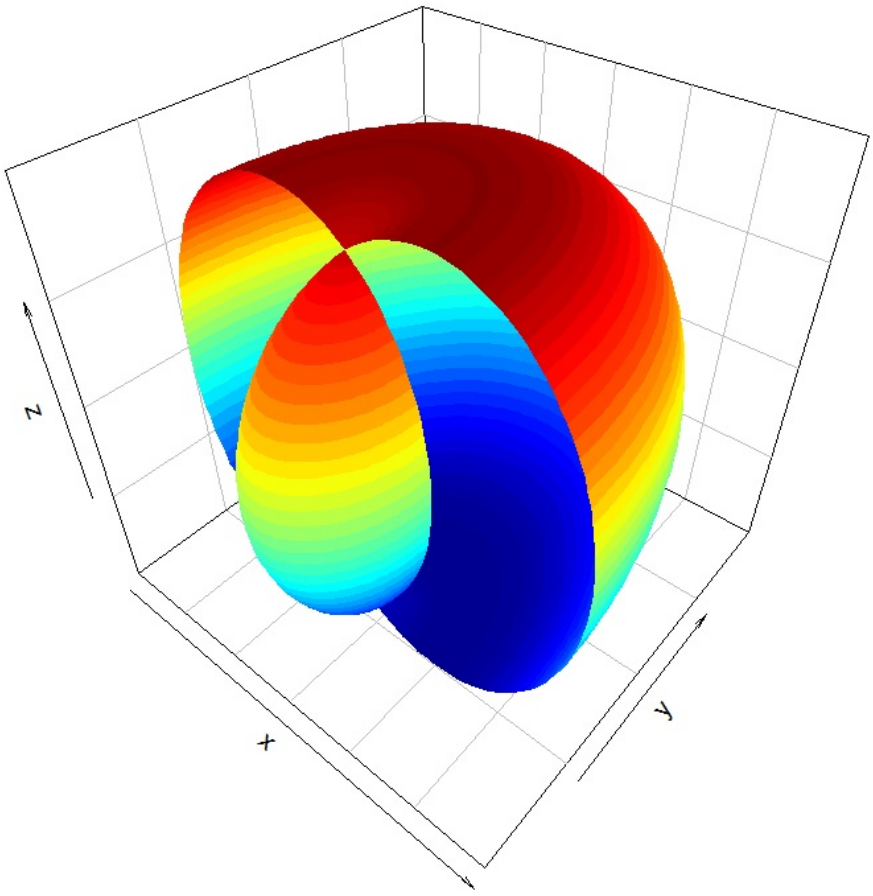
Université de Caen

Introduction aux graphiques avec 

Christophe Chesneau

http://www.math.unicaen.fr/~chesneau/



Caen, le 25 Août 2016

Table des matières

~ Note ~

L'objectif de ce document est de présenter les principales commandes graphiques offertes par le logiciel R. Contact : `christophe.chesneau@gmail.com`

Bonne lecture !

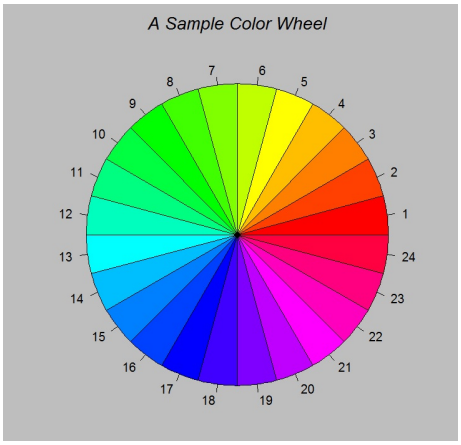
1 Point de départ

Pour avoir un aperçu des possibilités graphiques du logiciel R, on fait :

```
demo(graphics)
```

Taper sur la touche "Entrée" pour faire défiler plusieurs graphiques.

On obtient, entre autre, celui là :



Une fois que plus rien ne s'affiche, on fait :

```
dev.off()
```

Ces graphiques sont obtenus à l'aide de commandes. On en distingue trois sortes :

- celles qui créent une figure graphique dans une fenêtre,
- celles qui ajoutent une figure graphique à la fenêtre existante,
- celles qui modifient les paramètres du graphisme.

Pour illustrer ces commandes, on considèrera dès que possible le jeu de données "enquete" :

```
enquete = read.table("http://www.math.unicaen.fr/~chesneau/enquete.txt",
header = T)
attach(enquete)
str(enquete)
```

2 Créer un ou plusieurs graphiques

2.1 La commande `stripchart`

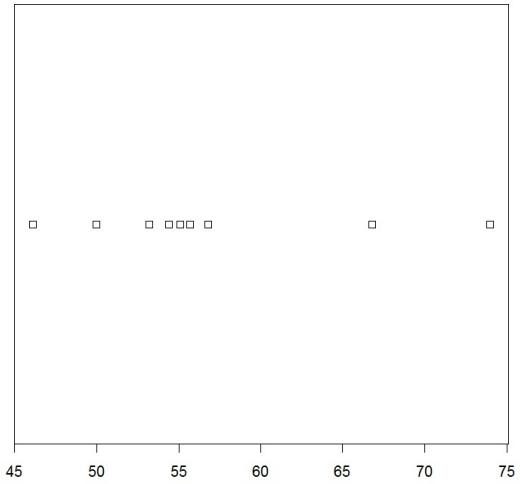
L'utilisation de base est `stripchart(x)`, où `x` désigne un vecteur numérique.

On affiche alors les valeurs ordonnées des éléments de `x` sur un axe permettant de juger de la dispersion des valeurs.

On fait :

```
stripchart(poids)
```

Cela renvoie :



2.2 La commande `dotchart`

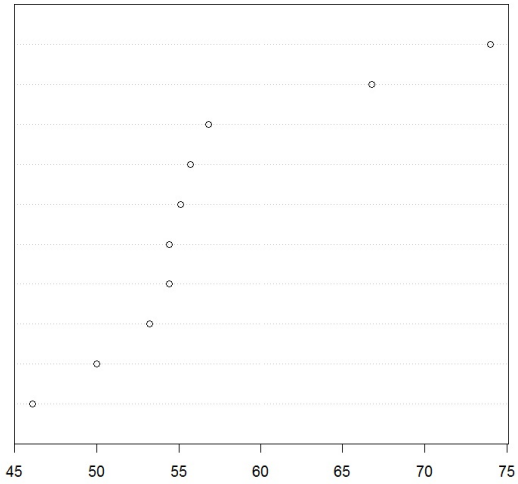
L'utilisation de base est `dotchart(x)`, où `x` désigne un vecteur numérique.

La valeur de chaque élément de `x` est affichée sur une ligne différente. Pour faciliter l'étude de la dispersion des valeurs, il convient d'ordonner les valeurs.

On fait :

```
dotchart(sort(poids))
```

Cela renvoie :



Notons que, si plusieurs graphiques se succèdent, seul le dernier apparait.

2.3 La commande `plot`

Utilisation de base

L'utilisation de base est `plot(x, y)`, où `x` et `y` sont 2 vecteurs de même longueur.

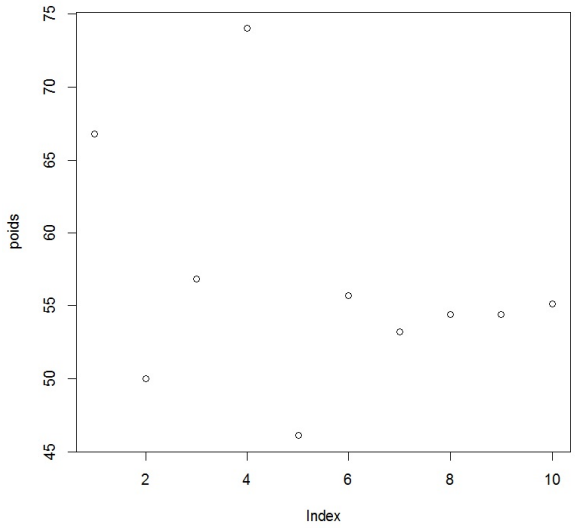
On construit alors un nuage de points dont le i-ème point est de coordonnées `(x[i], y[i])`.

Si un seul vecteur `y` est indiqué, on un nuage de points dont le i-ème point est de coordonnées `(i, y[i])`.

On fait :

```
plot(poids)
```

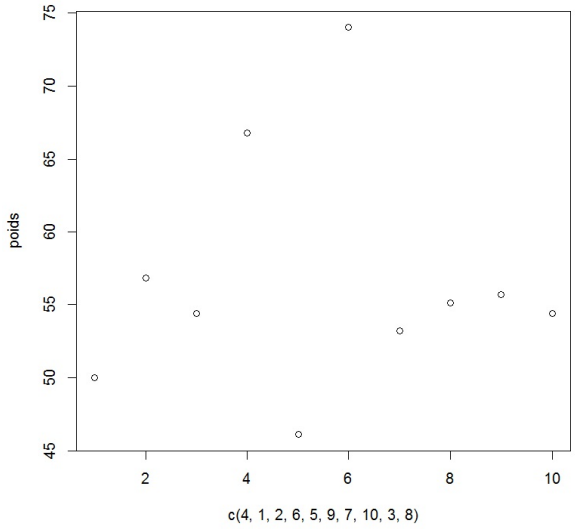
Cela renvoie :



On fait :

```
plot(c(4, 1, 2, 6, 5, 9, 7, 10, 3, 8), poids)
```

Cela renvoie :



Options graphiques

Il existe des options dans `plot` permettant de changer les paramètres graphiques. On les active en rajoutant une ou plusieurs commandes dans `plot`. Par exemple, on fait :

```
plot(poids, type = "l", lty = 2, axes = F, main = "poids des personnes")
```

Quelques options sont présentées ci-dessous.

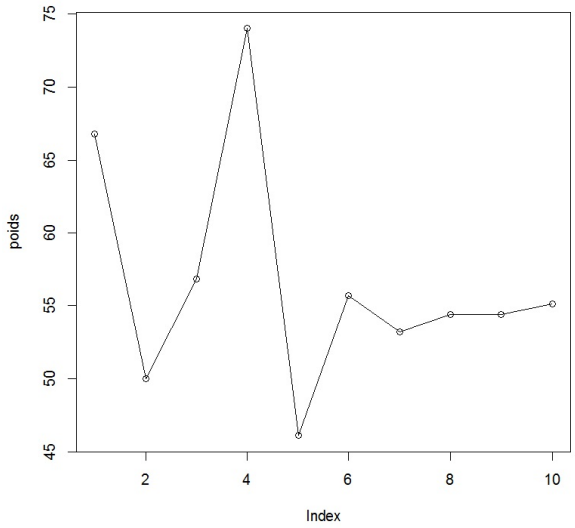
Option : `type`. On considère les commandes `type = "c"`

- si `c = p`, seul le nuage de point est construit (`p` est l'option par défaut),
- si `c = n`, seul l'encadrement est tracé,
- si `c = l`, les points sont reliés par une ligne,
- si `c = h`, des lignes verticales sont tracées,
- si `c = o` ou si `c = b`, les points sont marqués et reliés par une ligne.

On considère les commandes :

```
plot(poids, type = "o")
```

Cela renvoie :

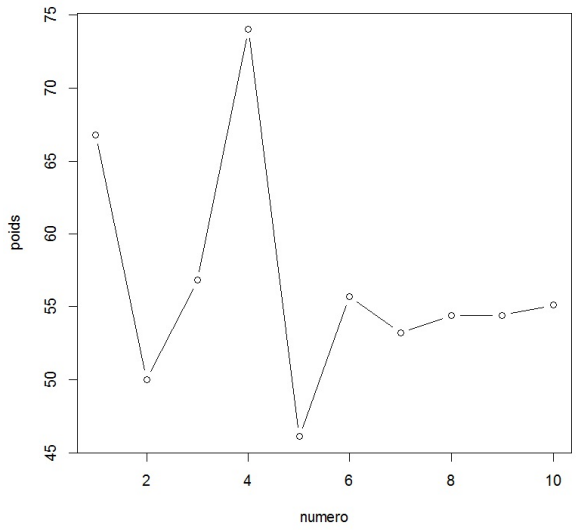


Option : `xlab`. Les commandes `xlab = "string"` et `ylab = "string"`, où `string` est une chaine de caractères, donnent un nom aux axes des coordonnées. Par défaut, ce sont les noms de `x` et `y`.

On considère les commandes :

```
plot(poids, type = "b", xlab = "numero")
```

Cela renvoie :

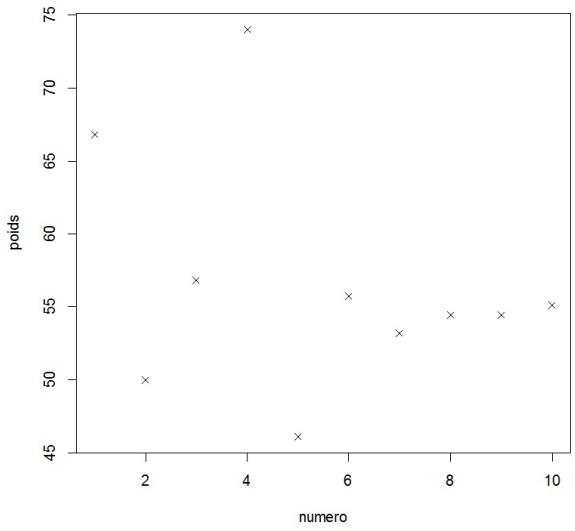


Option : `pch`. Les commandes `pch = n`, où `n` est un entier ou un caractère, changent la nature des points du graphique.

On considère les commandes :

```
plot(poids, pch = 4, xlab = "numero")
```

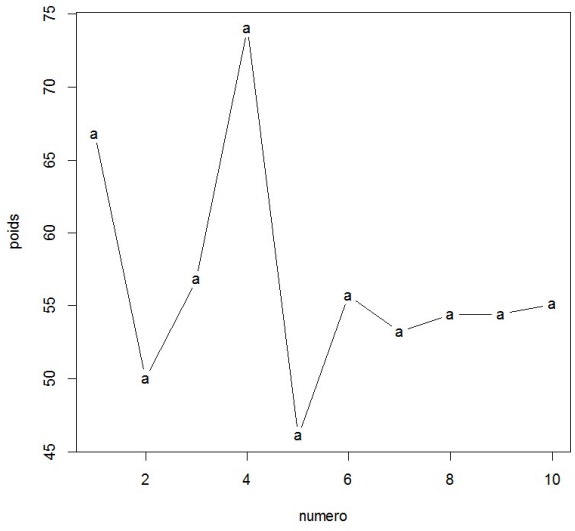
Cela renvoie :



On fait :

```
plot(poids, type = "b", pch = "a", xlab = "numero")
```

Cela renvoie :

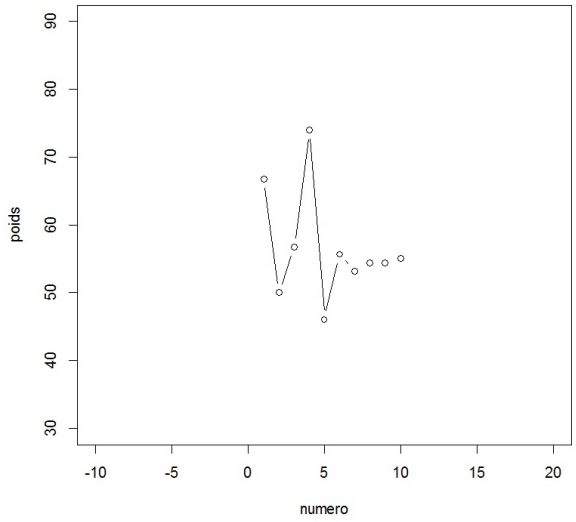


Option : `xlim`. Les commandes `xlim = c(a, b)` et/ou `ylim = c(a, b)`, où `a` et `b` sont deux nombres réels, imposent des limites aux axes.

