Fiche TD avec le logiciel R : tdr83

Les fonctions graphiques « classiques » 2D du paquet `ade4`

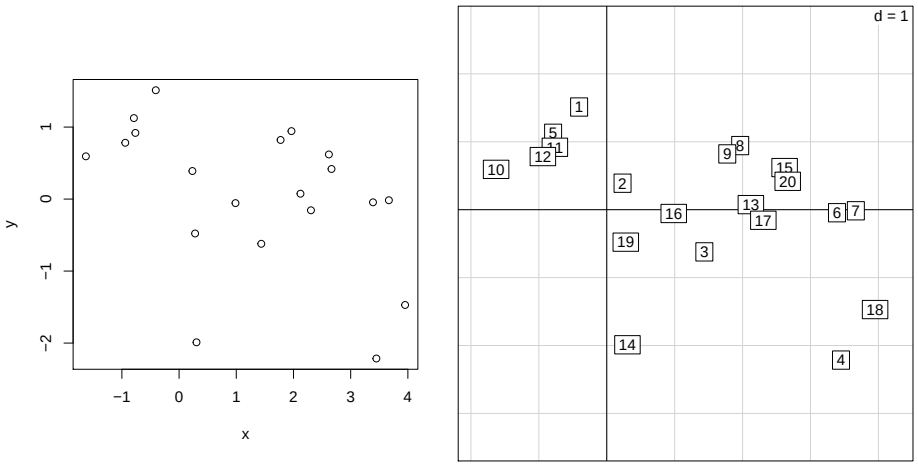
J.R. Lobry

Table des matières

1 Introduction

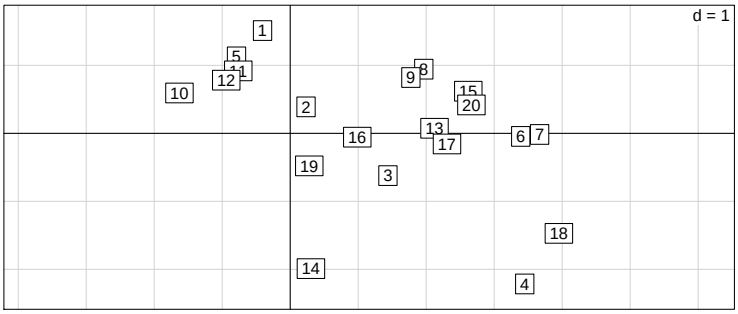
Le paquet `ade4` [1] propose un ensemble de fonctions destinées à manipuler les nuages de points et de vecteurs, la carte factorielle étant un des produits de base de l'analyse des données. Ces fonctions ont été complètement ré-implémentées dans le paquet `adegraphics` [2], on ne considère ici que les fonctions graphiques « classiques » d'`ade4`. Comparons les résultats dans le cas de la représentation d'un nuage de points :

```
set.seed(1)
x <- runif(20,-2,4)
y <- rnorm(20)
z <- data.frame(x,y)
par(mfrow = c(1,2))
plot(x, y)
library(ade4)
s.label(z)
```



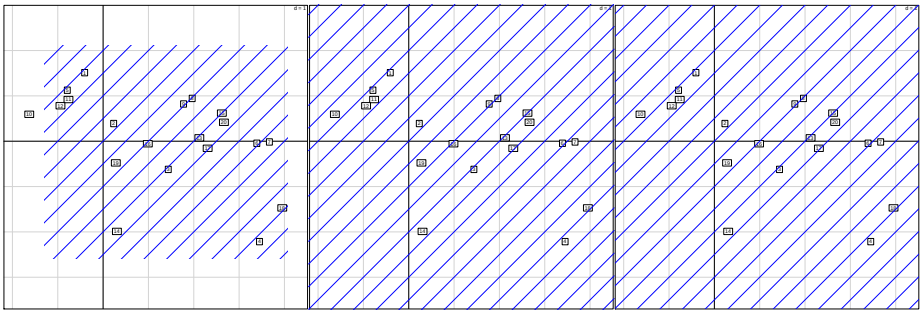
Le choix qui a été fait dans `ade4` est de remplir la totalité de la fenêtre graphique, sans marge. En effet, les cartes factorielles sont en général des graphes trop encombrés. Par défaut l'origine $(0, 0)$ appartient au graphe et on trace les deux axes. La repésentation étant euclidienne, par défaut l'échelle est la même sur les deux axes. On trace un quadrillage qui indique l'échelle commune. L'indication, en haut et à droite indique la valeur de la maille de ce quadrillage. Par défaut, les étiquettes des points sont placées dans un rectangle blanc et superposées. Typiquement le dessin de gauche est un graphe qui définit y comme fonction de x, celui de droite est une figure qui représente n points de coordonnées x et y. En jouant sur la taille du périphérique graphique utilisé, on utilisera au mieux la place disponible, en respectant l'identité des échelles, puisqu'en général, les représentations euclidiennes sont faites dans des bases orthonormées.

```
s.label(z)
```



En sortie de la fonction `s.label()`, les marges graphiques utilisées avant l'appel sont restaurées. Ceci explique que certaines régions du graphique généré ne soient pas directement accessibles. On peut y remédier en utilisant l'argument `xpd = TRUE` ou en redéfinissant les marges :

```
par(mfrow=c(1,3))
gribouille <- function(...){
 for(a in seq(-10,10, by = 0.5)){
   abline(c(a, 1), col = "blue", ...)
 }
}
s.label(z)
gribouille()
s.label(z)
gribouille(xpd = TRUE)
s.label(z)
par(mar = c(0.1, 0.1, 0.1, 0.1))
gribouille()
```



Ce type de représentation est utilisé par les fonctions de la famille `s.*`, ainsi que par la fonction générique `scatter()` :

```
ls("package:ade4", pattern = "^s\\.")
 [1] "s.arrow"       "s.chull"       "s.class"       "s.corcircle"   "s.distri"
 [6] "s.hist"        "s.image"       "s.kde2d"       "s.label"       "s.logo"
[11] "s.match"       "s.match.class" "s.multinom"    "s.traject"     "s.value"
ls("package:ade4", pattern = "^scatter\\.")
character(0)
```

2 Les fonctions de la famille `s.*`

2.1 Table ordonnée de présence/absence des arguments dans ces fonctions

Les arguments disponibles pour les fonctions de la famille de `s.*` sont résumés dans la figure 1 produite avec le code R suivant :

```
sqqc <- ls("package:ade4", patt="^s\\.")
lapply(sqqc, function(x) names(as.list(args(x)))) -> larg
names(larg) <- sqqc
larg <- lapply(larg, function(x) x[nchar(x)>1])
allargs <- sort(unique(unlist(larg)))
TabArg <- matrix(0, nrow = length(allargs), ncol = length(sqqc))
rownames(TabArg) <- allargs
colnames(TabArg) <- sqqc
for(j in 1:length(sqqc)) TabArg[,j] <-  allargs %in% larg[[j]]
afc <- dudi.coa(TabArg, scan = FALSE)
TabArg <- TabArg[order(afc$li[,1]), order(afc$co[,1])]
plot.new()
par(mar=c(0,0,0,0)+0.1)
nl <- length(allargs)
nc <- length(sqqc)
plot.window(xlim = c(0, nc+1), ylim = c(0, nl+2))
text(1, 1:nl, rownames(TabArg), cex = 0.6, pos = 2)
segments(0.8, 1:nl, nc+0.2, 1:nl, col = grey(0.7))
text(1:nc+0.5, nl+1, colnames(TabArg), srt = 30, pos = 3, cex = 0.8)
segments(1:nc, 0.8, 1:nc, nl+0.2, col = grey(0.7))
for(i in 1:nl) for(j in 1:nc) if(TabArg[i,j]==1) points(j,i, pch = 20)
```

2.2 `s.label()`

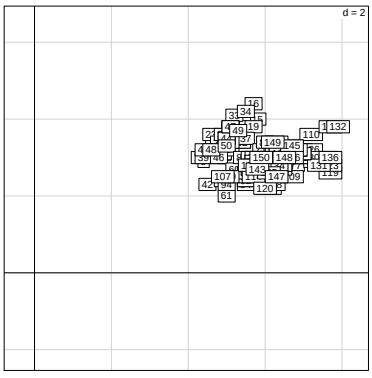
La fonction `s.label()` permet de représenter un nuage de points. Elle comporte de nombreux arguments :

```
args(s.label)
```

```
function (dfxy, xax = 1, yax = 2, label = row.names(dfxy), clabel = 1,
    pch = 20, cpoint = if (clabel == 0) 1 else 0, boxes = TRUE,
    neig = NULL, cneig = 2, xlim = NULL, ylim = NULL, grid = TRUE,
    addaxes = TRUE, cgrid = 1, include.origin = TRUE, origin = c(0,
        0), sub = "", csub = 1.25, possub = "bottomleft", pixmap = NULL,
    contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

Nous allons détailler l'utilisation de ces arguments car on les retrouve souvent dans les fonctions de la famille `s.*`. Le seul argument obligatoire est `dfxy` qui doit donner le nom d'un objet de la classe `data.frame` comportant au moins deux colonnes numériques. Par défaut, ce sont la première, `xax = 1`, et la deuxième colonne, `yax = 2`, qui sont utilisées pour l'axe des abscisses et l'axe des ordonnées, respectivement.

```
data(iris)
s.label(iris)
```



Les coordonnées factorielles étant souvent centrées, l'origine par défaut est en $(0, 0)$, et elle est incluse automatiquement. On peut désactiver cette option avec l'argument `include.origine` :

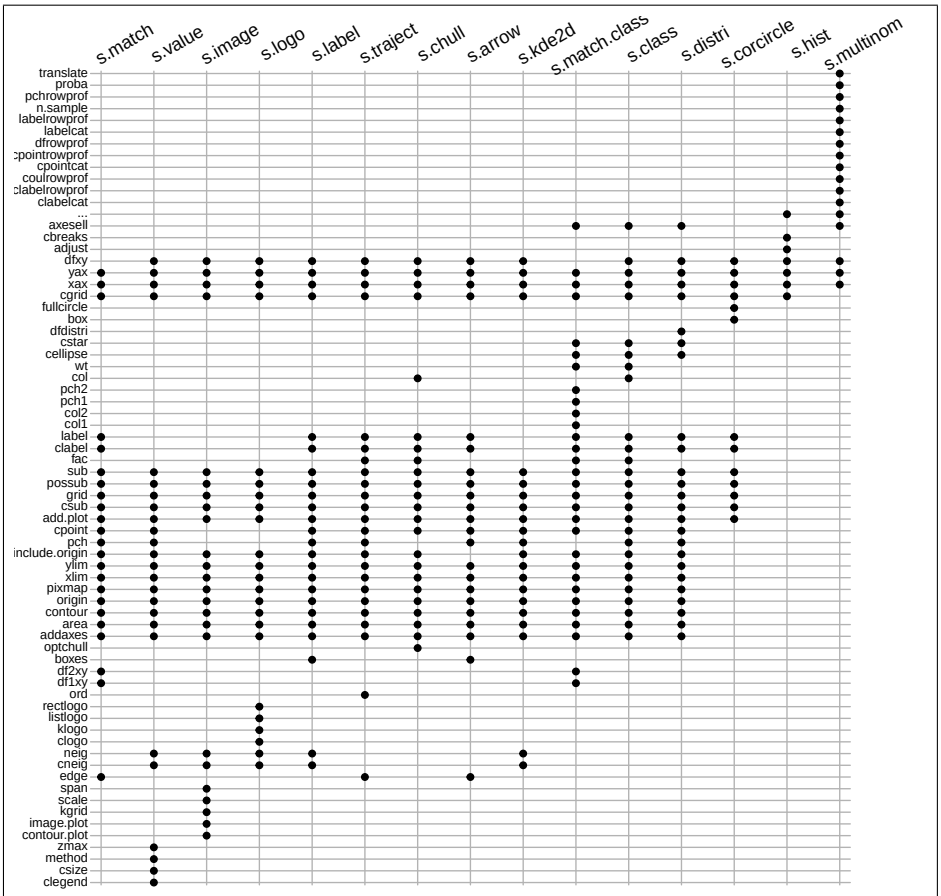
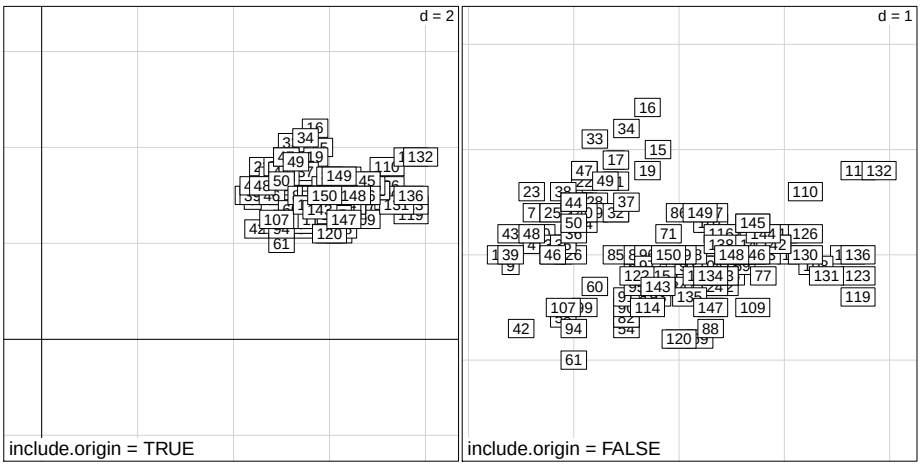


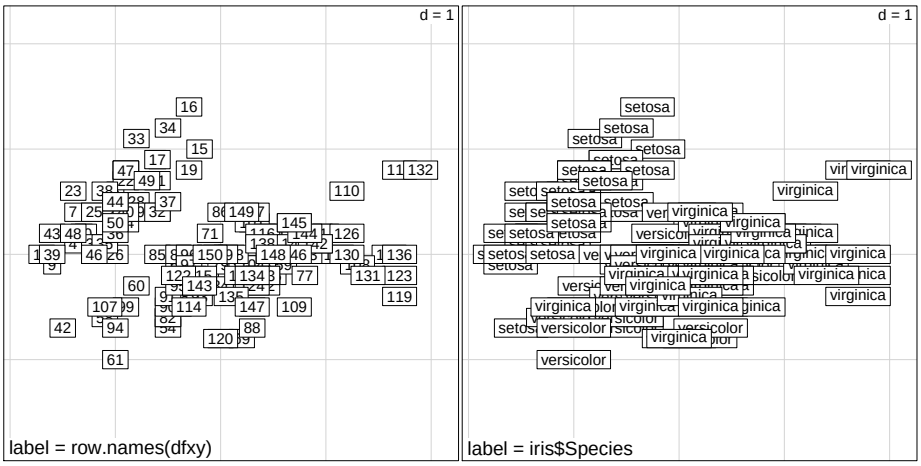
FIGURE 1 – Présentation synthétique des arguments disponibles pour les fonctions de la famille `s.*`. Les 70 arguments possibles sont disposés en ligne, les 15 fonctions de la famille `s.*` sont disposées en colonne. Un point de coordonnées (i, j) signifie que l'argument i est explicitement disponible pour la fonction j. L'ordre des lignes et des colonnes est selon le premier facteur de l'AFC de cette table de présence/absence. La fonction `s.multinom()` se démarque par le nombre d'arguments qui lui sont spécifique. Les fonctions `s.match()`, `s.value()`, `s.image()` et `s.logo()` se distinguent par leurs arguments additionnels. On note quatre arguments quasi-ubiquistes (*viz* `dfxy`, `yax`, `xax` et `cgrid`) ainsi qu'un noyau d'arguments très utilisés au milieu de la table. L'étude détaillée de la fonction `s.label()` permettra de couvrir les arguments les plus usuels.

