches
<code>symbols(x, y, ...)</code>	dessine aux coordonnées données par x et y des symboles (cercles, carrés, rectangles, étoiles, thermomètres ou "boxplots") dont les tailles, couleurs, ... sont spécifiées par des arguments supplémentaires
<code>termplot(mod.obj)</code>	graphe des effets (partiels) d'un modèle de régression (<code>mod.obj</code>)

Tab 5. Liste des fonctions graphiques principales

À chaque fonction graphique principale sont associés des paramètres consultables via l'aide-en-ligne de . Certains de ces paramètres sont identiques pour plusieurs fonctions graphiques dont voici les principales (les valeurs par défaut sont également présentées) 

<code>add = FALSE</code>	si TRUE superpose le graphe au graphe existant (s'il y en a un)
<code>axes = TRUE</code>	si FALSE ne trace pas les axes ni le cadre
<code>type="p"</code>	le type de graphe qui sera dessiné, "p" : points, "l" : lignes, "b" : points connectés par des lignes, "o" : idem mais les lignes recouvrent les points, "h" : lignes verticales, "s" : escaliers, les données étant représentées par le sommet des lignes verticales, "s" : idem mais les données étant représentées par le bas des lignes verticales
<code>xlim=, ylim=</code>	fixe les limites inférieures et supérieures des axes
<code>xlab=, ylab=</code>	annotations des axes, doivent être des variables de mode caractère
<code>main=</code>	titre principal, doit être une variable de mode caractère
<code>sub=</code>	sous-titre (écrit dans une police plus petite)

IV.4 Les fonctions graphiques secondaires Low-level plotting commands

 propose un ensemble de fonctions graphiques dites «secondaires qui ont une action sur un graphe déjà existant. Nous présentons dans le tableau 6. les fonctions graphiques secondaires principales.

<code>points(x, y)</code>	Ajoute des points (l'option <code>type=</code> peut être utilisée)
<code>lines(x, y)</code>	Idem mais avec des lignes
<code>text(x, y, labels, ...)</code>	Ajoute le texte spécifié par <code>labels</code> au coordonnées (x,y). Un usage typique sera: <code>plot(x, y, type="n"); text(x, y, names)</code>
<code>mtext(text, side=3, line=0, ...)</code>	Ajoute le texte spécifié par <code>text</code> dans la marge spécifiée par <code>side</code> (cf. <code>axis()</code> plus bas) ; <code>line</code> spécifie la ligne à partir du cadre de traçage
<code>segments(x0, y0, x1, y1)</code>	Trace des lignes des points (x0,y0) aux points (x1,y1)
<code>arrows(x0, y0, x1, y1, angle=30, code=2)</code>	Idem avec des flèches aux points (x0, y0) si <code>code=2</code> , aux points (x1,y1) si <code>code=1</code> , ou aux deux si <code>code=3</code> ; <code>angle</code> contrôle l'angle de la pointe par rapport à l'axe
<code>abline(a,b)</code>	Trace une ligne de pente <code>b</code> et ordonnée à l'origine <code>a</code>
<code>abline(h=y)</code>	Trace une ligne horizontale sur l'ordonnée <code>y</code>
<code>abline(v=x)</code>	Trace une ligne verticale sur l'abscisse <code>x</code>
<code>abline(lm.obj)</code>	Trace la droite de régression donnée par <code>lm.obj</code>
<code>rect(x1, y1, x2, y2)</code>	Trace un rectangle délimité à gauche par <code>x1</code> , à droite par <code>x2</code> , en bas par <code>y1</code> et en haut par <code>y2</code>
<code>polygon(x, y)</code>	Trace un polygone reliant les points dont les coordonnées sont données par <code>x</code> et <code>y</code>
<code>legend(x, y, legend)</code>	Ajoute la légende au point de coordonnées (x,y) avec les symboles données par <code>legend</code>
<code>title()</code>	Ajoute un titre et optionnellement un sous-titre
<code>axis(side, vect)</code>	Ajoute un axe en bas (<code>side=1</code>), à gauche (2), en haut (3) ou à droite (4) ; <code>vect</code> (optionnel) indique les abscisses (ou ordonnées) où les graduations seront

	tracées
rug(x)	Dessine les données x sur l'axe des x sous forme de petits traits verticaux
locator(n, type="n", ...)	retourne les coordonnées (x y) après que l'utilisateur ait cliqué n fois sur le graphe avec la souris ; également trace des symboles (type="p") ou des lignes (type="l") en fonction de paramètres graphiques optionnels (...); par défaut ne trace rien (type="n")

Tab 6. Liste des fonctions graphiques secondaires

IV.5 Les paramètres graphiques

La présentation des graphiques peut-être amélioré grâce aux paramètres graphiques. Il y a 68 paramètres graphiques. La liste détaillé de ces paramètres peut-être obtenu grâce à la commande `?par`. Voici une liste de paramètres graphiques couramment utilisés.