On considère les commandes :

```
plot(poids, type = "b", xlab = "numero", xlim = c(-10, 20), ylim = c(30,90))
```

Cela renvoie :

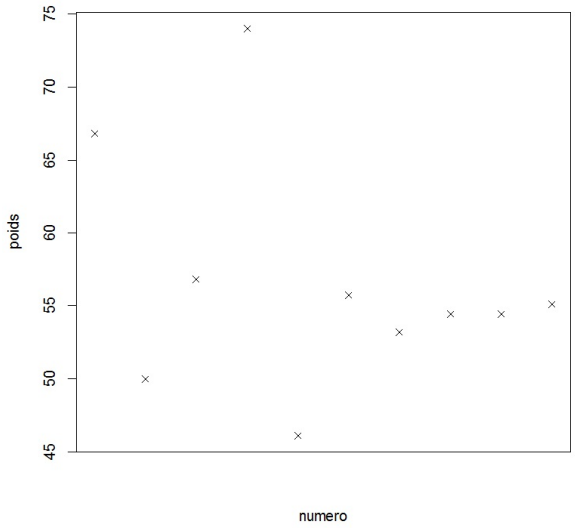


Option : `xaxt`. Les commandes `xaxt = "n"` et/ou `yaxt = "n"` effacent à la fois les tirets qui marquent les axes et les valeurs qui correspondent à ces tirets.

On fait :

```
plot(poids, type = "b", xlab = "numero", xlim= c(-10, 20), ylim = c(30, 90))
```

Cela renvoie :

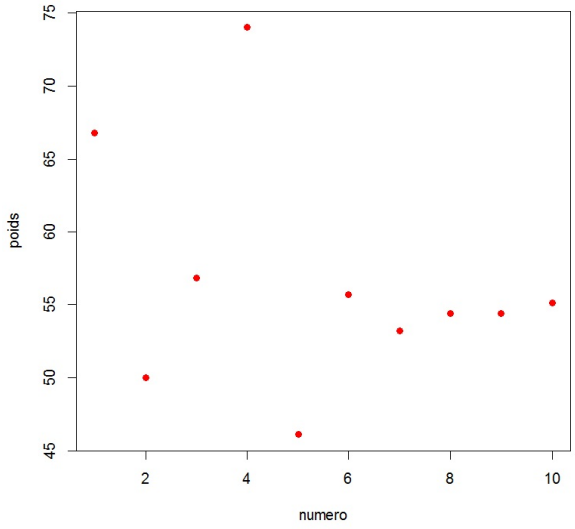


Option : `col`. Les commandes `col = "string"`, où `string` est une couleur : `red`, `yellow`, `green`, `blue`... ajoutent de la couleur.

On fait :

```
plot(poids, pch = 19, xlab = "numero", col = "red")
```

Cela renvoie :



On peut aussi utiliser :

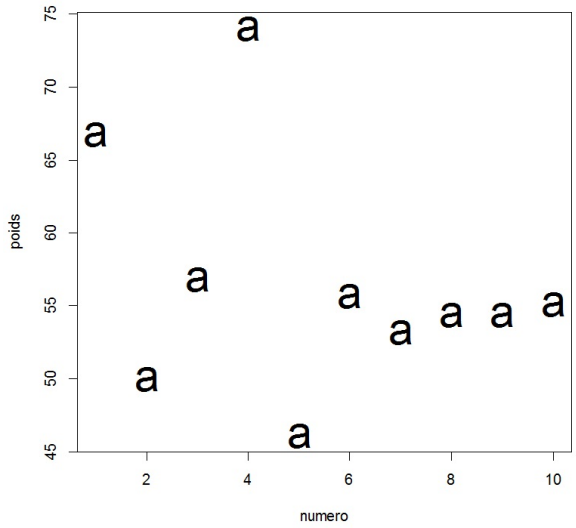
- des fonctions générant des couleurs par les commandes `col = rainbow(n)`, `col = heat.color(n)`, `col = terrain.colors(n)`, `col = topo.colors(n)`, `col = cm.colors(n)`, où `n` désigne un entier.
- des codes hexadécimaux avec les 2 premières uintés pour le rouge, les deux suivantes pour le vert et les deux dernières pour le bleu : `col = "#120019"`, `col = "#123418"`, `col = "#1200FF"`...

Option : `cex`. Les commandes `cex = l`, où `l` est un réel positif, multiplient la taille des caractères contenus dans la fenêtre par `l`. De même, les commandes `cex.axis = l` multiplient la taille des caractères indiquant les étiquettes des axes par `l`.

On fait :

```
plot(poids, pch = "a", xlab = "numero", cex = 3)
```

Cela renvoie :

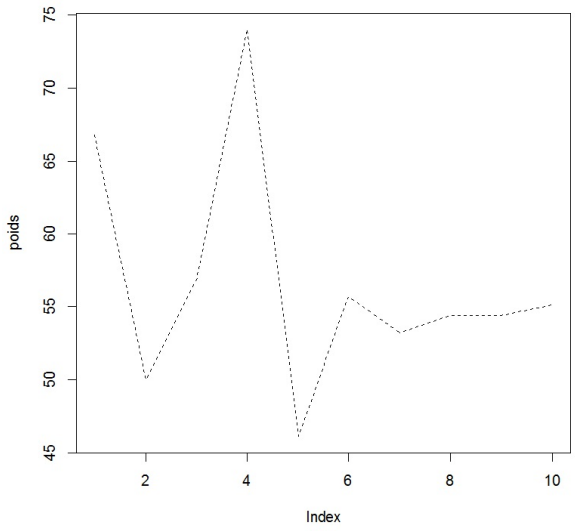


Option : `lty`. Les commandes `lty = m`, où `m` est un entier, changent la nature des lignes qui relient les points.

On considère les commandes :

```
plot(poids, type = "l", lty = 2)
```

Cela renvoie :

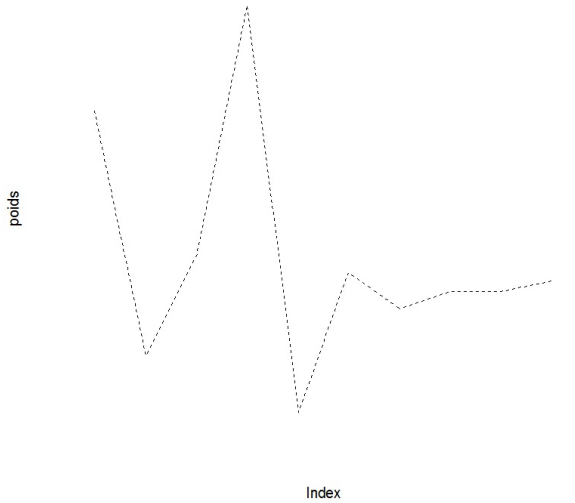


Option : `axes`. Les commandes `axes = F` effacent l'entourage de la fenêtre.

On fait :

```
plot(poids, type = "l", lty = 2, axes = F)
```

Cela renvoie :

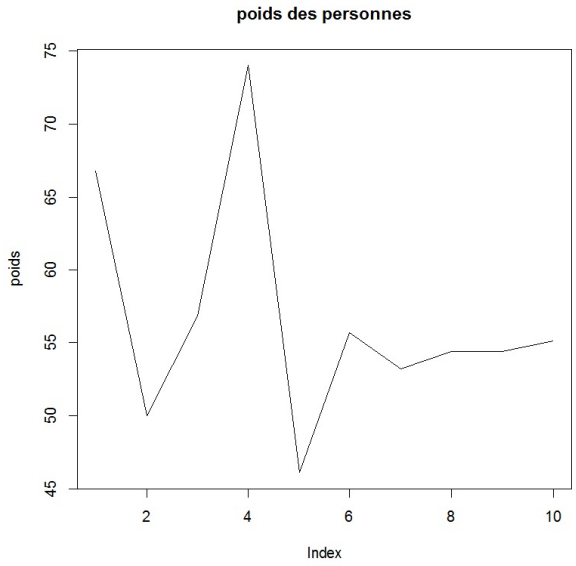


Option : `main`. Les commandes `main = "string"`, où `string` est une chaîne de caractères, mettent un titre au graphe.

On fait :

```
plot(poids, type = "l", main = "poids des personnes")
```

Cela renvoie :

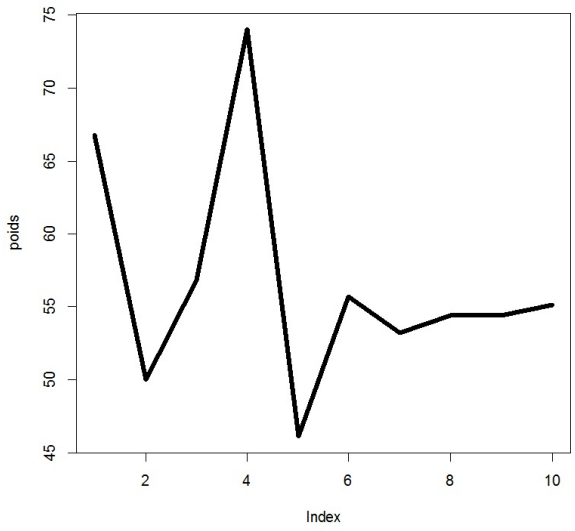


Option : `lwd`. Les commandes `lwd = m`, où `m` est un entier, changent l'épaisseur des lignes/traits du graphe.

On fait :

```
plot(poids, type = "l", lwd = 5)
```

Cela renvoie :



Pour plus de détails sur les options de `plot`, faire `help("plot")`. Cela vaut aussi pour les commandes à venir.

2.4 La commande `curve`

La commande `curve` sert à confectionner rapidement certaines courbes représentatives de fonctions.

On peut notamment l'utiliser pour représenter la densité et la fonction de répartition des lois d'une variable à densité.

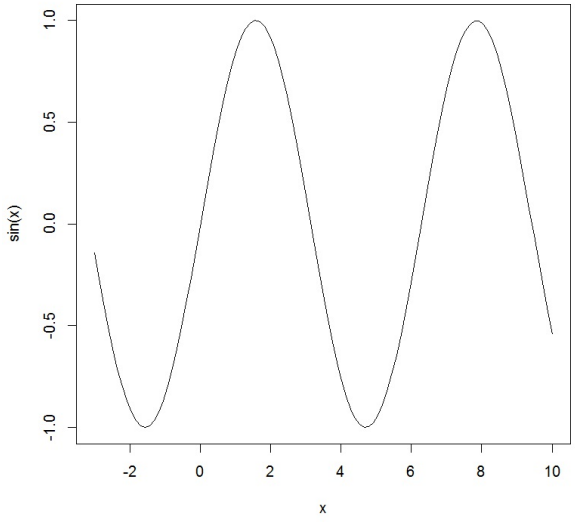
Pour dessiner la courbe représentative d'une fonction `f(x)` entre `a` et `b`, on fait :

`curve(f(x), a, b)`

On considère les commandes :

```
curve(sin(x), -3, 10)
```

Cela renvoie :

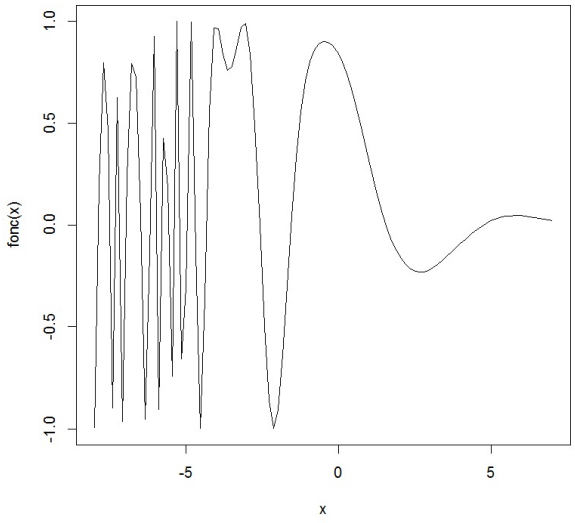


On considère les commandes :

```
fonc = function(x) { sin(cos(x) * exp( -x / 2)) }

curve(fonc, -8, 7)
```

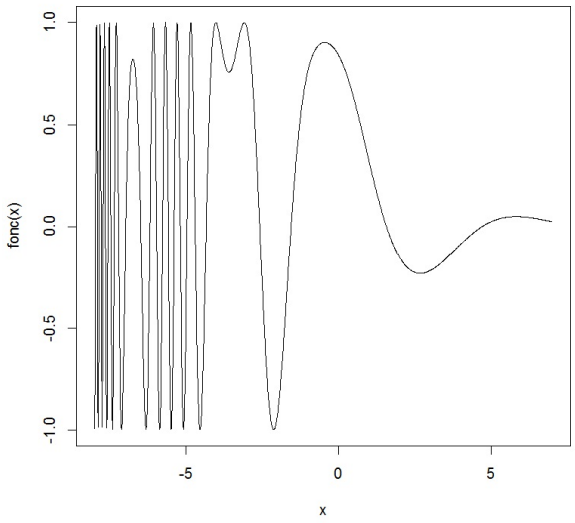
Cela renvoie :



On peut augmenter le nombre de points `n` à discrétiser entre `a` et `b` en faisant :

```
curve(fonc, -8, 7, n = 2001)
```

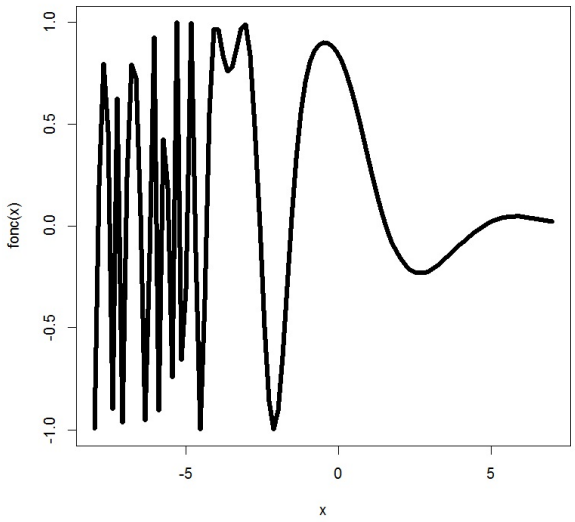
Cela renvoie :



On peut changer l'épaisseur de la courbe en faisant :

```
curve(fonc, -8, 7, lwd = 5)
```

Cela renvoie :



La plupart des options de `plot` (`col`, `lwd`...) peuvent aussi être rajoutées dans `curve`.

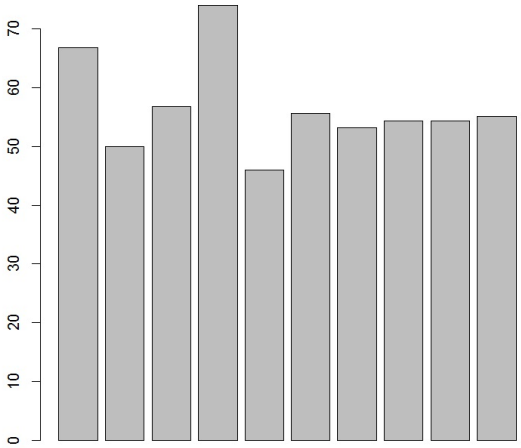
2.5 La commande `barplot`

Utilisation de base

Pour un vecteur `x` à n éléments, les commandes `barplot(x)` donnent n barres verticales, la i-ème barre étant de hauteur proportionnelle à `x[i]`. On fait :

```
barplot(poids)
```

Cela renvoie :



Utilisation en statistique

La commande `barplot` peut servir à :

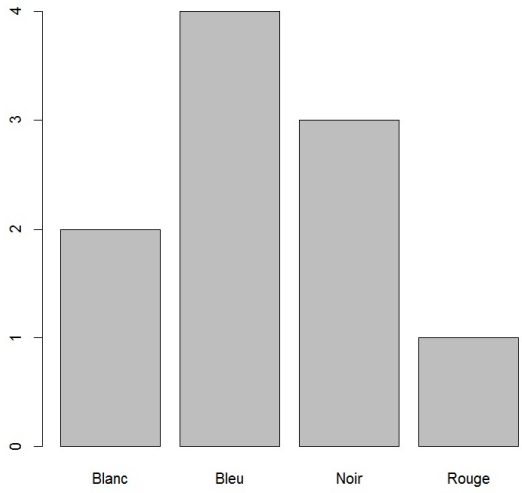
- représenter la série brute de données issues d'une variable quantitative (quand l'effectif n'est pas trop grand) en confectionnant un diagramme en barres ou un diagramme en bandeaux.
- représenter la distribution d'une variable qualitative (en effectifs et/ou en fréquence).

