Fiche TD avec le logiciel R : tdr75

Les paramètres graphiques

P^r Jean R. Lobry

Exploration systématique des paramètres graphiques accessibles avec la fonction `par()`. On parle ici des fonctions graphiques de base accessibles sans nécessiter l'installation du moindre paquet additionnel. D'aucuns diraient que c'est un peu l'équivalent de l'assembleur au temps des langages de programmation structurés. Ce n'est pas entièrement faux, à vous de voir si vous voulez savoir ce qui se passe sous le capot !

1 Introduction

1.1 Index rapide

Le code R suivant sert à générer dans le source du document LaTeX les commandes donnant le numéro des pages où sont traités les paramètres graphiques. Ceci permet d'avoir un index rapide et de vérifier que rien n'a été oublié. Si un nouveau paramètre graphique est introduit lors d'une mise à jour on aura quelque chose du type « **new** ?? ».

```
allpar <- par()                 # Liste des paramètres graphiques disponibles
aindexer <- sort(names(allpar)) # Leurs noms dans l'ordre alphabétique
n <- length(aindexer)           # Leur cardinal
for(i in 1:n){
  if(i == n - 1){      # On veut un "et" avant de finir la liste
    pf <- " et "
  } else {             # Sinon une virgule
    pf <- ", "
  }
  if(i == n) pf <- "." # Et un point final à la fin

  pn <- aindexer[i]    # Le nom du paramètre
  cat("\\textbf{", pn, "}~:~\\pageref{", pn, "}", pf, sep = "")
}
```

adj : 30, **ann** : 8, **ask** : 8, **bg** : 19, **bty** : 19, **cex** : 34, **cex.axis** : 34, **cex.lab** : 34, **cex.main** : 34, **cex.sub** : 34, **cin** : 5, **col** : 20, **col.axis** : 20, **col.lab** : 20, **col.main** : 20, **col.sub** : 20, **cra** : 5, **crt** : 35, **csi** : 5, **cxy** : 5, **din** : 4, **err** : 18, **family** : 22, **fg** : 23, **fig** : 37, **fin** : 37, **font** : 11, **font.axis** : 11, **font.lab** : 11, **font.main** : 11, **font.sub** : 11, **lab** : 17, **las** : 12, **lend** : 23, **lheight** : 38, **ljoin** : 25, **lmitre** : 39, **lty** : 26, **lwd** : 39, **mai** : 40, **mar** : 40, **mex** : 42, **mfcol** : 14, **mfg** : 18, **mfrow** : 14, **mgp** : 43, **mkh** : 44, **new** : 9, **oma** : 44, **omd** : 44, **omi** : 44, **page** : 7, **pch** : 12, **pin** : 47, **plt** : 37, **ps** : 14, **pty** : 27, **smo** : 44, **srt** : 35, **tck** : 46, **tcl** : 46, **usr** : 49, **xaxp** : 48, **xaxs** : 28, **xaxt** : 28, **xlog** : 9, **xpd** : 10, **yaxp** : 48, **yaxs** : 28, **yaxt** : 28, **ylbias** : 49 et **ylog** : 9.

1.2 Le jeu de données pour les exercices

Pour rendre les exercices proposés ici plus concrets, on utilise un petit jeu de données concernant la dépense annuelle par élève ou étudiant français en 2004 (en euro). La source des données est le ministère de l'éducation nationale, elles ont été reprises de [1]. La signification des acronymes est donnée dans la table 1 page 3. Définir l'objet `men` :

```
men <- structure(list(
  Depense = c(4400L, 4590L, 7400L, 10170L, 10490L, 12300L, 13760L, 9160L, 6700L)),
  class = "data.frame",
  row.names = c("Maternelle", "Primaire", "College", "Lycee general et technologique",
                "Lycee professionnel", "STS", "CPGE", "IUT", "Universites, hors IUT"))
```

1.3 Les paramètres graphiques étudiés

Il est question ici des paramètres graphiques de base accessibles avec la fonction `par()`. Les arguments de cette fonction sont :

```
args(par)
```

```
function (..., no.readonly = FALSE)
NULL
```

Acronymes	Signification
IUT	Instituts Universitaires de Technologie
CPGE	Classes Préparatoires aux Grandes Écoles
STS	Sections de Techniciens Supérieurs

Table 1: Signification des acronymes utilisés dans l'objet `men`.

Le seul argument explicite de la fonction `par()` est donc `no.readonly`, comme il est après l'argument point-point-point (`...`), on ne peut pas l'abréger. La documentation nous dit à propos de cet argument `no.readonly` :

```
logical; if TRUE and there are no other arguments, only parameters are returned
which can be set by a subsequent par() call on the same device.
```

Par défaut `no.readonly` est `FALSE`, donc un appel à `par()` nous donnera la liste de *tous* les paramètres graphiques, tandis qu'un appel à `par(no.readonly = TRUE)` ne nous donnera que la liste des paramètres graphiques qui sont *modifiables*. Ceci explique la construction typique trouvée dans les fonctions graphiques et donnée en exemple dans la documentation de la fonction `par()` :

```
ex <- function() {
   old.par <- par(no.readonly = TRUE) # all par settings which
                                      # could be changed.
   on.exit(par(old.par))
   ## ...
   ## ... do lots of par() settings and plots
   ## ...
   invisible() #-- now,  par(old.par)  will be executed
}
```

dont le but est de restaurer les paramètres graphiques auxquels l'utilisateur était habitué avant d'appeler la fonction. De combien de paramètres graphiques disposons nous en tout ?

```
allpar <- par()
length(allpar)
```
```
[1] 72
```

Nous avons donc à notre disposition pas moins de 72 paramètres graphiques ! Quels sont leurs noms ? Dans la documentation de la fonction `par()`, ces paramètres sont listés dans l'ordre alphabétique, faisons de même :

```
sort(names(allpar))
```
```
 [1] "adj"       "ann"       "ask"       "bg"        "bty"       "cex"
 [7] "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"   "cin"       "col"
[13] "col.axis"  "col.lab"   "col.main"  "col.sub"   "cra"       "crt"
[19] "csi"       "cxy"       "din"       "err"       "family"    "fg"
[25] "fig"       "fin"       "font"      "font.axis" "font.lab"  "font.main"
[31] "font.sub"  "lab"       "las"       "lend"      "lheight"   "ljoin"
[37] "lmitre"    "lty"       "lwd"       "mai"       "mar"       "mex"
[43] "mfcol"     "mfg"       "mfrow"     "mgp"       "mkh"       "new"
[49] "oma"       "omd"       "omi"       "page"      "pch"       "pin"
[55] "plt"       "ps"        "pty"       "smo"       "srt"       "tck"
[61] "tcl"       "usr"       "xaxp"      "xaxs"      "xaxt"      "xlog"
[67] "xpd"       "yaxp"      "yaxs"      "yaxt"      "ylbias"    "ylog"
```

Sur ces 72 paramètres, quels sont ceux qui sont directement modifiables ?

```
allparm <- par(no.readonly = TRUE)
sort(names(allparm))
```

```
 [1] "adj"       "ann"       "ask"       "bg"        "bty"       "cex"
 [7] "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"   "col"       "col.axis"
[13] "col.lab"   "col.main"  "col.sub"   "crt"       "err"       "family"
[19] "fg"        "fig"       "fin"       "font"      "font.axis" "font.lab"
[25] "font.main" "font.sub"  "lab"       "las"       "lend"      "lheight"
[31] "ljoin"     "lmitre"    "lty"       "lwd"       "mai"       "mar"
[37] "mex"       "mfcol"     "mfg"       "mfrow"     "mgp"       "mkh"
[43] "new"       "oma"       "omd"       "omi"       "pch"       "pin"
[49] "plt"       "ps"        "pty"       "smo"       "srt"       "tck"
[55] "tcl"       "usr"       "xaxp"      "xaxs"      "xaxt"      "xlog"
[61] "xpd"       "yaxp"      "yaxs"      "yaxt"      "ylbias"    "ylog"
```

```
length(allparm)
```

```
[1] 66
```

Il reste donc 66 paramètres graphiques modifiables. Quels sont les noms des 6 paramètres graphiques qui ne sont pas modifiables ?

```
allparnm <- allpar[!(names(allpar) %in% names(allparm))]
names(allparnm)
```

```
[1] "cin"  "cra"  "csi"  "cxy"  "din"  "page"
```

Comme il n'y a que 6 paramètres graphiques non modifiables, essayons de voir tout de suite leur signification.

1.4 Les paramètres graphiques non modifiables

Non modifiable signifie que l'on ne peut pas modifier la valeur de ces paramètres graphiques avec la fonction `par()` :

```
par(din = c(1,1))
```

```
Message d'avis :
In par(din = c(1, 1)) : le paramètre graphique "din" ne peut être changé
```

Mais non modifiable avec la fonction `par()` ne signifie pas qu'ils ne soient pas modifiables par ailleurs.

1.4.1 `par("din")`

Prenons par exemple le cas du paramètre graphique `din`, une abréviation pour *device dimensions, (width, height), in inches*, la taille du périphérique (largeur, hauteur) en pouces[1] :

```
(pardin <- par("din"))
```

```
[1] 7 7
```

Le périphérique fait ici approximativement 7 pouces de large par 7 pouces de haut, soit 17.78 cm de large par 17.78 cm de haut. Quand on travaille dans le mode interactif, ce qui est généralement le cas sous R, le périphérique graphique peut être redimensionné à la main. Un appel à `par("din")` permet alors de connaître la taille exacte du périphérique courant. Ceci est très pratique quand on utilise la fonction `Sweave()`[2] pour intégrer automatiquement des

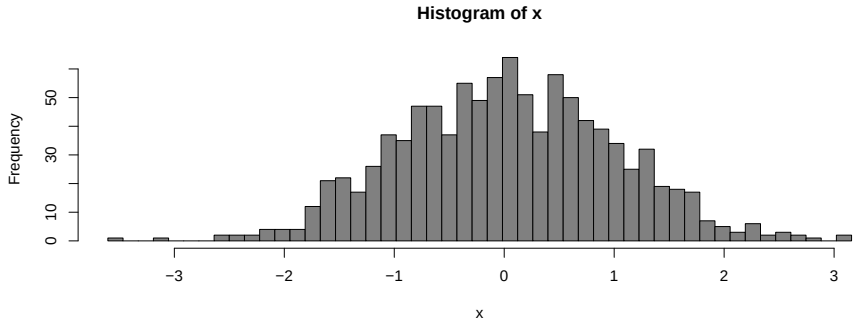
[1] En unités internationales 1 pouce vaut *exactement* 2.54 cm.

[2] Pour en savoir plus sur la fonction `Sweave()` voir la fiche *Comment rédiger un rapport avec la commande Sweave* à `http://pbil.univ-lyon1.fr/R/fichestd/tdr78.pdf`.

figures produites par R dans des documents LaTeX : on commence par essayer dans le mode interactif diverses tailles de périphériques jusqu'à ce que l'on soit satisfait du résultat. Un appel à `par("din")` donne alors directement la taille à utiliser dans le fragment de code.

```
<<demo, width = 10, height = 4>>=
x <- rnorm(1000)
hist(x, col = grey(0.5), breaks = seq(min(x), max(x), length = 50))
@
```

```
x <- rnorm(1000)
hist(x, col = grey(0.5), breaks = seq(min(x), max(x), length = 50))
```



D'une façon générale, on peut contrôler la taille des périphériques à leur ouverture. Par exemple `pdf(file = "essai.pdf", width = 10, height = 4)` ouvre un périphérique PDF de 10 pouces de large et de 4 pouces de haut. Toutes les commandes graphiques seront alors redirigées vers ce périphérique fichier jusqu'à l'appel à `dev.off()` pour fermer le périphérique.

Exercice. Faire un graphique quelconque, par exemple avec `plot(0)`, dans un fichier PDF. Ouvrir ensuite ce fichier PDF.

1.4.2 `par("cin")` et `par("cra")`

Ces deux paramètres donnent la taille des caractères, largeur et hauteur, exprimée en pouces et en pixels, respectivement. La table 2 donne les valeurs des périphériques courants. À noter, depuis la version R 2.7, un effort considérable a été fait pour que le périphérique graphique ouvert par défaut dans le mode interactif soit le plus proche possible de ce qui est obtenu avec le périphérique fichier `pdf`. Un grand merci à toute la R-core team.

1.4.3 `par("csi")` et `par("cxy")`

Ces paramètres donnent la taille des caractères par défaut dans le périphérique courant. Le paramètre `csi` donne la hauteur des caractère en pouces. Le paramètre `cxy` donne la largeur et la hauteur des caractères en coordonnées utilisateur. Ces valeurs peuvent changer pour un même périphérique, par exemple :

	nom	dev.larg	dev.haut	car.larg.pc	car.haut.pc	car.larg.pix	car.haut.pix
1	quartz	7.00	7.00	0.15	0.20	10.80	14.40
2	x11	6.99	6.99	0.15	0.20	14.40	19.23
3	pdf	7.00	7.00	0.15	0.20	10.80	14.40
4	jpeg	6.67	6.67	0.15	0.20	10.80	14.40
5	png	6.67	6.67	0.15	0.20	10.80	14.40
6	bmp	6.67	6.67	0.15	0.20	10.80	14.40
7	tiff	6.67	6.67	0.15	0.20	10.80	14.40

Table 2: Caractéristiques par défaut des périphériques courants dans R 3.3.1 (2016-06-21)

```
par(c("csi","cxy"))
```

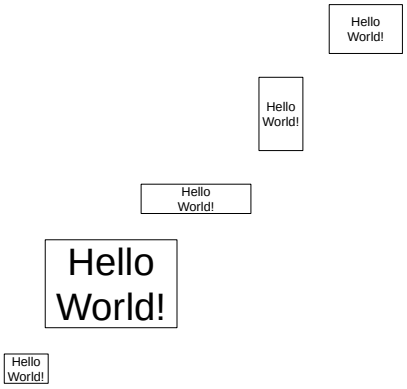
```
$csi
[1] 0.2
$cxy
[1] 0.02333733 0.11185451
```

```
par(mfrow = c(2,2))
par(c("csi","cxy"))
```

```
$csi
[1] 0.166
$cxy
[1] 0.05245265 0.27517600
```

Ces paramètres peuvent être utiles quand on a besoin de connaître la taille des chaînes de caractères écrites sur le périphérique, mais la documentation suggère plutôt d'utiliser dans ce cas les fonctions `strwidth()` et `strheight()`. Peut servir à encadrer du texte :

```
btext <- function(x, y, labels, cex = 1, epsl = 0.1, epsh = 0.1, ...){
  text(x, y, labels, cex = cex, pos = NULL, ...)
  larg <- strwidth(labels, cex = cex)
  haut <- strheight(labels, cex = cex)
  rect(x -larg/2 - epsl*larg, y - haut/2 -epsh*haut,
       x +larg/2 + epsl*larg, y + haut/2 + epsh*haut)
}
par(mar = c(0, 0, 0, 0))
plot.new()
plot.window(xlim = c(0, 10), ylim = c(0, 10))
msg <- "Hello\nWorld!"
btext(1, 1, msg)
btext(3, 3, msg, cex = 3)
btext(5, 5, msg, epsl = 1)
btext(7, 7, msg, epsh = 1)
btext(9, 9, msg, epsh = 0.5, epsl = 0.5)
```



1.4.4 par("page")

Le paramètre graphique **page** est une variable logique qui indique si le prochain graphique va se faire sur une nouvelle page ou non. Il sera faux en particulier quand il y a plusieurs graphiques par page :

```
plot(0)
par("page")
```

```
[1] TRUE
```

```
par(mfrow = c(2, 2)) # quatre graphiques par page
plot(0)
par("page")
```

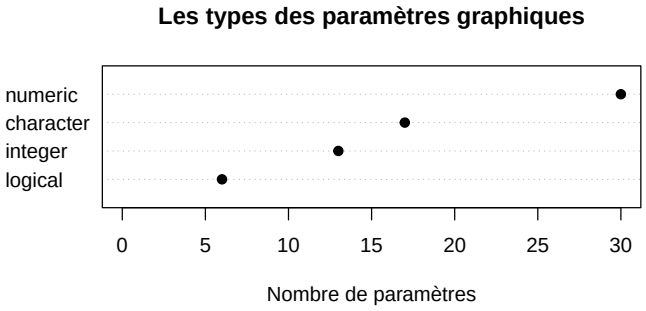
```
[1] FALSE
```

Je n'ai personnellement jamais utilisé ce paramètre, je ne saurais donner un exemple concret de son utilité, mais elle existe certainement pour ceux qui développent des fonctions graphiques sophistiquées.