```
par(mfrow=c(1,2))
s.label(iris, sub = "include.origin = TRUE")
s.label(iris, sub = "include.origin = FALSE", include.origin = FALSE)
```



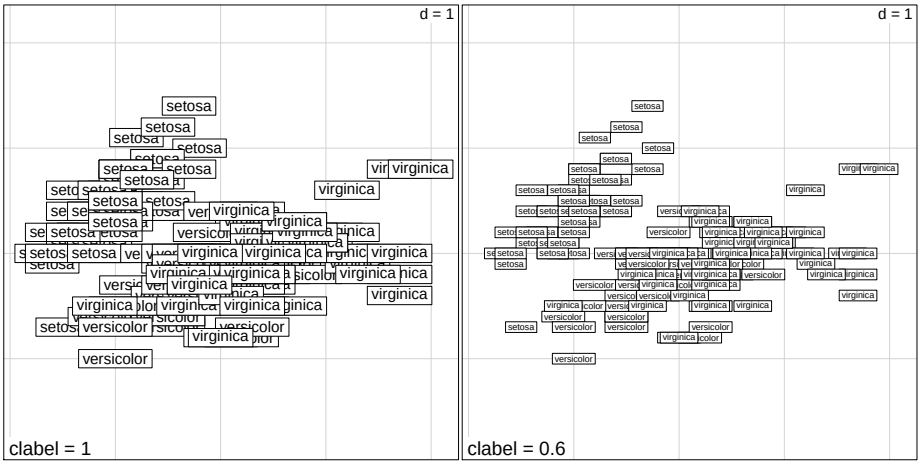
L'argument `label` permet de contrôler les noms utilisés pour les étiquettes. Par défaut, ce sont les noms des lignes du data frame utilisé, mais on peut utiliser autre chose :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, sub = "label = row.names(dfxy)")
s.label(iris, include.origin = FALSE, label = iris$Species, sub = "label = iris$Species")
```



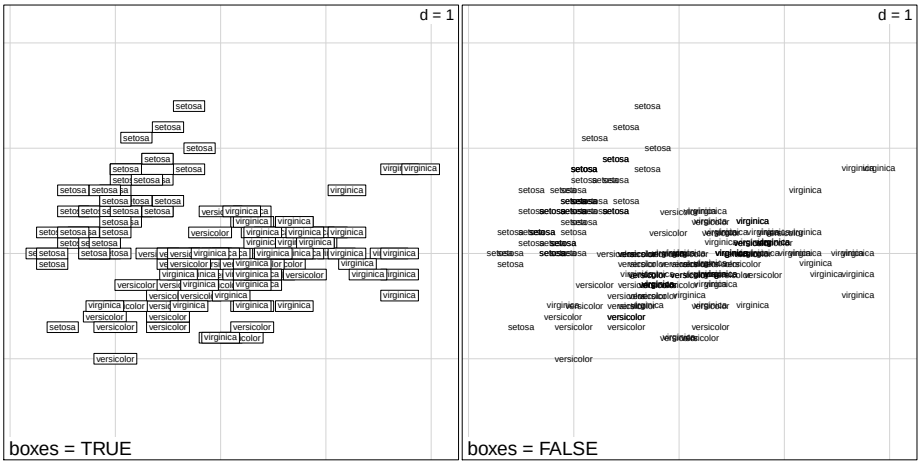
L'argument `clabel` permet de contrôler la taille des étiquettes :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, sub = "clabel = 1")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "clabel = 0.6")
```



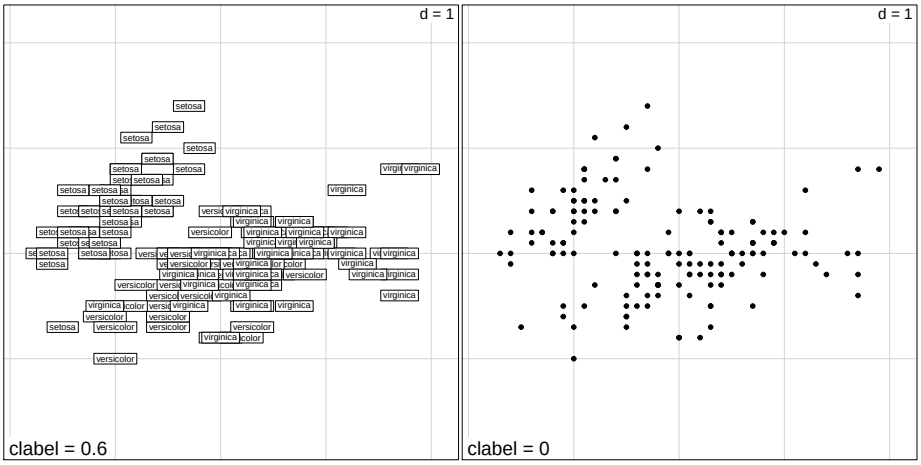
L'argument `boxes` permet de supprimer le cadre rectangulaire autour des étiquettes :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "boxes = TRUE")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, boxes = FALSE, sub = "boxes = FALSE")
```



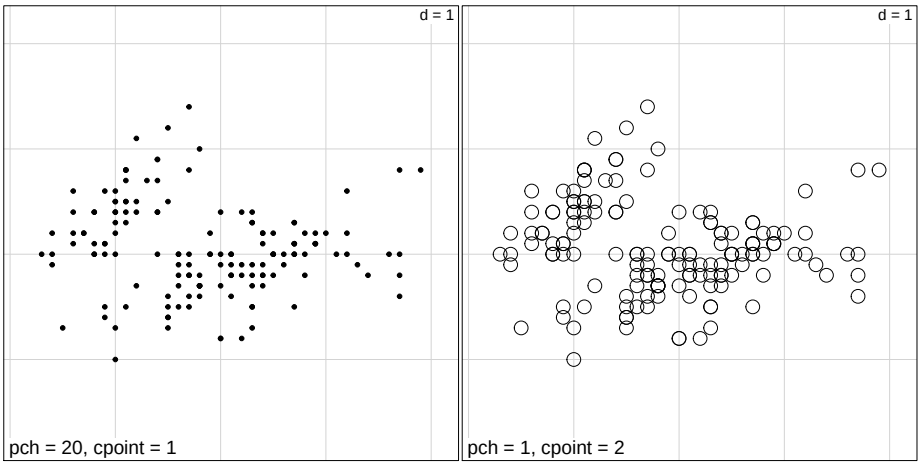
On peut supprimer complètement les étiquettes avec `clabel` = 0. Dans ce cas, ce sont des points qui seront utilisés pour la représentation graphique :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "clabel = 0.6")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0, sub = "clabel = 0")
```



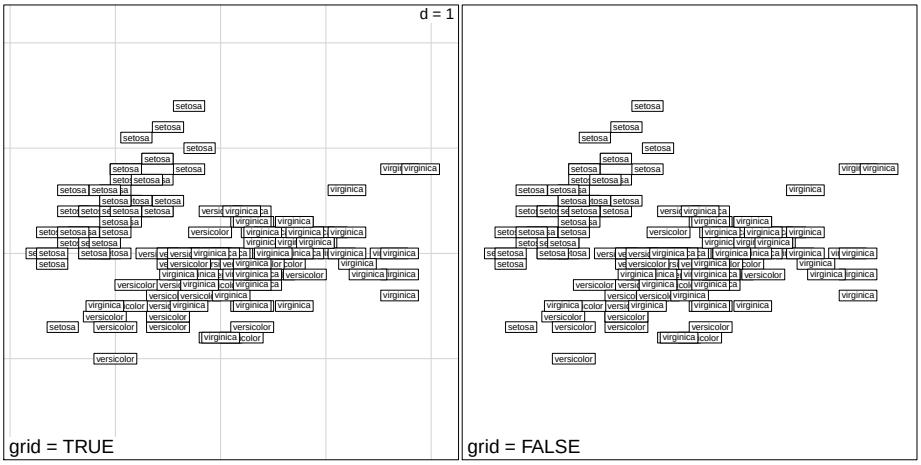
On peut alors contrôler le type et la taille des points avec les arguments `pch` et `cpoint`, respectivement :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0, sub = "pch = 20, cpoint = 1")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0, pch = 1, cpoint = 2, sub = "pch = 1, c
```



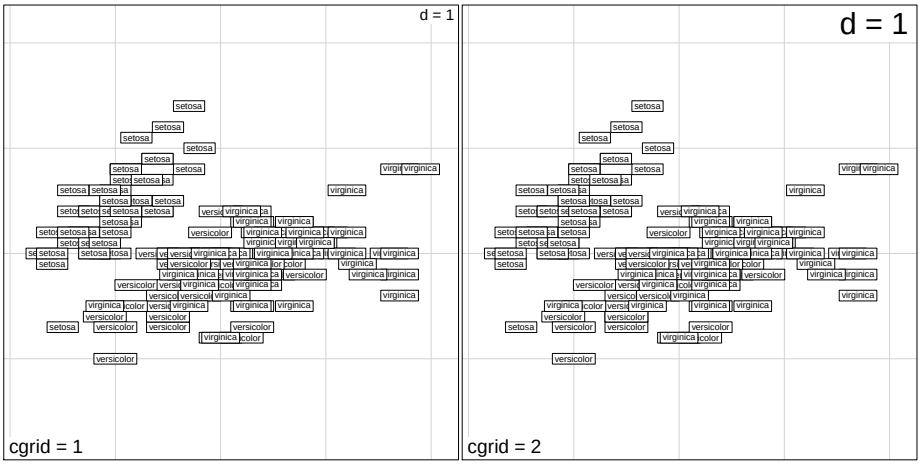
Le paramètre `grid` permet de supprimer le quadrillage :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "grid = TRUE")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, grid = FALSE, sub = "grid = FALSE")
```



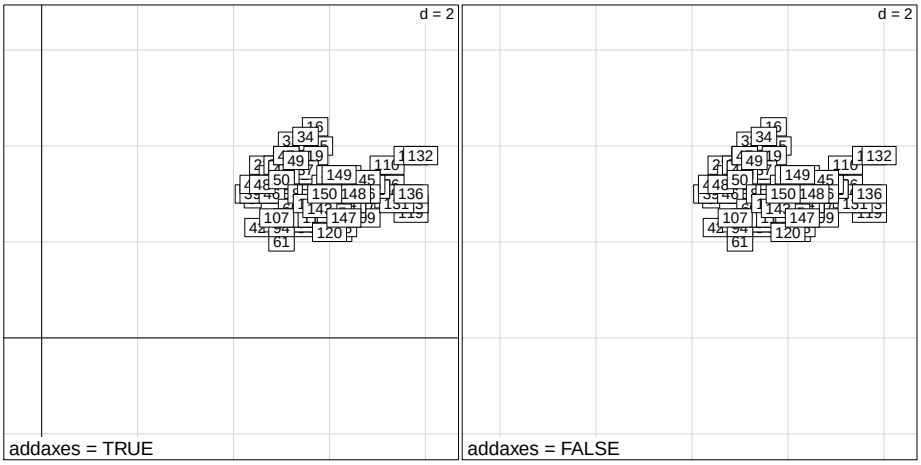
Le paramètre `cgrid` permet de contrôler la taille des caractères utilisés pour l'affichage de la taille de la maille du quadrillage :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "cgrid = 1")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, cgrid = 2, sub = "cgrid = 2")
```



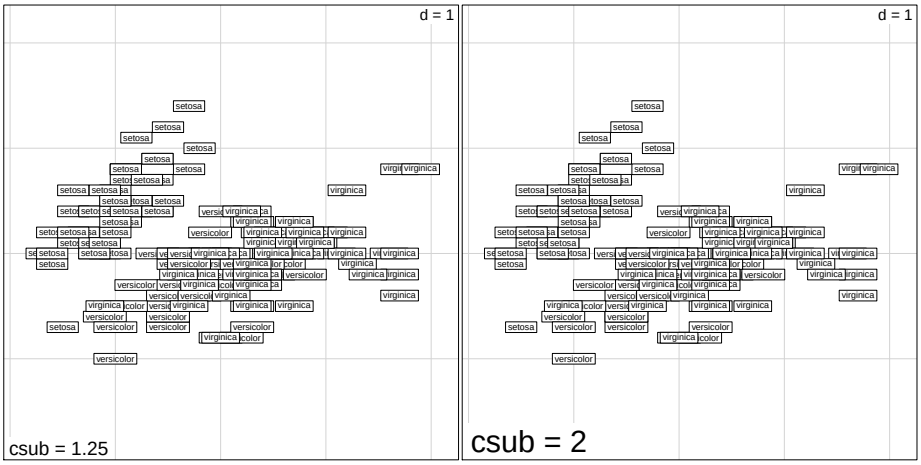
Le paramètre `addaxes` permet de supprimer l'affichage automatique des axes :

```
par(mfrow=c(1,2))
s.label(iris, sub = "addaxes = TRUE")
s.label(iris, addaxes = FALSE, sub = "addaxes = FALSE")
```