  On rappelle que cette distribution s'obtient en associant à chaque modalité m_i que peut prendre la variable, l'effectif n_i et/ou la fréquence n_i/n. Dans ce cas, on a toujours en abscisse, les modalités pouvant être prises par la variable et, en ordonnée, l'effectif ou la fréquence.

On fait :

```
barplot(table(couleur))
```

Cela renvoie :

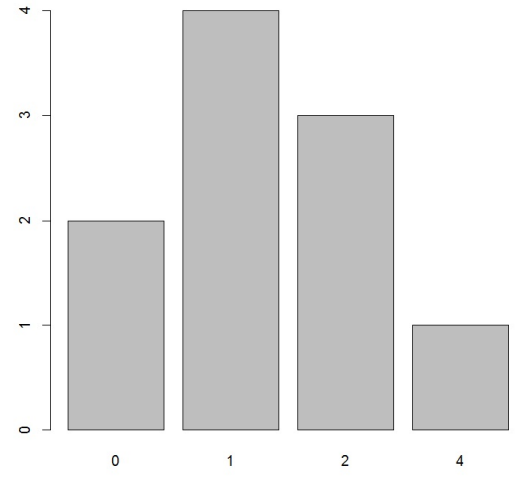


- représenter la distribution en effectif et/ou en fréquence d'une variable discrète.

  On fait :

```
barplot(table(nb))
```

  Cela renvoie :



Options graphiques

Comme pour `plot`, il existe des options dans `barplot` permettant de changer les paramètres graphiques. On les active en rajoutant une ou plusieurs commandes dans `barplot`.

Par exemple, on fait :

```
barplot(poids, xlim = c(-3, 33), width = 0.8)
```

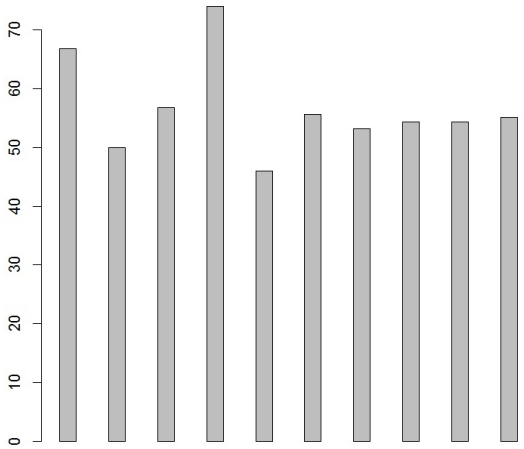
Quelques options sont présentées ci-dessous.

Option : `space`. Les commandes `space = l`, où `l` est un nombre réel, changent l'écartement entre les barres qui seront espacées de $\widehat{l}$ fois la largeur moyenne des barres.

On fait :

```
barplot(poids, space = 2)
```

Cela renvoie :

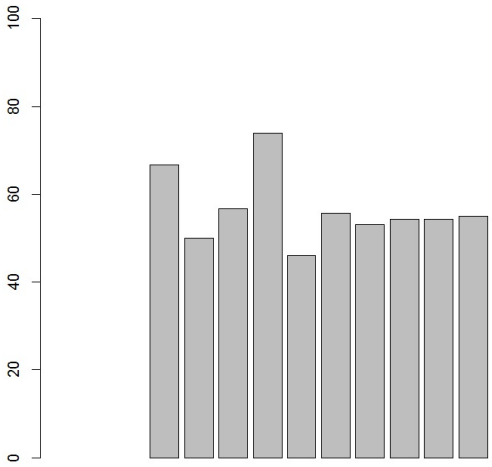


Option : `xlim`. Les commandes `xlim = c(a, b)` et/ou `ylim = c(a, b)`, où `a` et `b` sont deux nombres réels, imposent des limites aux axes.

On fait :

```
barplot(poids, xlim = c(-3, 13), ylim = c(0, 100))
```

Cela renvoie :

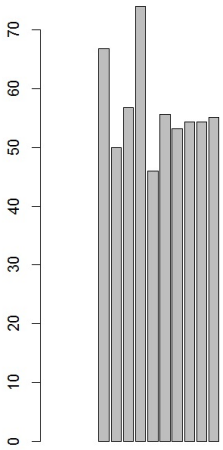


Option : `width`. Si l'option `xlim` a été utilisée, on peut spécifier la largeur des barres par `width = d`, où `d` est la largeur souhaitée.

On considère les commandes :

```
barplot(poids, xlim = c(-3, 33), width = 0.8)
```

Cela renvoie :

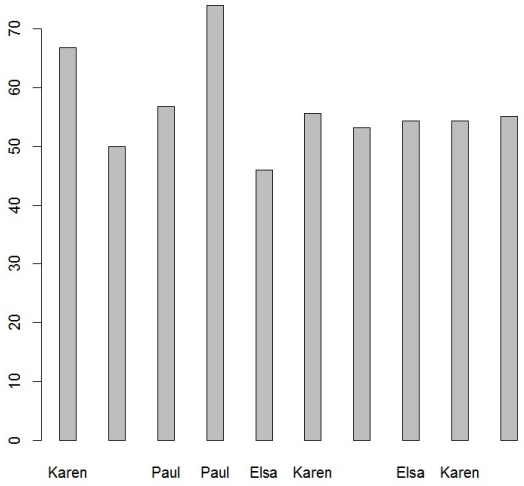


Option : `names.arg`. Les commandes `names.arg = string`, où `string` est un vecteur de chaîne de caractères, attribuent le nom du i-ème élément de `string` à la i-ème barre.

On considère les commandes :

```
prenoms = c("Karen", "Elodie", "Paul", "Paul", "Elsa", "Karen", "Aurelie",
"Elsa", "Karen", "Sophie")
barplot(poids, space = 2, names.arg = prenoms)
```

Cela renvoie :

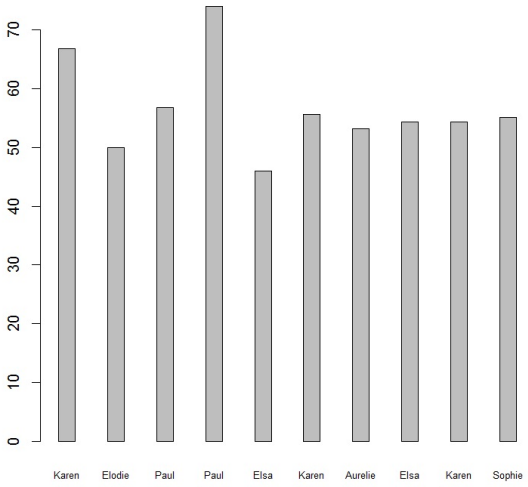


Les prénoms attribués prennent trop de place, c'est pourquoi ils ne sont pas tous affichés.

Une solution est :

```
barplot(poids, space = 2, names.arg = prenoms, cex.names = 0.4)
```

Cela renvoie :

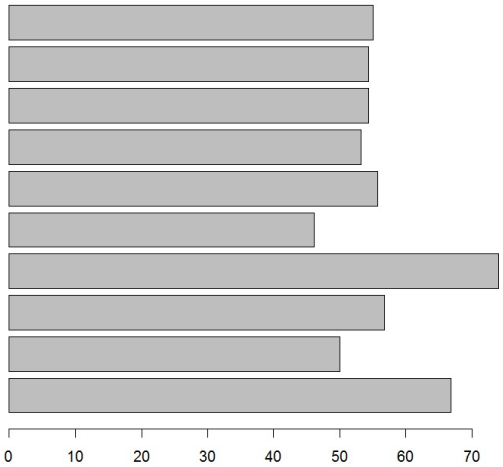


Option : `horiz`. Les commandes `horiz = L`, où `L` est `TRUE` ou `FALSE`, tracent les barres à l'horizontale si `L = TRUE` et à la verticale sinon.

On fait :

```
barplot(poids, horiz = TRUE)
```

Cela renvoie :



La plupart des options de `plot` (`col`, `lwd`...) peuvent aussi être rajoutées dans `barplot`.

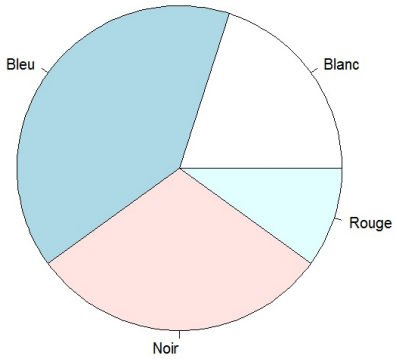
2.6 La commande `pie`

La commande basique est `pie(x)`, où `x` est un vecteur de longueur l. Elle constitue un diagramme à p secteurs. Elle est principalement utilisée pour représenter la distribution d'une variable qualitative à l modalités.

On fait :

```
pie(table(couleur))
```

Cela renvoie :



2.7 La commande `hist`

La commande basique est `hist(x)` ou `hist(x, prob = TRUE)`, où `x` est un vecteur de longueur n.

Cette commande :

- construit automatiquement une suite de l classes adjacentes $[\eta_0, \eta_1],]\eta_1, \eta_2], \ldots,]\eta_{l-1}, \eta_l]$ telles que l'ensemble des valeurs de `x` est inclus dans $[\eta_0, \eta_l]$ et ceci suivant une règle dite de Sturges. Par défaut, les classes sont de même amplitude.
- calcule automatiquement la distribution en effectifs de série des données `x` dans les classes ci-dessus, *ie* la série des n_j où n_j est l'effectif des données qui appartiennent à la classe $]\eta_{j-1}, \eta_j]$.
- construit l rectangles accolés. On distingue alors 2 possibilités :

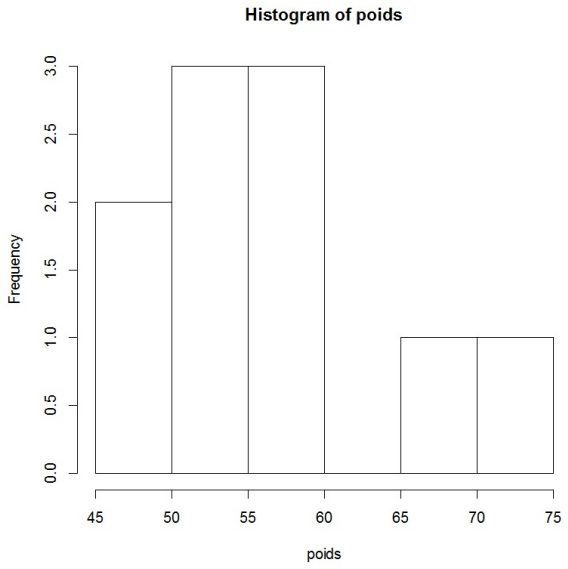
  Par défaut ; si `prob = FALSE` *:* la hauteur du rectangle pour la classe j est égale à n_j. Il s'agit alors d'un histogramme des effectifs. Sur le graphe, les ordonnées sont notées (Frequency) alors qu'en réalité, il s'agit bien des effectifs.

  Si `prob = TRUE` *:* la hauteur h_j du rectangle pour la classe j est égale à f_j/a_j où $f_j = n_j/n$ fréquence de la classe j et où $a_j = \eta_j - \eta_{j-1}$ amplitude de la classe j. Il s'agit alors d'un histogramme des fréquences. Sur le graphe, les ordonnées sont indiquées (Density).

On fait :

```
hist(poids)
```

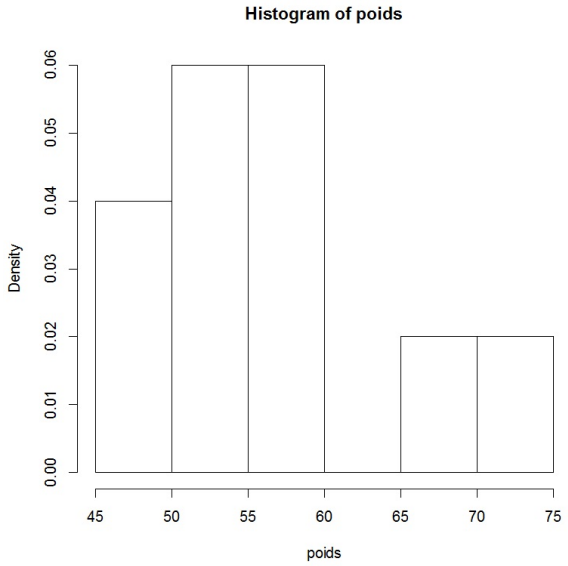
Cela renvoie :



On considère les commandes :

```
hist(poids, prob = TRUE)
```

Cela renvoie :



Options graphiques

Comme pour `plot`, il existe des options dans `hist` permettant de changer les paramètres graphiques. On les active en rajoutant une ou plusieurs commandes dans `hist`. Par exemple, on fait :

```
hist(poids, breaks = c(45, 60, 75), prob = T)
```

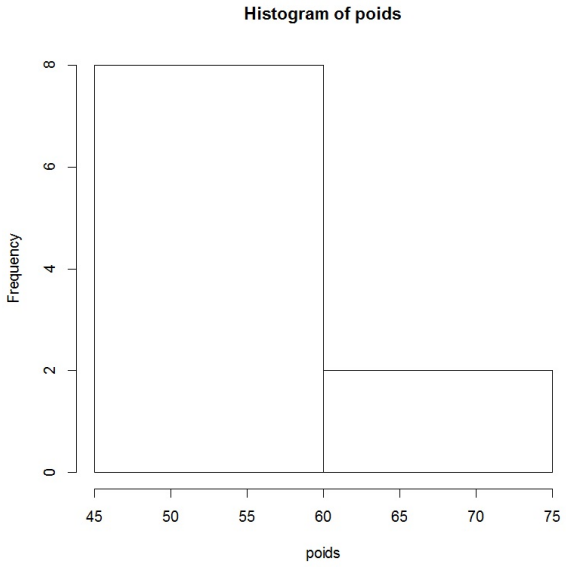
Quelques options sont présentées ci-dessous.

Option : `break`. Les commandes `breaks = x`, où `x` est un vecteur numérique, donnent les extrémités des classes.

On considère les commandes :

```
hist(poids,breaks = c(45, 60, 75))
```

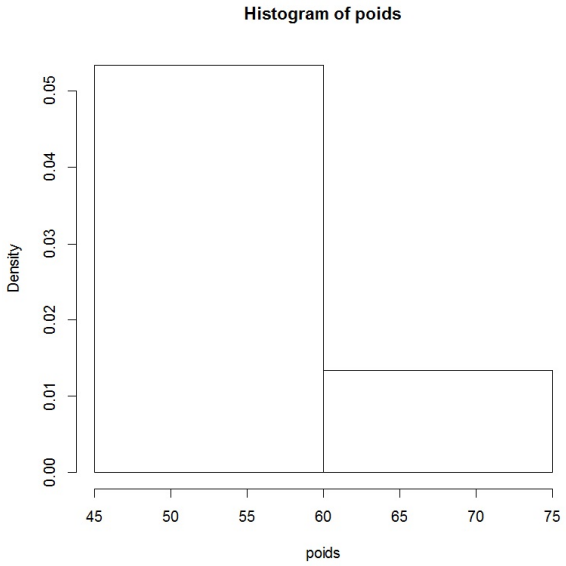
Cela renvoie :



On considère les commandes :

```
hist(poids,breaks = c(45, 60, 75), prob = TRUE)
```

Cela renvoie :

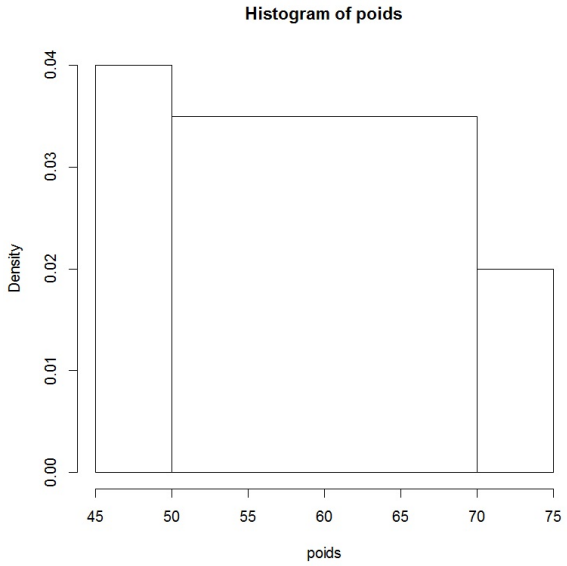


Notons que l'histogramme obtenu a même allure que le précédent mais on se trouve dans le deuxième cas et les hauteurs des rectangles sont maintenant des densités.