2 Les paramètres graphiques modifiables

Comme il y a beaucoup de paramètres graphiques modifiables, j'ai décidé de structurer leur exploration en fonction de leur type (logique, entier, numérique, chaîne de caractères). Le graphique suivant donne les effectifs de chaque classe :

```
resume <- table(sapply(allparm, class))
x <- as.vector(resume) ; names(x) <- names(resume)
dotchart(x[order(x)], xlim = c(0, max(x)), pch = 19,
xlab = "Nombre de paramètres", main = "Les types des paramètres graphiques")
```



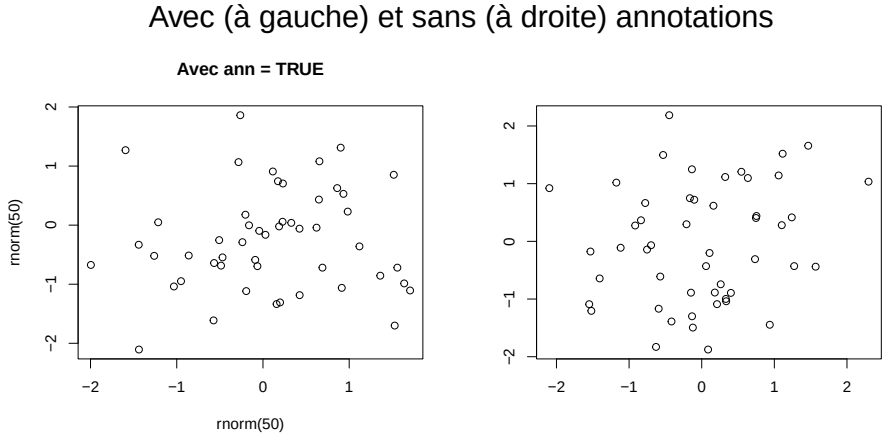
2.1 Les paramètres graphiques logiques

```
sort(names(allparm[sapply(allparm,class) == "logical"]))
```
```
[1] "ann"  "ask"  "new"  "xlog" "xpd"  "ylog"
```

2.1.1 par("ann")

PAR DÉFAUT les graphiques sont annotés, sauf si on demande explicitement qu'ils ne le soient pas. Par annotation, on entend ici le titre général et les légendes des deux axes.

```
par(mfrow = c(1, 2), oma = c(0, 0, 2, 0))
plot(rnorm(50), rnorm(50), main = "Avec ann = TRUE")
par(ann = FALSE)
plot(rnorm(50), rnorm(50), main = "Avec ann = FALSE")
mtext("Avec (à gauche) et sans (à droite) annotations", cex = 2, side = 3, outer = TRUE)
```



2.1.2 par("ask")

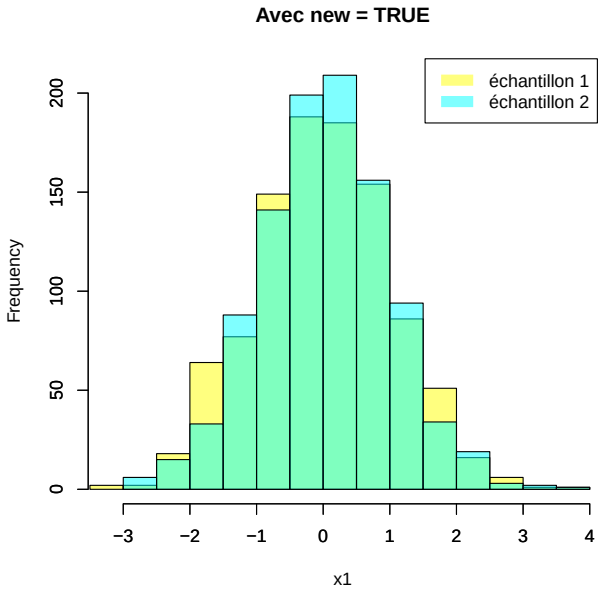
CONTRÔLE si l'on doit demander à l'utilisateur avant de tracer un nouveau graphique. Comparez le comportement dans les deux cas de figure :

```
example(points, ask = FALSE)
example(points, ask = TRUE)
```

2.1.3 par("new")

PARAMÈTRE permettant de contrôler si la fonction graphique de haut niveau utilisée ensuite doit effacer le périphérique avant de dessiner. Permet de superposer plusieurs figures, par exemple :

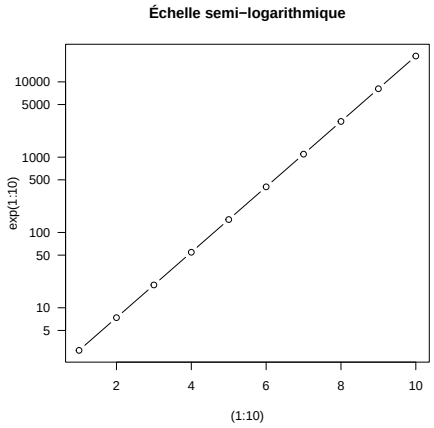
```
x1 <- rnorm(1000)
x2 <- rnorm(1000)
xlim <- range(c(x1, x2))
h1 <- hist(x1, plot = FALSE)
h2 <- hist(x2, plot = FALSE)
col1 <- rgb(1, 1, 0, 0.5)
col2 <- rgb(0, 1, 1, 0.5)
ylim <- range(c(h1$counts, h2$counts))
hist(x1, xlim = xlim, ylim = ylim, col = col1, main = "Avec new = TRUE")
par(new = TRUE, lend = "butt")
hist(x2, col = col2, xlim = xlim, ylim = ylim, ann = FALSE)
legend("topright", legend = c("échantillon 1", "échantillon 2"),
  lwd = 10, col = c(col1, col2))
```



2.1.4 par("xlog") et par("ylog")

Permettent d'avoir une échelle logarithmique en x ou y :

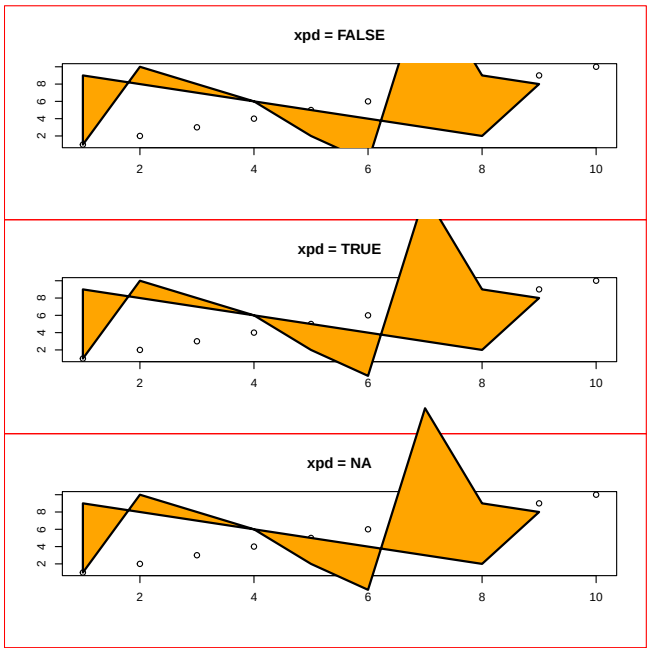
```
plot((1:10), exp(1:10), type = "b", log = "y",
     main = "Échelle semi-logarithmique", las = 1)
```



2.1.5 `par("xpd")`

Si on essaie de dessiner en dehors de la région utile, il ne se passera rien par défaut. Le paramètre `xpd` permet de déborder de la région utile ou même de la région de la figure. L'exemple suivant est adapté de la documentation de la fonction `polygon()` :

```
x <- c(1:9, 8:1)
y <- c(1, 2*(5:3), 2, -1, 20, 9, 8, 2:9)
par(mfrow = c(3, 1))
for(xpd in c(FALSE, TRUE, NA)) {
  plot(1:10, main = paste("xpd =", xpd), xlab = "", ylab = "")
  box("figure", col = "red")
  polygon(x, y, xpd = xpd, col = "orange", lty = 1, lwd = 2)
}
```



2.2 Les paramètres graphiques entiers

```
sort(names(allparm[sapply(allparm,class) == "integer"]))
```

```
 [1] "err"       "font"      "font.axis" "font.lab"  "font.main" "font.sub"
 [7] "lab"       "las"       "mfcol"     "mfg"       "mfrow"     "pch"
[13] "ps"
```

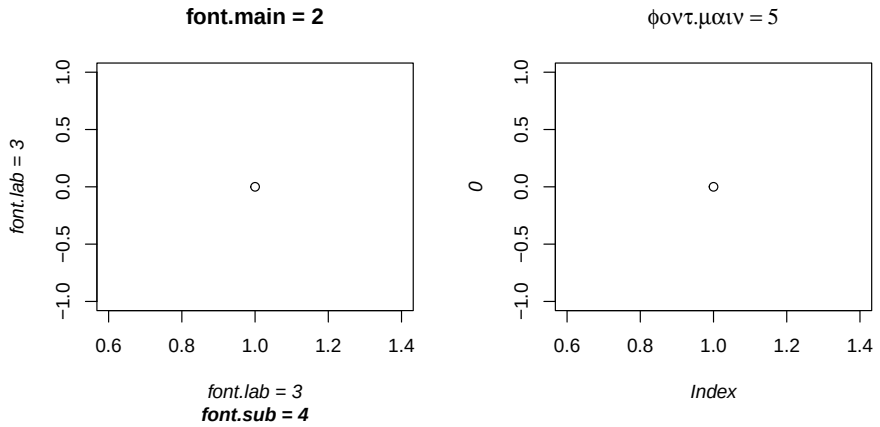
```
sort(sapply(allparm[sapply(allparm,class) == "integer"], length))
```

```
     err     font font.axis font.lab font.main font.sub      las      pch
       1        1         1        1         1        1        1        1
      ps    mfcol    mfrow      lab      mfg
       1        2        2        3        4
```

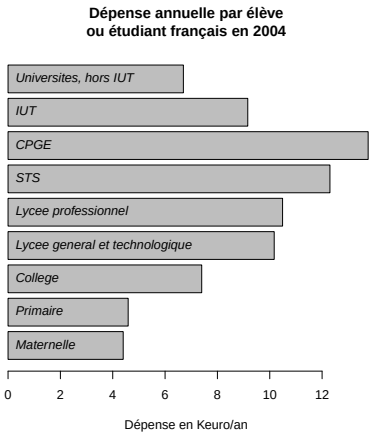
2.2.1 `par("font*")`

Les paramètres graphiques `font`, `font.axis`, `font.lab`, `font.main` et `font.sub` contrôlent la police de caractère. On a 1 pour normal, 2 pour gras, 3 pour italique et 4 pour gras et italique. Sur certains périphériques graphiques la valeur 5 correspond aux symboles. Exemple :

```
par(font.main = 2, font.lab = 3, font.sub = 4, mfrow = c(1, 2))
plot(0, main = "font.main = 2", xlab = "font.lab = 3", ylab = "font.lab = 3",
  sub = "font.sub = 4")
par(font.main = 5)
plot(0, main = "font.main = 5")
```



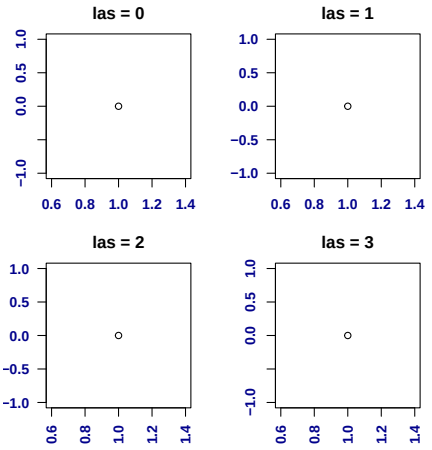
Exercice. Faire un graphique où le titre est en gras et le nom des établissements en italique :



2.2.2 `par("las")`

Contrôle le style des étiquettes des axes :

```
par(mfrow = c(2, 2), mar = c(2.5, 2.5, 2.5, 2.5))
for(i in 0:3) plot(0, las = i, main = paste("las =", i),
  font.axis = 2, col.axis = "darkblue")
```



À noter l'option `las = 1` qui permet de lire les étiquettes facilement car elles sont alors toutes horizontales.

2.2.3 `par("pch")`

Un paramètre graphique très utile pour contrôler le type de points utilisé dans les graphiques. Dans le mode interactif, voir la documentation de la fonction `points()` pour retrouver rapidement la liste des points disponibles (*cf* figure 1 page 13).

```
par(mar = c(0,0,5,0)+0.1)
Pex <- 3
ipch <- 0:25
np <- length(ipch)
k <- floor(sqrt(np))
dd <- c(-1, 1)/2
```

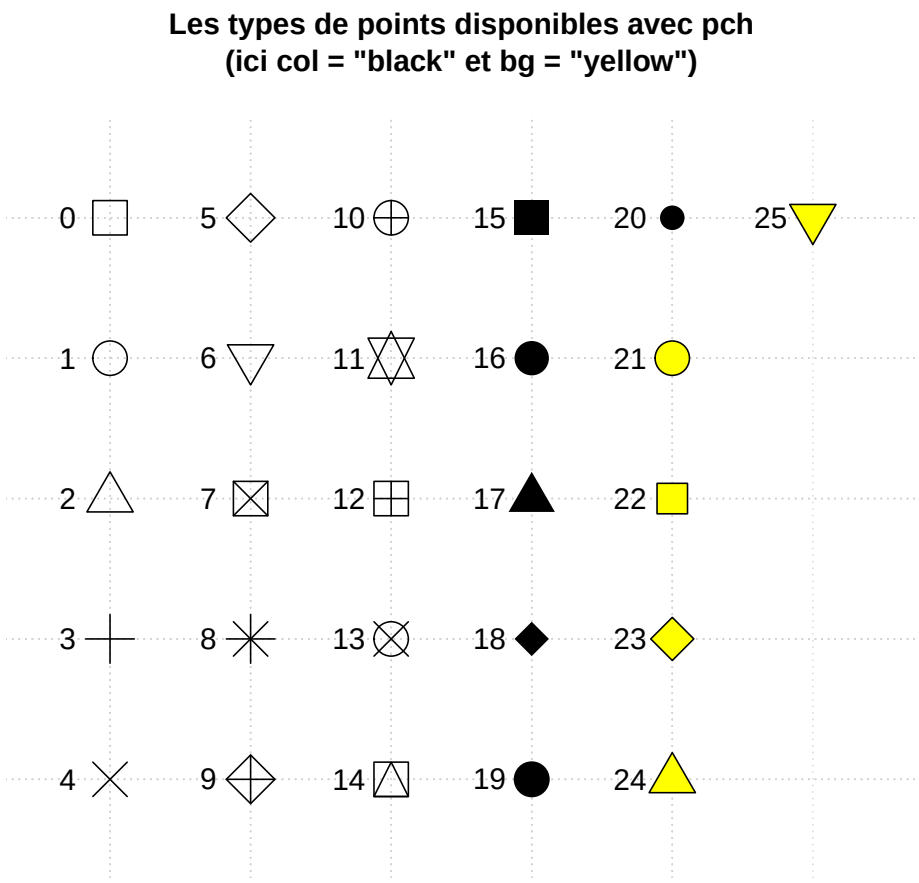
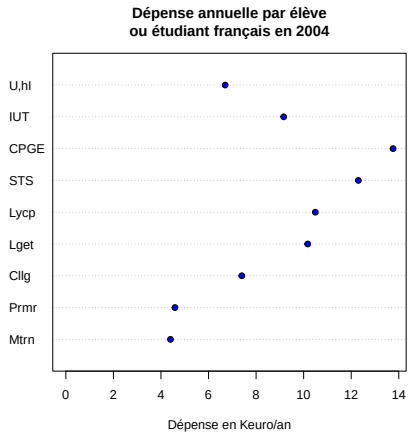