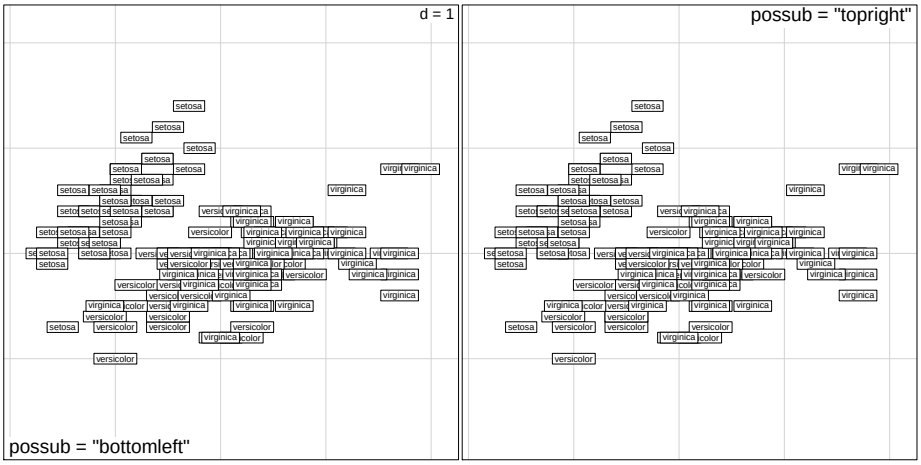
Le paramètre `csub` permet de contrôler la taille des caractères utilisés pour la légende :

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "csub = 1.25")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, csub = 2, sub = "csub = 2")
```



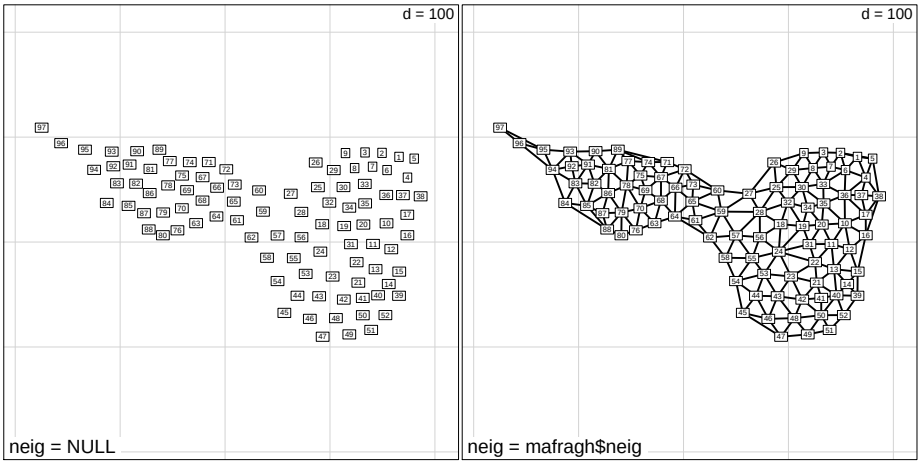
Le paramètre `possub` permet de contrôler la position de la légende, les valeurs possibles sont `"topleft"`, `"topright"`, `"bottomleft"`, `"bottomright"`.

```
par(mfrow=c(1,2))
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, sub = "possub = \"bottomleft\"")
s.label(iris, include.origin = FALSE, label = iris$Species, clabel = 0.6, possub = "topright", sub = "possub =
```



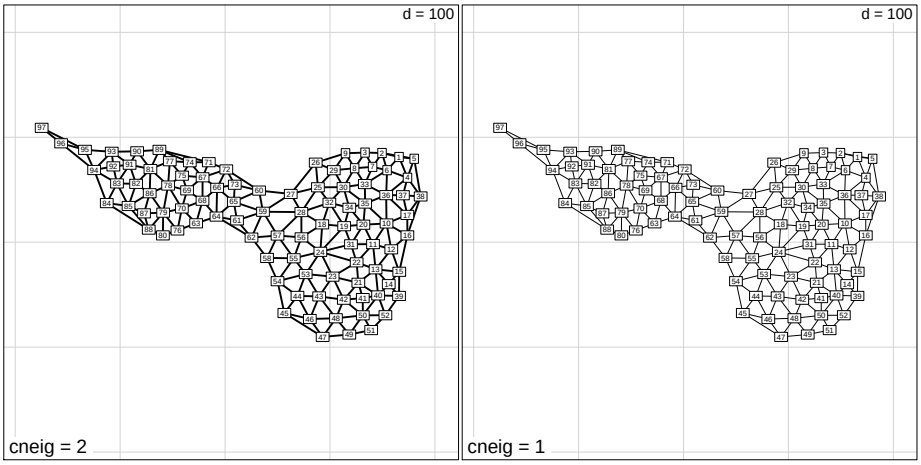
L'argument `neig` permet d'ajouter un graphe de voisinage :

```
par(mfrow=c(1,2))
data(mafragh)
s.label(mafragh$xy,include.origin=FALSE, clab = 0.5, addax=FALSE, sub = "neig = NULL")
s.label(mafragh$xy,include.origin=FALSE, clab = 0.5, addax=FALSE, neig=mafragh$neig, sub = "neig = mafragh$nei
```



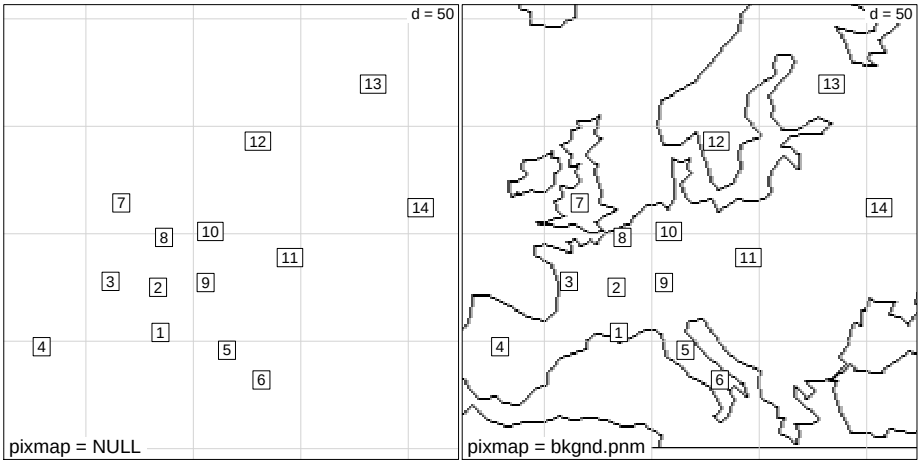
L'argument `cneig` contrôle l'épaisseur des lignes utilisées pour tracer le graphe de voisinage :

```
par(mfrow=c(1,2))
data(mafragh)
s.label(mafragh$xy,include.origin=FALSE, clab = 0.5, addax=FALSE, neig=mafragh$neig, sub = "cneig = 2")
s.label(mafragh$xy,include.origin=FALSE, clab = 0.5, addax=FALSE, neig=mafragh$neig, cneig = 1, sub = "cneig =
```



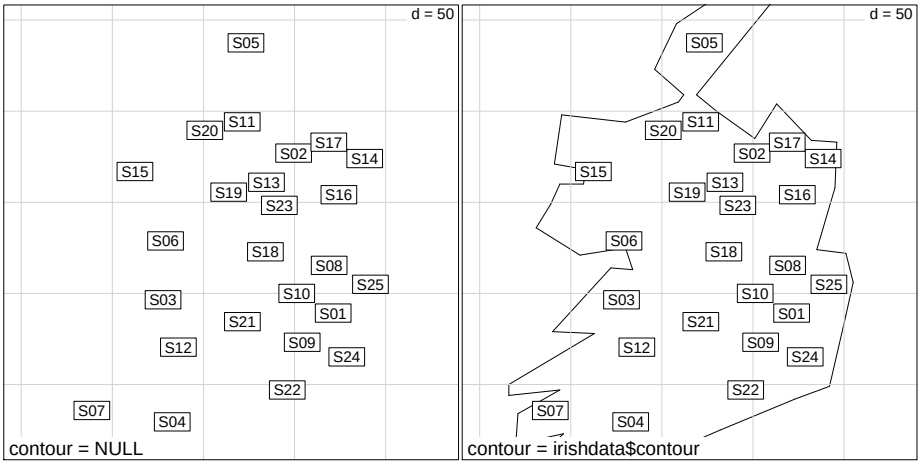
L'argument `pixmap` permet d'inclure en fond de carte un objet de classe `pixmap` :

```
library(pixmap)
data(sarcelles)
bkgnd.pnm <- read.pnm(system.file("pictures/sarcelles.pnm", package = "ade4"))
par(mfrow = c(1,2))
s.label(sarcelles$xy, include.origin = FALSE, addaxes = FALSE, sub = "pixmap = NULL")
s.label(sarcelles$xy, include.origin = FALSE, addaxes = FALSE, pixmap = bkgnd.pnm,
sub = "pixmap = bkgnd.pnm")
```



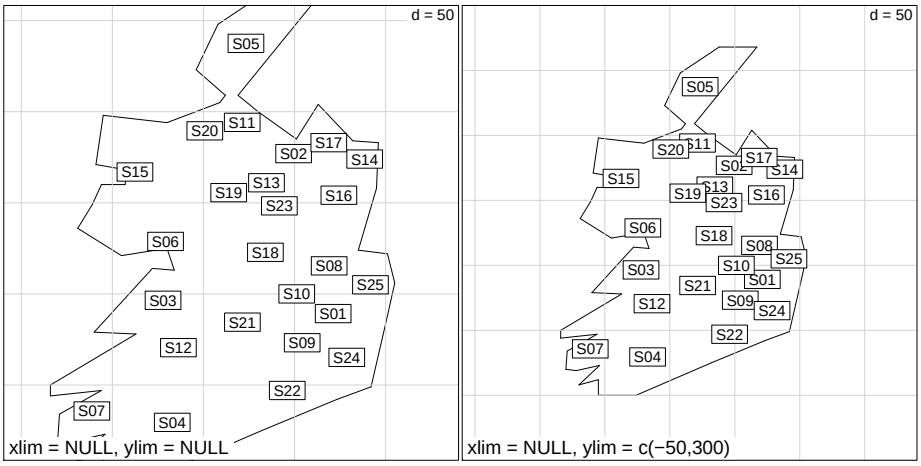
Le paramètre `contour` permet de tracer un contour en fond de carte. Le contour est donné par un `data.frame` à 4 colonnes, chaque ligne donne un trait (x_1, y_1, x_2, y_2).

```
data(irishdata)
par(mfrow = c(1,2))
s.label(irishdata$xy, inc = FALSE, addaxes = FALSE, sub = "contour = NULL")
s.label(irishdata$xy, inc = FALSE, addaxes = FALSE, contour = irishdata$contour, sub = "contour = irishdata$co
```



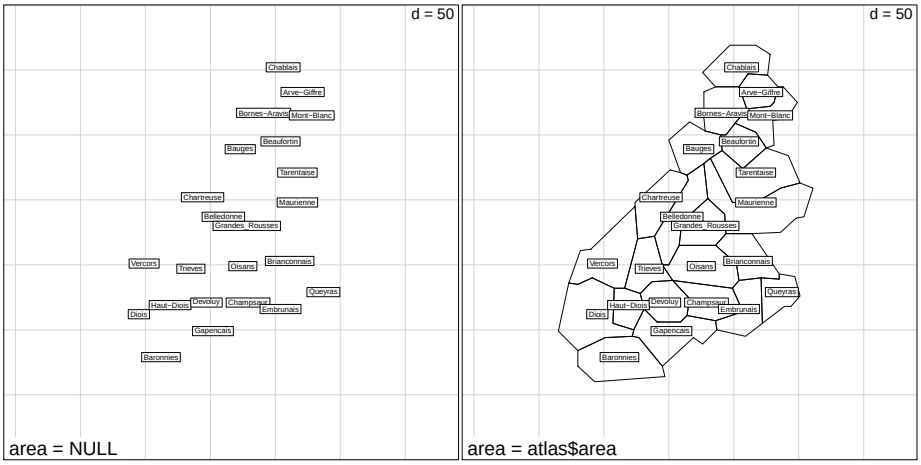
Les paramètres `xlim` et `ylim` permettent alors au contour de ne pas déborder du cadre :

```
par(mfrow = c(1,2))
s.label(irishdata$xy, inc = FALSE, addaxes = FALSE, contour = irishdata$contour, sub = "xlim = NULL, ylim = NU
s.label(irishdata$xy, inc = FALSE, addaxes = FALSE, contour = irishdata$contour,
  ylim = c(-50,300),
  sub = "xlim = NULL, ylim = c(-50,300)")
```



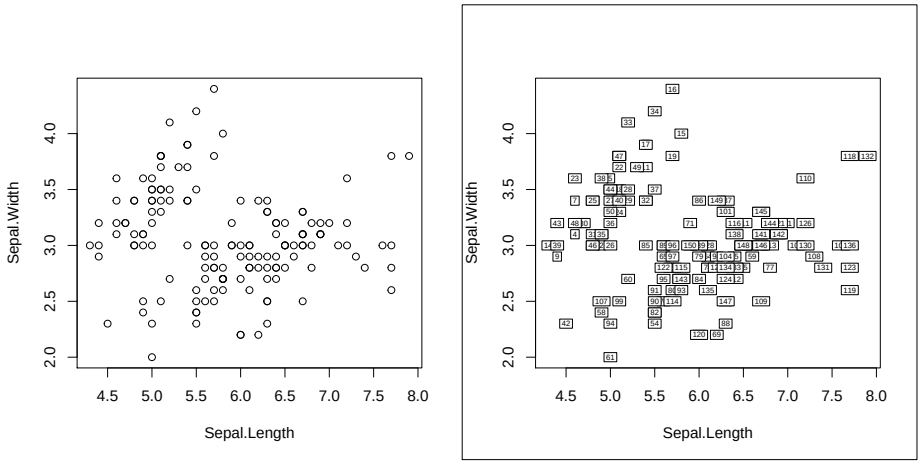
Le paramètre `area` permet de tracer un ensemble d'unités surfaciques en contour :

```
data(atlas)
par(mfrow=c(1,2))
s.label(atlas$xy, label = atlas$names.district, inc = F,addax=F, ylim = c(-50, 300), clab = 0.5, sub = "area =
s.label(atlas$xy, label = atlas$names.district, inc = F,addax=F, ylim = c(-50, 300), clab = 0.5, area = atlas$
```



L'argument `add.plot` permet d'ajouter le graphique à un graphique pré-existant :

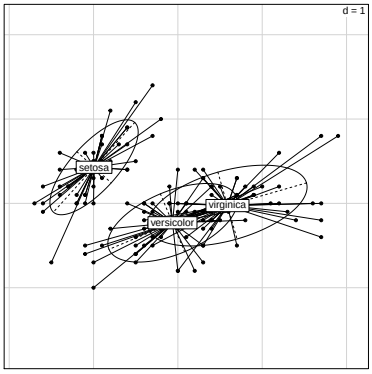
```
par(mfrow=c(1,2))
plot(iris[,1:2])
plot(iris[,1:2])
s.label(iris, clab = 0.5, add.plot = TRUE)
```



2.3 `s.class()`

Cette fonction permet le tracé de nuages de points avec représentation de classes de points, par exemple :

```
s.class(iris, iris$Species, include.origin = FALSE)
```



Nous ne reprendrons pas ici les arguments déjà envisagés avec la fonction `s.label()` pour nous focaliser sur ceux qui sont spécifiques à `s.class()`.