On fait :

```
hist(poids,breaks = c(45, 50, 70, 75))
```

Cela renvoie :



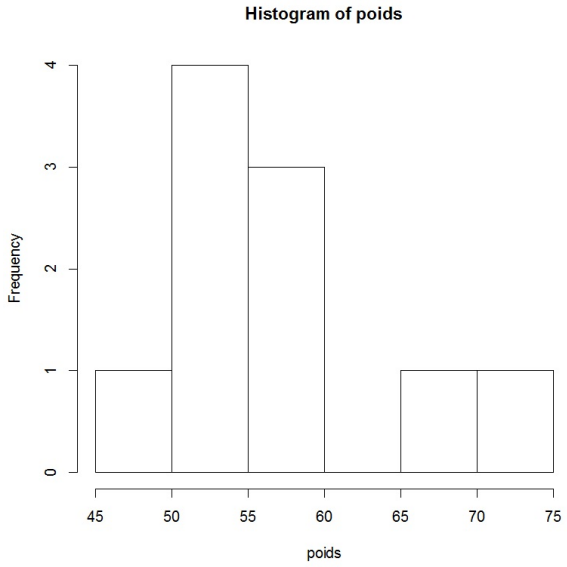
Ainsi, le fait que les classes ne soient plus d'amplitudes égales active l'option `prob = TRUE`.

Option : `right`. Les commandes `right = L`, où `L` est `TRUE` ou `FALSE`, considèrent des intervalles de la forme $]\eta_{j-1}, \eta_j]$ si `L = TRUE`, sinon de la forme $[\eta_{j-1}, \eta_j[$.

On fait :

```
hist(poids, right = FALSE)
```

Cela renvoie :



La plupart des options de `plot` (`col`, `lwd`...) peuvent aussi être rajoutées dans `hist`.

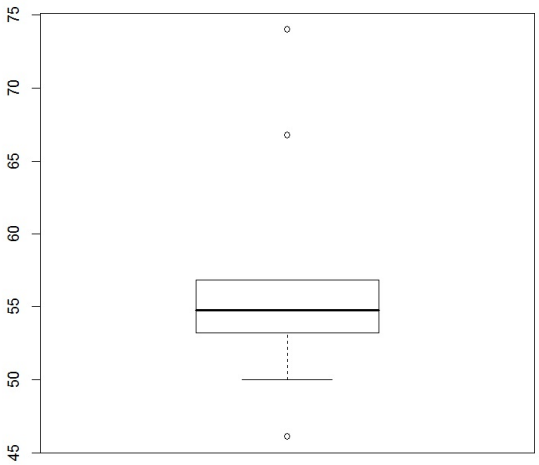
2.8 La commande `boxplot`

La commande basique est `boxplot(x)`, où `x` est un vecteur de longueur n. Elle construit la boîte à moustaches de `x`.

On fait :

```
boxplot(poids)
```

Cela renvoie :



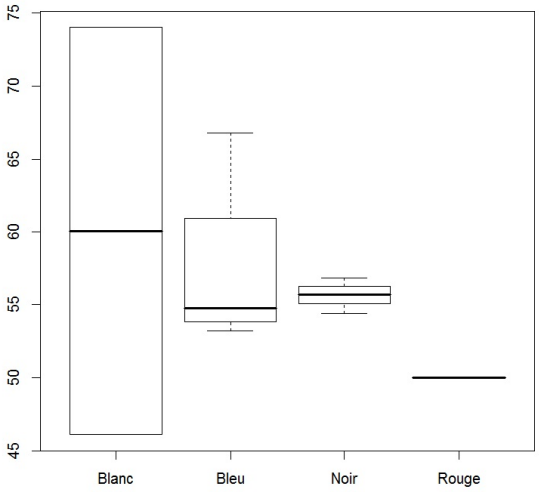
Pour construire une boîte à moustaches de `x` pour chaque élément d'un vecteur `y`, on fait :

`boxplot(x ~ y)`

On fait :

```
boxplot(poids ~ couleur)
```

Cela renvoie :



2.9 Avoir plusieurs graphiques sur la même fenêtre

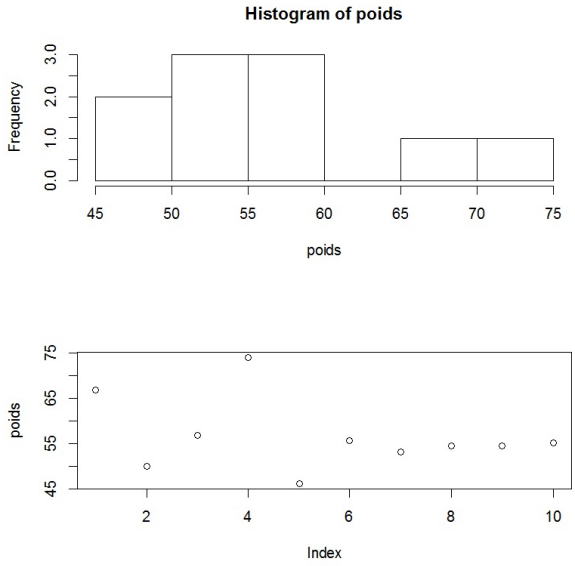
Les commandes `par(mfrow = c(k, l))`, où `k` et `l` sont des entiers, servent à découper l'écran en `k` lignes et `l` colonnes. Lorsque plusieurs commandes créant un graphique se succèdent, ces graphiques se positionnent par ligne sur les cases ainsi créées.

Si on veut qu'ils se positionnent par colonne, on fait `par(mfcol = c(k, l))`.

On fait :

```
par(mfrow = c(2, 1))
hist(poids)
plot(poids)
```

Cela renvoie :



Pour revenir à la configuration de départ, on fait `par(mfrow = c(1, 1))`.

2.10 Avoir plusieurs graphiques successifs

Les commandes `par(ask = TRUE)` affichent plusieurs graphiques successivement.

On considère les commandes :

```
par(ask = TRUE)

hist(poids)

plot(poids)
```

Pour revenir à la configuration de départ, on fait `par(ask = FALSE)`.

3 Additionner un graphique à un graphique existant

3.1 La commande `points`

L'utilisation de base est `points(x, y)`, où `x` et `y` sont 2 vecteurs de même longueur.

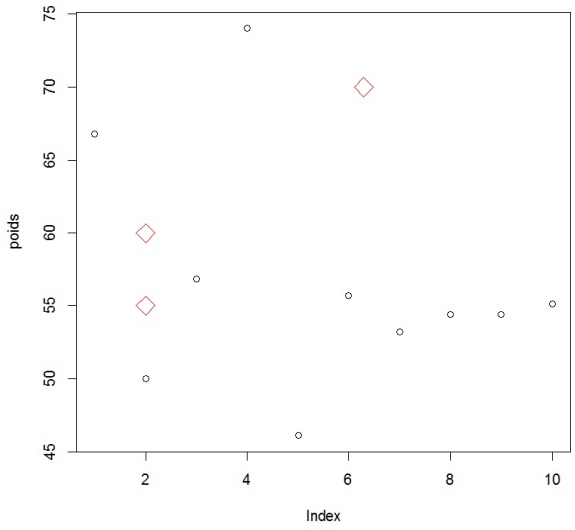
Cela ajoute à la figure existante un nuage de points associé à `(x, y)`.

On fait :

```
plot(poids)

points(c(2, 2, 6.3), c(55, 60, 70), cex = 2, pch = 5, col = "red")
```

Cela renvoie :



3.2 La commande `lines`

L'utilisation de base est `lines(x, y)`, où `x` et `y` sont 2 vecteurs de même longueur.

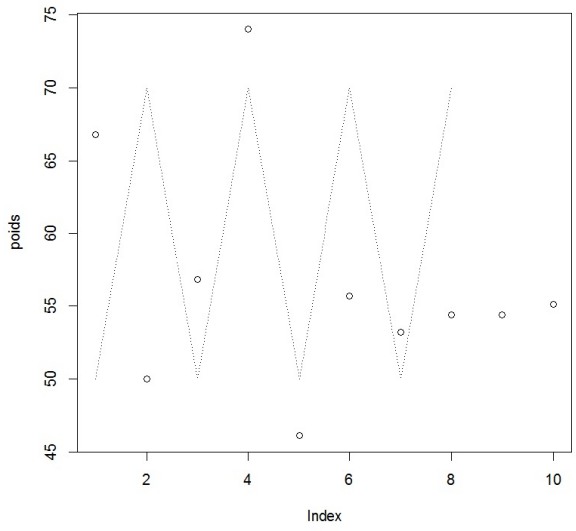
Cela ajoute à la figure existante une ligne reliant les points du nuage de points associé à `(x, y)`.

On fait :

```
plot(poids)

lines(1:8, rep(60, 8) + c(-10, 10), lty = 3)
```

Cela renvoie :



3.3 La commande `text`

L'utilisation de base est `text(x, y, string)`, où `x` et `y` sont 2 vecteurs de même longueur, et `string` est un vecteur chaîne de caractères.

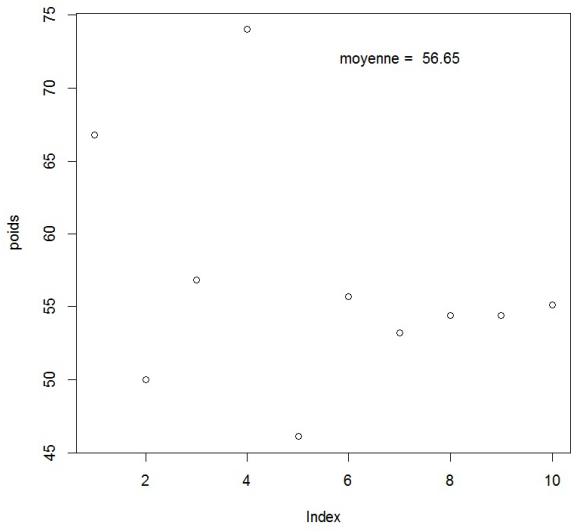
Cela attribue le nom du i-ème élément de `string` au point de coordonnées `(x[i], y[i])`.

On fait :

```
plot(poids)

text(7, 72, paste("moyenne = ", mean(poids)))
```

Cela renvoie :

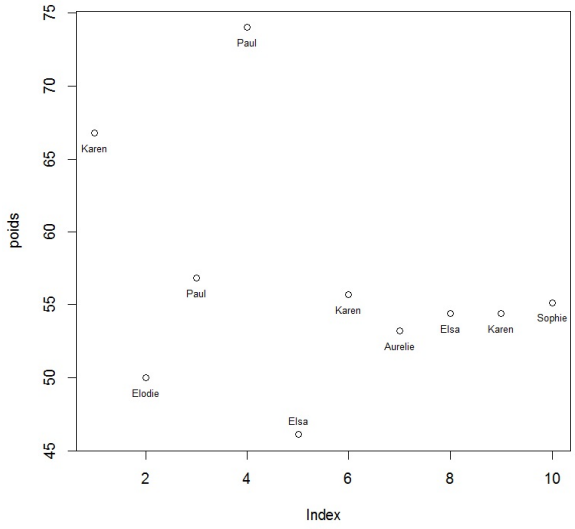


On considère les commandes :

```
plot(poids)

text(poids + c(rep(-1, 4), 1, rep(-1, 5)), prenoms, cex = 0.7)
```

Cela renvoie :



3.4 La commande `abline`

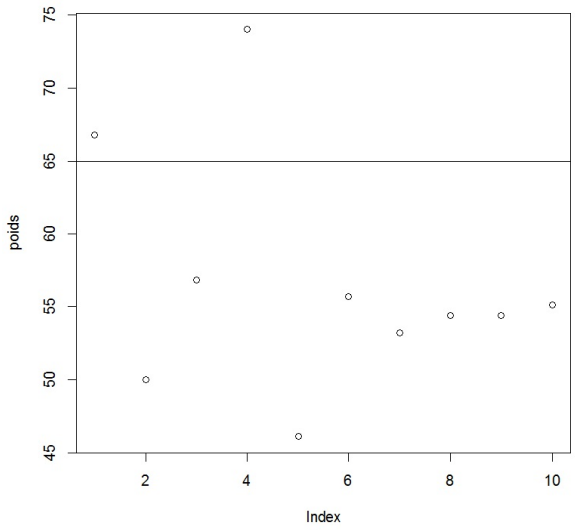
- `abline(h = y)` trace une ligne horizontale de coordonnée y,
- `abline(v = x)` trace une ligne verticale de coordonnée x,
- `abline(a, b)` trace la droite d'équation : $y = a + bx$.

On fait :

```
plot(poids)

abline(h = 65)
```

Cela renvoie :



3.5 Les commandes `curve(f(x), add = TRUE)`

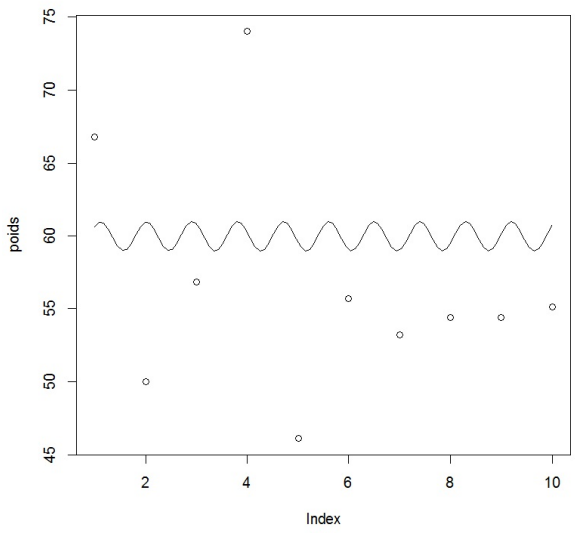
Les commandes `curve(f(x), add = TRUE)` ajoutent à la fenêtre courante le graphe de la fonction `f(x)`.

On fait :

```
plot(poids)

curve(60 + sin(7 * x), add = T)
```

Cela renvoie :



3.6 La commande `segments`

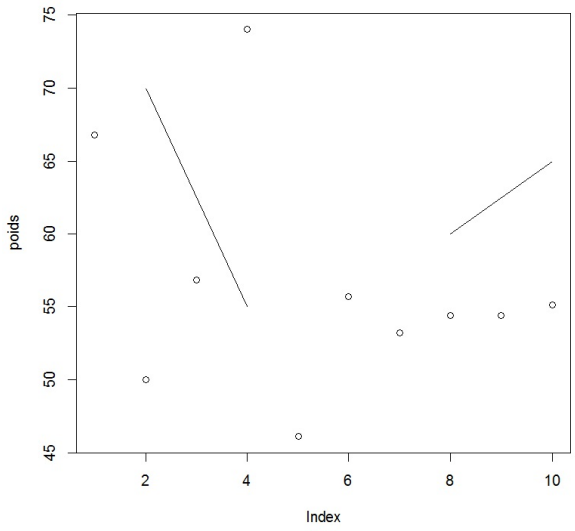
Les commandes `segments(x1, y1, x2, y2)`, où `x1`, `y1`, `x2` et `y2` sont des vecteurs, tracent des segments entre les points de coordonnées : `(x1[i], y1[i])` et `(x2[i], y2[i])`.

On fait :

```
plot(poids)

segments(c(4, 8), c(55, 60), c(2, 10), c(70, 65))
```

Cela renvoie :



Si on remplace `segments` par `arrows`, on trace des flèches entre les points de coordonnées : `(x1[i], y1[i])` et `(x2[i], y2[i])`.