Figure 1: Le type des points pour les graphiques est contrôlé avec le paramètre `pch`. Les points de 0 à 20 sont d'une seule couleur contrôlée avec le paramètre `col`. Les points de 21 à 25 ont en plus une couleur de remplissage contrôlée avec le *paramètre* `bg` de la fonction `points()`.

```
rx <- dd + range(ix <- ipch%/%k)
ry <- dd + range(iy <- 3 + (k - 1) - ipch%%k)
pch <- as.list(ipch)
plot(rx, ry, type = "n", axes = FALSE, xlab = "",
    ylab = "",
main = "Les types de points disponibles avec pch\n(ici col = \"black\" et bg = \"yellow\")")
 abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
 for (i in 1:np) {
    pc <- pch[[i]]
    points(ix[i], iy[i], pch = pc, col = "black", bg = "yellow",
        cex = Pex)
    text(ix[i] - 0.3, iy[i], pc, col = "black", cex = 1.2)
}
```

Exercice. Modifier le code suivant pour avoir des points bleus cerclés de noir :

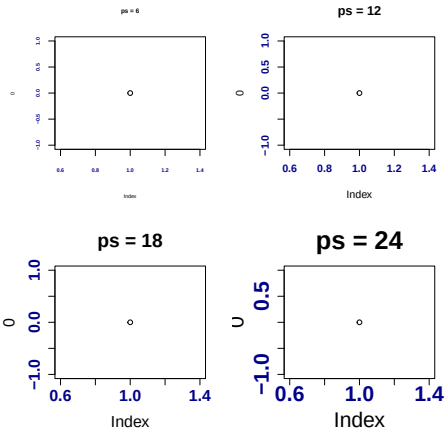
```
dotchart(men$Depense/1000, labels = abbreviate(rownames(men)),
  xlab = "Dépense en Keuro/an",
  main = "Dépense annuelle par élève\nou étudiant français en 2004",
  xlim = c(0, max(men/1000)))
```



2.2.4 `par("ps")`

Contrôle de la taille en points des caractères. On n'utilise habituellement pas ce paramètre graphique directement.

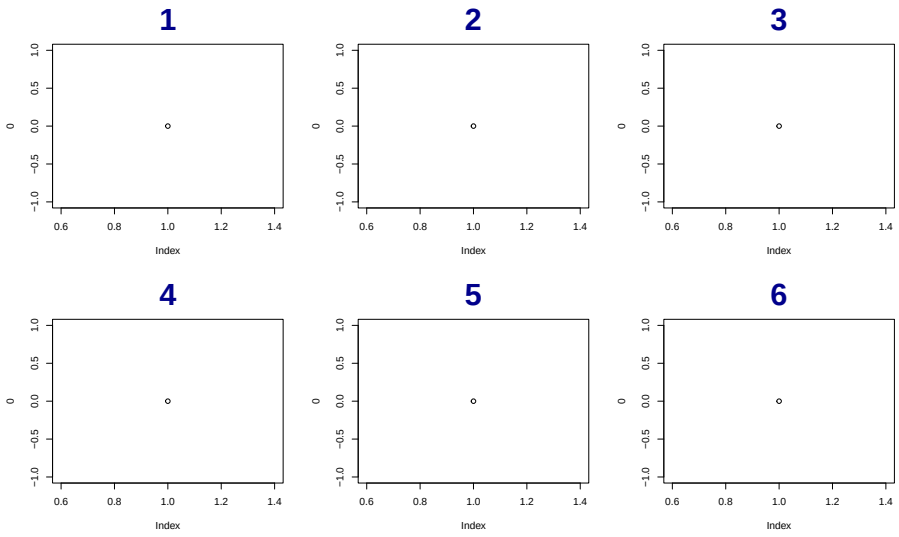
```
par(mfrow=c(2,2))
ps <- par("ps")
for(i in seq(ps/2,2*ps,le=4)){
par(ps = i)
 plot(0,main=paste("ps =",i), font.axis = 2, col.axis = "darkblue")
}
```



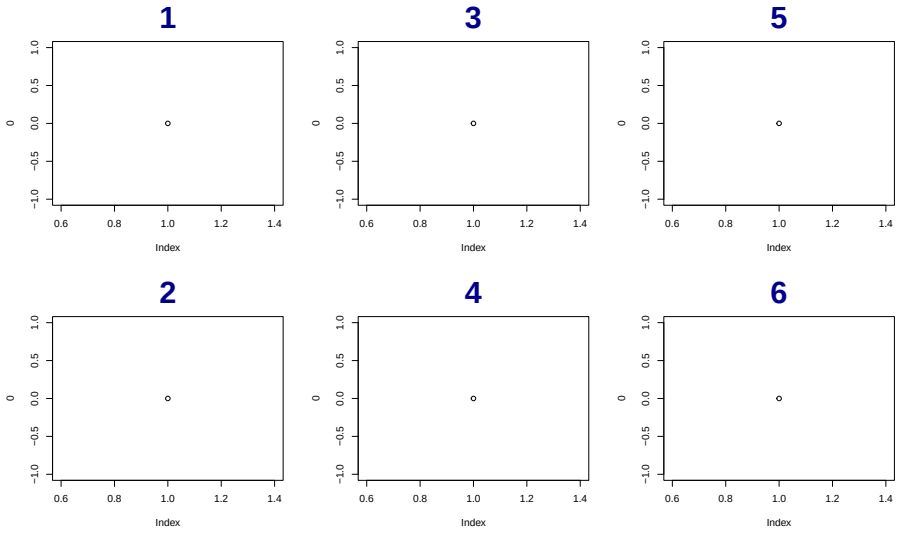
2.2.5 `par("mfrow")` et `par("mfcol")`

Contrôle du découpage de la fenêtre avec un remplissage en ligne avec `mfrow` et un remplissage en colonne avec `mfcol` :

```
par(mfrow = c(2,3))
for(i in 1:6) plot(0, main = i,cex.main=3,col.main="darkblue")
```




```
par(mfcol = c(2,3))
for(i in 1:6) plot(0, main = i,cex.main=3,col.main="darkblue")
```

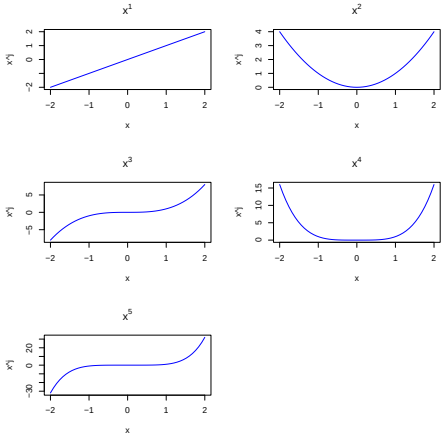


On peut utiliser la fonction utilitaire `n2mfrow()` :

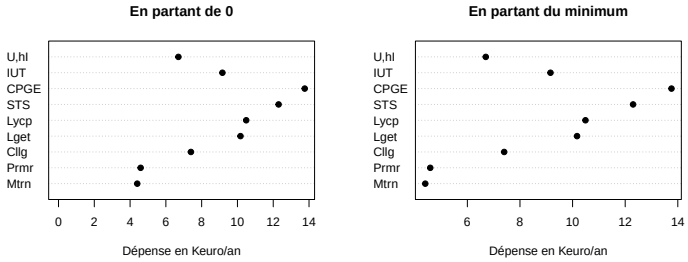
```
example(n2mfrow, ask = FALSE)
```

```
n2mfrw require(graphics)
n2mfrw n2mfrow(8) # 3 x 3
[1] 3 3

n2mfrw n <- 5 ; x <- seq(-2, 2, length.out = 51)

n2mfrw ## suppose now that 'n' is not known {inside function}
n2mfrw op <- par(mfrow = n2mfrow(n))

n2mfrw for (j in 1:n)
n2mfrw     plot(x, x^j, main = substitute(x^ exp, list(exp = j)), type = "l",
```

```
n2mfrw    col = "blue")

n2mfrw sapply(1:14, n2mfrow)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    2    3    2    3    3    3    3    3     4     4     4     4     4
[2,]    1    1    1    2    2    2    3    3    3     3     3     3     4     4

n2mfrw sapply(1:14, n2mfrow, asp=16/9)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    2    2    2    2    2    2    3    3     3     3     3     3     3
[2,]    1    1    2    2    3    3    4    3    3     4     4     4     5     5
```

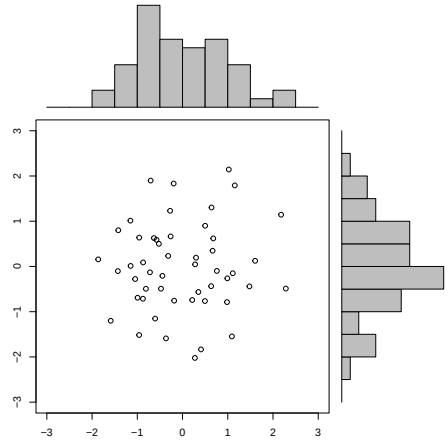


Exercice. Juxtaposer deux graphiques pour mettre en évidence ce qui se passe selon que l'on choisisse de commencer l'axe à zéro ou bien au minimum des valeurs observées :



NOTEZ qu'avec `mfrow` et `mfcol` les sous-graphiques ont tous la même hauteur et la même largeur. Il est possible de faire des choses plus sophistiquées avec la fonction `layout()`. L'exemple suivant est tiré de la documentation de cette dernière fonction :

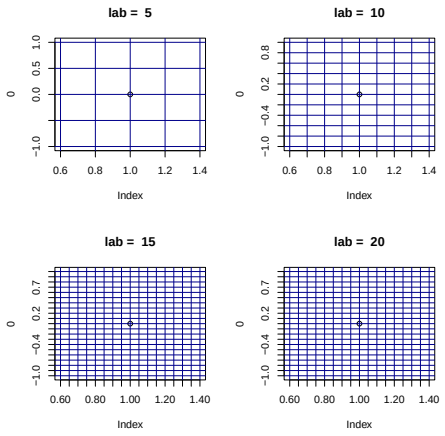
```
x <- pmin(3, pmax(-3, stats::rnorm(50)))
y <- pmin(3, pmax(-3, stats::rnorm(50)))
xhist <- hist(x, breaks=seq(-3,3,0.5), plot=FALSE)
yhist <- hist(y, breaks=seq(-3,3,0.5), plot=FALSE)
top <- max(c(xhist$counts, yhist$counts))
xrange <- c(-3,3)
yrange <- c(-3,3)
nf <- layout(matrix(c(2,0,1,3),2,2,byrow=TRUE), c(3,1), c(1,3), TRUE)
par(mar=c(3,3,1,1))
plot(x, y, xlim=xrange, ylim=yrange, xlab="", ylab="")
par(mar=c(0,3,1,1))
barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space=0)
par(mar=c(3,0,1,1))
barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space=0, horiz=TRUE)
```



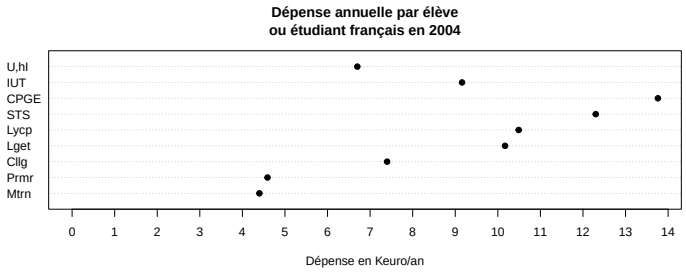
2.2.6 `par("lab")`

Un vecteur numérique de la forme `c(x, y, len)` qui modifie la façon dont les axes sont annotés. Les valeurs de `x` et `y` donnent le nombre approximatif de graduations et `len` n'est pas implémenté (R 4.1.1). La valeur par défaut est `c(5, 5, 7)`.

```
par(mfrow = c(2, 2))
for(i in seq(5, 20, by = 5)){
par(lab = c(i, i, 7))
 plot(0, main = paste("lab = ", i))
 grid(col = "darkblue", lty = 1)
}
```



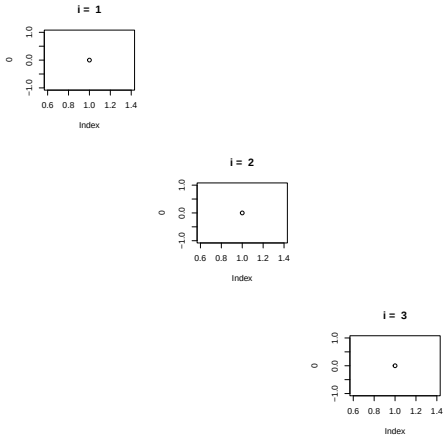
Exercice. Augmenter le nombre de graduations de façon à obtenir le graphique suivant :



2.2.7 `par("mfg")`

Après un découpage en plusieurs figures avec `mfrow` ou `mfcol`, ce paramètre permet de choisir la sous-figure dans laquelle on veut dessiner.

```
plot.new()
par(mfrow=c(3, 3))
for(i in 1:3){
  par(mfg = c(i, i))
  plot(0,main=paste("i = ",i))
}
```



2.2.8 `par("err")`

Le paramètre `err` n'est pas implémenté (4.1.1).