```
 args(s.class)
```

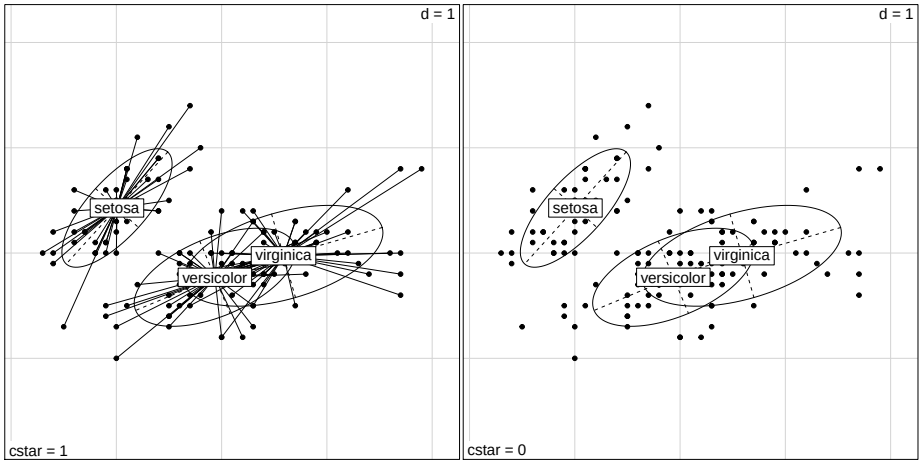
```
function (dfxy, fac, wt = rep(1, length(fac)), xax = 1, yax = 2,
    cstar = 1, cellipse = 1.5, axesell = TRUE, label = levels(fac),
    clabel = 1, cpoint = 1, pch = 20, col = rep(1, length(levels(fac))),
    xlim = NULL, ylim = NULL, grid = TRUE, addaxes = TRUE, origin = c(0,
        0), include.origin = TRUE, sub = "", csub = 1, possub = "bottomleft",
    cgrid = 1, pixmap = NULL, contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

```
 args.s.class <- names(as.list(args(s.class)))
 args.s.label <- names(as.list(args(s.label)))
 args.s.class[!args.s.class %in% args.s.label]
```

```
[1] "fac"      "wt"       "cstar"    "cellipse" "axesell"  "col"
```

L'argument `fac`, obligatoire, est un facteur distribuant les lignes de `dfxy` en classes. L'argument `wt` donne la pondération des points de `dfxy` utilisée pour calculer les moyennes (centre des étoiles) et ellipses de dispersion. Par défaut, on utilise une pondération uniforme. Le paramètre `cstar` contrôle la longueur des segments tracés :

```
 par(mfrow=c(1,2))
 s.class(iris, iris$Species, include.origin = FALSE, sub = "cstar = 1")
 s.class(iris, iris$Species, include.origin = FALSE, cstar = 0, sub = "cstar = 0")
```



L'argument `cellipse` définit la longueur des axes des ellipses. Les ellipses d'ade4 sont des résumés graphiques et non des régions de confiance. Il n'y a pas de règle

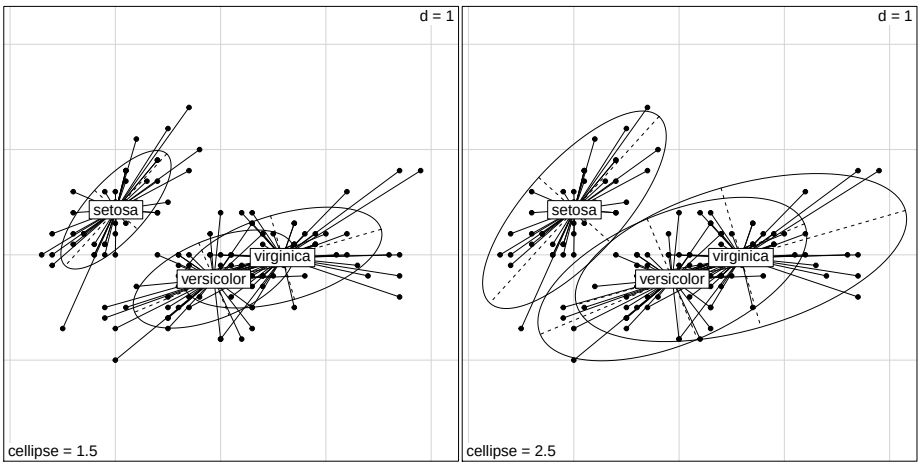
pour définir la taille de l'ellipse. On peut simplement dire que si le nuage est un échantillon aléatoire simple d'une loi normale bivariée, la probabilité p d'être dans l'ellipse de taille k est :

$$p = 1 - \mathrm{e}^{-\frac{k^2}{2}}$$

```
1-exp(-0.5*(1.5)^2)
[1] 0.6753475
1-exp(-0.5*(2.5)^2)
[1] 0.9560631
```

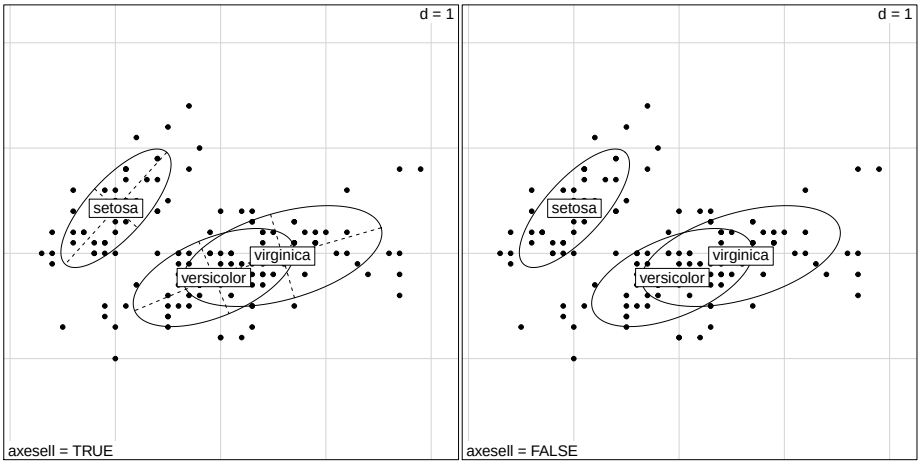
Environ 67% des points sont dans l'ellipse $k = 1.5$ et 95% dans l'ellipse $k = 2.5$.

```
par(mfrow=c(1,2))
s.class(iris, iris$Species, include.origin = FALSE, sub = "cellipse = 1.5")
s.class(iris, iris$Species, include.origin = FALSE, cellipse = 2.5, sub = "cellipse = 2.5")
```



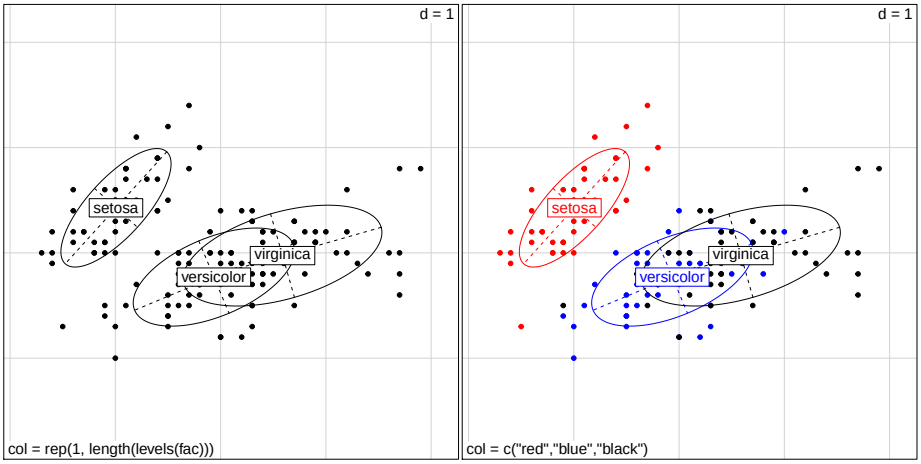
L'argument `axesell` permet de supprimer le tracé des axes des ellipses :

```
par(mfrow=c(1,2))
s.class(iris, iris$Species, include.origin = FALSE, cstar = 0, sub = "axesell = TRUE")
s.class(iris, iris$Species, include.origin = FALSE, cstar = 0, axesell = FALSE, sub = "axesell = FALSE")
```



L'argument `col` permet de donner une couleur pour les différentes classes de points :

```
par(mfrow=c(1,2))
s.class(iris, iris$Species, include.origin = FALSE, cstar = 0, sub = "col = rep(1, length(levels(fac)))")
s.class(iris, iris$Species, include.origin = FALSE, cstar = 0,
  col = c("red","blue","black"), sub = "col = c(\"red\",\"blue\",\"black\")")
```



2.4 `s.chull()`

```
args(s.chull)
```

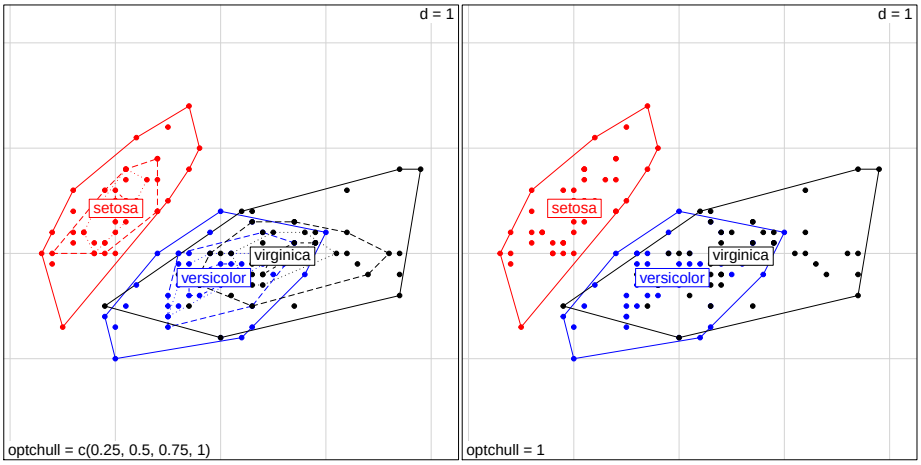
```
function (dfxy, fac, xax = 1, yax = 2, optchull = c(0.25, 0.5,
    0.75, 1), label = levels(fac), clabel = 1, cpoint = 0, col = rep(1,
    length(levels(fac))), xlim = NULL, ylim = NULL, grid = TRUE,
    addaxes = TRUE, origin = c(0, 0), include.origin = TRUE,
    sub = "", csub = 1, possub = "bottomleft", cgrid = 1, pixmap = NULL,
    contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

```
args.s.chull <- names(as.list(args(s.chull)))
args.s.chull[!args.s.chull %in% c(args.s.label, args.s.class)]
```

```
[1] "optchull"
```

`s.chull()` permet de tracer les polygones de contour pour chacune des classes de points définies par les niveaux du facteur. Ils sont obtenus par peeling. Le contour au niveau 1, si présent dans `optchull`, contient tous les points. Le contour de niveau 0.75, si présent dans `optchull`, contient au plus 75 % des points (c'est le premier contour défini par peeling qui contient au plus 75 % des points). Le contour de niveau 0.50, si présent dans `optchull`, est le premier contour défini par peeling qui contient au plus 50 % des points. Le contour de niveau 0.25, si présent dans `optchull`, est le premier contour défini par peeling qui contient au plus 25 % des points. Les contours contenant moins de 3 points ne sont pas tracés.

```
par(mfrow=c(1,2))
s.chull(iris, iris$Species, include.origin = FALSE, col = c("red", "blue", "black"), cpoint = 1,
   sub = "optchull = c(0.25, 0.5, 0.75, 1)")
s.chull(iris, iris$Species, include.origin = FALSE, col = c("red", "blue", "black"), cpoint = 1, optchull = 1,
```



Pour lisser les polygones représentant les groupes, on peut utiliser la fonction `xspline()` et jouer sur son paramètre `shape`. Le code suivant a été utilisé pour produire la figure 2 illustrant l'effet de ce paramètre.