3.7 La commande `title`

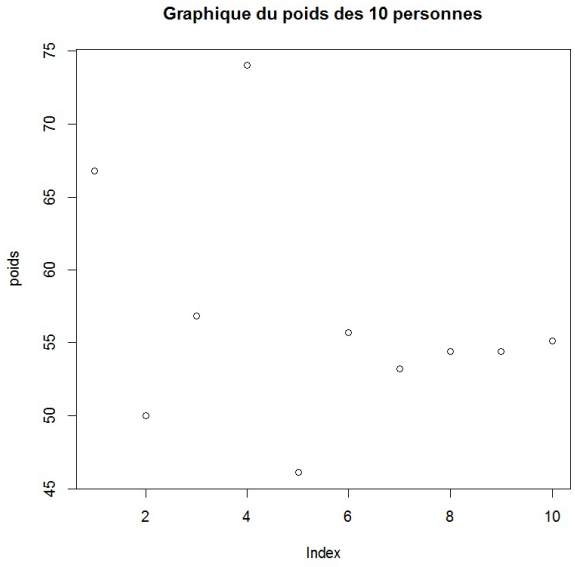
Les commandes `title(string)` ajoutent un titre qui est la chaîne de caractères `string`.

On fait :

```
plot(poids)

title(paste("Graphique du poids des", length(poids), "personnes"))
```

Cela renvoie :



3.8 La commande `axis`

Les commandes `axis(k, vec1, labels = vec2)`, où

- `k` est 1, 2, 3 ou 4 (1 correspond à l'axe des x, 2 à l'axe des y, 3 à l'axe du sommet de la fenêtre et 4 à l'axe à droite de la fenêtre),
- `vec1` est un vecteur qui indique où les tirets doivent être marqués,
- `vec2` un vecteur de chaine de caractères qui donne le nom de chacun des tirets,

changent la configuration de l'axe correspondant à `k`.

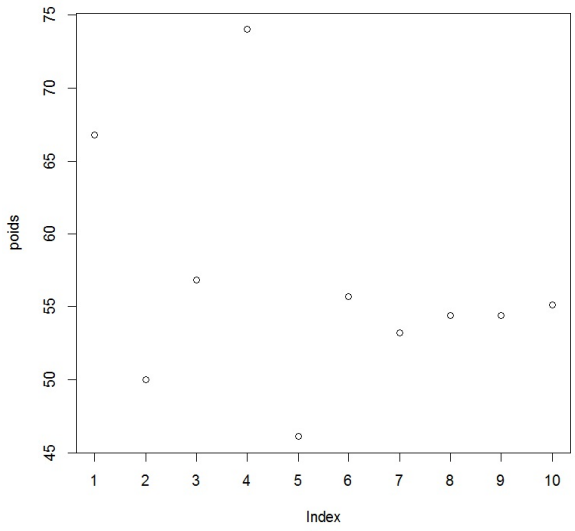
Avant d'utiliser `axis`, il faut inclure dans le plot `xaxt = "n"` et/ou `yaxt = "n"` pour effacer l'écriture automatique des étiquettes des axes.

On fait :

```
plot(poids, xaxt = "n")

axis(1, 1:10, 1:10)
```

Cela renvoie :



3.9 La commande `legend`

Les commandes `legend(x, y, legend)` ajoutent une légende de contenu `legend` au point de coordonnées `(x, y)`.

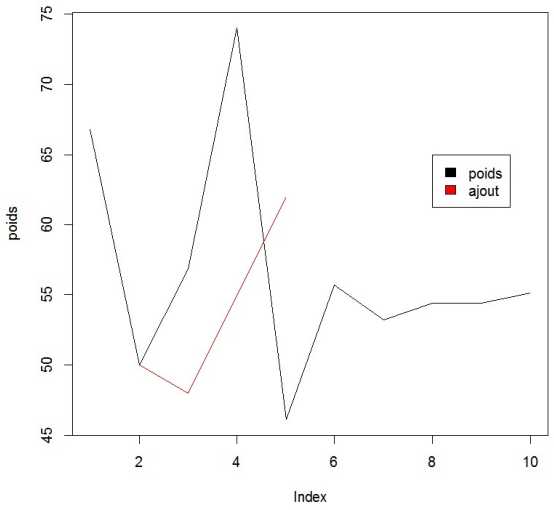
On fait :

```
plot(poids, type = "l")

lines(c(2, 3, 5), c(50, 48, 62), col = "red")

legend(8, 65, legend = c("poids", "ajout"), fill = c("black", "red"))
```

Cela renvoie :



Plus simplement, on peut faire `x = "string"` et ignorer `y`, où `string` est l'endroit où l'on veut mettre la légende : `bottomright`, `bottom`, `bottomleft`, `left`, `topleft`, `top`, `topright`, `right` et `center`.

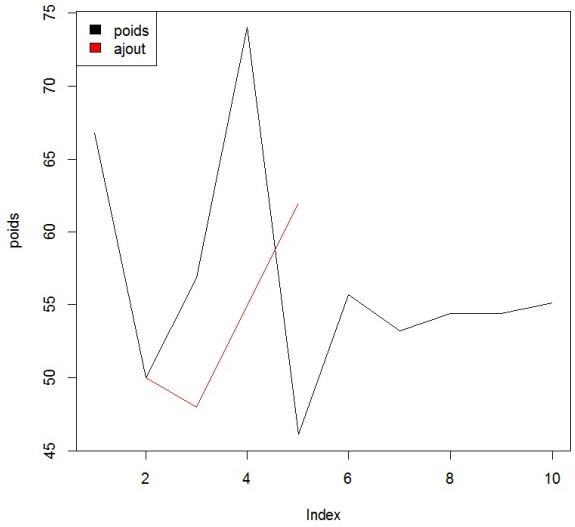
On considère les commandes :

```
plot(poids, type = "l")

lines(c(2, 3, 5), c(50, 48, 62), col = "red")

legend("topleft", legend = c("poids", "ajout"), fill = c("black", "red"))
```

Cela renvoie :



3.10 La commande `grid`

Une fois le graphique affiché, on peut mettre un quadrillage au fond en faisant `grid()`.

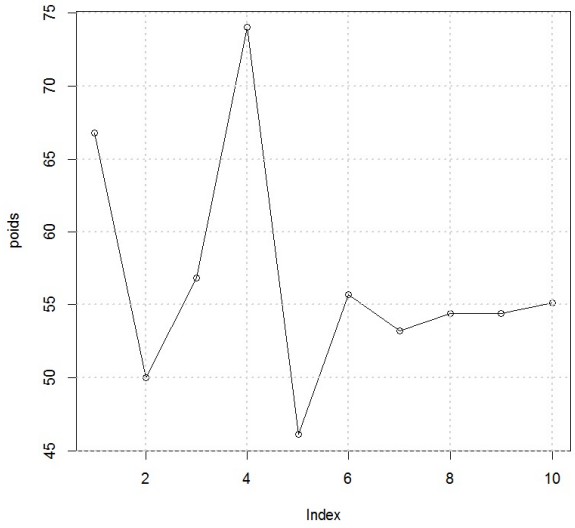
De nombreuses options graphiques existent alors (`lwd`, `lty`...).

On fait :

```
plot(poids, type = "o")

grid(lwd = 2)
```

Cela renvoie :



3.11 Les problèmes d'échelles

Quand on ajoute une ou plusieurs figures dans une figure existante, il n'est pas sûr que celles-ci rentrent dans le cadre. Pour éviter cet inconvénient, il faut préciser les limites inférieures et supérieures de l'ordonnée en utilisant un vecteur `vec` de longueur 2 qui donne cette information et écrire dans `plot`, les commandes `ylim = vec`. Si on veut agir sur l'échelle des x, on utilise `xlim = vec`.

Une figure j est généralement définie par un couple de vecteurs (xj, yj). On calcule alors

`limx = range(x1, x2,...)` et `limy = range(y1, y2,...)`.

On trace le cadre du graphique en faisant :

```
plot(x1, y1, xlim = limx, ylim = limy, type = "n")
```

On peut alors ajouter des figures avec les commandes déjà vues.

4 ggplot2 : pour commencer

Il existe des outils évolués pour faire des graphiques sophistiqués avec le logiciel R.

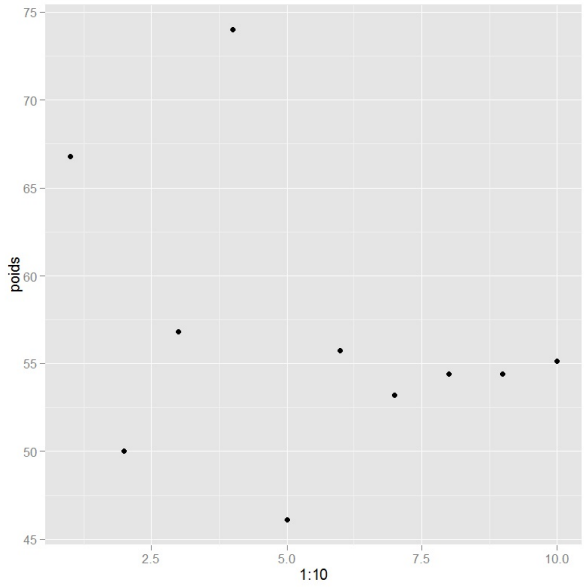
Quelques exemples "pas à pas" utilisant la librairie `ggplot2` avec le jeu de données "enquete" sont présentés ci-dessous. En premier lieu, on fait :

```
library(ggplot2)
```

On fait :

```
ggplot(enquete, aes(x = 1:10, y = poids)) + geom_point()
```

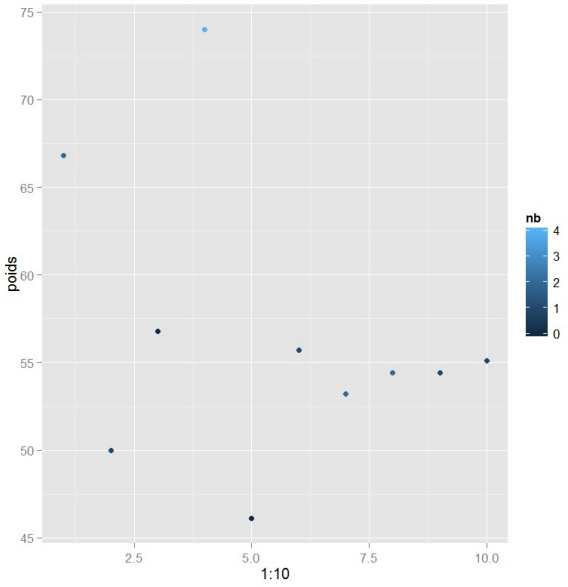
Cela renvoie :



On considère les commandes :

```
ggplot(enquete, aes(x = 1:10, y = poids)) + geom_point(aes(color = nb))
```

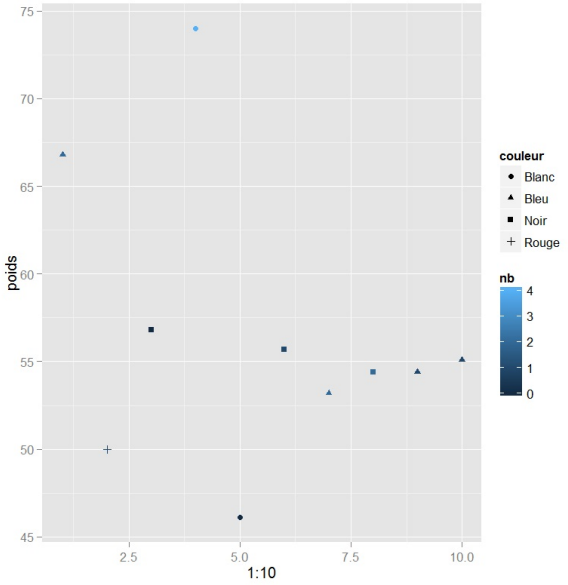
Cela renvoie :



On considère les commandes :

```
p1 = ggplot(enquete, aes(x = 1:10, y = poids)) + geom_point(aes(color = nb,
shape = couleur))
p1
```

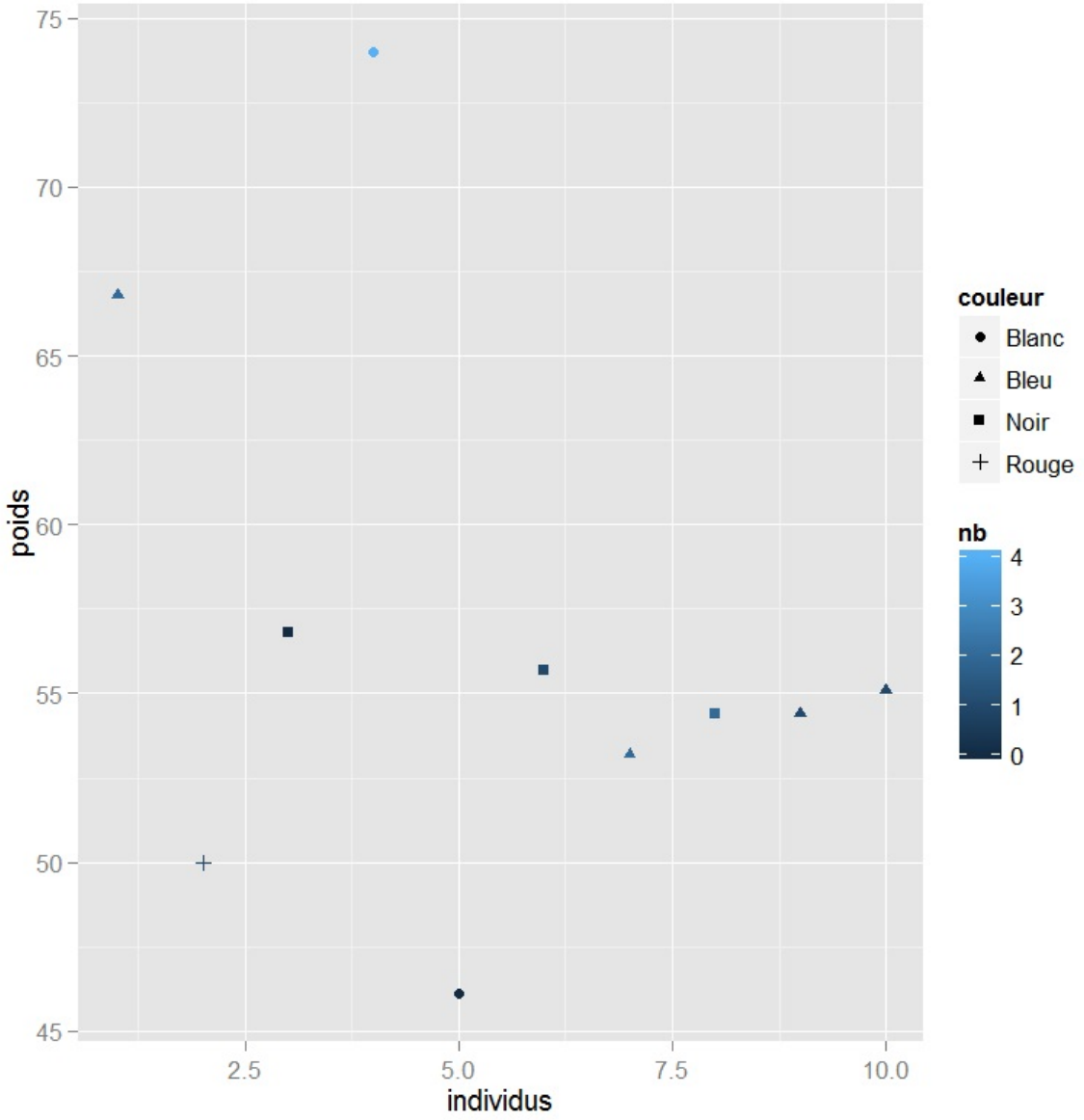
Cela renvoie :



On considère les commandes :

```
p1 + scale_x_continuous(name = "individus")
```