2.3 Les paramètres graphiques de type chaîne de caractères

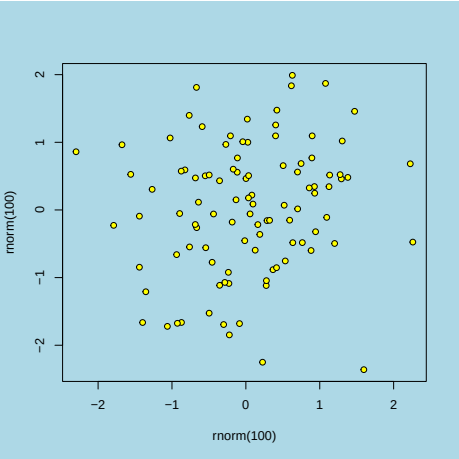
```
sort(names(allparm[sapply(allparm,class) == "character"]))
```

```
 [1] "bg"       "bty"      "col"      "col.axis" "col.lab"  "col.main" "col.sub"
 [8] "family"   "fg"       "lend"     "ljoin"    "lty"      "pty"      "xaxs"
[15] "xaxt"     "yaxs"     "yaxt"
```

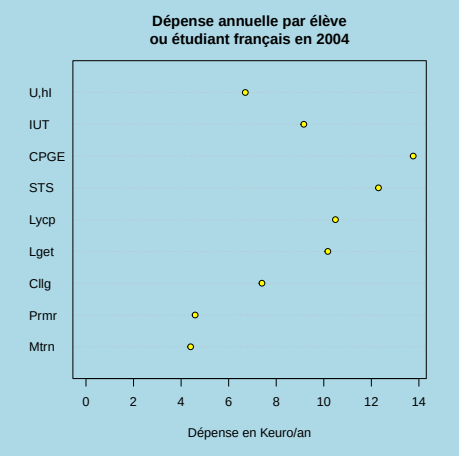
2.3.1 par("bg")

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")
```



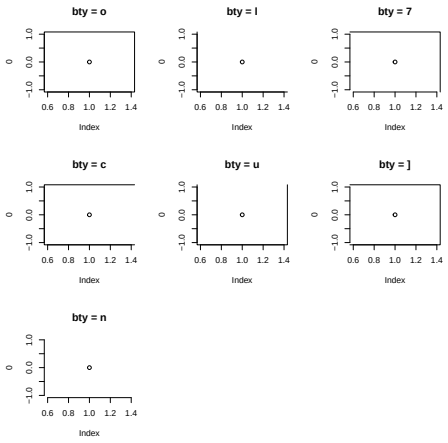
Exercice. Faire la représentation graphique suivante :



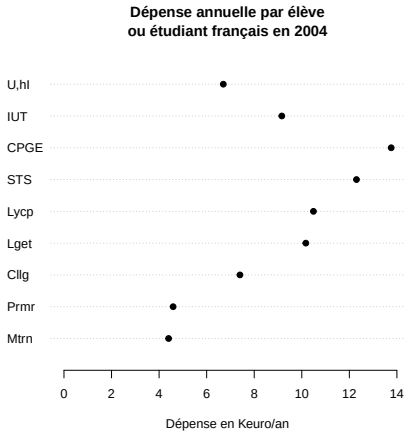
2.3.2 par("bty")

Contrôle le style du cadre à tracer autour du graphique :

```
vals <-  c("o", "l", "7", "c", "u", "]","n")
par(mfrow = n2mfrow(length(vals)))
for(bty in vals) plot(0, main = paste("bty =",bty), bty = bty)
```



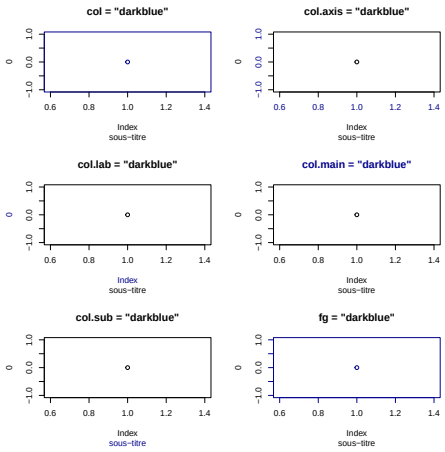
Exercice. Faire la représentation graphique suivante :



2.3.3 `par("col*")`

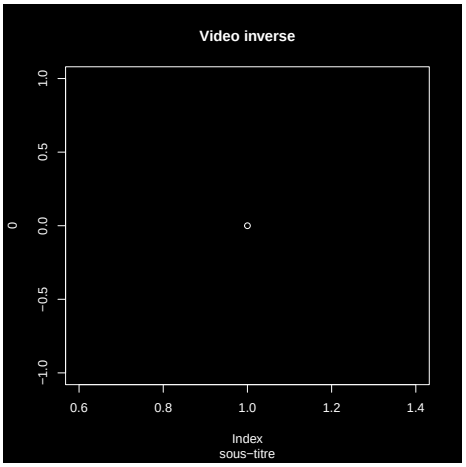
Ces paramètres contrôlent la couleur des éléments du graphique :

```
par(mfrow = n2mfrow(6))
opar <- par(no.readonly = TRUE)
par(col = "darkblue")
plot(0, main = "col = \"darkblue\"", sub = "sous-titre")
par(col = "black", col.axis = "darkblue")
plot(0, main = "col.axis = \"darkblue\"", sub = "sous-titre")
par(col.axis = "black", col.lab = "darkblue")
plot(0, main = "col.lab = \"darkblue\"", sub = "sous-titre")
par(col.lab = "black", col.main = "darkblue")
plot(0, main = "col.main = \"darkblue\"", sub = "sous-titre")
par(col.main = "black", col.sub = "darkblue")
plot(0, main = "col.sub = \"darkblue\"", sub = "sous-titre")
par(col.sub = "black", fg = "darkblue")
plot(0, main = "fg = \"darkblue\"", sub = "sous-titre")
```

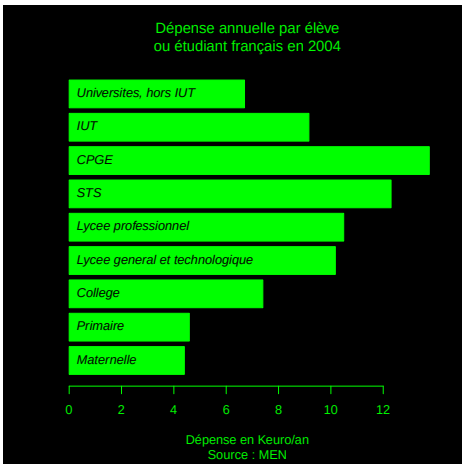


Notez que les paramètre `col*` ne vous permettent pas de contrôler la couleur des graduations des axes, il faut utiliser `fg` pour cela. Pour faire un graphique en inverse video :

```
par(bg = "black", fg = "white", col.axis = "white", col.lab = "white",
col.main = "white", col.sub="white")
plot(0, main = "Video inverse", sub = "sous-titre")
```



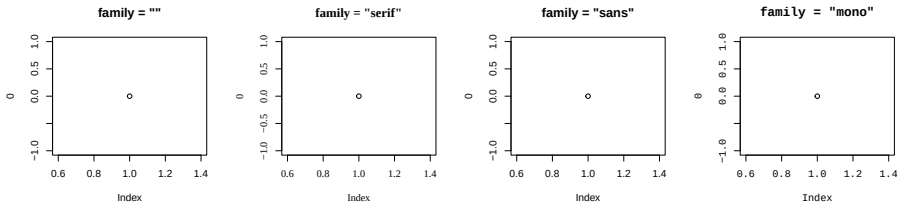
Exercice. Faire la représentation graphique suivante :



2.3.4 `par("family")`

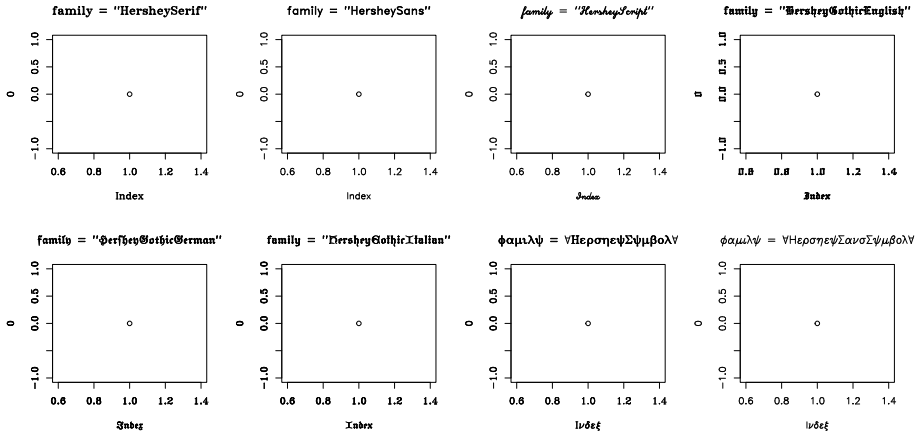
Contrôle la police de caractère utilisée, mais cela ne marche pas sur tous les périphériques. On utilise un périphérique PDF ici.

```
pdf("parfamily.pdf", width=10, height=2.5)
vals <- c("", "serif", "sans", "mono")
par(mfrow = c(1,4))
for(i in vals){
par(family = i)
plot(0, main = paste("family = \"",i,"\"", sep = ""))
}
dev.off()
```



On a aussi droit à toute la famille des polices `Hershey` :

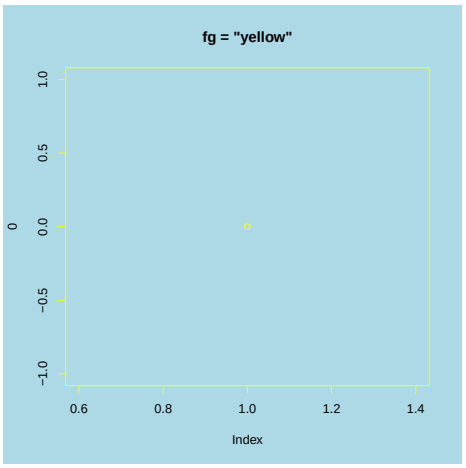
```
pdf("parHershey.pdf", width=10, height=5)
vals <- c("HersheySerif", "HersheySans", "HersheyScript", "HersheyGothicEnglish",
"HersheyGothicGerman", "HersheyGothicItalian","HersheySymbol", "HersheySansSymbol")
par(mfrow = c(2,4))
for(i in vals){
par(family = i)
plot(0,main = paste("family = \"",i,"\"", sep = ""))
}
dev.off()
```



2.3.5 `par("fg")`

Contrôle de la couleur pour dessiner :

```
par(fg = "yellow", bg = "lightblue")
plot(0, main = "fg = \"yellow\"")
```



Voir `par("col*")` pour contrôler le reste.

2.3.6 `par("lend")`

QUAND on trace des lignes un peu épaisses, il devient important de contrôler le style utilisé pour l'extrémité des lignes (`lend` signifie *line end*, c'est-à-dire la fin des lignes). R possède trois styles différents :

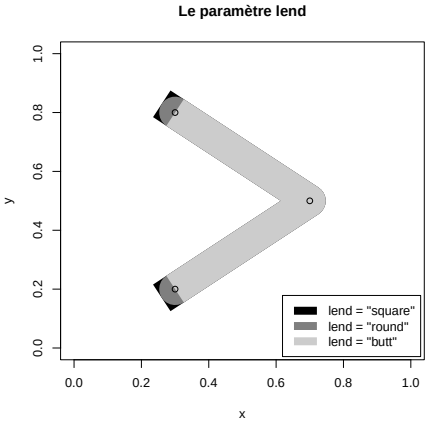
1° `square` : extrémité plate étendue au delà de la fin de la ligne ;

2° `round` : extrémité arrondie de la ligne ;

3° `butt` : extrémité plate à la fin exacte de la ligne.

La différence entre les trois styles de fin de ligne est bien visualisée par le graphique produit par le code[3] suivant :

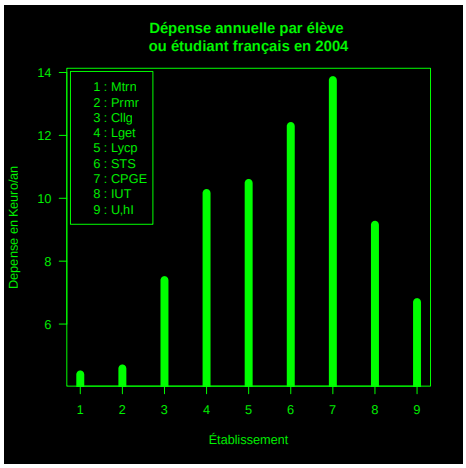
```
x <- c(.3, .7, .3)
y <- c(.2, .5, .8)
par(lend = "square")
plot(x, y, type = "l", lwd = 40, col = "black", xlim = c(0,1), ylim = c(0,1),
main = "Le paramètre lend")
par(lend = "round")
lines(x, y, lwd = 40, col = "grey50")
par(lend = "butt")
lines(x, y, lwd = 40, col = "grey80")
points(x, y)
legend("bottomright", inset = 0.01, legend = c("lend = \"square\"",
"lend = \"round\"","lend = \"butt\""),
lty = 1,
lwd = 10, col = c("black","grey50","grey80"))
```



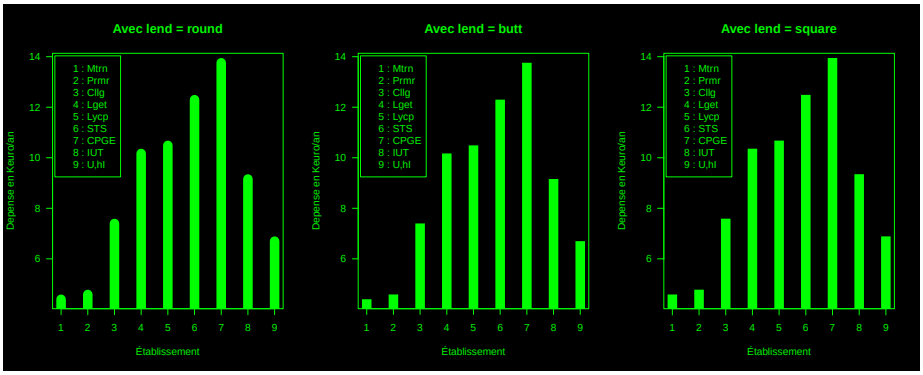
Exercice. En partant du code donnant la représentation graphique suivante :

```
par(bg = "black", fg = "green", col.axis = "green", col.lab = "green",
col.main = "green", col.sub = "green", lab = c(10,5,7))
plot(men$Depense/1000, ylab = "Depense en Keuro/an",
main = "Dépense annuelle par élève \nou étudiant français en 2004", col = "green",
type = "h", las = 1, lwd = 10, xlab = "Établissement")
legend("topleft", inset = 0.01, legend = paste(1:nrow(men), abbreviate(rownames(men)), sep = " : "))
```

[3] Adapté de la figure 3.7 du livre de Paul Murrell [2] dont le code est disponible en ligne à `http://www.stat.auckland.ac.nz/~paul/RGraphics/custombase-linesimple.R`



Faire trois graphiques juxtaposés pour mettre en évidence l'effet du paramètre `lend` :



2.3.7 `par("ljoin")`

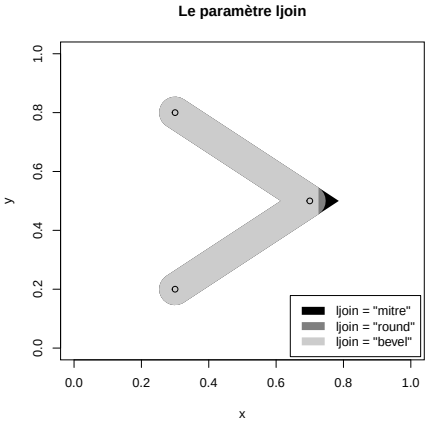
Ce paramètre ne peut être modifié que *via* un appel à la fonction `par()` et non en tant qu'argument des fonctions graphiques. Quand on trace des lignes un peu épaisses, il devient important de contrôler le style utilisé pour la jointure des lignes (`ljoin` signifie *line join*). Ce paramètre influence également l'aspect des angles produits par `rect()` et `polygon()`. R possède trois styles différents :

1. `mitre` : jointure pointue (sauf si l'angle entre les lignes est trop faible, dans ce cas le style utilisé sera `bevel` pour éviter d'avoir des angles trop pointus. Le point de transition est contrôlé par le paramètre `lmitre`).
2. `round` : jointure arrondie.
3. `bevel` : jointure plate.

La différence entre les trois styles de jointure est bien visualisée par le graphique produit par le code[4] suivant :

[4] adapté de la figure 3.7 du livre de Paul Murrell [2] dont le code est disponible en ligne à `http://www.stat.auckland.ac.nz/~paul/RGraphics/custombase-linesimplÃĭe.R`

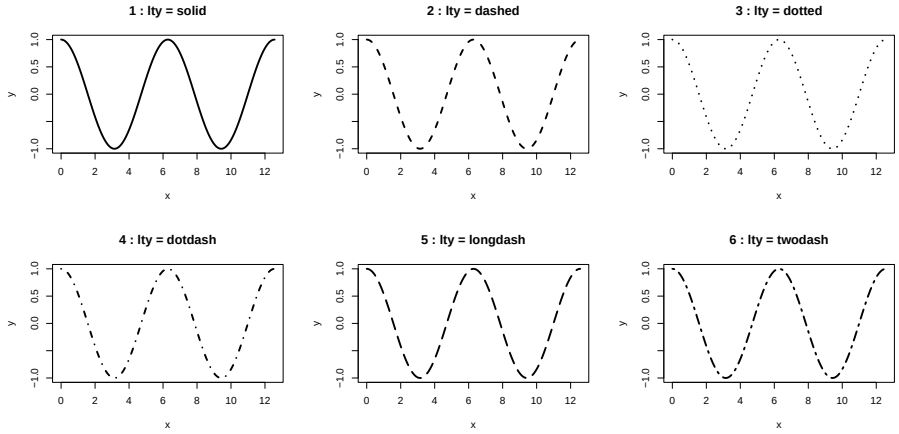
```
x <- c(.3, .7, .3)
y <- c(.2, .5, .8)
par(ljoin = "mitre")
plot(x, y, type = "l", lwd = 40, col = "black", xlim = c(0,1), ylim = c(0,1),
main = "Le paramètre ljoin")
par(ljoin = "round")
lines(x, y, lwd = 40, col = "grey50")
par(ljoin = "bevel")
lines(x, y, lwd = 40, col = "grey80")
points(x, y)
par(lend = "butt")
legend("bottomright", inset = 0.01, legend = c("ljoin = \"mitre\"",
"ljoin = \"round\"","ljoin = \"bevel\""),
lty = 1,
lwd = 10, col = c("black","grey50","grey80"))
```



2.3.8 `par("lty")`

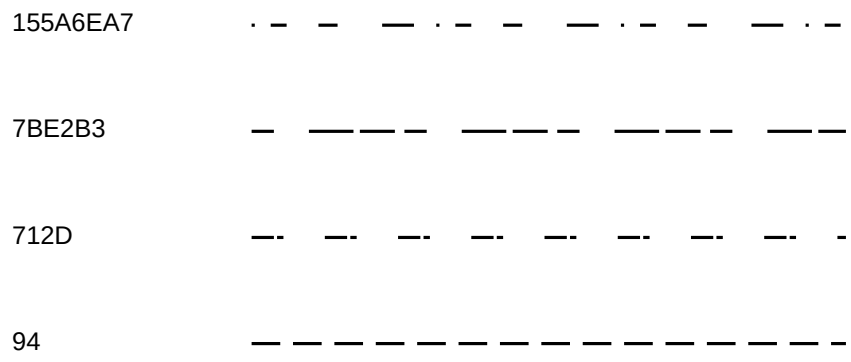
Un paramètre très utile pour choisir le type de ligne à tracer. On peut également le spécifier avec un entier : 1 pour `solid`, 2 pour `dashed`, 3 pour `dotted`, 4 pour `dotdash`, 5 pour `longdash` et 6 pour `twodash`.