adj	Contrôle la justification du texte (0 à gauche, 0.5 centré, 1 à droite)
bg	Spécifie la couleur de l'arrière-plan (ex : bg="red", bg="blue", ...). La liste des 657 couleurs disponibles est affichée avec colors().
bty	Contrôle comment le cadre est tracé, valeurs permises : "o", "l", "7", "c", "u" ou "]" (le cadre ressemblant au caractère correspondant) ; bty="n" supprime le cadre
cex	Une valeur qui contrôle la taille des caractères et des symboles par rapport au défaut ; les paramètres suivants ont le même contrôle pour les nombres sur les axes, cex.axis, les annotations des axes, cex.lab, le titre, cex.main, le sous-titre, cex.sub
col	Contrôle la couleur des symboles ; comme pour cex il y a : col.axis, col.lab, col.main, col.sub
font	Un entier qui contrôle le style du texte (1 : normal, 2 : italique, 3 : gras, 4 : gras italique) ; comme pour cex il y a : font.axis, font.lab, font.main, font.sub
las	Un entier qui contrôle comment sont disposées les annotations des axes (0 : parallèles aux axes, 1 : horizontales, 2 : perpendiculaires aux axes, 3 : verticales)
lty	Contrôle le type de ligne tracée, peut être un entier (1 : continue, 2 : tirets, 3 : points, 4 : points et tirets alternés, 5 : tirets longs, 6 : tirets courts et longs alternés), ou ensemble de 8 caractères maximum (entre "0" et "9") qui spécifie alternativement la longueur, en points ou pixels, des éléments tracés et des blancs, par exemple lty="44" aura le même effet que lty=2
lwd	Une valeur numérique qui contrôle la largeur des lignes
mar	Un vecteur de 4 valeurs numériques qui contrôle l'espace entre les axes et le bord de la figure de la forme c(bas, gauche, haut, droit), les valeurs par défaut sont c(5.1, 4.1, 4.1, 2.1)
mfc0l	Un vecteur de forme c(nr,nc) qui partitionne la fenêtre graphique en une matrice de nr lignes et nc colonnes, les graphes sont ensuite dessinés en colonne
mfrow	Idem mais les graphes sont ensuite dessinés en ligne
pch	Contrôle le type de symbole, soit un entier entre 1 et 25, soit n'importe quel caractère entre guillemets
ps	Un entier qui contrôle la taille en points du texte et des symboles
pty	Un caractère qui spécifie la forme du graphe, "s" : carrée, "m" : maximale
tck	Une valeur qui spécifie la longueur des graduations sur les axes en fraction du plus petit de la largeur ou de la hauteur du graphe ; si tck=1 une grille est tracée
tcl	Une valeur qui spécifie la longueur des graduations sur les axes en fraction de la hauteur d'une ligne de texte (défaut tcl=-0.5)
xaxt	si xaxt="n" l'axe des x est défini mais pas tracé (utile avec axis(side=1, ...))
yaxt	si yaxt="n" l'axe des y est défini mais pas tracé (utile avec axis(side=2, ...))

Tab 7. Liste des options graphiques

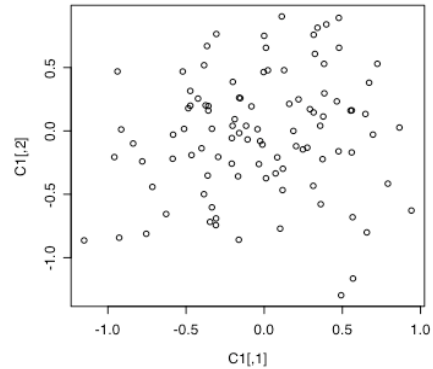
Nous allons illustrer le potentiel graphique de \mathbb{R} à travers les fonctions `plot()`, `pie()`, `boxplot(x,y)` et `hist()`

IV.6 Exemple d'utilisation de fonctions graphiques de

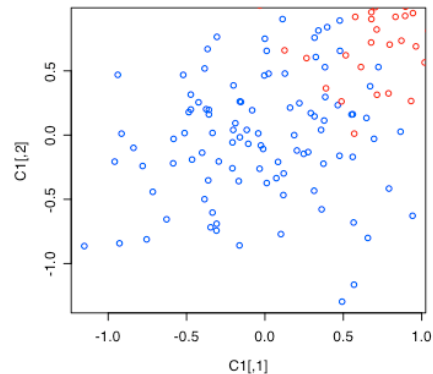
IV.6.1 utilisation de la fonction `plot()`

```
C1 = matrix(rnorm(200, sd = 0.5), ncol = 2)
C2 = matrix(rnorm(200, mean = 1, sd = 0.5), ncol = 2)
mat = rbind(C1, C2)
```

```
plot(C1)
```

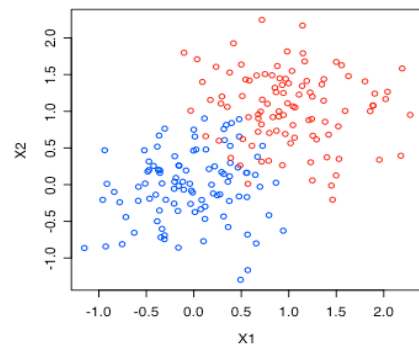


```
plot(C1, col = "blue")
points(C2, col = "red")
```



représentation d'un nuage de points

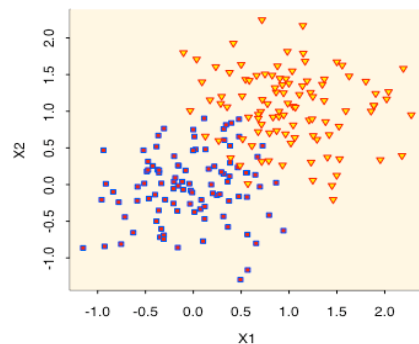
```
plot(C1, col = "blue",
      xlim = range(mat[, 1]),
      ylim = range(mat[, 2]),
      main = "représentation d'un nuage de points",
      xlab = "X1", ylab = "X2")
points(C2, col = "red")
```



représentation d'un nuage de points

```
plot(1,
      xlim = range(mat[, 1]),
      ylim = range(mat[, 2]),
      main = "représentation d'un nuage de points",
      xlab = "X1", ylab = "X2",
      bty = "l", tcl = -.25)
rect(-3, -3, 3, 3, col = "cornsilk")

points(C1, col = "blue", pch = 22, bg = "red")
points(C2, col = "red", pch = 25, bg = "yellow")
```



Graphique 1. utilisation de la fonction `plot()`

Commentons les commandes qui ont permis de générer les graphiques 1.1, 1.2, 1.3 et 1.4.