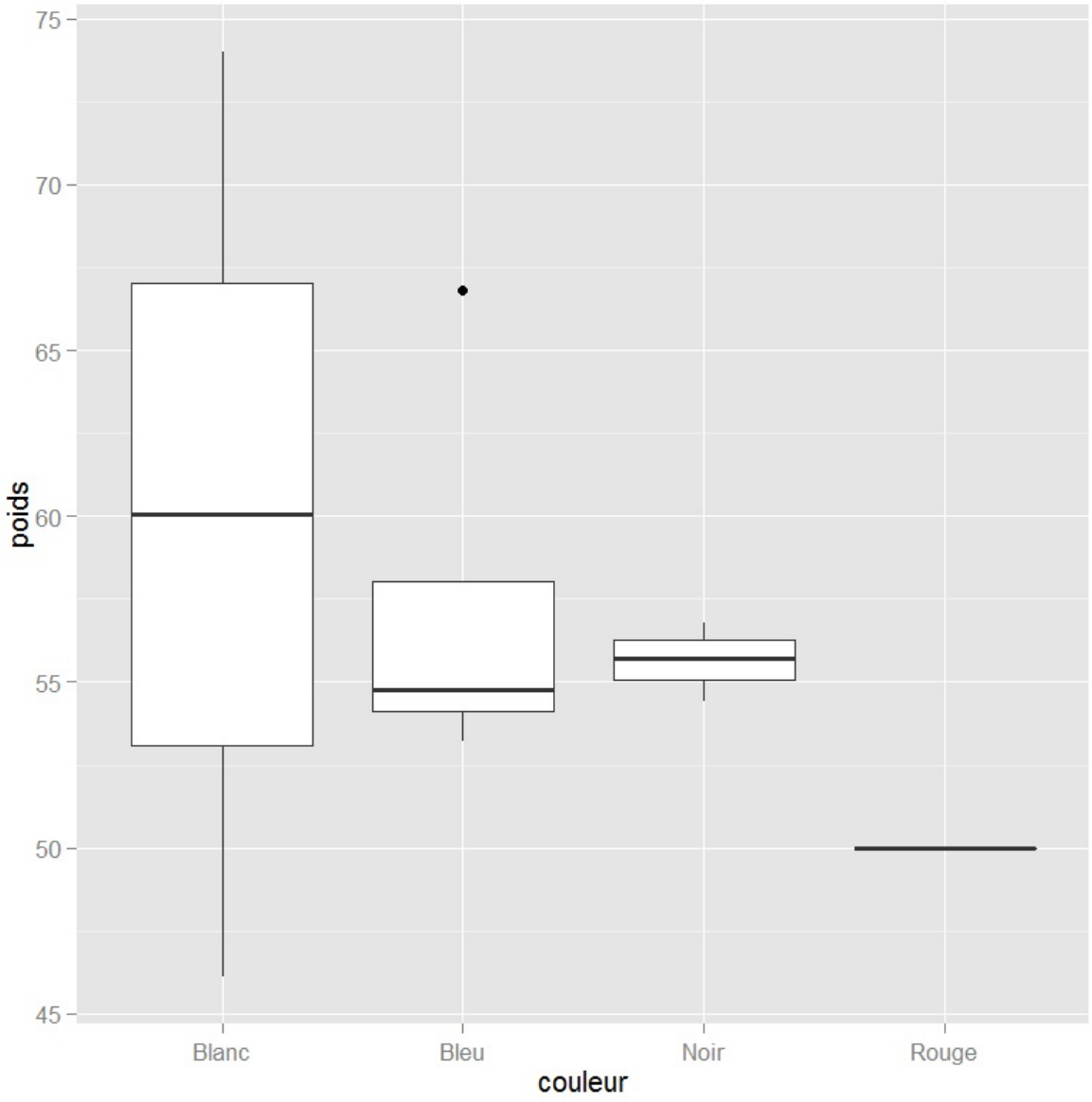
Cela renvoie :



On fait :

```
ggplot(enquete, aes(x = couleur, y = poids)) + geom_boxplot()
```

Cela renvoie :



Avec ggplot2 et du travail, d'autres graphiques sophistiqués sont possibles.

La documentation est disponible ici :

http://docs.ggplot2.org/current/

Pour finir

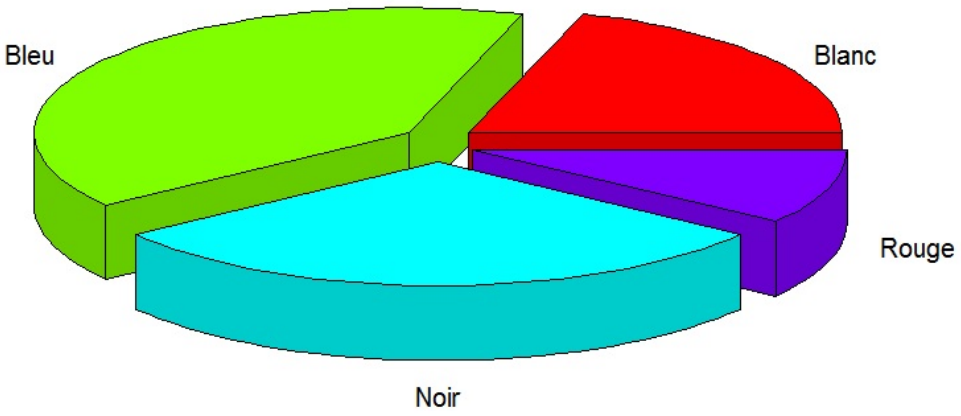
De nombreuses librairies, non présentées ici, permettent de faire des graphiques évolués. Citons entre autre `graphics`, `lattice`, `grid` et `plotrix`.

On termine avec un simple et joli camembert utilisant la librairie `plotrix` :

```
library(plotrix)

pie3D(table(couleur), explode = 0.1, labelcex = 1)
```

Cela renvoie :

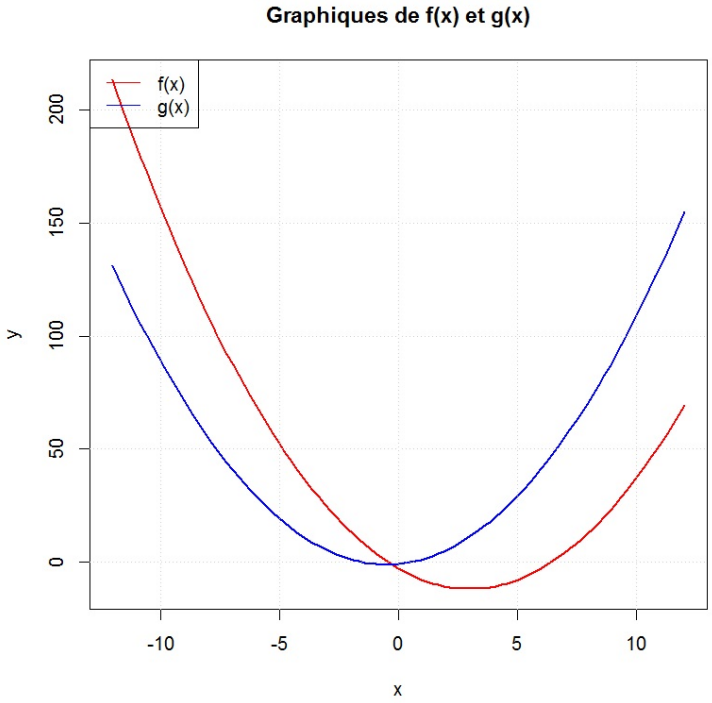


5 Exercices

Exercice 1. On considère les 2 fonctions :

$$f(x) = x^2 - 6x - 3, \qquad g(x) = x^2 + x - 1.$$

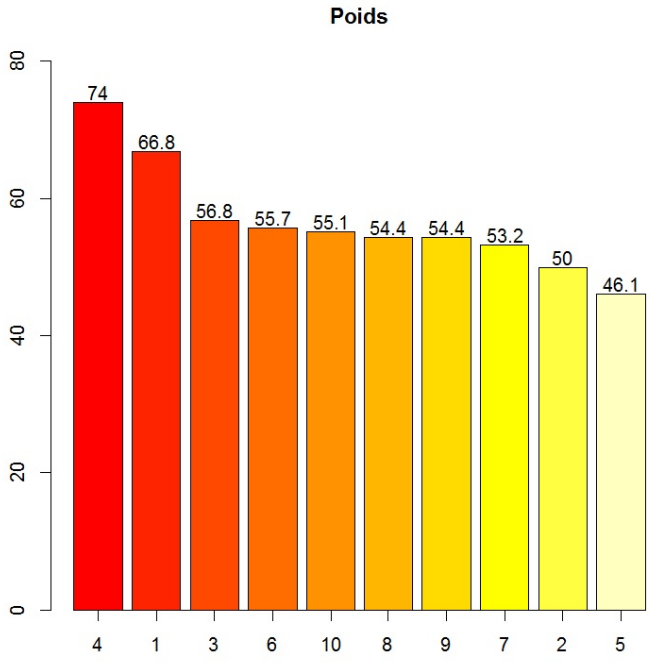
Proposer des commandes R permettant d'obtenir le graphique suivant :



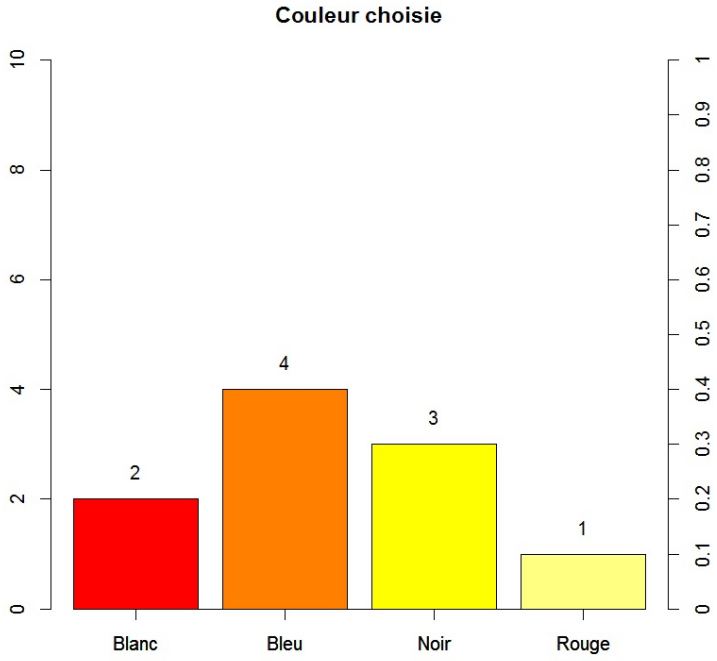
Exercice 2. On considère le jeu de données "enquete" :

```
enquete = read.table("http://www.math.unicaen.fr/~chesneau/enquete.txt",
header = T)
attach(enquete)
```

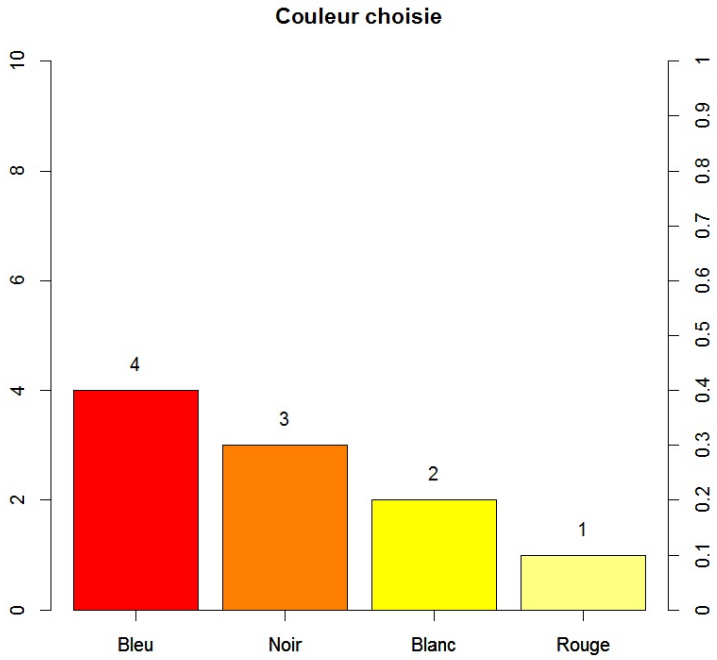
1. Proposer des commandes R permettant d'obtenir le graphique suivant :



2. Proposer des commandes R permettant d'obtenir le graphique suivant :



3. Proposer des commandes R permettant d'obtenir le graphique suivant :



Exercice 3. On considère le jeu de données "nba" :

```
nba = read.table("http://www.math.unicaen.fr/~chesneau/nba.txt",
header = T, sep = ",")
```

On dispose :

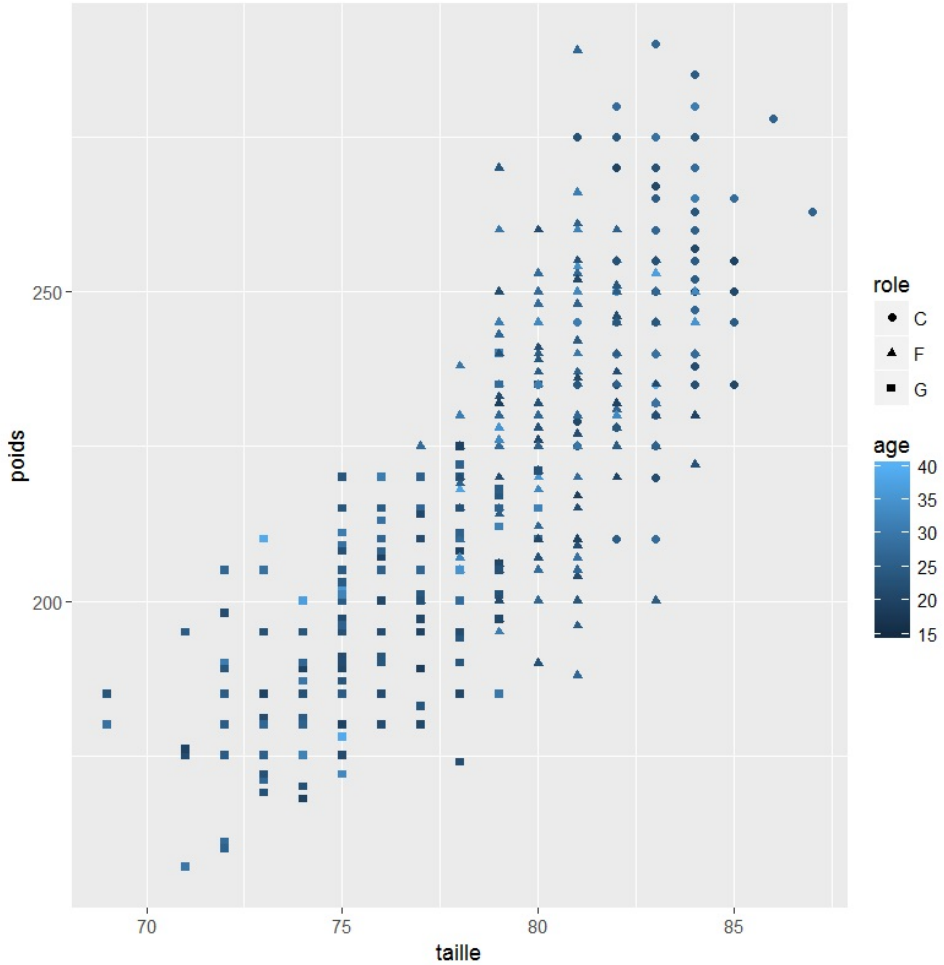
- de l'identité du joueur (variable `Joueur`),
- de leur rôle sur le terrain (variable `X2`),
- de leur taille (variable `X1`),
- de leur poids (variable `Y`),
- de leur âge (variable `X3`).

1. Préciser la nature des variables étudiées.
2. Renommer les labels des variables par : `joueur`, `role`, `taille`, `poids` et `age`.
3. Représenter la dispersion des poids des basketteurs avec la commande `stripchart`, puis sa distribution avec la commande `hist`.
4. Représenter la distribution des tailles des basketteurs avec la commande `hist`.