```
s.potatoe <- function (dfxy, fac, xax = 1, yax = 2, col.border = rep(1, length(levels(fac))), col.fill = rep(1
    dfxy <- data.frame(dfxy)
    opar <- par(mar = par("mar"))
    par(mar = c(0.1, 0.1, 0.1, 0.1))
    on.exit(par(opar))
    x <- dfxy[, xax]
    y <- dfxy[, yax]
    for(f in levels(fac)){
             xx <- x[fac == f]
             yy <- y[fac == f]
      xc <- chull(xx, yy)
      qui <- which(levels(fac) == f)
      border <- col.border[qui]
      col <- col.fill[qui]
      xspline(xx[xc], yy[xc], shape = shape, open = open,
        border = border, col = col, ...)
    }
}
col.border <- c("red","green","blue")
col.fill <- c(rgb(1,0,0,0.3), rgb(0,1,0,0.3), rgb(0,0,1,0.3))
par(mfrow = c(3, 3))
bkg <- function(...) s.class(iris, iris$Species, inc = FALSE, cell = 0, adda = FALSE, cst = 0, col =col.border
for(shape in seq(-1, 1, length = 9)){
  bkg(sub = paste("shape =", round(shape, 2)), csub = 2.5)
  s.potatoe(iris, iris$Species, col.border = col.border, shape = shape, col.fill = col.fill)
}
```

2.5 `s.arrow()`

```
args(s.arrow)
```

```
function (dfxy, xax = 1, yax = 2, label = row.names(dfxy), clabel = 1,
    pch = 20, cpoint = 0, boxes = TRUE, edge = TRUE, origin = c(0,
        0), xlim = NULL, ylim = NULL, grid = TRUE, addaxes = TRUE,
    cgrid = 1, sub = "", csub = 1.25, possub = "bottomleft",
    pixmap = NULL, contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

```
args.s.arrow <- names(as.list(args(s.arrow)))
args.s.arrow[!args.s.arrow %in% c(args.s.label, args.s.class, args.s.chull)]
```

```
[1] "edge"
```

`s.arrow()` trace des vecteurs de l'origine aux points, l'argument `edge` permet de neutraliser le tracé de la flèche en fin de segment :

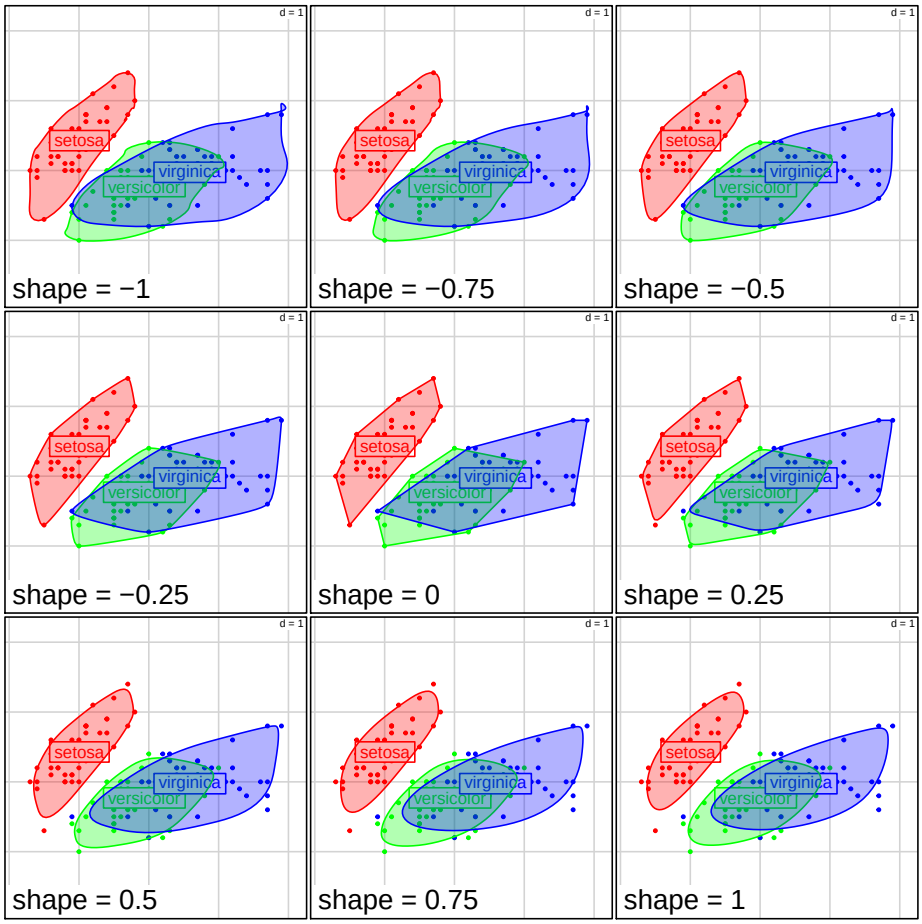
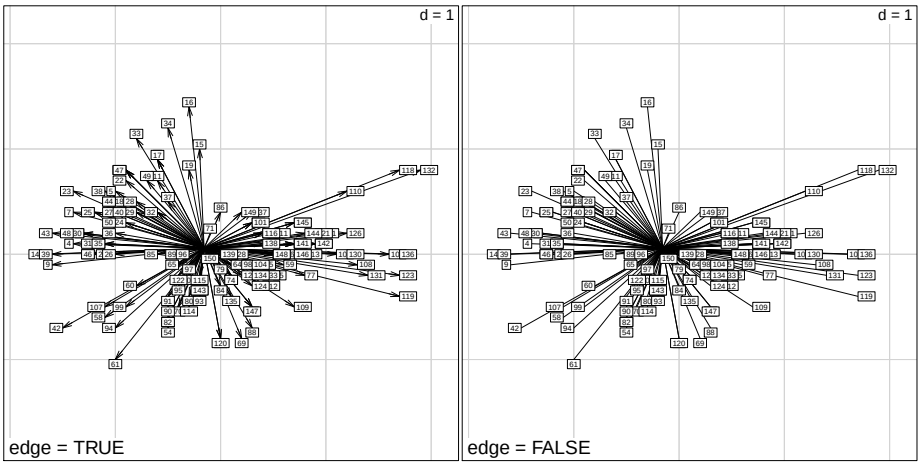


FIGURE 2 – Démonstration de l'effet du paramètre `shape` de la fonction standard `xspline()` sur l'aspect des patatoïdes représentant des classes de points. Avec `shape = 0` on retrouve au centre le rendu de `s.chull(optchull = 1)`. Attention : l'utilisation ici de couleurs semi-transparentes pour le remplissage des patatoïdes n'est pas supportée par tous les périphériques graphiques.

```
par(mfrow=c(1, 2))
s.arrow(iris, origin = colMeans(iris[,1:2]), clab = 0.5, sub = "edge = TRUE")
s.arrow(iris, origin = colMeans(iris[,1:2]), clab = 0.5, edge = FALSE,
sub = "edge = FALSE")
```



2.6 s.hist()

```
args(s.hist)
```

```
function (dfxy, xax = 1, yax = 2, cgrid = 1, cbreaks = 2, adjust = 1,
    ...)
NULL
```

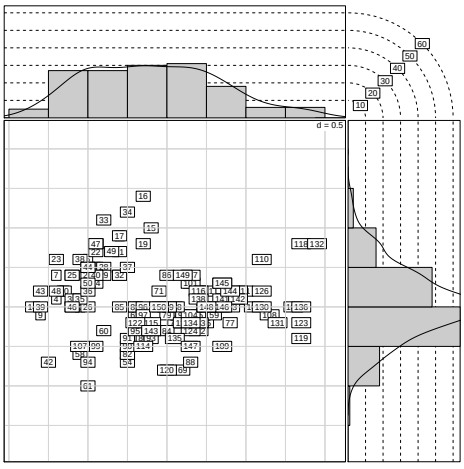
```
args.s.hist <- names(as.list(args(s.hist)))
args.s.hist[!args.s.hist %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow)]
```

```
[1] "cbreaks" "adjust"  "..."
```

Cette fonction représente un nuage de points avec les distributions marginales :

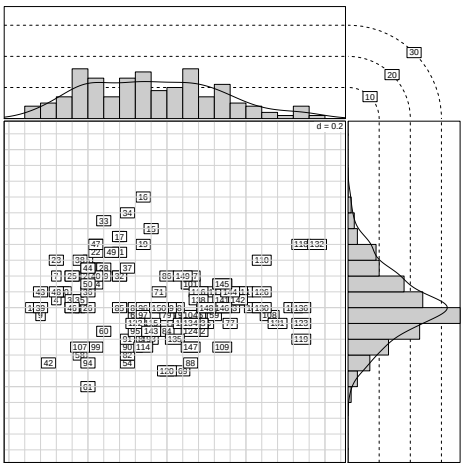
```
s.hist(iris, inc = FALSE)
```

```
[1] 10 20 30 40 50 60
```



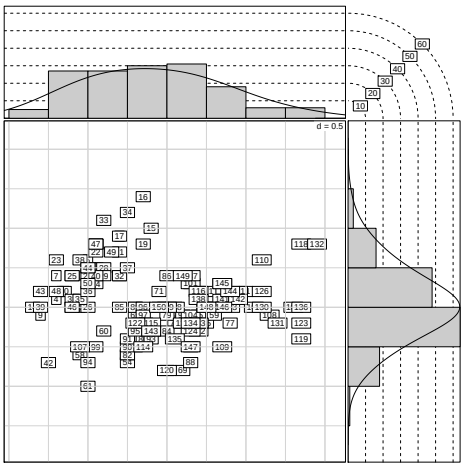
Cette fonction faisant appel à `layout()`, on ne peut pas l'utiliser simultanément avec un appel à `par(mfrow = c(1, 2))`. L'argument `cbreaks` permet de contrôler l'amplitude des classes pour le tracé des histogrammes :

```
s.hist(iris, inc = FALSE, cbreaks = 5)
```
```
[1] 10 20 30
```



L'argument `adjust` est transmis à la fonction `density()` :

```
s.hist(iris, inc = FALSE, adjust = 2)
```
```
[1] 10 20 30 40 50 60
```



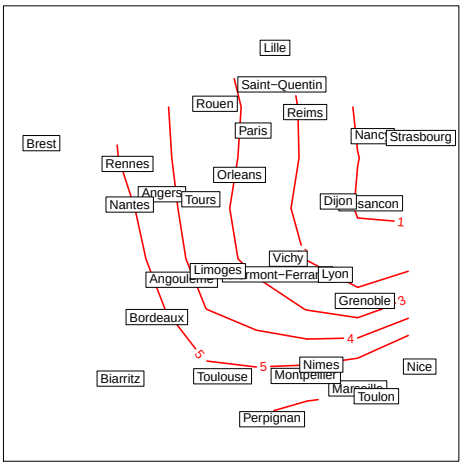
Les arguments supplémentaires sont pris en charge par la fonction `s.label()`.

2.7 `s.image()`

```
args(s.image)
```
```
function (dfxy, z, xax = 1, yax = 2, span = 0.5, xlim = NULL,
    ylim = NULL, kgrid = 2, scale = TRUE, grid = FALSE, addaxes = FALSE,
    cgrid = 0, include.origin = FALSE, origin = c(0, 0), sub = "",
    csub = 1, possub = "topleft", neig = NULL, cneig = 1, image.plot = TRUE,
    contour.plot = TRUE, pixmap = NULL, contour = NULL, area = NULL,
    add.plot = FALSE)
NULL
```
```
args.s.image <- names(as.list(args(s.image)))
args.s.image[!args.s.image %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist)]
```
```
[1] "z"            "span"         "kgrid"        "scale"        "image.plot"
[6] "contour.plot"
```

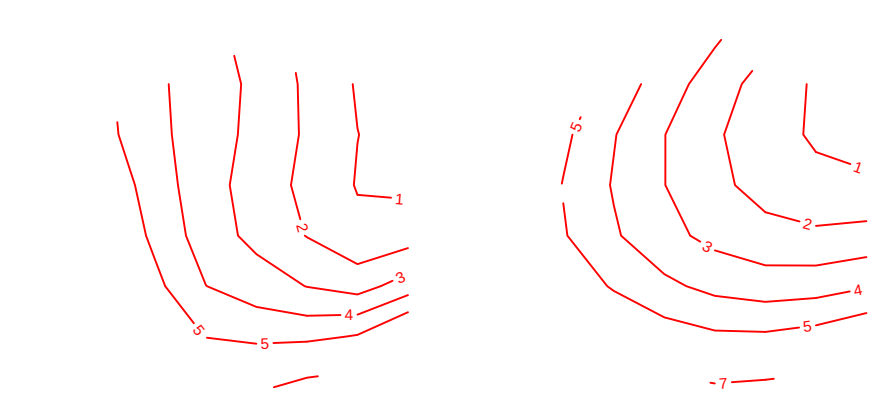
La fonction `s.image()` permet de faire des représentations en courbes de niveau d'une variable spatialisée. Par exemple, la température moyenne pour 30 villes françaises au mois de janvier :

```
data(t3012)
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE)
s.label(t3012$xy, add.plot = TRUE)
```



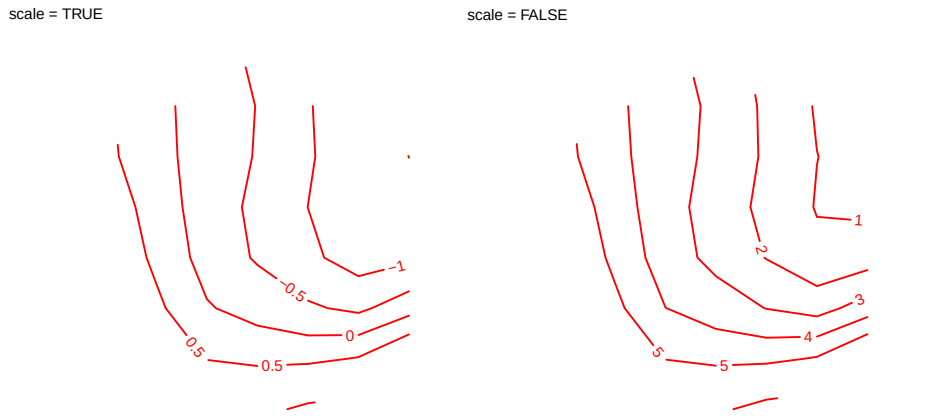
L'argument `span` contrôle l'intensité du lissage :

```
par(mfrow = c(1,2))
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, scale = FALSE, image.plot = FALSE, sub = "sp
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, scale = FALSE, image.plot = FALSE, span = 2,
sub = "span = 2")
```



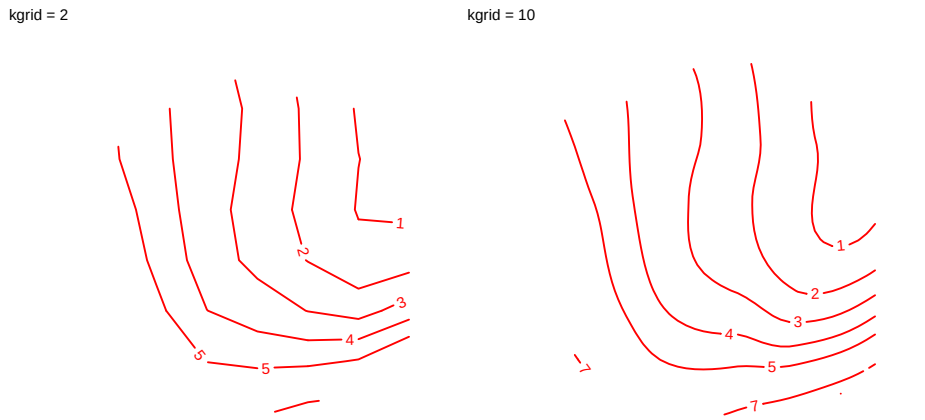
L'argument `scale` contôle le centrage et la réduction des données :

```
par(mfrow = c(1,2))
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, sub = "scale = TRUE")
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE,
sub = "scale = FALSE")
```