Le graphique 1.1 présente le graphe de base de la fonction `plot()`. `C1` est une matrice composée de deux vecteurs. La fonction `plot()` trace le graphe bivarié de la première colonne de `C1` sur la deuxième colonne de `C1`. On aurait également pu taper la commande suivante `plot(C1[,1], C1[,2])`

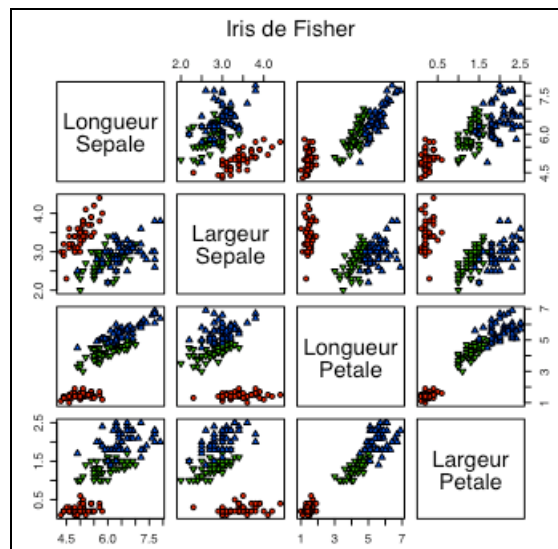
On peut spécifier la couleur des points grâce à l'option `col`. La liste des couleurs disponibles (657 couleurs) est fournie par la fonction `colors()`. Le graphique 1.2 présente, en plus de l'utilisation de l'option `col`, l'utilisation de la fonction `points()`. Cette fonction permet de rajouter des points à un graphique existant. Un problème subsiste : La longueur des axes ne s'adapte pas au nuage de points complet mais juste au nuage de points du premier plot.

Le graphique 1.3 propose une réponse à ce problème grâce aux options `xlim` et `ylim`. Notons que la fonction `range()` retourne le minimum et le maximum des arguments rentrés en paramètre. On peut par ailleurs donner des noms aux axes grâce aux options `xlab` et `ylab` et un titre au graphique grâce à l'option `main`.

Comme l'illustre le graphique 1.4, La fonction `plot()`, grâce à sa flexibilité, fournit un outil au pouvoir esthétique performant. Nous avons, dans un premier temps, défini l'architecture du graphique. L'option `bty` contrôle la forme du cadre tandis que l'option `tcl` spécifie la longueur des graduations. La fonction `rect()` trace un rectangle délimité par les quatre premiers arguments rentrés en paramètres. Une fois l'architecture du graphique achevée, il ne reste plus qu'à placer les points via la fonction `points()`. Notons l'utilisation des options `pch` et `bg` qui spécifient respectivement la forme et la couleur de fond des points.

IV.6.2 Utilisation de la fonction `plot()` dans le cadre d'une `data.frame`

```
plot(iris[,1:4],
     bg = c("red", "green3", "blue")[iris[,5]],
     pch = c(21, 25, 24)[iris[,5]],
     main = "Iris de Fisher",
     labels =
       c("Longueur\nSepale",
         "Longueur\nSepale",
         "Longueur\nPetale",
         "Longueur\nPetale"
       )
     )
```



Graphique 2. Utilisation de la fonction `plot()` sur une `data.frame`

Commentons les commandes qui ont permis de générer le graphique 2. Les données d'Iris sont stockées dans une `data.frame`. Dans le cadre d'une `data.frame`, la fonction `plot()` ne se contente pas de tracer le graphe bivarié du premier élément de la `data.frame` sur le deuxième mais trace tous les graphes bivariés. On peut obtenir le même résultat sur une matrice si on utilise la fonction `pairs()`.

```

pairs(matrice_iris[ ,1:4],
      bg = c("red", "green3", "blue")[iris[ ,5]],
      pch = c(21, 25, 24)[iris[ ,5]],
      main = "Iris de Fisher",
      labels = c("Longueur\nSepale",
                "Largeur\nSepale",
                "Longueur\nPetale",
                "Largeur\nPetale")
)

```

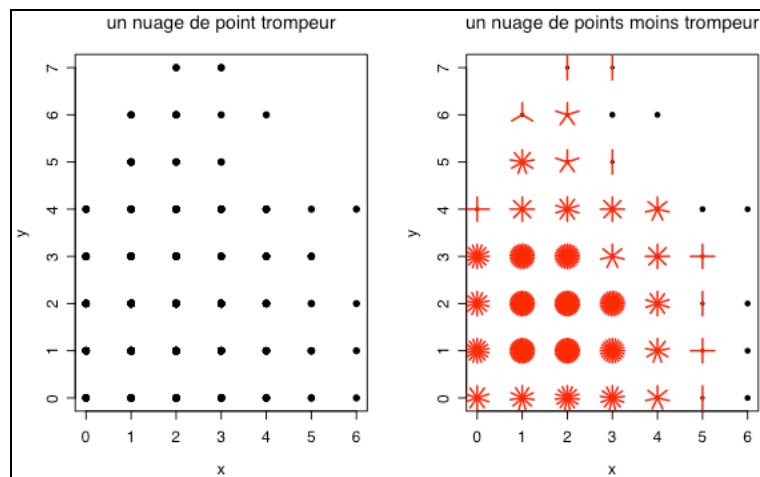
IV.6.3 Utilisations de la fonction `sunflowerplot()` ☐ Un exemple fleuri

Une difficulté des nuages de points vient de la superposition des points. Une manière sympathique d’appréhender ce problème est proposée par la fonction `sunflowerplot()`.

```

n = 500
x = rpois(n, lambda = 2)
y = rpois(n, lambda = 2)
layout(t(matrix(1:2)))
plot(x, y, pch = 19, main = "un nuage de point trompeur")
sunflowerplot(x, y, pch = 19, main = "un nuage de points moins trompeur")

```



Graphique 3. Illustration de la fonction `sunflowerplot()`

Commentons les commandes qui ont permis de générer le graphique ☐. Le nombre de pétale indique le nombre de superposition. Notons que la fonction `rpois()` génère des données aléatoires suivant une loi de poisson.