5. Représenter la distribution de la variable `role` avec la commande `pie`.
6. Étudier graphiquement les paramètres statistiques de `poids` en fonction de `role`.
7. Étudier graphiquement les paramètres statistiques de `taille` en fonction de `role`.
8. Étudier graphiquement les paramètres statistiques de `poids` en fonction de `age`.
9. Étudier graphiquement les paramètres statistiques de `taille` en fonction de `age`.
10. Représenter le nuage de points associés à `(taille, poids)`. Sur ce même graphique, tracer la fonction :

$$f(x) = e^{-4.59678} x^{2.28551}, \qquad x \in [66, 88].$$

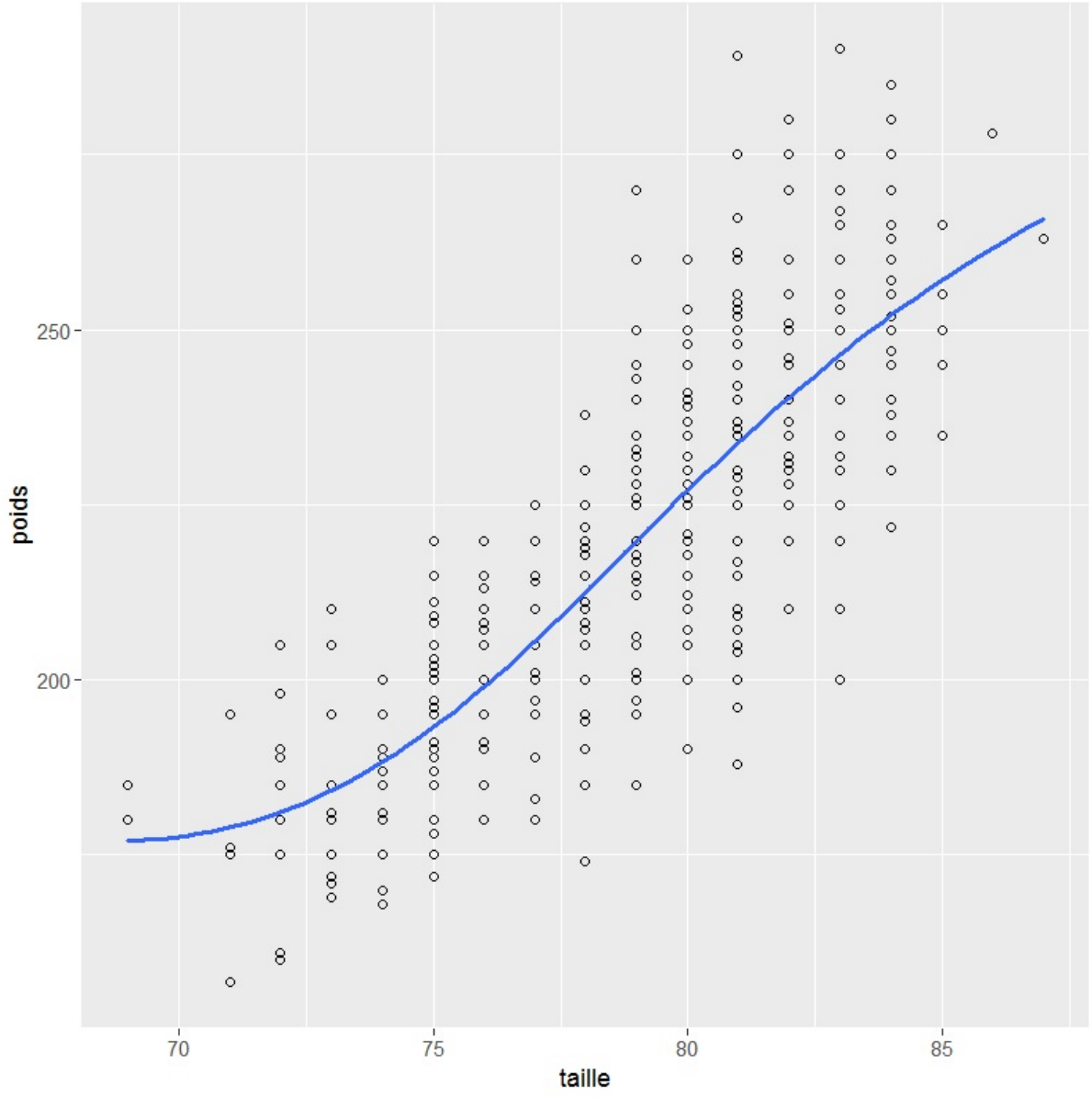
11. Proposer des commandes R utilisant la librairie `ggplot2` permettant d'obtenir le graphique suivant :



12. Commenter les commandes suivantes et le graphique obtenu :

```
ggplot(nba, aes(x = taille, y = poids)) + geom_point(shape=1)
+ geom_smooth(, se = FALSE)
```

Cela renvoie :



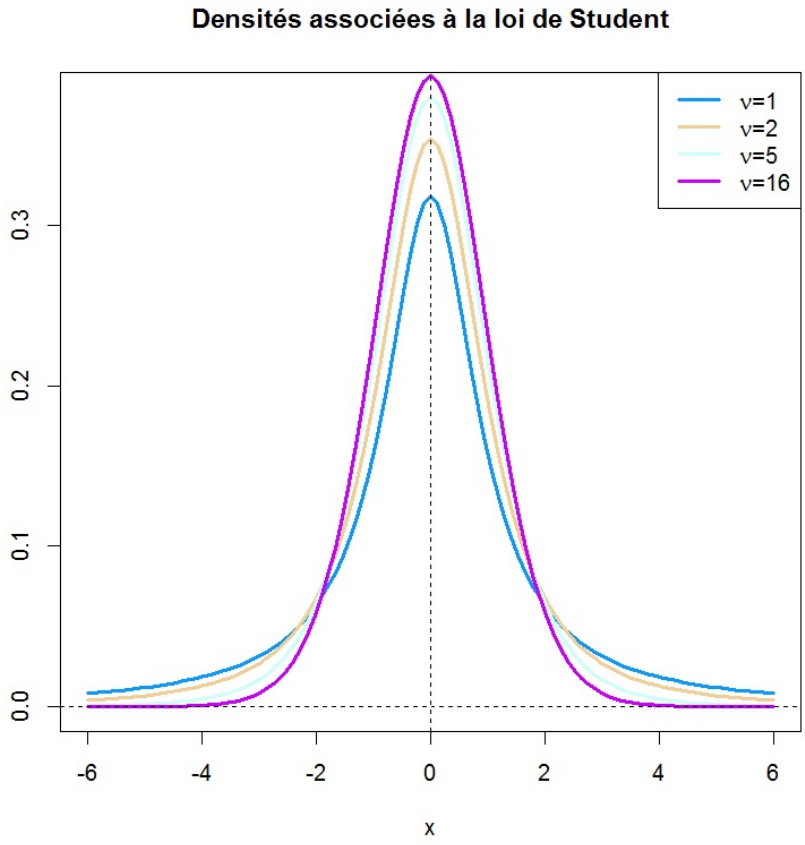
Exercice 4. Soit $\nu > 0$. On appelle "densité associée à la loi de Student $\mathcal{T}(\nu)$" la fonction :

$$f_\nu(x) = \frac{1}{\sqrt{\nu\pi}} \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \qquad x \in \mathbb{R},$$

où Γ désigne la fonction gamma d'Euler définie par :

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt, \qquad x > 0.$$

1. Créer dans R la fonction `tdens(x, nu)` représentant la fonction $f_\nu(x)$.
2. À partir de cette fonction R, proposer des commandes R permettant d'obtenir le graphique suivant :



Exercice 5. Soient

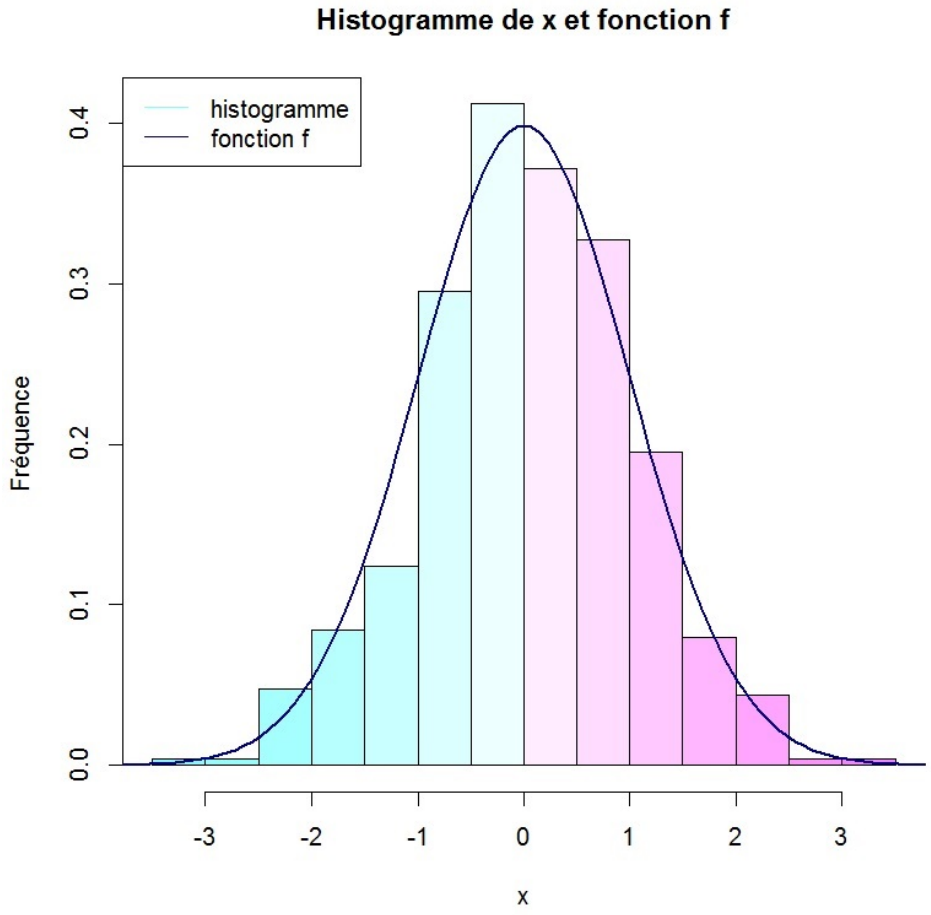
- `x` le vecteur numérique dont les éléments sont disponibles ici :

  `http://www.math.unicaen.fr/~chesneau/norm.txt`

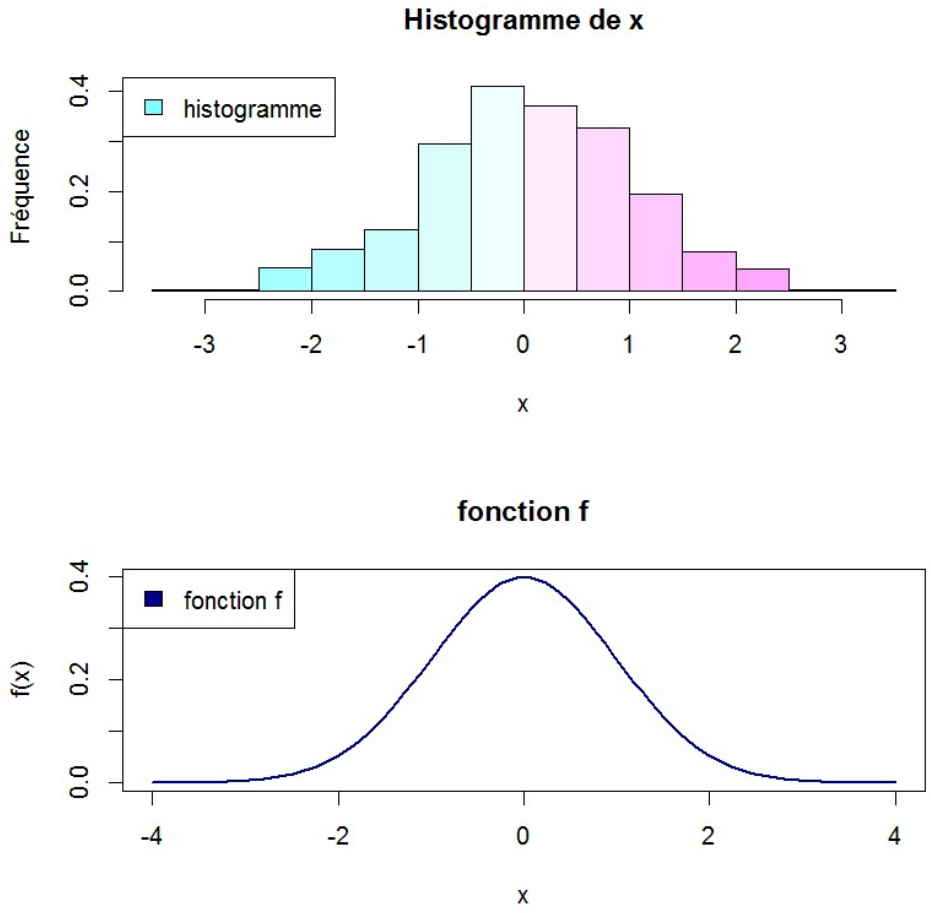
- la fonction `f` représentant $f : \mathbb{R} \to [0, \infty[$:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

1. Proposer des commandes permettant d'obtenir le graphique suivant :



2. Proposer des commandes permettant d'obtenir le graphique suivant :



Exercice 6. Reproduire et comprendre l'enjeu des commandes suivantes :

```
x = rnorm(18)
y = rnorm(18)
plot(x, y, xlab = "", ylab = "", xlim = c(-2, 2), ylim = c(-2, 2), pch = 22,
col = "red", bg = "yellow", bty = "l", tcl = 0.4, main = "Nuage de points")
grid(lwd = 1)
```

Exercice 7. Reproduire et comprendre l'enjeu des commandes suivantes :

```
plot(1, 1, xlim = c(1, 5.5), ylim = c(0, 7), type = "n", xlab = "", ylab = "")
text(1:5, rep(6, 5), labels = c(0:4), cex = 1:5, col = 1:5)
points(1:5, rep(5, 5), cex = 1:5, col = 1:5, pch = 0:4)
text((1:5) + 0.4, rep(5, 5), cex = 0.6, (0:4))
points(1:5, rep(4, 5), cex = 2, pch = (5:9))
text((1:5) + 0.4, rep(4, 5), cex = 0.6, (5:9))
points(1:5, rep(3, 5), cex = 2, pch = (10:14))
text((1:5) + 0.4, rep(3, 5), cex = 0.6, (10:14))
points(1:5, rep(2, 5), cex = 2, pch = (15:19))
text((1:5) + 0.4, rep(2, 5), cex = 0.6, (15:19))
points((1:6)*0.8 + 0.2, rep(1, 6), cex = 2, pch = (20:25))
text((1:6)*0.8 + 0.5, rep(1, 6), cex = 0.6, (20:25))
```

6 Solutions

Solution 1. On fait :

```
f = function(x) { x^2 - 6*x -3 }
g = function(x) { x^2 + x - 1 }
curve(f(x), -12, 12, col = "red", xlab = "x", ylab = "y", lwd = 2,
main = "Graphiques de f(x) et g(x)")
curve(g(x), col = "blue", lwd = 2, add = TRUE)
grid()
legend("topleft", col = c("red", "blue"), lty=1, legend = c("f(x)", "g(x)"))
```

Solution 2.

1. On fait :

   ```
   bppoids = barplot(sort(poids, decreasing = T), ylim = c(0, 80), main = "Poids",
     col = heat.colors(10))
   text(bppoids, sort(poids, decreasing=T) + 1.5, sort(poids, decreasing = T))
   axis(1, at = bppoids,labels = order(poids, decreasing = T))
   ```

2. On fait :

   ```
   tcoul = table(couleur)
   bpcoul = barplot(tcoul, ylim = c(0,10), main = "Couleur choisie",
   col = heat.colors(4))
   text(bpcoul, tcoul + 0.5, tcoul)
   axis(4, at = (0:10),labels = seq(0, 1, by = 0.1))
   axis(1, at = bpcoul,labels = names(tcoul))
   ```

3. On fait :

```
tcoul = table(couleur)
bpcoulbis = barplot(sort(tcoul, decreasing = T), ylim = c(0, 10),
main = "Couleur choisie", col = heat.colors(4))
text(bpcoulbis, sort(tcoul, decreasing = T) + 0.5, sort(tcoul, decreasing = T))
axis(4, at = (0:10), labels = seq(0, 1, by = 0.1))
axis(1, at = bpcoulbis, labels = names(tcoul)[order(tcoul, decreasing = T)])
```

Solution 3.

1. On fait :

```
str(nba)
```

Cela renvoie :

```
'data.frame':   505 obs. of  5 variables:
 $ Joueur: Factor w/ 505 levels "Aaron Brooks",..: 360 198 391 438  ...
 $ X2    : Factor w/ 3 levels "C","F","G": 3 3 3 3 3 3 3 3 3 3 ...
 $ X1    : int  69 69 71 71 71 71 72 72 72 72 ...
 $ Y     : int  180 185 175 176 195 157 180 205 175 185 ...
 $ X3    : int  29 24 22 20 25 30 25 27 28 30 ...
```

On dispose alors de la nature des variables considérées (`X2` est une variable qualitative avec 3 modalités `"C"`, `"F"` et `"G"`, `X1` est une variable qualitative à valeurs entières...).