L'argument `kgrid` contrôle le nombre de points utilisé localement pour estimer les courbes de niveau :

```
par(mfrow = c(1,2))
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE, sub = "kg
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE,
kgrid = 10, sub = "kgrid = 10")
```



L'argument `image.plot` contrôle si une représentation en niveaux de gris doit être tracée :

```
par(mfrow = c(1,2))
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = TRUE, scale = FALSE, kgrid = 10
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE,
kgrid = 10, sub = "image.plot = FALSE")
```

image.plot = TRUE

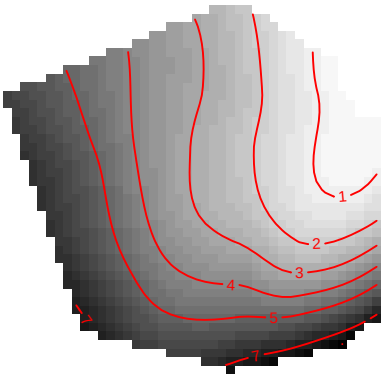
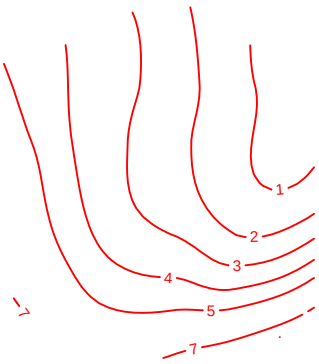


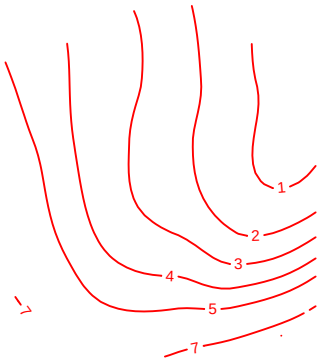
image.plot = FALSE



L'argument `contour.plot` contrôle si les courbes de niveau doivent être tracées :

```
par(mfrow = c(1,2))
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE, kgrid = 1
s.image(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, image.plot = FALSE, scale = FALSE,
kgrid = 10, contour.plot = FALSE, sub = "contour.plot = FALSE")
```

contour.plot = TRUE



contour.plot = FALSE

2.8 `s.kde2d()`

```
args(s.kde2d)
```

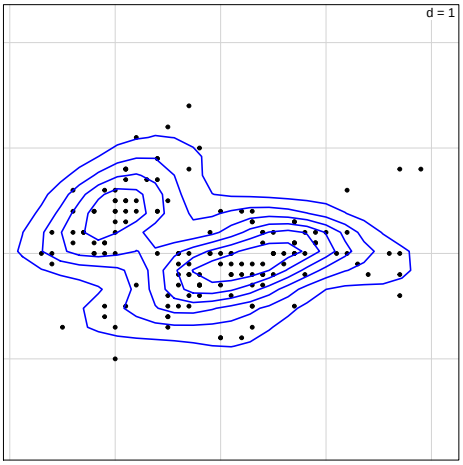
```
function (dfxy, xax = 1, yax = 2, pch = 20, cpoint = 1, neig = NULL,
    cneig = 2, xlim = NULL, ylim = NULL, grid = TRUE, addaxes = TRUE,
    cgrid = 1, include.origin = TRUE, origin = c(0, 0), sub = "",
    csub = 1.25, possub = "bottomleft", pixmap = NULL, contour = NULL,
    area = NULL, add.plot = FALSE)
NULL
```

```
args.s.kde2d <- names(as.list(args(s.kde2d)))
args.s.kde2d[!args.s.kde2d %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, args.s.
```

```
character(0)
```

La fonction `s.kde2d()` n'a pas d'argument que nous n'ayons déjà traité. Elle permet de tracer une estimation de la densité locale des points :

```
library(MASS)
s.kde2d(iris, include.origin=FALSE)
```



2.9 s.logo()

```
args(s.logo)
```

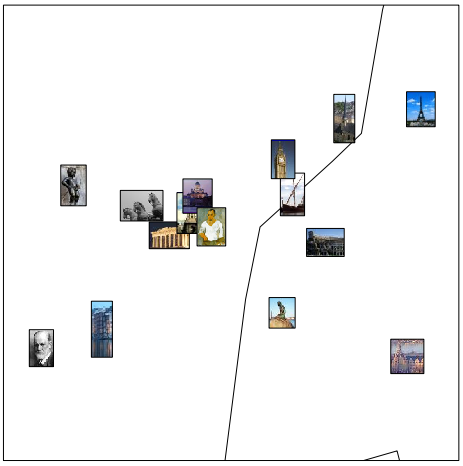
```
function (dfxy, listlogo, klogo = NULL, clogo = 1, rectlogo = TRUE,
    xax = 1, yax = 2, neig = NULL, cneig = 1, xlim = NULL, ylim = NULL,
    grid = TRUE, addaxes = TRUE, cgrid = 1, include.origin = TRUE,
    origin = c(0, 0), sub = "", csub = 1.25, possub = "bottomleft",
    pixmap = NULL, contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

```
args.s.logo <- names(as.list(args(s.logo)))
args.s.logo[!args.s.logo %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, args.s.im
```

```
[1] "listlogo" "klogo"    "clogo"    "rectlogo"
```

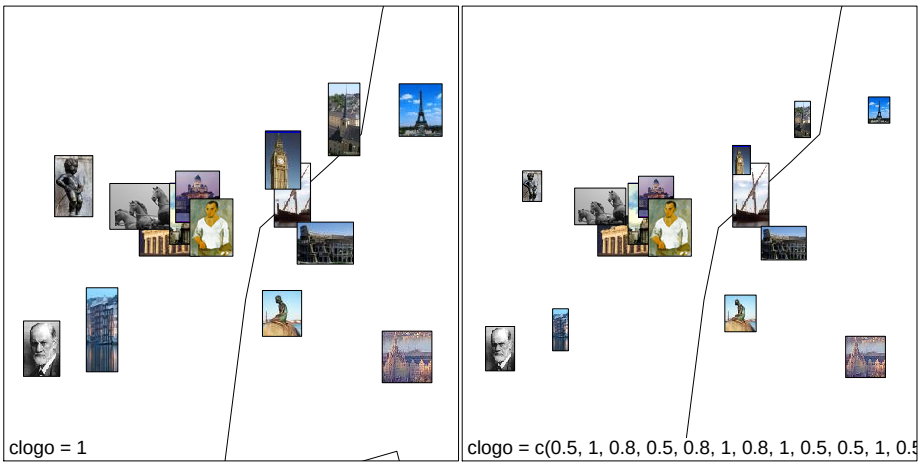
Cette fonction permet de placer les pictogrammes au format pixmap donnés par `listlogo` sur un plan. Ils sont affichés dans l'ordre donné par `klogo`, par exemple :

```
data(capitales)
index <- order(match(names(capitales$logo), tolower(rownames(capitales$df))))
s.logo(capitales$xy, capitales$logo, klogo = index, clogo=0.8, include.origin = FALSE,
grid = FALSE, addaxes = FALSE, area = capitales$area)
```



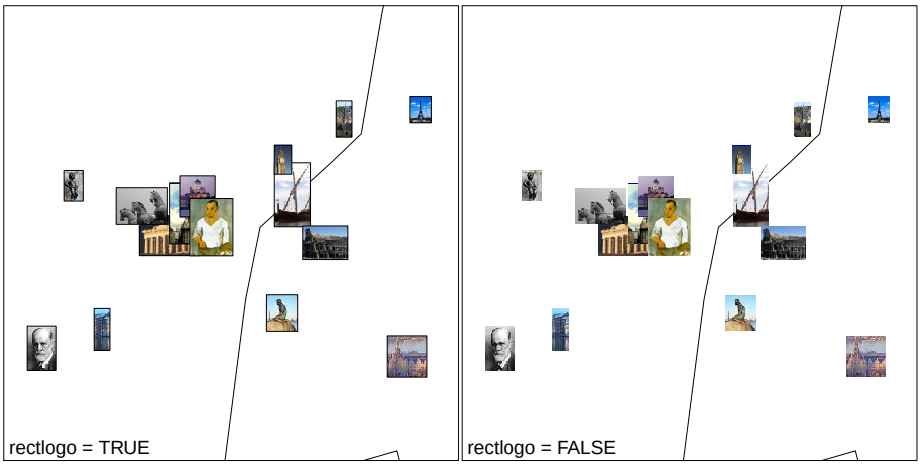
L'argument `clogo` permet de contrôler la taille des pictogrammes :

```
par(mfrow=c(1,2))
s.logo(capitales$xy, capitales$logo, klogo = index, include.origin = FALSE,
grid = FALSE, addaxes = FALSE, area = capitales$area, sub = "clogo = 1")
s.logo(capitales$xy, capitales$logo, klogo = index,
clogo = c(0.5, 1, 0.8, 0.5, 0.8, 1, 0.8, 1, 0.5, 0.5, 1, 0.5, 0.8, 0.8, 0.8), include.origin = FALSE,
grid = FALSE, addaxes = FALSE, area = capitales$area, sub = "clogo = c(0.5, 1, 0.8, 0.5, 0.8, 1, 0.8, 1, 0.5,
```



L'argument `rectlogo` permet de neutraliser le tracé du cadre autour des pictogrammes :

```
par(mfrow=c(1,2))
s.logo(capitales$xy, capitales$logo, klogo = index,
clogo = c(0.5, 1, 0.8, 0.5, 0.8, 1, 0.8, 1, 0.5, 0.5, 1, 0.5, 0.8, 0.8, 0.8), include.origin = FALSE,
grid = FALSE, addaxes = FALSE, area = capitales$area, sub = "rectlogo = TRUE")
s.logo(capitales$xy, capitales$logo, klogo = index,
clogo = c(0.5, 1, 0.8, 0.5, 0.8, 1, 0.8, 1, 0.5, 0.5, 1, 0.5, 0.8, 0.8, 0.8), include.origin = FALSE,
grid = FALSE, addaxes = FALSE, area = capitales$area, sub = "rectlogo = FALSE", rectlogo = FALSE)
```



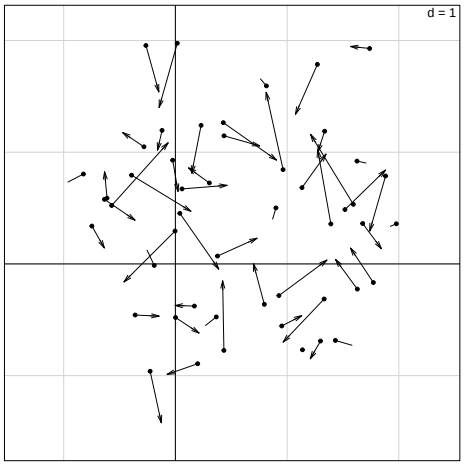
2.10 `s.match()`

```
args(s.match)
function (df1xy, df2xy, xax = 1, yax = 2, pch = 20, cpoint = 1,
    label = row.names(df1xy), clabel = 1, edge = TRUE, xlim = NULL,
    ylim = NULL, grid = TRUE, addaxes = TRUE, cgrid = 1, include.origin = TRUE,
    origin = c(0, 0), sub = "", csub = 1.25, possub = "bottomleft",
    pixmap = NULL, contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

```
args.s.match <- names(as.list(args(s.match)))
args.s.match[!args.s.match %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, args.s.
```
```
[1] "df1xy" "df2xy"
```

Cette fonction ne comporte pas d'argument spécifique à part les deux data frames obligatoires `df1xy` et `df2xy`. Elle permet la représentation de coordonnées appariées :

```
X <- data.frame(x = runif(50,-1,2), y = runif(50,-1,2))
Y <- X + rnorm(100, sd = 0.3)
s.match(X,Y, clab = 0)
```



2.11 s.multinom()

```
args(s.multinom)
```
```
function (dfxy, dfrowprof, translate = FALSE, xax = 1, yax = 2,
    labelcat = row.names(dfxy), clabelcat = 1, cpointcat = if (clabelcat ==
        0) 2 else 0, labelrowprof = row.names(dfrowprof), clabelrowprof = 0.75,
    cpointrowprof = if (clabelrowprof == 0) 2 else 0, pchrowprof = 20,
    coulrowprof = grey(0.8), proba = 0.95, n.sample = apply(dfrowprof,
        1, sum), axesell = TRUE, ...)
NULL
```
```
args.s.multinom <- names(as.list(args(s.multinom)))
args.s.multinom[!args.s.multinom %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, a
```
```
 [1] "dfrowprof"     "translate"     "labelcat"      "clabelcat"     "cpointcat"
 [6] "labelrowprof"  "clabelrowprof" "cpointrowprof" "pchrowprof"    "coulrowprof"
[11] "proba"         "n.sample"
```

Cette fonction comporte de nombreux arguments qui lui sont spécifiques. Elle n'est pas traitée ici car non essentielle dans les graphiques liés aux méthodes exploratoires.