IV.6.4 Utilisation de la fonction `hist()`

Intéressons nous à la taille de 237 étudiants de DEUG MASS disponibles dans le jeu de données `survey` du packages MASS. Nous reprenons ici l’analyse de J.R Lobry du tutorial «**Programmation Statistique avec R – Graphique de base**» ☐.

```

library(MASS)
data(survey)
mat = matrix(c(1:2), 1, 2)
layout(mat)

hist(survey$Height, col = "yellow", border = "red",
      main = paste("Taille de", nrow(survey), " étudiants"),

```



```

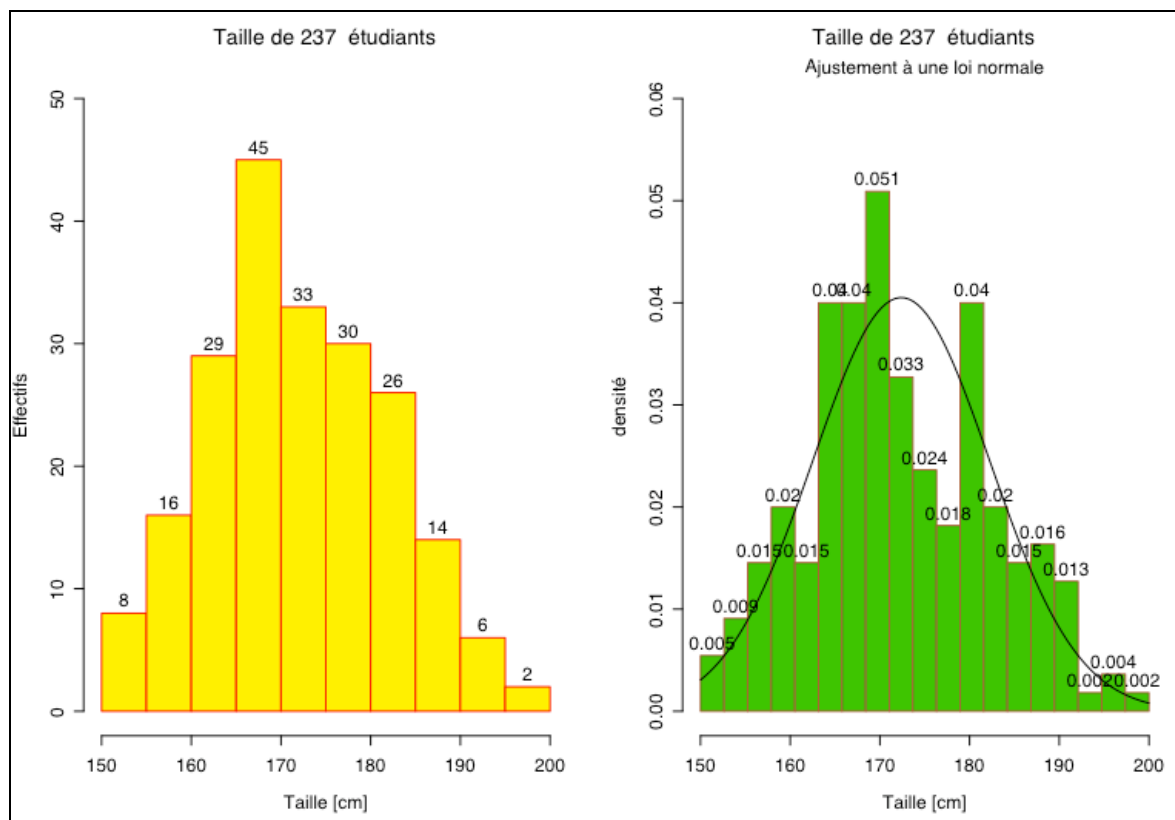
xlab = "Taille [cm]", ylab = "Effectifs", ylim = c(0, 50),
labels = TRUE)

hist(survey$Height, breaks = seq(from = 150, to = 200, length = 20),
col = "green3", border = "sienna",
main = paste("Taille de", nrow(survey), " étudiants"),
xlab = "Taille [cm]", ylab = "densité",
proba = TRUE, labels = TRUE, ylim = c(0, 0.06))

x = seq(from = 150, to = 200, length = 100)
lines(x, dnorm(x, mean(survey$Height, na.rm = TRUE),
sd(survey$Height, na.rm = TRUE)

)
)
mtext("Ajustement à une loi normale")

```



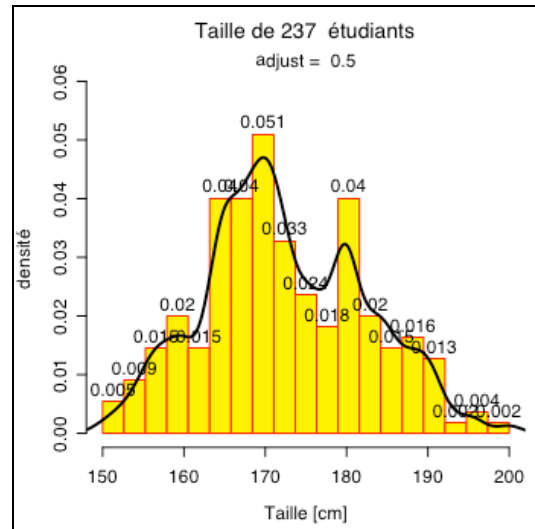
Graphique 4. Utilisation de la fonction hist()

Commentons les commandes qui ont permis de générer le graphique. Une première étape consiste, via la fonction `library()`, à charger le package MASS dans lequel est stocké le jeu de données à analyser (survey). On souhaite illustrer les résultats de l'analyse par deux histogrammes présentés sur le même graphique. On utilise donc la fonction `layout()`. Nous constatons au premier appel de la fonction `hist` des similarités avec la fonction `plot` dans l'utilisation des paramètres (`col`, `xlab`, `ylab`, `main`, ...). Certaines options sont spécifiques à la fonction `hist`. L'option `labels = TRUE` affiche les fréquences absolues au sommet de chaque barre. Comme l'illustre le deuxième histogramme, on peut préférer utiliser les fréquences relatives, (`proba = TRUE`) ceci facilitant la superposition de distributions de référence (`lines()`).

Le problème des histogrammes est que le choix du découpage en intervalles est assez arbitraire. On peut le contrôler avec le paramètre `breaks` comme l'illustre les commandes du deuxième histogramme.