```
par(mfrow = c(2, 3))
vals <- c("solid", "dashed", "dotted", "dotdash", "longdash",  "twodash")
x <- seq(0, 4*pi, length = 255) ; y <- cos(x)
for(i in seq_len(length(vals))) {
  par(lty = vals[i])
  plot(x, y, type = "l", main = paste(i, ": lty =", vals[i]), lwd = 2)
}
```



On peut avoir un contrôle beaucoup plus fin en utilisant une chaîne de caractères ayant un nombre pair (huit au maximum) de chiffres hexadécimaux strictement positifs qui donnent les longueurs des traits dessinés puis omis.

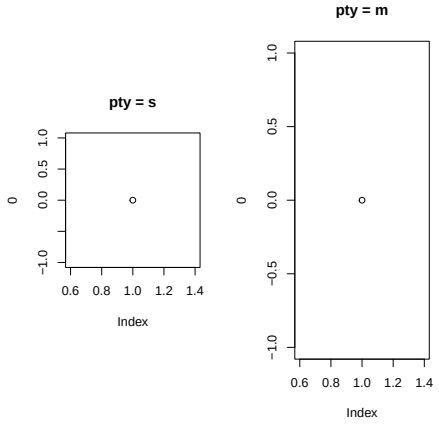
```
set.seed(1)
phd <- c(1:9, LETTERS[1:6])
par(mar = c(0, 0, 0, 0), lend = "butt")
plot.new()
plot.window(xlim = c(0, 10), ylim = c(1, 9))
for(n in 2*(1:4)){
  lty <- paste0(sample(phd, n, replace = TRUE), collapse = "")
  par(lty = lty)
  text(0, n, lty, pos = 4)
  lines(c(3, 10), c(n, n), lty = lty, lwd = 2)
}
```



2.3.9 `par("pty")`

Contrôle si on veut une figure carrée ou de taille maximale.

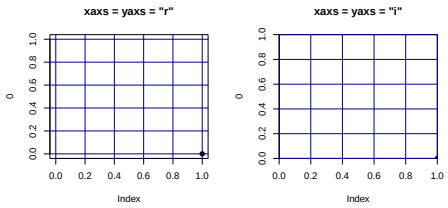
```
par(mfrow = c(1, 2))
vals <- c("s", "m")
for(pty in vals) {
  par(pty = pty)
  plot(0, main = paste("pty =", pty))
}
```



2.3.10 `par("xaxs")` et `par("yaxs")`

Contrôle le style des axes, permet de neutraliser l'extension de 4 % de `xlim` et `ylim` :

```
par(mfrow = c(1,2))
par(xaxs = "r", yaxs = "r")
plot(0, xlim = 0:1, ylim = 0:1, main = "xaxs = yaxs = \"r\"", pch = 19)
grid(col= "darkblue", lty = 1)
par(xaxs = "i", yaxs = "i")
plot(0, xlim = 0:1, ylim = 0:1, main = "xaxs = yaxs = \"i\"", pch = 19)
grid(col = "darkblue", lty = 1)
```

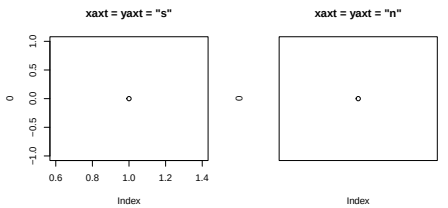


La valeur par défaut `"r"` permet d'éviter de perdre des points écrasés sur les axes.

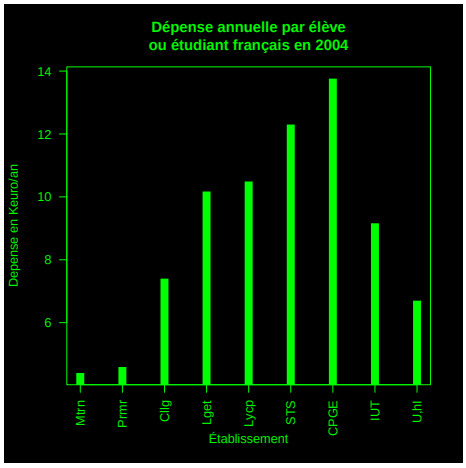
2.3.11 `par("xaxt")` et `par("yaxt")`

Contrôle de l'affichage des axes :

```
par(mfrow = c(1,2))
par(xaxt = "s", yaxt = "s")
plot(0, main = "xaxt = yaxt = \"s\"")
par(xaxt = "n", yaxt = "n")
plot(0, main = "xaxt = yaxt = \"n\"")
```



Exercice. Neutraliser l'affichage de l'axe horizontal, puis à l'aide de la fonction `axis()` ajouter un axe dont les étiquettes sont les noms abrégés des établissements :



2.4 Les paramètres graphiques numériques

```
sort(names(allparm[sapply(allparm,class) == "numeric"]))
```

```
 [1] "adj"      "cex"      "cex.axis" "cex.lab"  "cex.main" "cex.sub"  "crt"
 [8] "fig"      "fin"      "lheight"  "lmitre"   "lwd"      "mai"      "mar"
[15] "mex"      "mgp"      "mkh"      "oma"      "omd"      "omi"      "pin"
[22] "plt"      "smo"      "srt"      "tck"      "tcl"      "usr"      "xaxp"
[29] "yaxp"     "ylbias"
```

```
sort(sapply(allparm[sapply(allparm,class) == "numeric"],length))
```

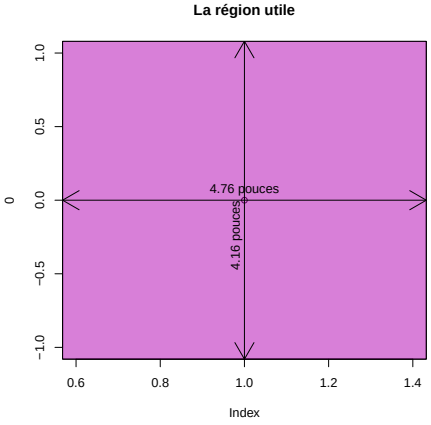
```
    adj      cex cex.axis  cex.lab cex.main  cex.sub      crt  lheight   lmitre
      1        1        1        1        1        1        1        1        1
    lwd      mex      mkh      smo      srt      tck      tcl   ylbias      fin
      1        1        1        1        1        1        1        1        2
    pin      mgp     xaxp     yaxp      fig      mai      mar      oma      omd
      2        3        3        3        4        4        4        4        4
    omi      plt      usr
      4        4        4
```

```
table(sort(sapply(allparm[sapply(allparm,class) == "numeric"],length)))
```

```
 1  2  3  4
17  2  3  8
```

On définit une fonction utilitaire pour mettre en évidence la région utile :

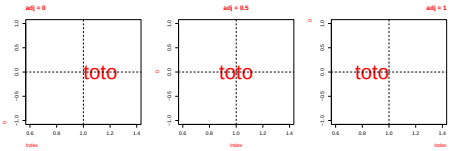
```
showmeplot <- function() {
  p <- par("usr")
  midx <- (p[1] + p[2])/2
  midy <- (p[3] + p[4])/2
  rect(xleft = p[1], ybottom = p[3], xright = p[2], ytop= p[4], col = rgb(0.7, 0, 0.7, 0.5))
  arrows(x0 = p[1], y0 = midy, x1 = p[2], y1 = midy, code = 3)
  text(x = midx, y = midy, labels = paste(round(par("pin")[1],2),"pouces"),
    xpd = NA, pos = 3)
  arrows(x0 = midx, y0 = p[3], x1 = midx, y1 = p[4], code = 3)
  text(x = midx, y = p[3] +(p[4] - p[3])/2, labels = paste(round(par("pin")[2], 2),
    "pouces"), xpd = NA, pos = 2, srt = 90)
}
plot(0, main = "La région utile")
showmeplot()
```



2.4.1 `par("adj")`

Contrôle la justification des chaînes de caractères.

```
par(mfrow = c(1,3))
vals <- c(0, 0.5, 1)
for(adj in vals){
  par(adj = adj)
  plot(0, main = paste("adj =", adj), col.lab = "red", col.main = "red", type = "n")
  text(1, 0, "toto", col = "red", cex = 4)
  abline(h = 0, lty = 2)
  abline(v = 1, lty = 2)
}
```



Remarque : la fonction `text()` permet aussi un contrôle dans le sens vertical avec son *argument* `adj` qui est un vecteur de deux éléments, `adj[1]` pour l'alignement dans le sens horizontal et `adj[2]` pour l'alignement dans le sens vertical, la figure 2 a été produite avec le code ci-après, et la figure 3 avec le même code sauf `text(srt = 45)`. La fonction `text()` possède également un paramètre `pos` qui permet de positionner rapidement le texte (*cf* figure 4).