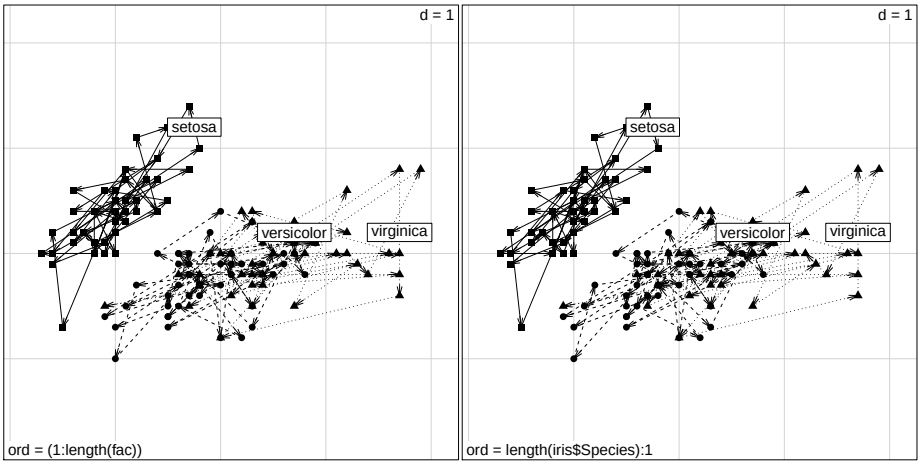
2.12 s.traject()

```
args(s.traject)
```
```
function (dfxy, fac = factor(rep(1, nrow(dfxy))), ord = (1:length(fac)),
    xax = 1, yax = 2, label = levels(fac), clabel = 1, cpoint = 1,
    pch = 20, xlim = NULL, ylim = NULL, grid = TRUE, addaxes = TRUE,
    edge = TRUE, origin = c(0, 0), include.origin = TRUE, sub = "",
    csub = 1, possub = "bottomleft", cgrid = 1, pixmap = NULL,
    contour = NULL, area = NULL, add.plot = FALSE)
NULL
```

```
 args.s.traject <- names(as.list(args(s.traject)))
 args.s.traject[!args.s.traject %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, arg
[1] "ord"
```

Cette fonction permet la représentation de trajectoires, l'argument `ord` contrôle l'odre de tracé des vecteurs.

```
 par(mfrow=c(1,2))
 s.traject(iris, iris$Species, inc = FALSE, sub = "ord = (1:length(fac))")
 s.traject(iris, iris$Species, inc = FALSE, ord = length(iris$Species):1, sub = "ord = length(iris$Species):1")
```

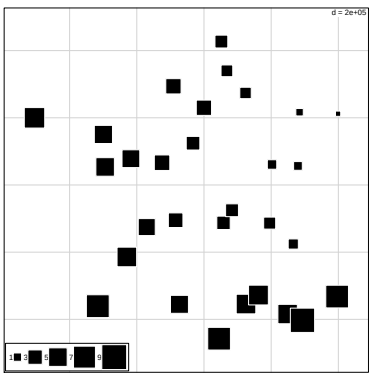


2.13 `s.value()`

```
 args(s.value)
function (dfxy, z, xax = 1, yax = 2, method = c("squaresize",
    "greylevel"), zmax = NULL, csize = 1, cpoint = 0, pch = 20,
    clegend = 0.75, neig = NULL, cneig = 1, xlim = NULL, ylim = NULL,
    grid = TRUE, addaxes = TRUE, cgrid = 0.75, include.origin = TRUE,
    origin = c(0, 0), sub = "", csub = 1, possub = "topleft",
    pixmap = NULL, contour = NULL, area = NULL, add.plot = FALSE)
NULL
 args.s.value <- names(as.list(args(s.value)))
 args.s.value[!args.s.value %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, args.s.
[1] "method"  "zmax"    "csize"   "clegend"
```

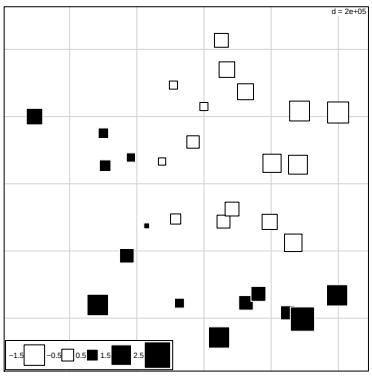
Cette fonction permet de représenter une valeur sur un nuage de points, par exemple pour la température moyenne de 30 villes françaises au mois de janvier :

```
 s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE)
```



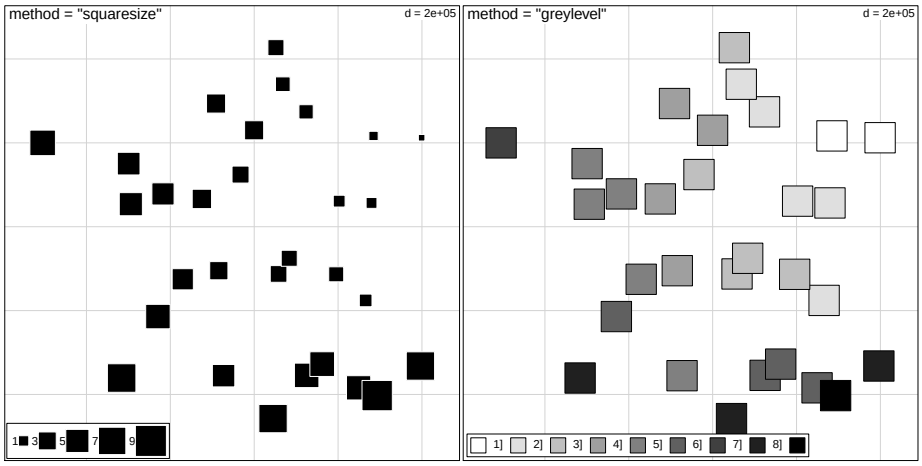
Les valeurs positives sont représentées par des carrés noirs et les valeurs négatives par des carrés blancs :

```
s.value(t3012$xy, scale(t3012$temp[,"Jan"]/10), contour = t3012$contour, inc = FALSE)
```



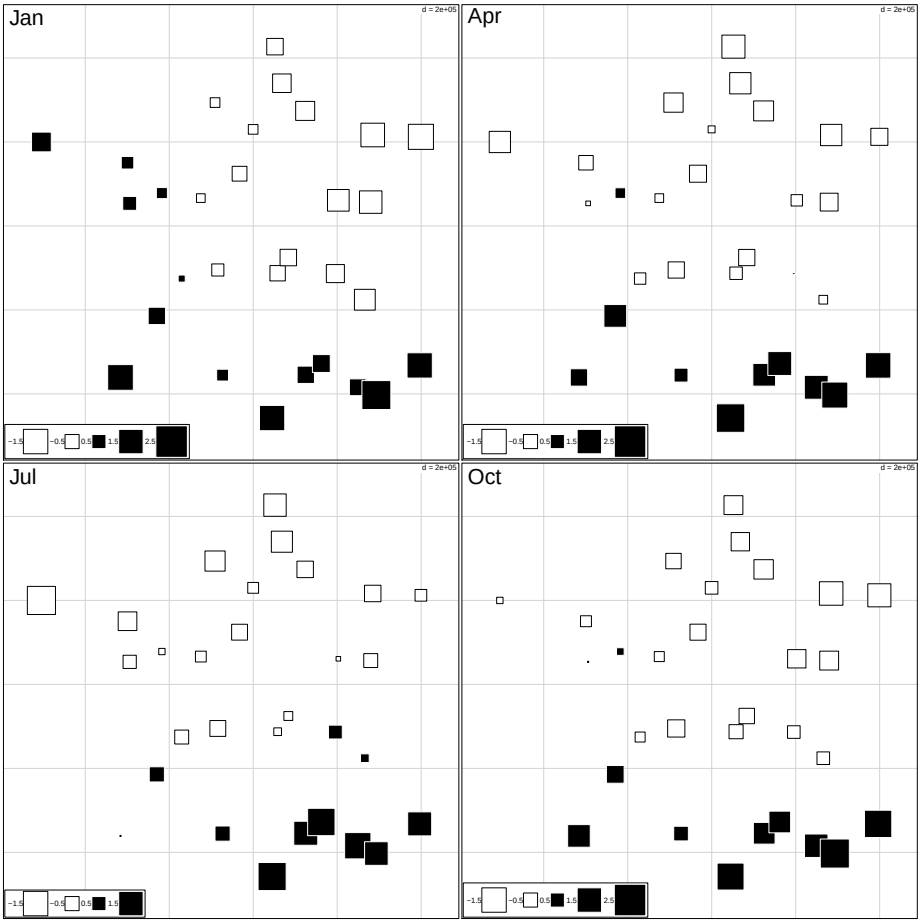
L'argument `method` permet de choisir entre des symboles de taille proportionelle ou des niveaux de gris :

```
par(mfrow=c(1,2))
s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE, sub = "method = \"squaresize\""
s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE, method = "greylevel", sub = "me
```



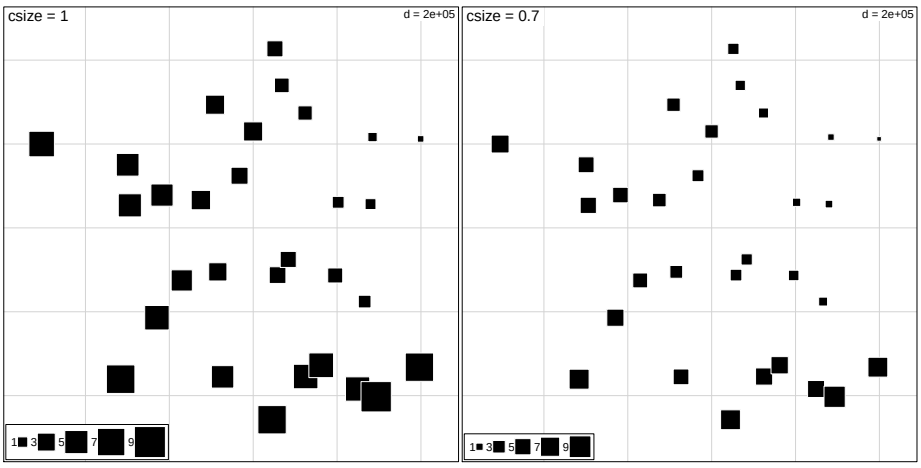
L'argument `zmax` contrôle la taille maximum des symboles, il permet ainsi d'utiliser une échelle commune à plusieurs graphiques :

```
par(mfrow=c(2,2))
z <- scale(t3012$temp)
for(i in seq(1,12,by=3))
s.value(t3012$xy, z[,i], contour = t3012$contour, inc = FALSE, zmax = max(abs(z)), sub = colnames(t3012$temp)[
```



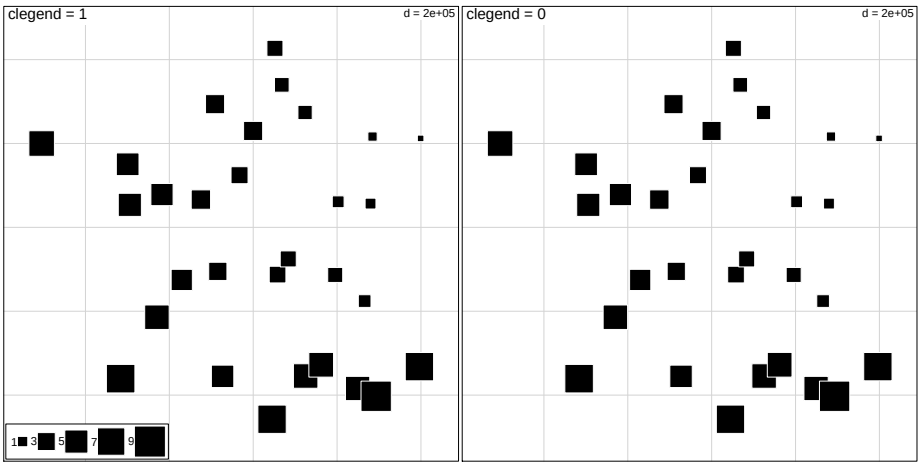
L'argument `csize` permet de contrôler la taille des symboles :

```
par(mfrow=c(1,2))
s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE, sub = "csize = 1")
s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE, csize = 0.7, sub = "csize = 0.7
```



L'argument `clegend` contrôle la taille de la légende :

```
par(mfrow=c(1,2))
s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE, sub = "clegend = 1")
s.value(t3012$xy, t3012$temp[,"Jan"]/10, contour = t3012$contour, inc = FALSE, clegend = 0, sub = "clegend = 0
```



2.14 s.distri()

```
args(s.distri)
```

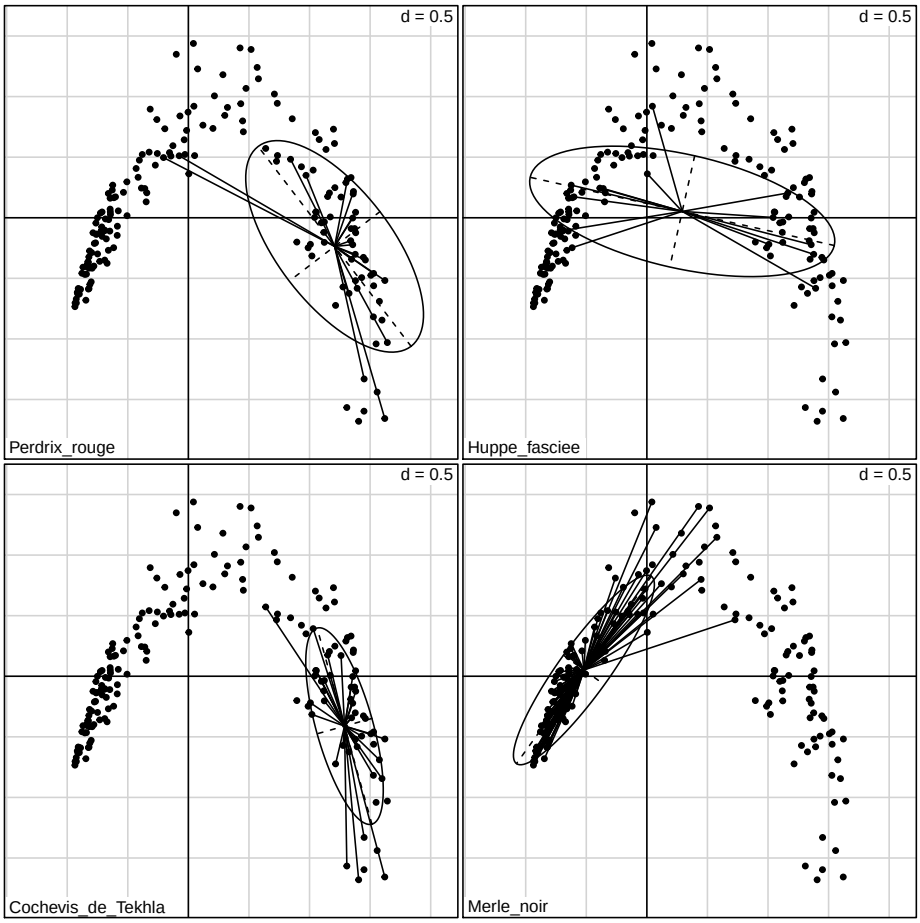
```
function (dfxy, dfdistri, xax = 1, yax = 2, cstar = 1, cellipse = 1.5,
    axesell = TRUE, label = names(dfdistri), clabel = 0, cpoint = 1,
    pch = 20, xlim = NULL, ylim = NULL, grid = TRUE, addaxes = TRUE,
    origin = c(0, 0), include.origin = TRUE, sub = "", csub = 1,
    possub = "bottomleft", cgrid = 1, pixmap = NULL, contour = NULL,
    area = NULL, add.plot = FALSE)
NULL
```

```
args.s.distri <- names(as.list(args(s.distri)))
args.s.distri[!args.s.distri %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist, args.
```

```
[1] "dfdistri"
```