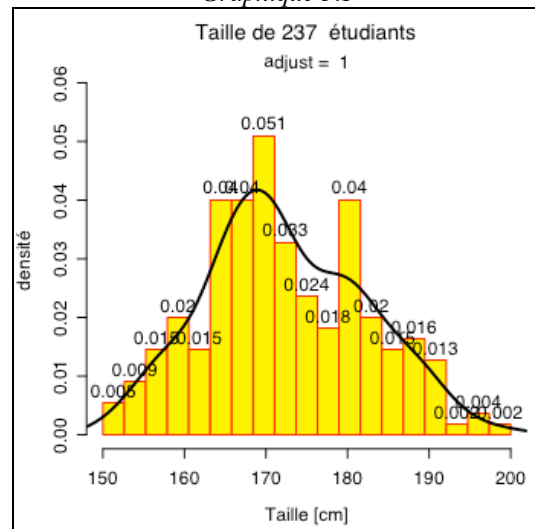
Le choix du découpage en intervalle est un problème délicat qui risque de biaiser notre perception des données. L'histogramme est donc un outil délicat à manipuler. De nos jours, on préfère utiliser des estimateurs locaux de la densité des points et explorer différentes échelles \square ce qu'illustrent les graphiques 5.1, 5.2, 5.3 et 5.4.

```
adj = 0.5
dst = density(survey$Height, na.rm = TRUE, adjust = adj)
hist(survey$Height,
     breaks = seq(from = 150, to = 200, length = 20),
     col = "yellow", border = "red",
     main = paste("Taille de", nrow(survey), " étudiants"),
     xlab = "Taille [cm]", ylab = "densité",
     proba = TRUE, labels = TRUE, ylim = c(0, 0.06))
lines(dst$x, dst$y, lwd = 2)
mtext(paste("adjust = ", adj))
```



Graphique 5.1

```
adj = 1
dst = density(survey$Height, na.rm = TRUE, adjust = adj)
hist(survey$Height,
     breaks = seq(from = 150, to = 200, length = 20),
     col = "yellow", border = "red",
     main = paste("Taille de", nrow(survey), " étudiants"),
     xlab = "Taille [cm]", ylab = "densité",
     proba = TRUE, labels = TRUE, ylim = c(0, 0.06))
lines(dst$x, dst$y, lwd = 2)
mtext(paste("adjust = ", adj))
```

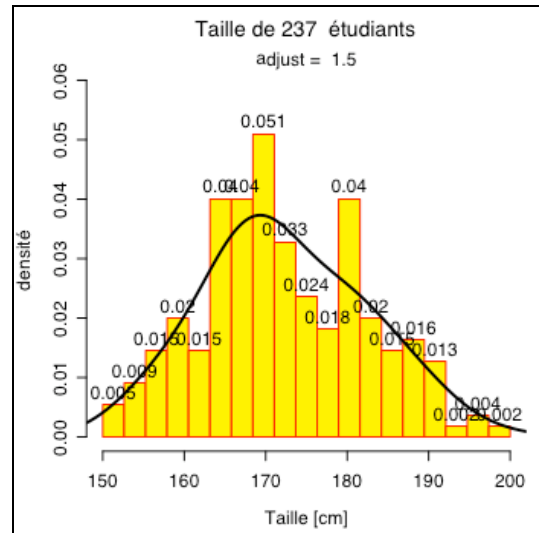


Graphique 5.2

```

adj = 1.5
dst = density(survey$Height, na.rm = TRUE, adjust = adj)
hist(survey$Height,
     breaks = seq(from = 150, to = 200, length = 20),
     col = "yellow",
     border = "red",
     main = paste("Taille de", nrow(survey), " étudiants"),
     xlab = "Taille [cm]", ylab = "densité",
     proba = TRUE, labels = TRUE, ylim = c(0, 0.06))
lines(dst$x, dst$y, lwd = 2)
mtext(paste("adjust = ", adj))

```

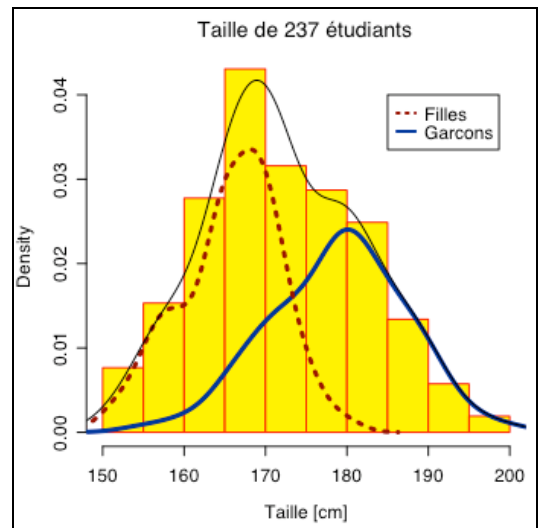


Graphique 5.3

```

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 = "yellow",
     border = "red",
     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$y, lwd = 3, lty = 3,
      col = "darkred" )
lines(dst$x, dst$y)
legend(185, 0.04, legend = c("Filles", "Garçons"),
      col = c("darkred", "darkblue"),
      lty = c(3, 1), lwd = 2, pt.cex = 2 )

```



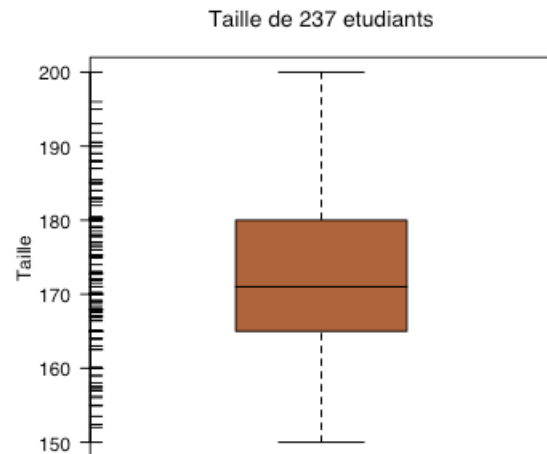
Graphique 5.4

Commentons les commandes qui ont permis de générer les graphiques 5.1, 5.2, 5.3 et 5.4. Le paramètre important de la fonction `density()` est le paramètre `adjust`. La valeur par défaut du paramètre `adjust` est 1. Si `adjust` est inférieur à 1, on s'intéresse aux petites variations, à la nature discrète de la variable. En revanche, si `adjust` est supérieur 1, on lisse le signal on s'intéresse aux comportement global de la variable, aux grosses variations.