```
par(mar = c(4, 4, 3, 0) + 0.1, xaxs = "r", yaxs = "r")
n <- 3
plot(expand.grid(1:n, 1:n), pch = 3, col = grey(0.5),
xlim = c(0, n + 1), ylim = c(0, n + 1), cex = 5,
xlab = "adj[1]", ylab = "adj[2]", xaxt = "n", yaxt = "n", bty = "n",
main = "Le paramètre adj de text()")
adj <- c(1, 0.5, 0)
#grid()
axis(1, at = 1:n, paste(adj))
axis(2, at = 1:n, paste(adj), las = 1)
msg <- "Ô, temps !"
for (i in 1:n)
  for (j in 1:n)
    text(i, j, msg, adj = c(adj[i], adj[j]), cex = 1.5)
```

Exercice. Faire un graphique où les noms des établissements sont justifiés à droite comme ci-dessous :

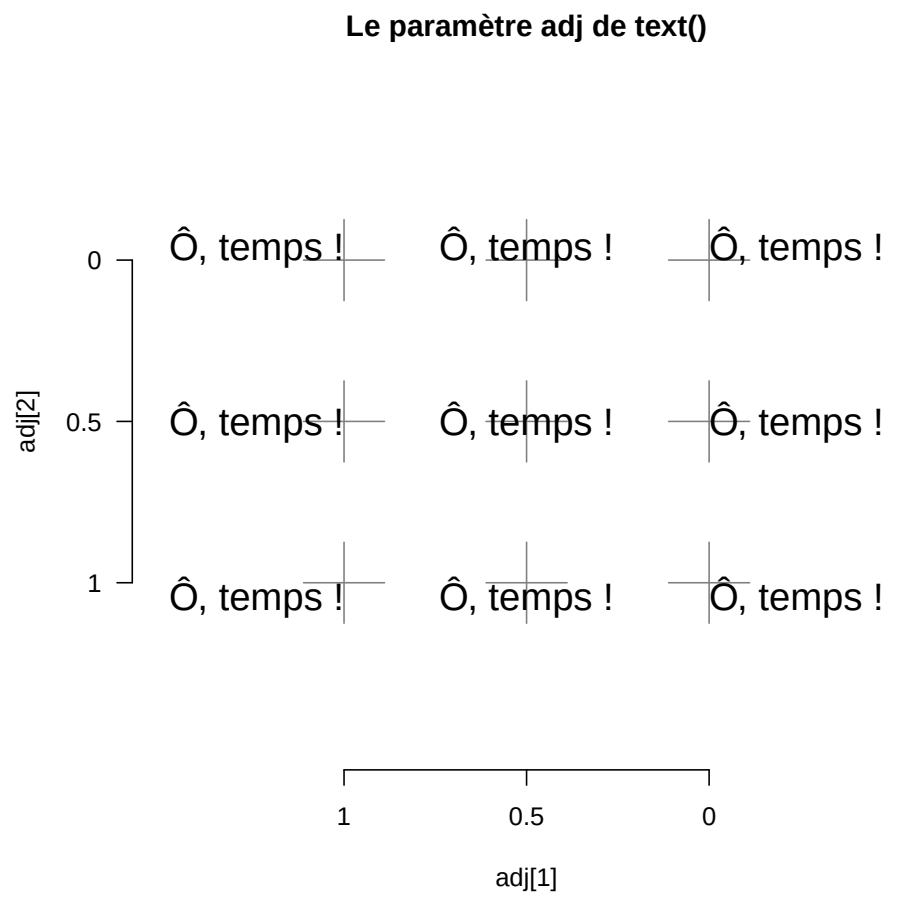


Figure 2: Contrôle fin de l'alignement des textes avec *l'argument* `adj` de la fonction `text()`. L'argument `adj` est un vecteur de deux éléments, `adj[1]` pour l'alignement dans le sens horizontal et `adj[2]` pour l'alignement dans le sens vertical. Une valeur de 1 signifie que le texte va être justifié à droite et en haut, pour les sens horizontal et vertical, respectivement. Une valeur de 0 signifie que le texte va être justifié à gauche et en bas, pour le sens horizontal et vertical, respectivement. Une valeur de 0.5 signifie que le texte va être centré, pour les deux sens. La justification dans le sens vertical n'est pas forcément honorée de façon précise par tous les périphériques graphiques parce qu'elle implique de connaître la hauteur de tous les caractères, et cette information n'est pas toujours disponible. Dans le sens vertical, notez la position du signe diacritique sur le 'O' (*i.e.* Ô) et celle de la hampe descendante du 'p' par rapport à la ligne de base.

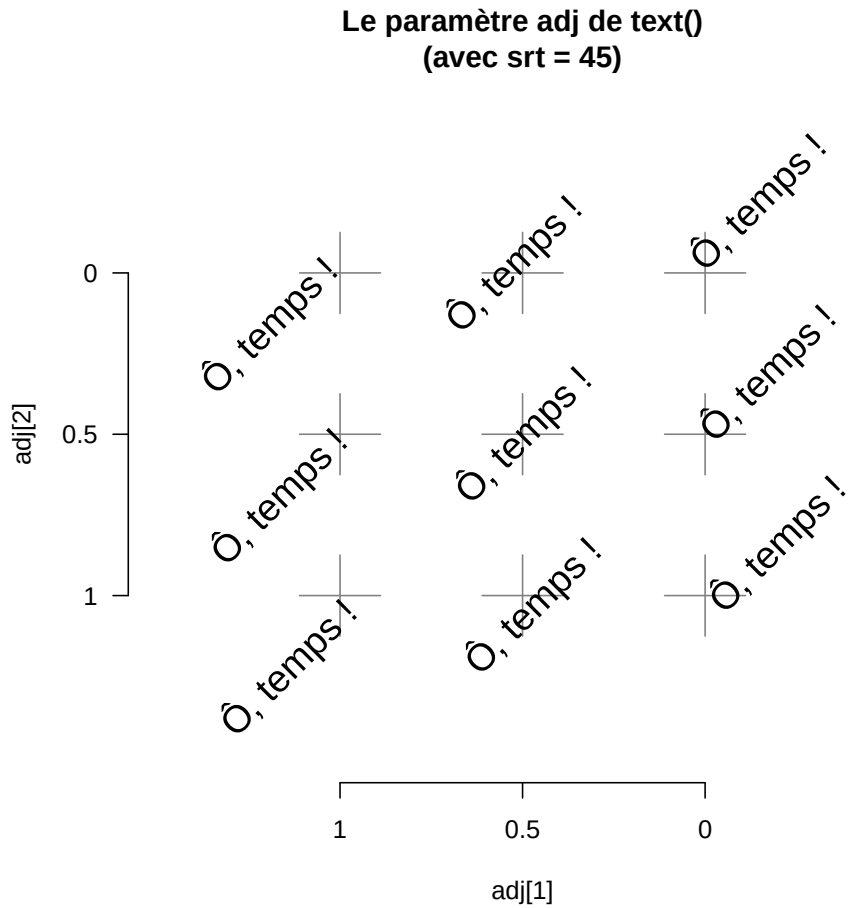


Figure 3: Contrôle de fin de l'alignement des textes (*cf* figure 2) avec en plus l'argument `srt` de la fonction `text()` ayant une valeur de `45`.

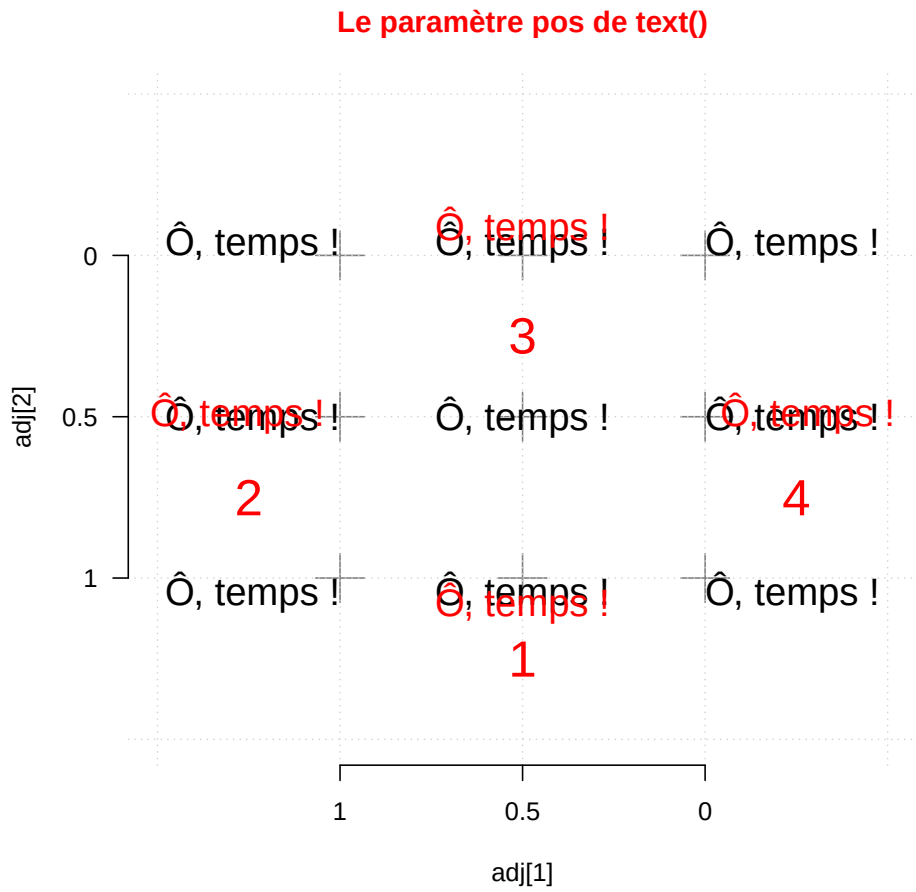
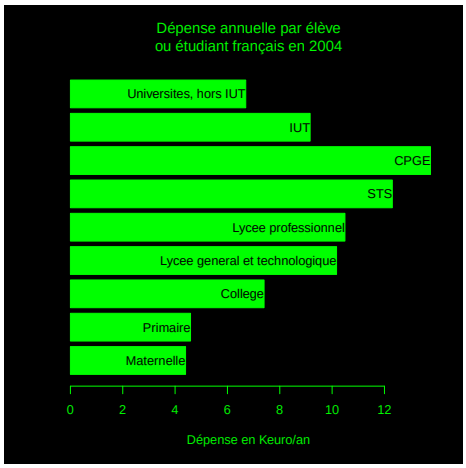


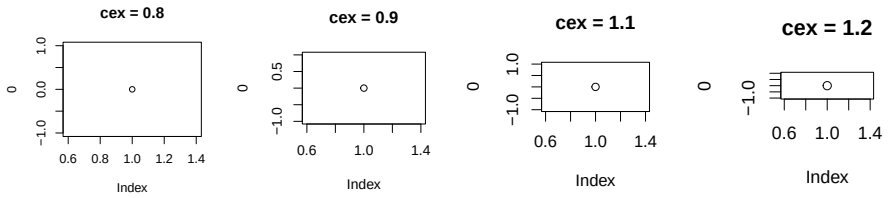
Figure 4: Contrôle rapide de l'alignement des textes avec l'argument `pos` de la fonction `text()`. Une valeur de 1 écrit en-dessous, une valeur de 2 à gauche, une valeur de 3 au-dessus et une valeur de 4 à droite. Notez que par rapport à ce qui est obtenu avec le paramètre `adj` il y a un décalage qui est contrôlé avec le paramètre `offset`. Par défaut celui-ci vaut la moitié de la largeur d'un caractère (`par("cxy")[1]`).



2.4.2 `par("cex*")`

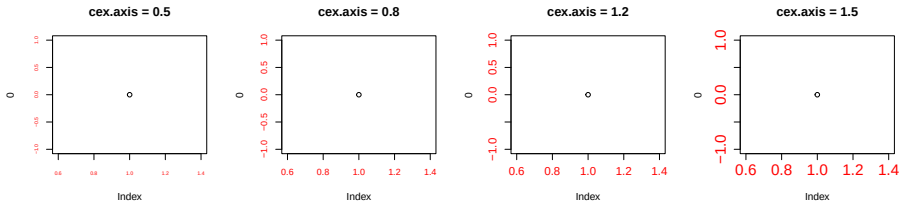
Ces paramètres contrôlent la taille relative des chaînes de caractères. Notez que les marges s'adaptent automatiquement à la taille des caractères :

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



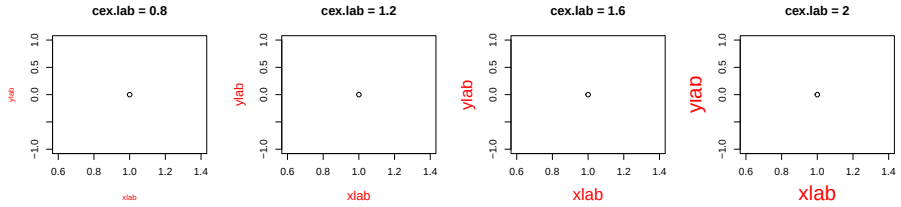
Mais les marges ne sont pas modifiées avec les autres paramètres graphiques de type `cex.*`.

```
par(mfrow = c(1, 4))
for(cex in seq(from = 0.5, to = 1.5, length = 4))
{
  par(cex.axis = cex)
  plot(0, main = paste("cex.axis =", round(cex,1)), col.axis = "red")
}
```

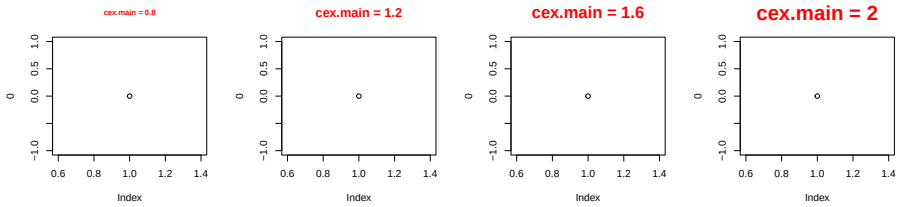



```
par(mfrow = c(1, 4))
for(cex in seq(from = 0.8, to = 2, length = 4))
{
```

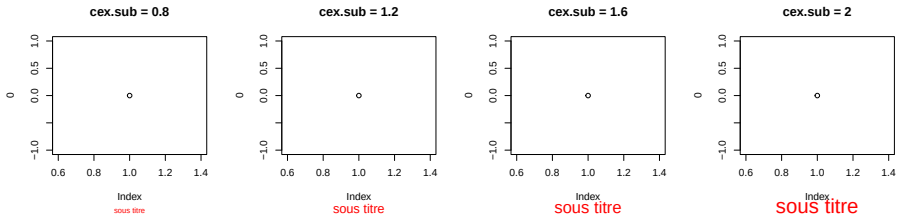
```
  par(cex.lab = cex)
  plot(0, main = paste("cex.lab =", round(cex,1)), col.lab = "red", xlab = "xlab", ylab = "ylab")
}
```




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



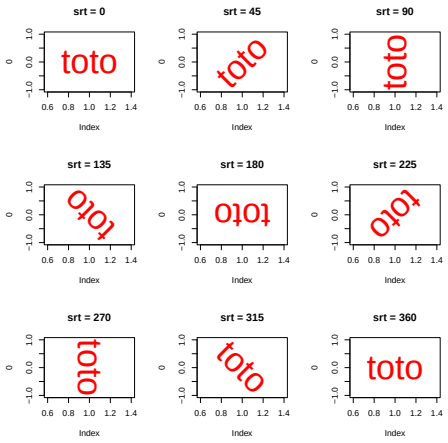

```
par(mfrow = c(1, 4))
for(cex in seq(from = 0.8, to = 2, length = 4))
{
  par(cex.sub = cex)
  plot(0, main = paste("cex.sub =", round(cex,1)), col.sub = "red", sub = "sous titre")
}
```



2.4.3 par("crt") et par("srt")

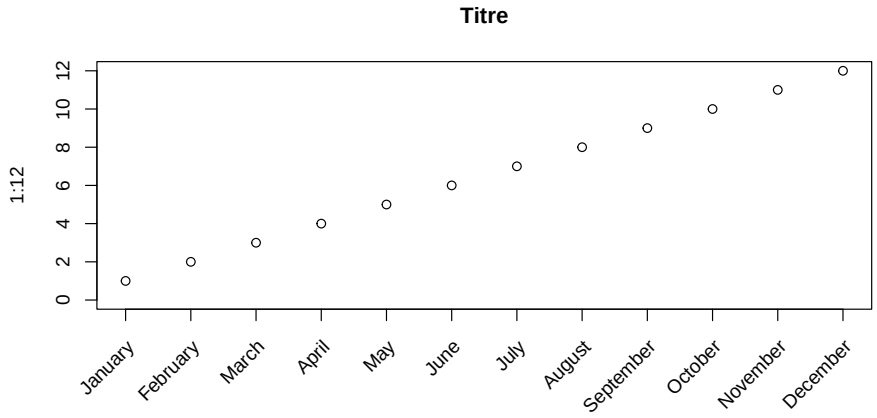
Contrôle de la rotation des chaînes de caractères :

```
par(mfrow = c(3, 3))
for(srt in seq(from = 0, to = 360, length = 9))
{
  par(srt = srt)
  plot(0, main = paste("srt =", round(srt,1)), type = "n")
  text(1,0,"toto", cex = 4, col = "red")
}
```

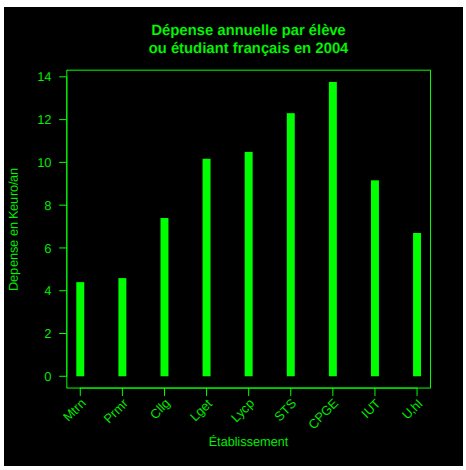


Peut être utile dans certaines situations particulières :

```
plot(1:12,1:12, main = "Titre", xaxt = "n", ylim = c(0,12), xlab = "")
axis(side = 1, at = 1:12, labels = FALSE)
text(x = 1:12, y = -2, labels = month.name, xpd = NA, srt = 45, adj = c(1,1))
```



Exercice. Faire la représentation graphique suivante :



2.4.4 par("fig") et par("plt")

Donnent les coordonnées en unités normalisées ($\in [0,1]$) de la région de la figure et de la région utile (plot).

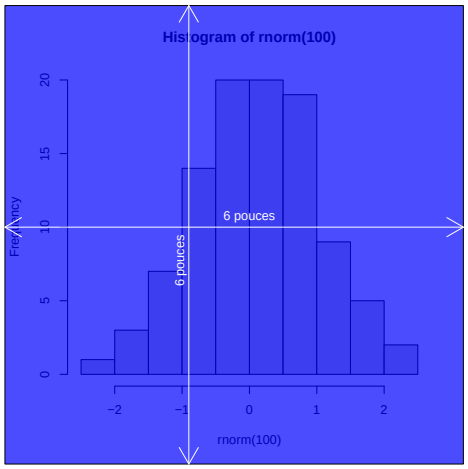
```
par("fig")
[1] 0 1 0 1
par("plt")
[1] 0.1366667 0.9300000 0.1700000 0.8633333
```

Pas très utile en pratique. On contrôle plutôt cela avec les marges exprimées en lignes de texte, ce qui est bien plus commode.

2.4.5 par("fin")

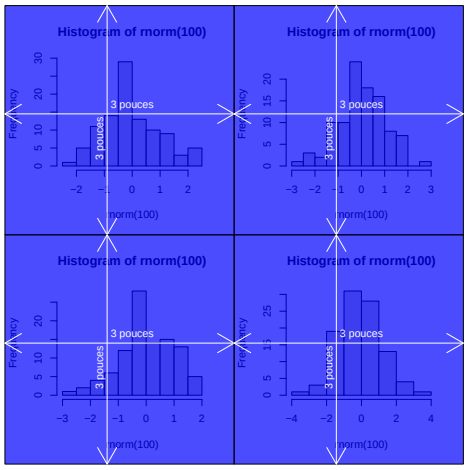
Dimensions de la figure en pouces. La fonction utilitaire `dimfig()` suivante colorie cette région pour la mettre en évidence :