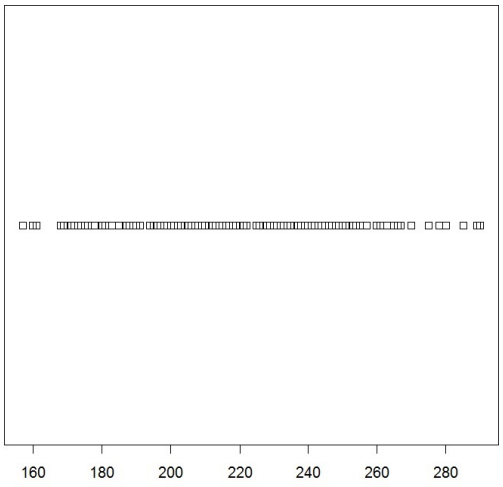
2. On fait :

```
names(nba) = c("joueurs", "role", "taille", "poids", "age")
attach(nba)
```

3. Pour représenter simplement la dispersion des poids des basketteurs, on fait :

```
stripchart(poids)
```

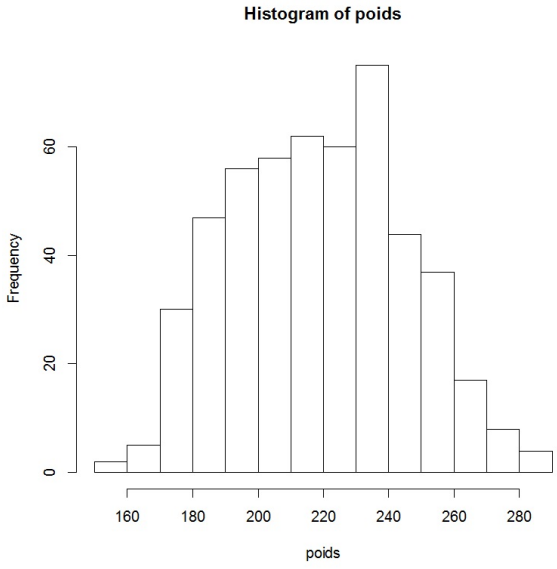
Cela renvoie :



La distribution associée est donnée par les commandes :

```
hist(poids)
```

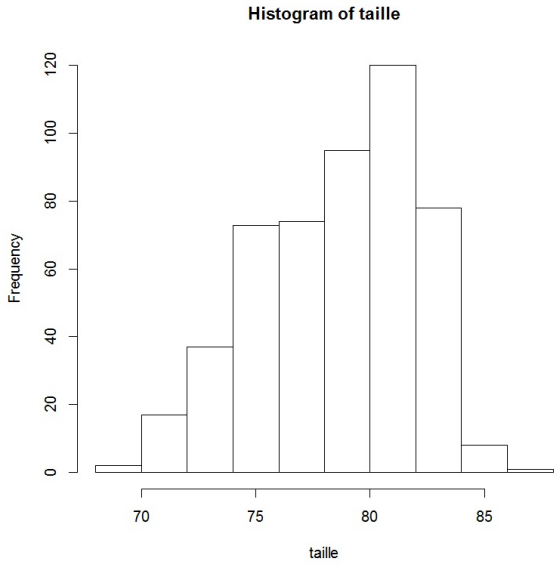
Cela renvoie :



4. Pour représenter simplement la distribution des tailles des basketteurs, on fait :

```
hist(taille)
```

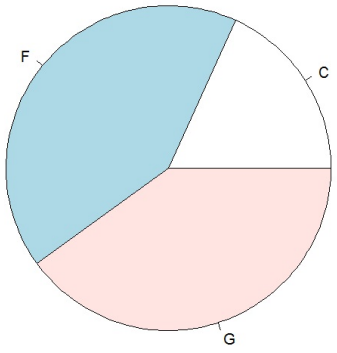
Cela renvoie :



5. La distribution de la variable `role` avec la commande `pie` peut être représentée avec les commandes :

```
pie(table(role))
```

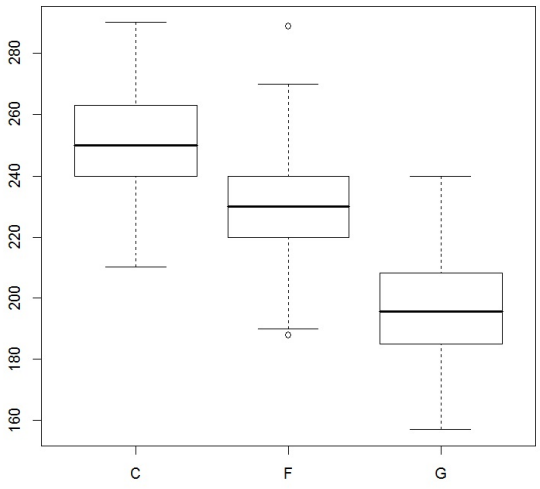
Cela renvoie :



6. Les paramètres statistiques de `poids` en fonction de `role` peuvent se visualiser avec des boîtes à moustaches :

   ```
   boxplot(poids ~ role)
   ```

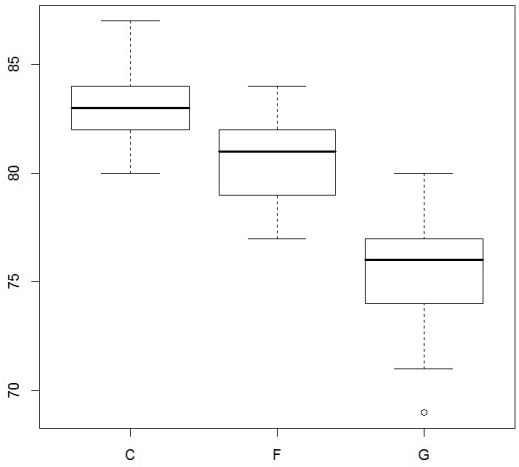
   Cela renvoie :



7. Les paramètres statistiques de `taille` en fonction de `role` peuvent se visualiser avec des boîtes à moustaches :

   ```
   boxplot(taille ~ role)
   ```

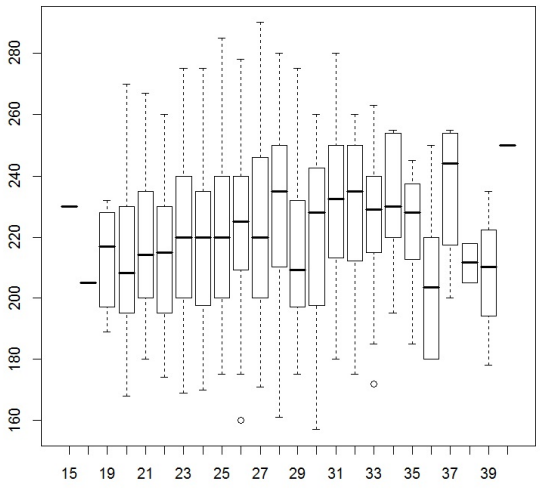
   Cela renvoie :



8. Les paramètres statistiques de `poids` en fonction de `age` peuvent se visualiser avec des boîtes à moustaches :

   ```
   boxplot(poids ~ age)
   ```

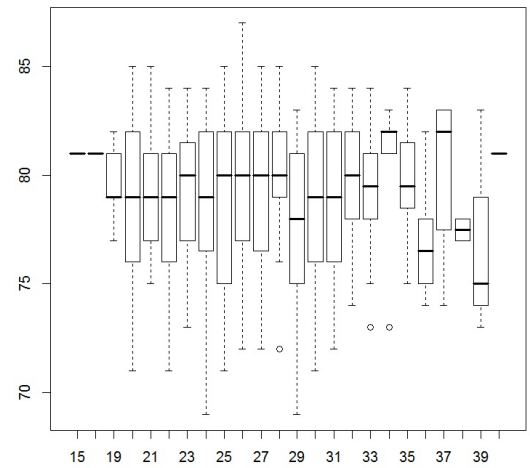
   Cela renvoie :



9. Les paramètres statistiques de `taille` en fonction de `age` peuvent se visualiser avec des boîtes à moustaches :

   ```
   boxplot(taille ~ age)
   ```

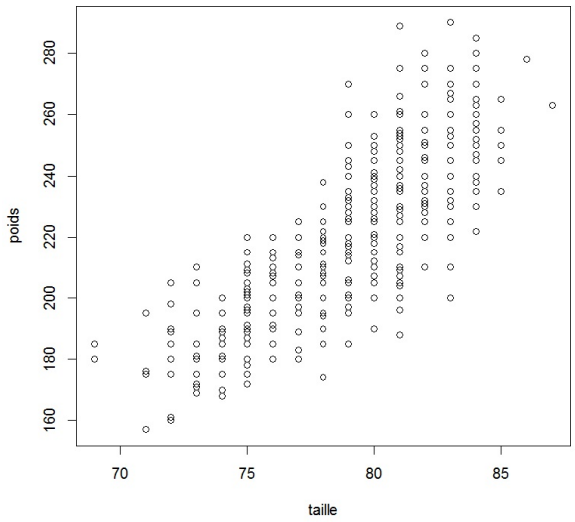
   Cela renvoie :



10. Pour construire le nuage de points associé à `(taille, poids)`, on fait :

    ```
    plot(taille, poids)
    ```

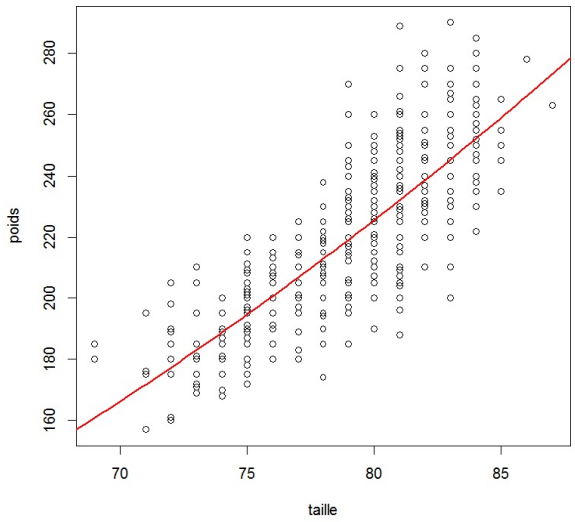
    Cela renvoie :



    On trace la fonction demandée en faisant :

    ```
    f = function(x){ exp(-4.59678) * x^(2.28551) }
    curve(f, 66, 88, col = "red", lwd = 2, add = TRUE)
    ```

    Cela renvoie :



11. On propose les commandes :

```
library(ggplot2)
ggplot(nba, aes(x = taille, y = poids)) +
geom_point(aes(color = age, shape = role))
```

12. On construit le nuage de points associé à `(taille, poids)` et on trace une courbe estimée qui ajuste au mieux ce nuage.

Solution 4.

1. On propose les commandes :

```
tdens = function(x, nu) {
(1 / sqrt(nu * pi)) * (gamma((nu + 1)/2) / gamma(nu / 2)) *
(1 + x^2 / nu)^(-(nu + 1) / 2)
}
```

2. On fait :

```
curve(tdens(x, 1), xlim = c(-6, 6), ylim = c(0, 0.38), col = "#0C99F9", lwd = 3,
ylab = "", main = "Densités associées à la loi de Student")
curve(tdens(x, 2), col = "#F0CF99", lwd = 3, add = TRUE)
curve(tdens(x, 5), col = "#CFFFF9", lwd = 3, add = TRUE)
curve(tdens(x, 16), col = "#CC00F9", lwd = 3, add = TRUE)
abline(h = 0, lty = 2)
abline(v = 0, lty = 2)
legend("topright",
legend = c(
expression(paste(nu, "=1")),
expression(paste(nu, "=2")),
expression(paste(nu, "=5")),
expression(paste(nu, "=16"))),
lwd = 3, col = c("#0C99F9", "#F0CF99", "#CFFFF9", "#CC00F9"))
```

Solution 5.

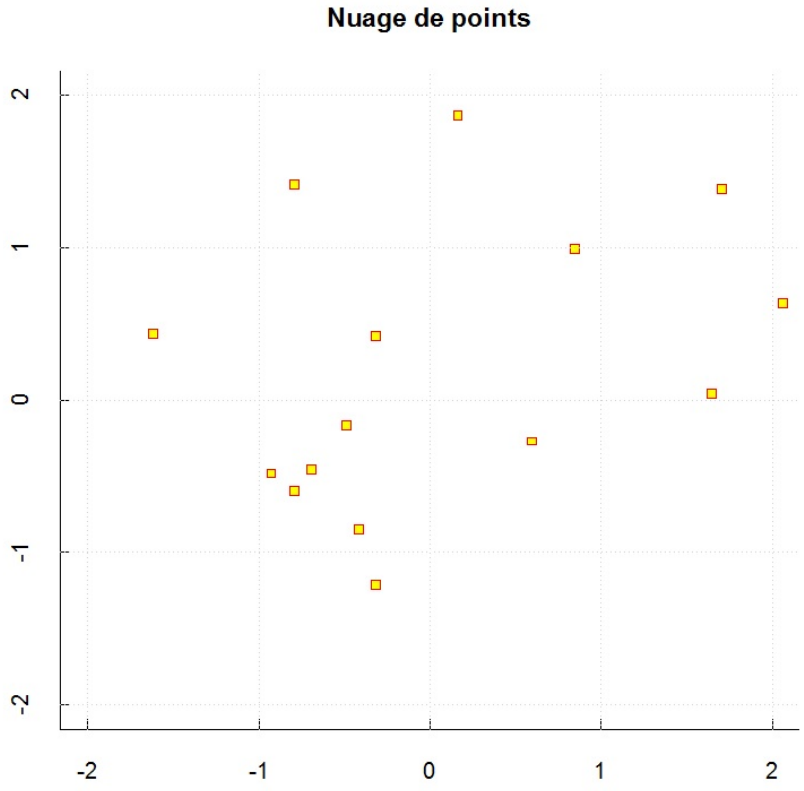
1. On propose les commandes suivantes :

```
x = scan("http://www.math.unicaen.fr/~chesneau/norm.txt", sep = ",")
hist(x, main = "Histogramme de x et fonction f", ylab = "Fréquence",
freq = FALSE, col = cm.colors(14))
f = function(x) { (1/sqrt(2 * pi)) * exp(- x^2 / 2) }
curve(f, -4, 4, col = "darkblue", lwd = 2, add = T)
legend("topleft", col = c(cm.colors(1), "darkblue"), lty = 1,
legend = c("histogramme", "fonction f"))
```

2. On propose les commandes suivantes :

```
par(mfrow = c(2,1))
hist(x, main = "Histogramme de x", ylab = "Fréquence", freq = FALSE,
col = cm.colors(14))
legend("topleft", fill = cm.colors(2),  legend = "histogramme")
curve(f, -4, 4, col = "darkblue", lwd = 2, main = "fonction f")
legend("topleft", fill = "darkblue",  legend = "fonction f")
```

Solution 6. On simule 18 points dont les coordonnées sont des réalisations d'un couple de variables aléatoires réelles (X, Y), avec X et Y suivant chacune la loi normale $\mathcal{N}(0, 1)$. Ensuite, on représente le nuage de points en utilisant des options graphiques spécifiques sur le format des points. Pour finir, on met un quadrillage au fond du graphique. Cela renvoie :



Solution 7. En utilisant plusieurs commandes graphiques usuelles, on a

- créer un graphique vide (dans lequel on a)
- afficher les chiffres 0, 1, 2, 3 et 4, avec une taille croissante et des couleurs changeantes,
- afficher les "points" codés 0, 1, 2, 3 et 4 dans l'option `pch`, toujours avec une taille croissante et des couleurs changeantes,
- afficher les "points" codés 5, 6, 7, 8 et 9 dans l'option `pch`,
- afficher les "points" codés 10, 11, 12, 13 et 14 dans l'option `pch`,
- afficher les "points" codés 15, 16, 17, 18 et 19 dans l'option `pch`,
- afficher les "points" codés 20, 21, 22, 23, 24 et 25 dans l'option `pch`.

Cela renvoie :