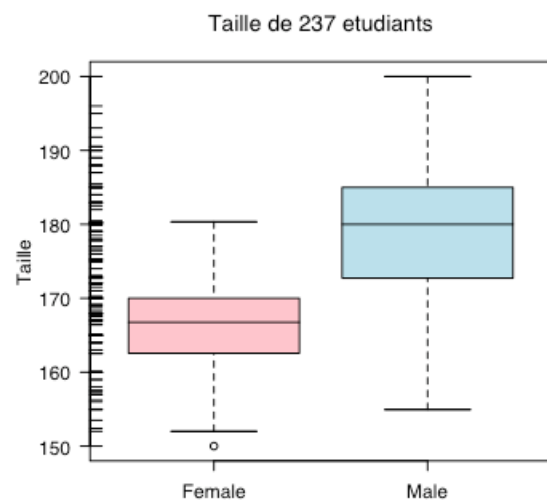
Un autre avantage des estimateurs locaux de la densité par rapport aux histogrammes est qu'il permette de superposer facilement plusieurs distributions. En effet, il peut être intéressant de distinguer les filles des garçons.

IV.6.5 Utilisation de la fonction `boxplot()`

```
boxplot(survey$Height,  
        col = "sienna",  
        main = paste("Taille de", nrow(survey),  
                     "etudiants"),  
        ylab = "Taille",  
        las = 1  
        )  
rug(survey$Height, side = 2)
```



```
boxplot(survey$Height ~ survey$Sex,  
        col = c("lightpink", "lightblue"),  
        main = paste("Taille de", nrow(survey),  
                     "etudiants"),  
        ylab = "Taille",  
        las = 1  
        )  
rug(survey$Height, side = 2)
```



Graphique 6.1 & Graphique 6.2. Utilisation de la fonction `boxplot()`

Commentons les commandes qui ont permis de générer les graphiques 6.1 et 6.2. On retrouve l'utilisation désormais classique des arguments des fonctions graphiques. Notons l'utilisation de la fonction `rug()` qui permet de représenter les données par des graduations sur les axes.

IV.6.6 Utilisation de la fonction `pie()`

```
vente.tarte = c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
names(vente.tarte) = c("Cerise", "Framboise", "Pomme",
                      "Abricot", "Pêche", "Fraise")
pie(vente.tarte,
    col = c("tan", "green3", "plum",
            "royalblue", "red", "peru"),
    border = NA,
    main = "Ventes de tartes")
```



Graphique 7. Utilisation de la fonction `pie()`

Commentons les commandes qui ont permis de générer le graphique. Les commandes sont grosso modo les mêmes que pour les autres fonctions graphiques. Le camembert représente la proportion de tarte vendue en fonction du fruit qui la compose.

IV.7 Les packages `grid` et `lattice`

Les packages `grid` et `lattice` proposent un outil de visualisation de données multivariées particulièrement adaptées pour l'exploration de relations ou d'interactions entre variables. L'idée principale derrière le package `lattice` est celle des graphes multiples conditionnés. Un graphe bivarié entre deux variables sera découpé en plusieurs graphes en fonction des valeurs d'une troisième variable.

La plupart des fonctions de `lattice` prennent pour argument principal une formule, par exemple `~x | y . y~x | z` signifie que le graphe de y en fonction de x sera dessiné en plusieurs sous-graphes en fonction des valeurs de z .

Le tableau ci-dessous indique les principales fonctions du package `lattice`.

<code>barchart(~y x)</code>	Histogramme des valeurs de y en fonction de celles de x
<code>bwplot(~y x)</code>	Graphe «boîtes à moustaches»
<code>densityplot(~ x)</code>	Graphe de fonctions de densité
<code>dotplot(y ~ x)</code>	Graphe de Cleveland (graphes superposés ligne par ligne et colonne par colonne)
<code>histogram(~ x)</code>	Histogrammes des fréquences de x
<code>qqmath(~ x)</code>	Quantiles de x en fonction des valeurs attendues selon une distribution théorique
<code>stripplot(y ~ x)</code>	Graphe unidimensionnel, x doit être numérique, y peut être un facteur
<code>qq(y ~ x)</code>	Quantiles pour comparer deux distributions, x doit être numérique, y peut être numérique, caractère ou facteur mais doit avoir deux «niveaux»
<code>xyplot(y ~ x)</code>	Graphes bivariés (avec de nombreuses fonctionnalités)
<code>levelplot(z~x*y)</code>	Graphe en couleur des valeurs de z aux coordonnées fournies par x et y (x , y et z sont tous de même longueur)
<code>splom(~ x)</code>	Matrice de graphes bivariés
<code>parallel(~ x)</code>	Graphe de coordonnées parallèles

Pour accéder aux fonctions du package `lattice`, il faut au préalable chargé le package en mémoire via la commande `library(lattice)`.

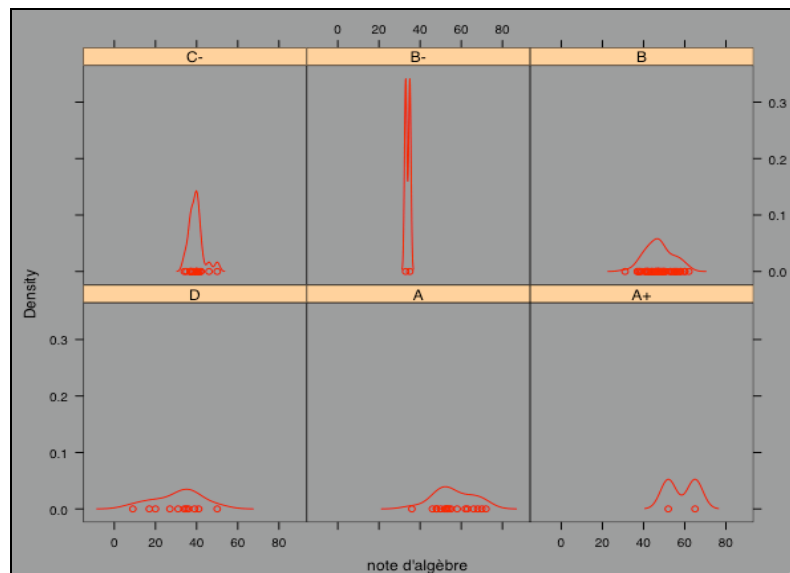
Pour illustrer l'utilisation des fonctions `densityplot()`, `bwplot()`, `histogram()`, `xyplot()` et `splom()`, nous allons utiliser le jeu de données `deug` disponible dans le package `ade4`. `deug` fournit les résultats de 104 étudiants sur 9 matières (Algebra, Analysis, Proba, Informatic, Economy, Option1, Option2, English, Sport). `deug` fournit également les résultats globaux pour chacun des étudiants (A, B, C, D). On souhaite évaluer l'impact de la note d'algèbre sur le résultat global.

```
x = deug$tab$Algebra
y = deug$result
```