```
dimfig <- function() {
  p <- par("usr")
  f <- par("plt")
  dx <- (p[2] - p[1])/(f[2] - f[1])
  dy <- (p[4] - p[3])/(f[4] - f[3])
  rect(xleft = p[1] - f[1]*dx, ybottom = p[3] -f[3]*dy, xright = p[2] + (1 - f[2])*dx,
    ytop = p[4] + (1 - f[4])*dy, col = rgb(0, 0, 1, 0.7), xpd = NA)
  arrows(x0 = p[1] - f[1]*dx, y0 = p[3] + (p[4] - p[3])/2, x1 = p[2] + (1 - f[2])*dx,
    y1 = p[3] + (p[4] - p[3])/2, xpd = NA, code = 3, col = "white")
  text(x = p[1] + (p[2] - p[1])/2, y = p[3] + (p[4] - p[3])/2,
    labels = paste(round(par("fin")[1], 1), "pouces"), pos = 3, xpd = NA, col = "white")
  arrows(x0 = p[1] + (p[2] - p[1])/3, y0 = p[3] - f[3]*dy, x1 = p[1] + (p[2] - p[1])/3,
    y1 = p[4] + (1 - f[4])*dy, xpd = NA, code = 3, col = "white")
  text(x = p[1] + (p[2] - p[1])/3, y = p[3] + (p[4] - p[3])/2.1,
    labels = paste(round(par("fin")[2], 1), "pouces"), pos = 2, srt = 90,
    xpd = NA, col = "white")
}
hist(rnorm(100))
dimfig()
```



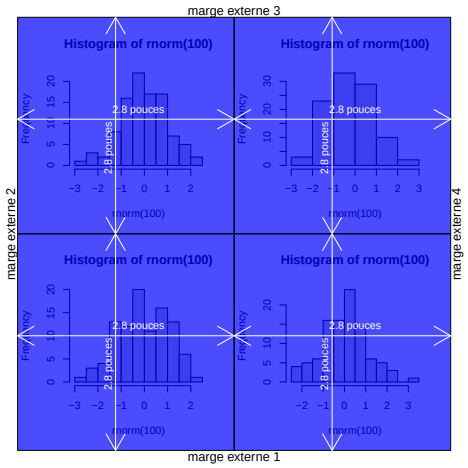
En cas de figures multiples :

```
par(mfrow=c(2,2))
for(i in 1:4) {
  hist(rnorm(100))
  dimfig()
}
```



En cas de marge externe, moins de place est disponible pour les figures :

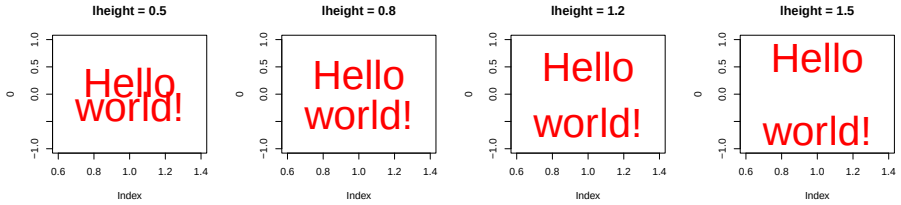
```
par(oma = rep(1,4))
par(mfrow=c(2,2))
for(i in 1:4) {
  hist(rnorm(100))
  dimfig()
  mtext(side = i, text = paste("marge externe", i), outer = TRUE) }
```



2.4.6 `par("lheight")`

Contrôle de l'interligne des chaînes de caractères :

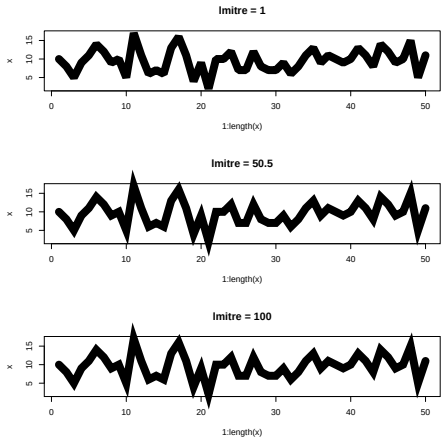
```
par(mfrow=c(1,4))
for(lheight in seq(from = 0.5, to = 1.5, length = 4))
{
  par(lheight=lheight)
  plot(0, main = paste("lheight =", round(lheight,1)), type = "n")
  text(1,0,"Hello\nworld!", cex = 4, col = "red")
}
```



2.4.7 par("lmitre")

Il contrôle le passage automatique du mode de raboutage des lignes de "mitre" à "bevel". Tous les périphériques n'honorent pas ce paramètre.

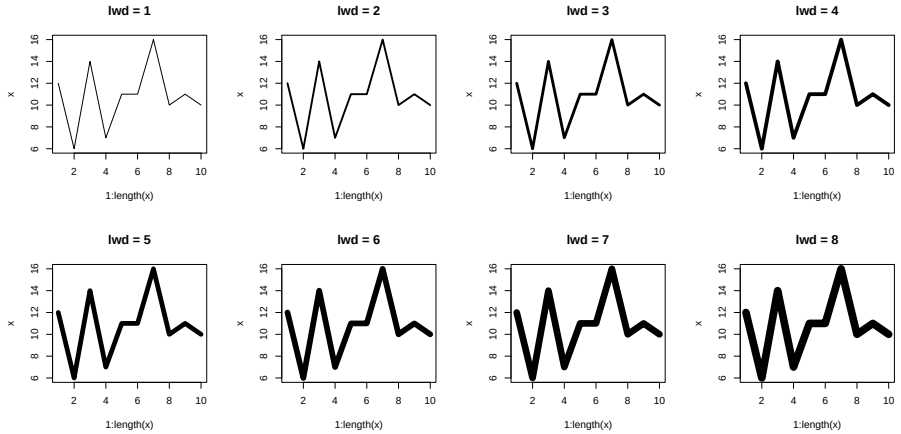
```
par(mfrow=c(3,1))
x <- rpois(50,10)
for(lmitre in seq(from = 1, to = 100, length = 3))
{
  par(lmitre=lmitre, ljoin = "mitre", xpd = NA)
  plot(1:length(x),x, main = paste("lmitre =", round(lmitre,1)), type = "l", lwd = 10)
}
```



2.4.8 par("lwd")

Contrôle de l'épaisseur du trait :

```
par(mfrow = c(2, 4), xpd = NA)
x <- rpois(10,10)
for(lwd in 1:8)
  plot(1:length(x),x, main = paste("lwd =", round(lwd,1)), type = "l", lwd = lwd)
```



2.4.9 `par("mar")` et `par("mai")`

Contrôlent des marges internes. La taille de la région utile est contrôlée par la taille des marges avec `par("mar")` exprimée en lignes. La valeur par défaut est de `c(5, 4, 4, 2) + 0.1` :

```
par("mar")
```

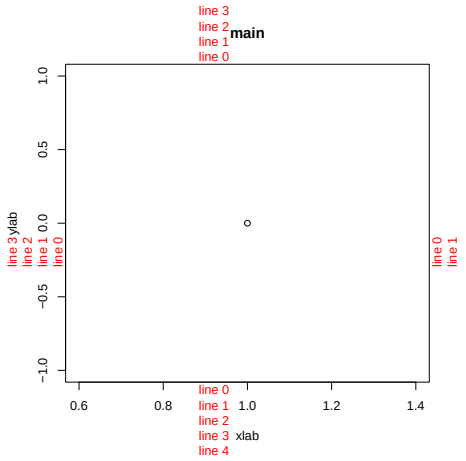
```
[1] 5.1 4.1 4.1 2.1
```

Dans un graphique avec les marges par défaut nous disposons donc dans l'ordre des aiguilles d'une montre :

1. 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.
2. À gauche 4 lignes : une ligne pour les graduations, une ligne pour les labels des graduations, une ligne vide, une ligne de légende.
3. En haut : 4 lignes pour le titre.
4. À droite : 2 lignes vides.

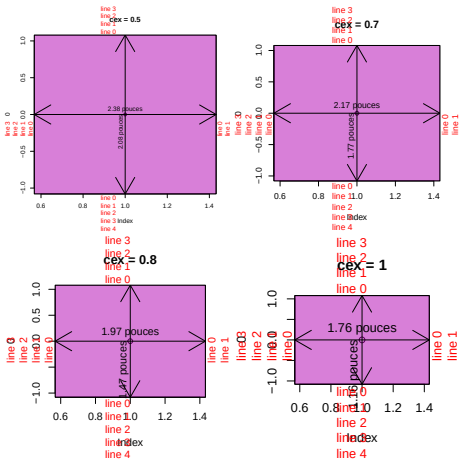
plus une bordure vide d'un dixième de ligne tout autour. Graphiquement :

```
plot(0, xlab = "xlab", ylab = "ylab", main = "main")
for(i in 0:4) mtext(text = paste("line",i), side = 1, line = i, col = "red", adj = 0.4)
for(i in 0:3) mtext(text = paste("line",i), side = 2, line = i, col = "red", adj = 0.4)
for(i in 0:3) mtext(text = paste("line",i), side = 3, line = i, col = "red", adj = 0.4)
for(i in 0:2) mtext(text = paste("line",i), side = 4, line = i, col = "red", adj = 0.4)
```



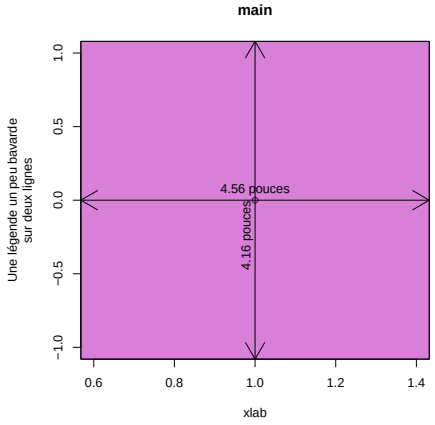
Le fait que les marges soient exprimées en unités de lignes de texte est très commode puisque leur taille va s'adapter automatiquement à la taille des caractères :

```
par(mfrow=c(2,2))
for(cex in seq(from = 0.5, to = 1, length = 4))
{
  par(cex=cex)
  plot(0, main = paste("cex =", round(cex,1)))
  showmeplot()
  for(i in 0:4) mtext(text = paste("line",i), side = 1, line = i, col = "red", adj = 0.4, cex = cex)
for(i in 0:3) mtext(text = paste("line",i), side = 2, line = i, col = "red", adj = 0.4, cex = cex)
for(i in 0:3) mtext(text = paste("line",i), side = 3, line = i, col = "red", adj = 0.4, cex = cex)
for(i in 0:1) mtext(text = paste("line",i), side = 4, line = i, col = "red", adj = 0.4, cex = cex)
}
```

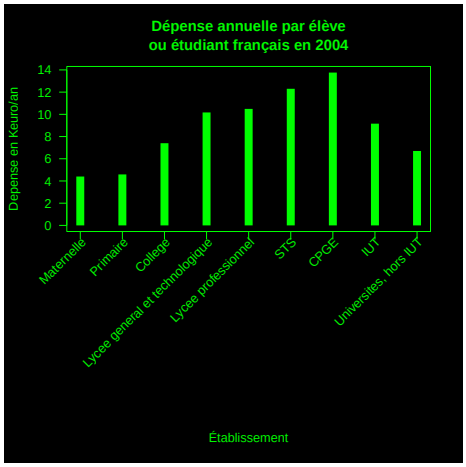


Du coup on n'utilise qu'assez peu `par("mai")` qui donne la taille des marges en pouces. Supposons que nous ayons besoin de deux lignes pour la légende de l'axe vertical, avec les valeurs par défaut des marges, il nous manque une ligne. Il suffit d'en ajouter une et le tour est joué :

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



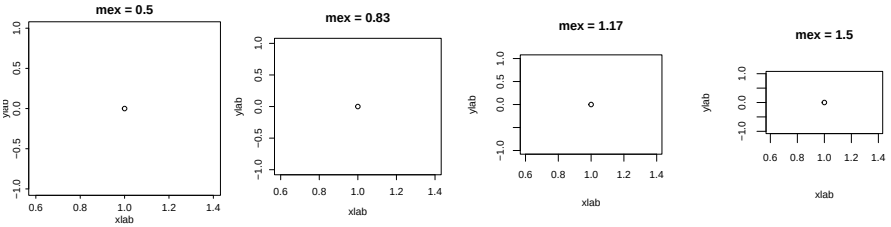
Exercice. Augmenter la marge en bas de la figure pour pouvoir faire la représentation suivante :



2.4.10 `par("mex")`

Contrôle le facteur d'expansion des marges.

```
par(mfrow = c(1, 4))
for( mex in seq(from = 0.5, to = 1.5, length = 4)){
  par(mex=mex)
  plot(0, xlab = "xlab", ylab ="ylab", main = paste("mex =", round(mex, 2)))
}
```



Mais comme la taille des marges s'adapte automatiquement à la taille des caractères, il est rare que l'on ait besoin de le modifier.

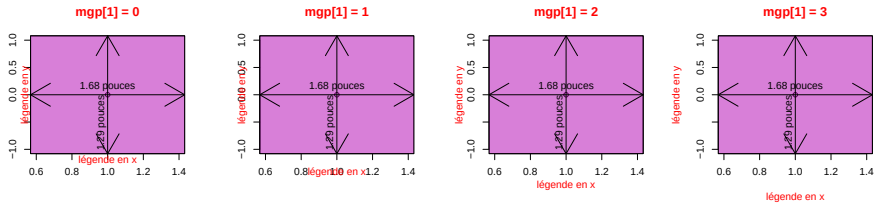
2.4.11 par("mgp")

Le paramètre utile pour contrôler le positionnement relatif des éléments des marges en x et y est `mgp`. Les valeurs par défaut sont :

```
par("mgp")
```
```
[1] 3 1 0
```

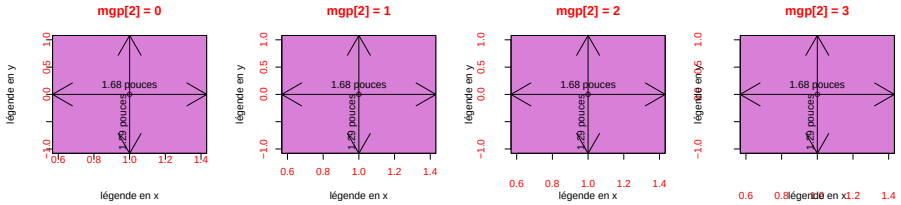
Ce sont des valeurs en unités `mex` (lignes de texte) pour les légendes des axes (par défaut 3) pour les étiquettes des graduations (par défaut 1) et pour la ligne de l'axe (par défaut 0). Le premier paramètre contrôle la distance relative dans les marges de la légende des axes :

```
par(mfrow = c(1,4))
for( mgp1 in seq(from = 0, to = 3, length = 4)){
  par(mgp = c(mgp1, 1, 0))
  plot(0, xlab = "légende en x", ylab ="légende en y", main = paste("mgp[1] =",
    round(mgp1, 2)), col.lab = "red", col.main = "red")
  showmeplot()
}
```



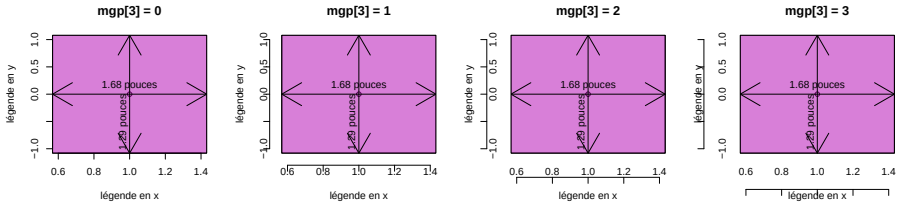
Le deuxième paramètre contrôle la distance relative dans les marges des étiquettes des graduations :