L'objectif de cette étude est de découper les valeurs des notes d'analyse suivant les résultats finaux des 104 étudiants. On souhaite donc découper l'ensemble des valeurs de x en tranches et, pour chacune de ces tranches, construire une courbe de densité (`densityplot()`), une boîte à moustaches (`bwplot()`), un histogramme (`histogram()`). Ces fonctions sont dans le package `lattice`.

IV.7.1 Utilisation de la fonction `densityplot()`

```
densityplot(~ x | y,
xlab = " note d'algèbre ",
col = "red")
```



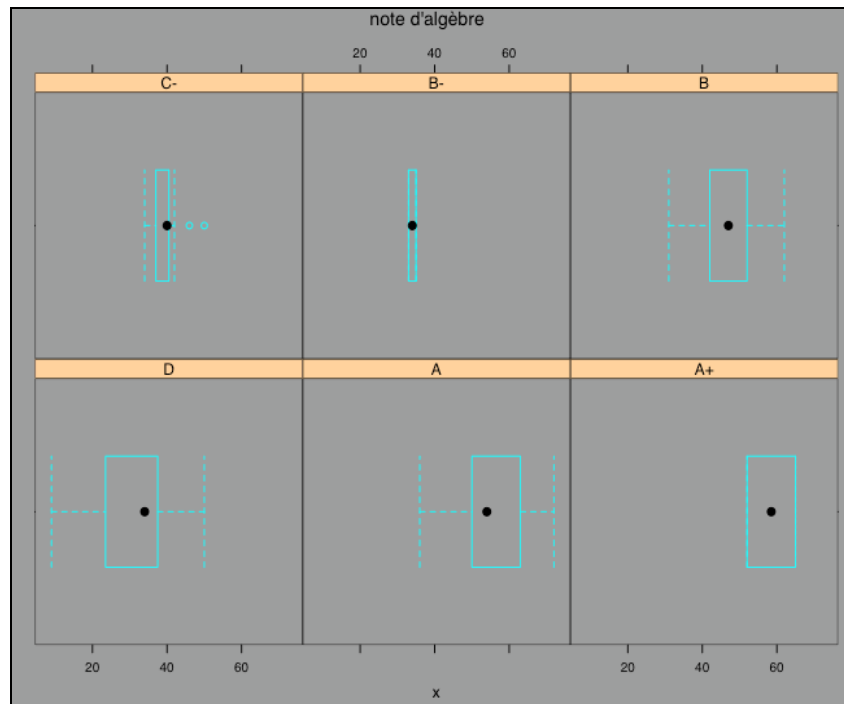
Graphique 8. Utilisation de la fonction `densityplot()`

Comme on devait s'y attendre, les meilleurs étudiants en analyse auraient tendance à obtenir les meilleurs résultats globaux.

IV.7.2 Utilisation de la fonction `bwplot()`

La fonction `bwplot()` fournit des informations globalement identiques aux graphiques précédent mais sous la forme de boîte à moustaches.

```
bwplot(~ x | y,
main = "note d'algèbre")
```

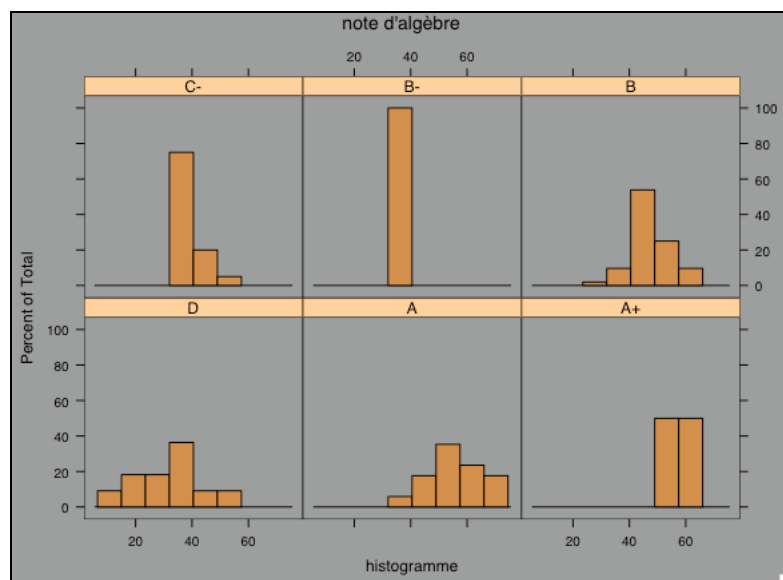


Graphique 9. Utilisation de la fonction bwplot()

IV.7.3 Utilisation de la fonction histogram()

On peut vouloir extraire des informations sous la forme d'histogramme. C'est ce que permet la fonction histogram()

```
histogram(~ x | y,
col = "peru",
main = "note d'algèbre",
xlab = "histogramme")
```

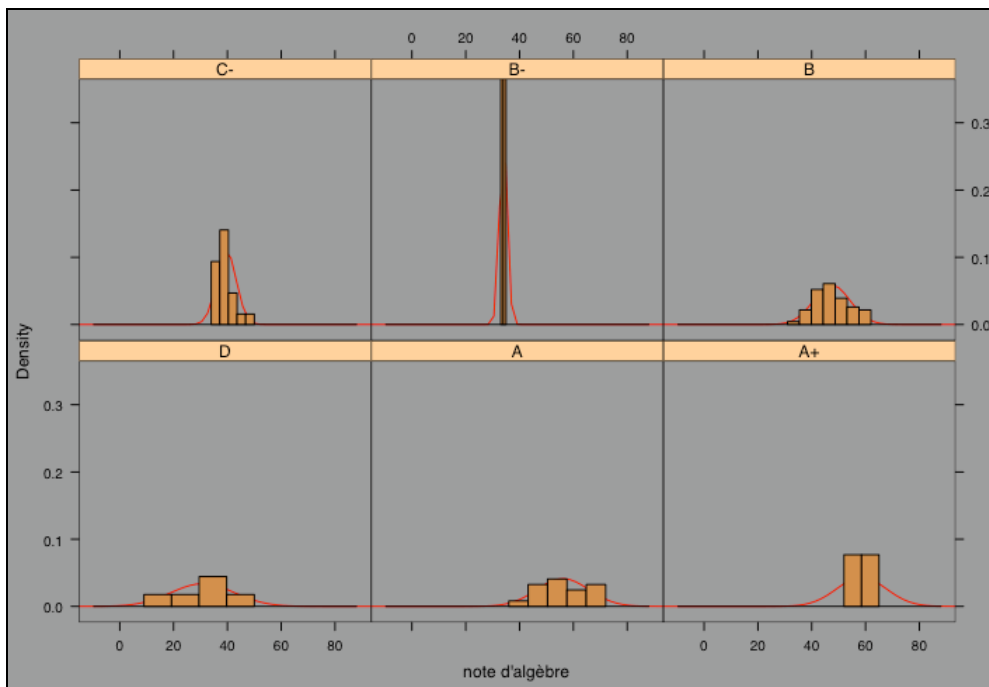


Graphique 10. Utilisation de la fonction histogram()

IV.7.4 Utilisation de l'argument panel

```
densityplot(~ x | y, xlab = "note d'algèbre ",
panel = function(x, ...)
{
panel.mathdensity(dmath=dnorm, args=list(mean=mean(x), sd=sd(x)), col="red")
panel.histogram(x, breaks = NULL, col = "peru")
}
}
```

)

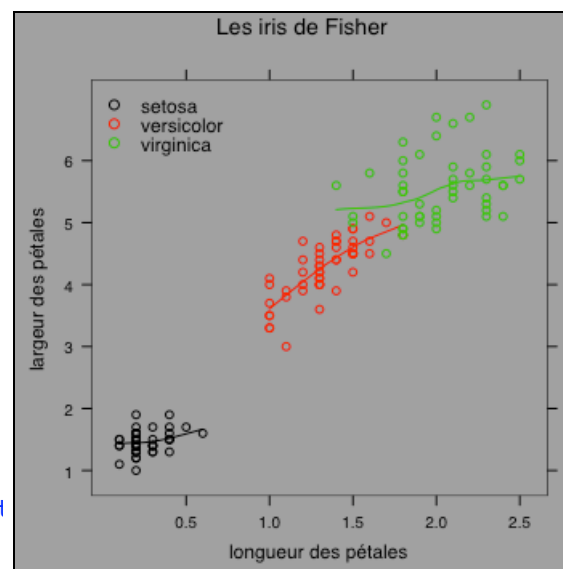


Graphique 11. Utilisation de l'argument panel

Commentons les commandes qui ont permis de générer le graphique 10. La fonction `densityplot()` produit un graphe par sous-échantillon. L'argument `panel` prend en argument une fonction. Dans cet exemple, nous avons construit une fonction qui fait appel à deux fonctions prédéfinies par le package `lattice` : `panel.mathdensity` et `panel.histogram`. `Panel` définit les analyses qui doivent être effectuées pour chaque sous population.

IV.7.5 Utilisation de la fonction `xyplot()`

```
xyplot(iris$Petal.Length ~ iris$Petal.Width,
       groups=iris$Species,
       type = c("p", "smooth"),
       col = 4:6,
       xlab = "longueur des pétales",
       ylab = "largeur des pétales",
       main = "Les iris de Fisher",
       span = 0.75,
       key = list(
         x = 0.15,
         y = 0.85,
         points=list(
           col=4:6,
           pch = 1
         )
       ),
       t e x = l
list(levels(iris$Species))
)
```



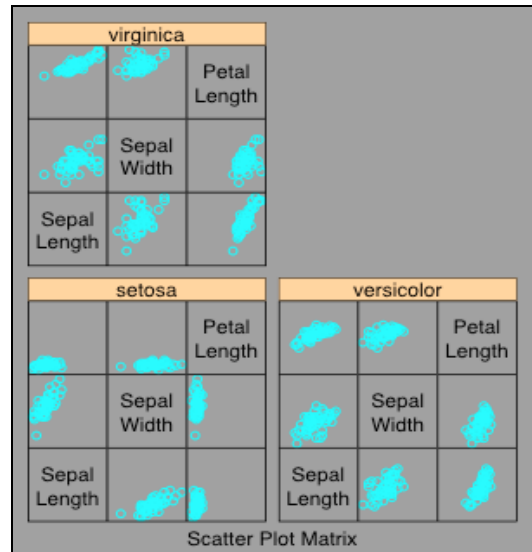
Graphique 12 Utilisation de la fonction `xyplot()`

IV.7.6 Utilisation de la fonction `splom()`


```

splom(~iris[1:3] | Species, data = iris,
      pscales = 0,
      varnames = c("Sepal\nLength",
                  "Sepal\nWidth",
                  "Petal\nLength"
                  )
)

```



Graphique 12. Utilisation de la fonction splom()

Commentons les commandes qui nous ont permis d'obtenir le graphique 12. La fonction splom() permet donc de tracer tous les graphes bivariés d'une matrice (à l'instar de la fonction pairs()) mais permet en plus de partitionner les graphes selon la valeur d'une autre variable (une variable qualitative par exemple).