```
par(mfrow = c(1,4))
for( mgp2 in seq(from = 0, to = 3, length = 4)){
  par(mgp = c(3, mgp2, 0))
  plot(0, xlab = "légende en x", ylab ="légende en y", main = paste("mgp[2] =",
    round(mgp2, 2)), col.axis = "red", col.main = "red")
  showmeplot()
}
```



Le troisième paramètre contrôle la position relative de l'axe et de ses graduations :

```
par(mfrow = c(1,4))
for( mgp3 in seq(from = 0, to = 3, length = 4)){
  par(mgp = c(3, 1, mgp3))
  plot(0, xlab = "légende en x", ylab ="légende en y", main = paste("mgp[3] =",
    round(mgp3, 2)))
  showmeplot()
}
```



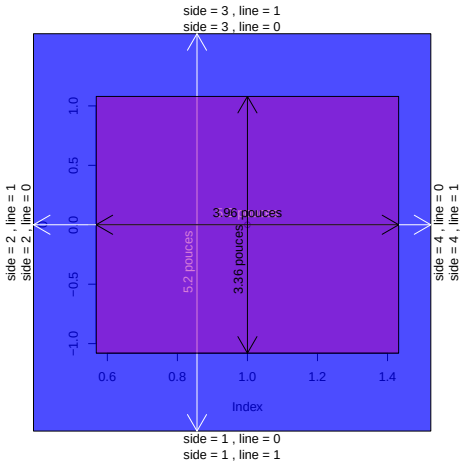
2.4.12 `par("mkh")` et `par("smo")`

Non implémentés (R 3.1.1).

2.4.13 `par("oma")`, `par("omi")` et `par("omd")`

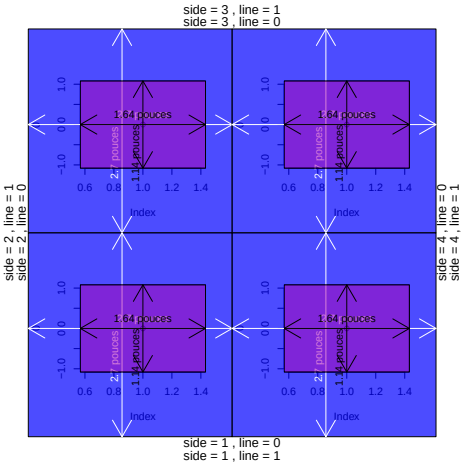
On peut ajouter des marges externes (exprimées en lignes) à la figure :

```
par(oma = rep(2,4))
plot(0)
dimfig()
showmeplot()
for(side in 1:4){
  mtext(paste("side =", side, ", line = 0"), side = side, line = 0, outer = TRUE)
  mtext(paste("side =", side, ", line = 1"), side = side, line = 1, outer = TRUE)
}
```



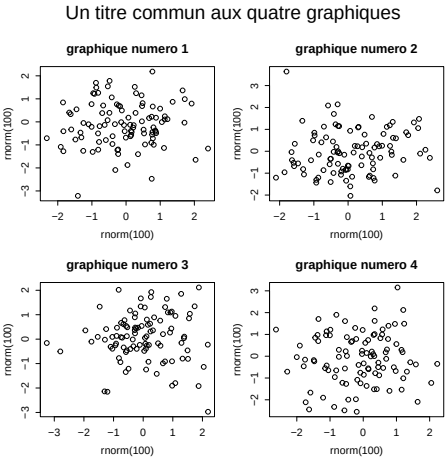
En cas de graphiques multiples, la marge externe est commune :

```
par(oma = rep(2,4), mfrow = c(2,2))
for(i in 1:4){
plot(0)
dimfig()
showmeplot()
}
for(side in 1:4){
  mtext(paste("side =", side, ", line = 0"), side = side, line = 0, outer = TRUE)
  mtext(paste("side =", side, ", line = 1"), side = side, line = 1, outer = TRUE)
}
```

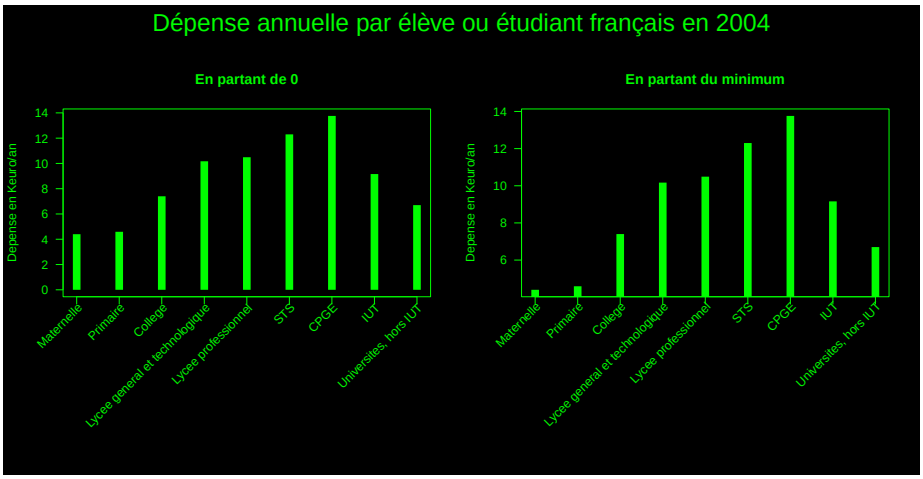


On peut ainsi facilement mettre un titre commun à quatre graphiques comme dans l'exemple ci-dessous.

```
par(oma = c(0, 0, 3, 0), mfrow = c(2,2), mex = 0.7)
for(i in 1:4){
  plot(rnorm(100), rnorm(100), main = paste("graphique numero", i))
}
mtext("Un titre commun aux quatre graphiques", side = 3, line = 1, outer = TRUE, cex = 1.5)
```



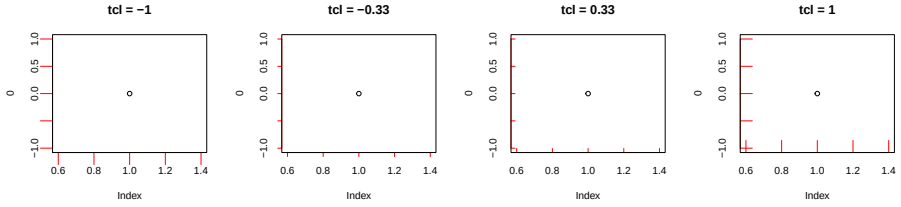
Exercice. Faire la représentation graphique suivante :



2.4.14 `par("tck")` et `par("tcl")`

Contrôle la longueur des graduations. Le plus simple est d'utiliser `tcl` exprimé en hauteur de lignes de textes.

```
par(mfrow = c(1, 4))
for( tcl in seq(from = -1, to = 1, length = 4)){
  par(tcl = tcl, fg = "red", col = "black")
  plot(0, main = paste("tcl =", round(tcl, 2)))
}
```



2.4.15 `par("tmag")` (obsolète)

Le paramètre `tmag` était censé modifier la taille relative des caractères du titre (dans R 2.3.0), mais impossible de le faire marcher. J'ai donc posé la question sur la liste de diffusion R-help :

```
From: Jean lobry <lobry_at_biomserv.univ-lyon1.fr>
Date: Wed 19 Apr 2006 - 17:28:37 EST

Dear list,

I'm trying to understand the graphical parameters by a systematic exploration of
the par() function (if you are interested by the result it's here
http://pbil.univ-lyon1.fr/R/pdf/tdr75.pdf, the comments are all in french
but it's pure R code under Sweave).

I have a problem with par(tmag) illustrated by the following code:

#####################################################
par(mfrow = c(2, 2))
for(tmag in seq(from = 0.5, to = 2, length = 4)){
   par(tmag = tmag)
   plot(0, main = paste("tmag =", round(tmag, 2))) }
#####################################################

From the documentation, tmag is "[a] number specifying the enlargement of
text of the main title relative to the other annotating text of the plot", so that
my understanding is that the main title size should have been modified between
the four figures. However, on my devices (quartz, pdf, png, or x11) the main
```

```
title size is not affected between the four figures.

What am I missing here? I have RSiteSearch("tmag") without success.

> version
               _
platform powerpc-apple-darwin7.9.0
arch     powerpc
os       darwin7.9.0
system powerpc, darwin7.9.0
status
major 2
minor 2.0
year 2005
month 10
day 06
svn rev 35749
language R

Any hint would be greatly appreciated,

Jean
```

À peine une heure quarante quatre minutes et trente secondes plus tard, la réponse est déjà postée par un célèbre contributeur oxfordien :

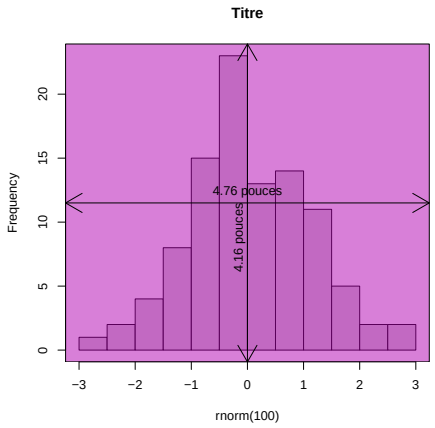
```
From: Prof Brian Ripley <ripley_at_stats.ox.ac.uk>
Date: Wed 19 Apr 2006 - 19:04:38 EST

It's not used by the internal code of title(), and I believe it is ancient
R history. Use "cex.main" instead.

You are using 2.2.0. 2.3.0 is in code freeze, but I will tidy up the
documentation in 2.3.0 and remove this in 2.4.0.
```

C'était donc simplement une option obsolète. On voit ici toute la force d'un logiciel libre comme R, la documentation a été amendée dès la version 2.3.0 et l'option enlevée dès la version 2.4.0. Que demander de plus ?

2.4.16 `par("pin")`

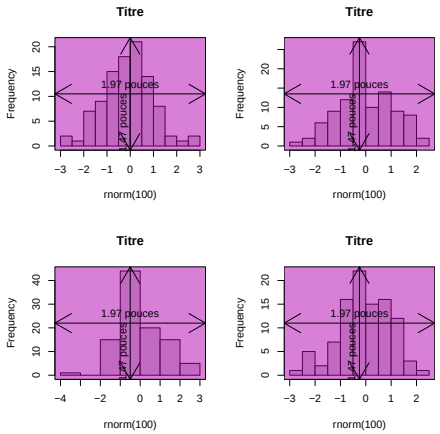
Dimensions de la région utile (plot) en pouces :

```
hist(rnorm(100), main = "Titre")
showmeplot()
```



Avec des graphiques multiples :

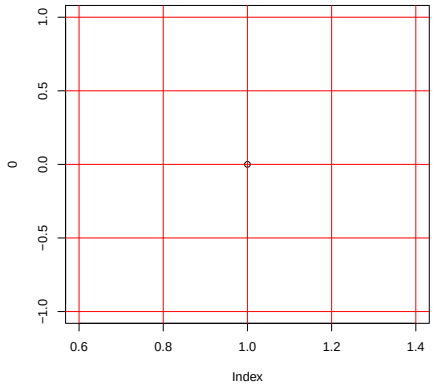
```
par(mfrow=c(2,2))
for(i in 1:4){
  hist(rnorm(100), main = "Titre")
  showmeplot()
}
```



2.4.17 `par("xaxp")` et `par("yaxp")`

Donnent les coordonnées des graduations. Simple en échelle linéaire :

```
plot(0)
xt <- par("xaxp")
xloc <- seq(from = xt[1], to = xt[2],length=xt[3]+1)
abline(v = xloc, col = "red")
yt <- par("yaxp")
yloc <- seq(from = yt[1], to = yt[2],length=yt[3]+1)
abline(h = yloc, col = "red")
```

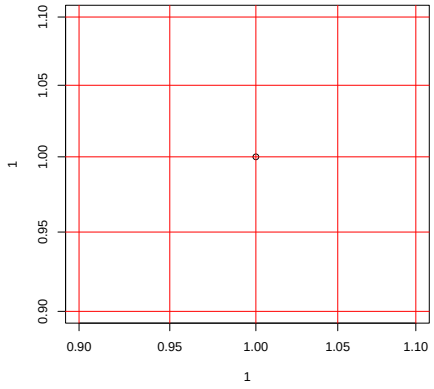


En coordonnées logarithmiques quand l'étendue est faible, `par("xaxp")[3]` est négatif, mais sinon c'est comme dans le cas linéaire :

```
plot(x = 1,y = 1, xlim = c(0.9,1.1), ylim = c(0.9,1.1), log = "xy")
xt <- par("xaxp")
xt[3]
```

```
[1] -4
```

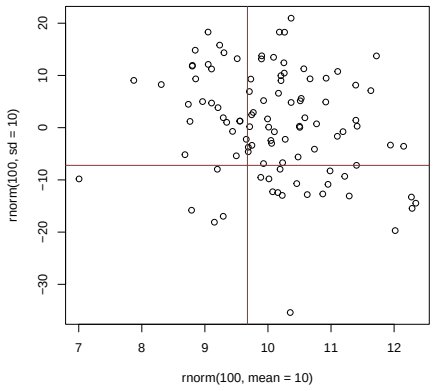
```
xloc <- seq(from = xt[1], to = xt[2],length=-xt[3]+1)
abline(v = xloc, col = "red")
yt <- par("yaxp")
yloc <- seq(from = yt[1], to = yt[2],length=-yt[3]+1)
abline(h = yloc, col = "red")
```



2.4.18 `par("usr")`

L'appel à `par("usr")` nous donne les coordonnées en unités utilisateur de la région utile (*plot region*). Par exemple pour diviser la région en quatres zones égales :

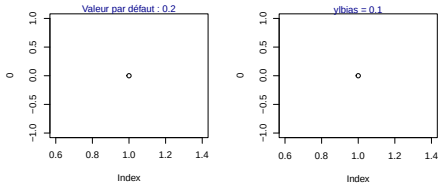
```
plot(rnorm(100, mean = 10), rnorm(100, sd = 10))
p <- par("usr")
abline(v = p[1] + (p[2] - p[1])/2, col = "red")
abline(h = p[3] + (p[4] - p[3])/2, col = "red")
```



2.4.19 `par("ylbias")`

Ce paramètre a été introduit assez tardivement dans la version R 2.14.0 en octobre 2011. Il contrôle la position du texte affiché dans les marges par les fonction `axis()` et `mtext()`. La valeur par défaut de 0.2 convient bien, ce paramètre a été introduit pour pouvoir reproduire ce qui se passait sur les périphériques graphiques `X11()` et `windows()` qui utilisaient une valeur de 0.1 avant la version 2.14.0.

```
par(mfrow = c(1, 2))
plot(0)
mtext("Valeur par défaut : 0.2", col = "darkblue")
par(ylbias = 0.1)
plot(0)
mtext("ylbias = 0.1", col = "darkblue")
```



TODO: A positive real value used in the positioning of text in the margins by axis and mtext. The default is in principle device-specific, but currently 0.2 for all of R's own devices. Set this to 0.2 for compatibility with R $<$ 2.14.0 on x11 and windows() devices.

References

[1] J. Marseille. *Les bons chiffres pour ne pas voter nul en 2007*. Perrin, Paris, France, 2007.

[2] P. Murrell. *R Graphics*. Computer Science & Data Analysis. Chapman & Hall/CRC, New York, 2006.