Cette fonction permet la représentation d'une distribution de fréquences donnée dans l'argument `dfdistri` :

```
data(rpjdl)
 xy <- dudi.coa(rpjdl$fau, scan = FALSE)$li
 par(mfrow = c(2,2))
 for (i in c(1,5,8,20)){
 s.distri(xy, rpjdl$fau[,i], sub = rpjdl$frlab[i])
}
```



2.15 s.corcircle()

```
 args(s.corcircle)
```

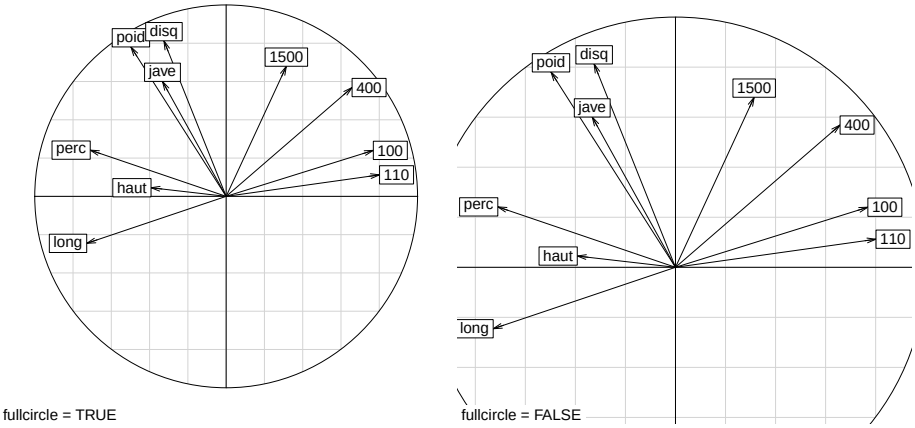
```
function (dfxy, xax = 1, yax = 2, label = row.names(df), clabel = 1,
    grid = TRUE, sub = "", csub = 1, possub = "bottomleft", cgrid = 0,
    fullcircle = TRUE, box = FALSE, add.plot = FALSE)
NULL
```

```
 args.s.corcircle <- names(as.list(args(s.corcircle)))
 args.s.corcircle[!args.s.corcircle %in% c(args.s.label, args.s.class, args.s.chull, args.s.arrow, args.s.hist,
```

```
[1] "fullcircle" "box"
```

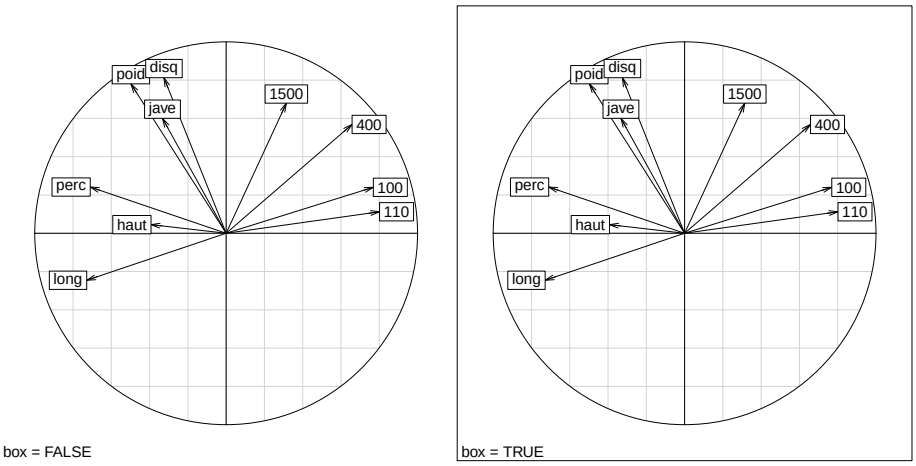
Cette fonction permet le tracé des cartes factorielles du type cercle de corrélation, c'est-à- dire la projection des vecteurs de la base canonique (les variables de départ) sur les différents plans factoriels. L'argument `fullcircle` permet de neutraliser l'utilisation du cercle complet par défaut :

```
 data (olympic)
 par(mfrow=c(1,2))
 acp <- dudi.pca(olympic$tab, scannf = FALSE)
 s.corcircle(acp$co, sub = "fullcircle = TRUE")
 s.corcircle(acp$co, fullcircle = FALSE, sub = "fullcircle = FALSE")
```



L'argument `box` permet de tracer un cadre autour du graphique :

```
data (olympic)
par(mfrow=c(1,2))
acp <- dudi.pca(olympic$tab, scannf = FALSE)
s.corcircle(acp$co, sub = "box = FALSE")
s.corcircle(acp$co, box = TRUE, sub = "box = TRUE")
```



3 La fonction générique `scatter()`

3.1 Table ordonnée de présence/absence des arguments dans ces fonctions

La fonction `scatter()` est générique : son comportement va s'adapter automatiquement en fonction de la classe de l'argument `x` qui lui est transmis. Le code suivant a été utilisé pour générer la figure 3 synthétisant les arguments disponibles pour ces fonctions.

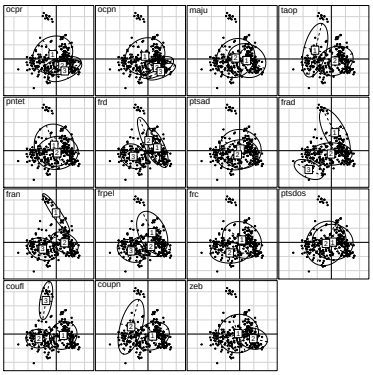
```
scatterqqc <- paste("ade4:::scatter", c("acm", "coa", "dudi", "fca", "nipals", "pco"), sep = ".")
larg <- vector("list", 6)
larg[[1]] <- names(as.list(args(ade4:::scatter.acm)))
larg[[2]] <- names(as.list(args(ade4:::scatter.coa)))
larg[[3]] <- names(as.list(args(ade4:::scatter.dudi)))
```

```
larg[[4]] <- names(as.list(args(ade4:::scatter.fca)))
larg[[5]] <- names(as.list(args(ade4:::scatter.nipals)))
larg[[6]] <- names(as.list(args(ade4:::scatter.pco)))
#lapply(scatterqqc, function(x) names(as.list(args(x)))) -> larg
names(larg) <- scatterqqc
larg <- lapply(larg, function(x) x[nchar(x)>1])
allargs <- sort(unique(unlist(larg)))
TabArg <- matrix(0, nrow = length(allargs), ncol = length(scatterqqc))
rownames(TabArg) <- allargs
colnames(TabArg) <- scatterqqc
for(j in 1:length(scatterqqc)) TabArg[,j] <-  allargs %in% larg[[j]]
afc <- dudi.coa(TabArg, scan = FALSE)
TabArg <- TabArg[order(afc$li[,1]), order(afc$co[,1])]
plot.new()
par(mar=c(0,0,0,0)+0.1)
nl <- length(allargs)
nc <- length(scatterqqc)
plot.window(xlim = c(0, nc+1), ylim = c(0, nl+2))
text(0.8, 1:nl, rownames(TabArg), cex = 0.8, pos = 2)
segments(0.8, 1:nl, nc+0.2, 1:nl, col = grey(0.7))
text(1:nc+0.5, nl+1, colnames(TabArg), srt = 30, pos = 3, cex = 1)
segments(1:nc, 0.8, 1:nc, nl+0.2, col = grey(0.7))
for(i in 1:nl) for(j in 1:nc) if(TabArg[i,j]==1) points(j,i, pch = 20)
```

3.2 Graphiques par défaut de `scatter()`.

3.2.1 `scatter.acm()`

```
data(lascaux)
acm <- dudi.acm(lascaux$ornem, scannf = FALSE)
scatter(acm)
```



3.2.2 `scatter.coa()`

```
data(housetasks)
coa <- dudi.coa(housetasks, scannf = FALSE)
scatter(coa)
```

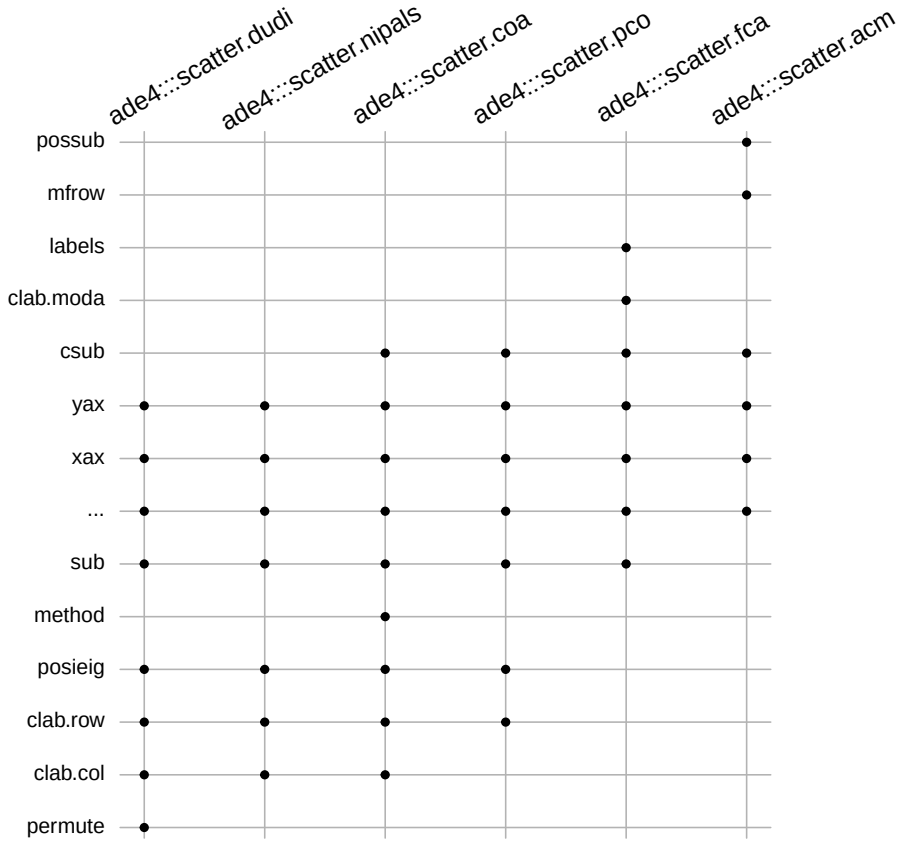
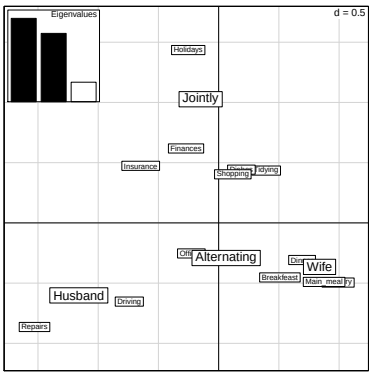
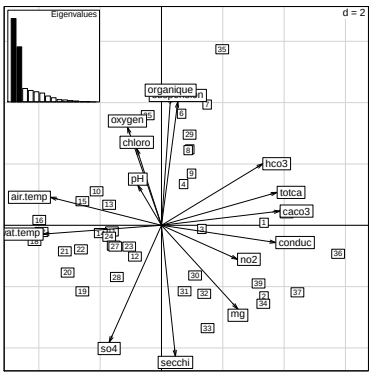


FIGURE 3 – Présentation synthétique des arguments disponibles pour les fonctions de la famille `scatter`. Les 14 arguments possibles sont disposés en ligne, les 6 fonctions sont disposées en colonne. Un point de coordonnées (i, j) signifie que l'argument i est explicitement disponible pour la fonction j. L'ordre des lignes et des colonnes est selon le premier facteur de l'AFC de cette table de présence/absence.



3.2.3 scatter.dudi()

```
data(rhone)
pca <- dudi.pca(rhone$tab, scannf = FALSE)
scatter(pca)
```

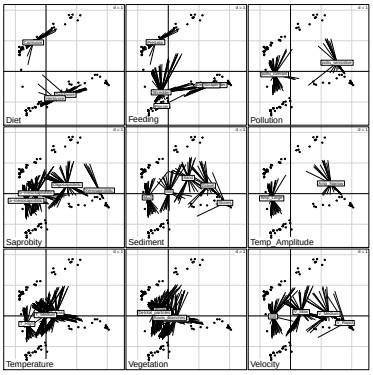


3.2.4 scatter.fca()

```
data(coleo)
coleo.fuzzy <- prep.fuzzy.var(coleo$tab, coleo$col.blocks)
```

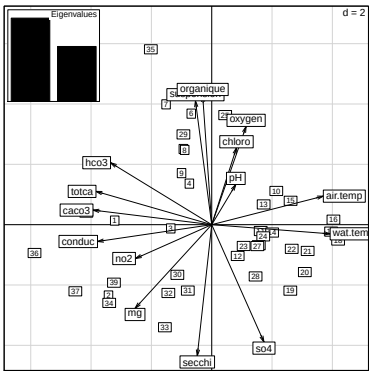
```
2 missing data found in block 1
1 missing data found in block 3
2 missing data found in block 4
```

```
fca <- dudi.fca(coleo.fuzzy, scannf = FALSE)
scatter(fca, labels = coleo$moda.names)
```



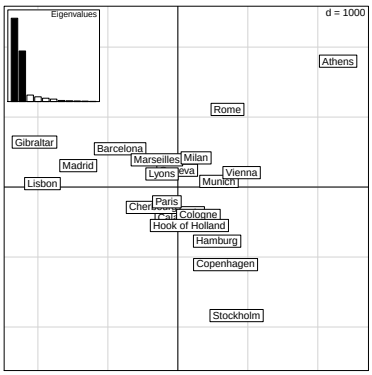
3.2.5 scatter.nipals()

```
data(rhone)
rhone$tab[1,1] <- NA
nip <- nipals(rhone$tab)
scatter(nip)
```



3.2.6 scatter.pco()

```
data(eurodist)
pco <- dudi.pco(eurodist, scannf = FALSE)
scatter(pco)
```

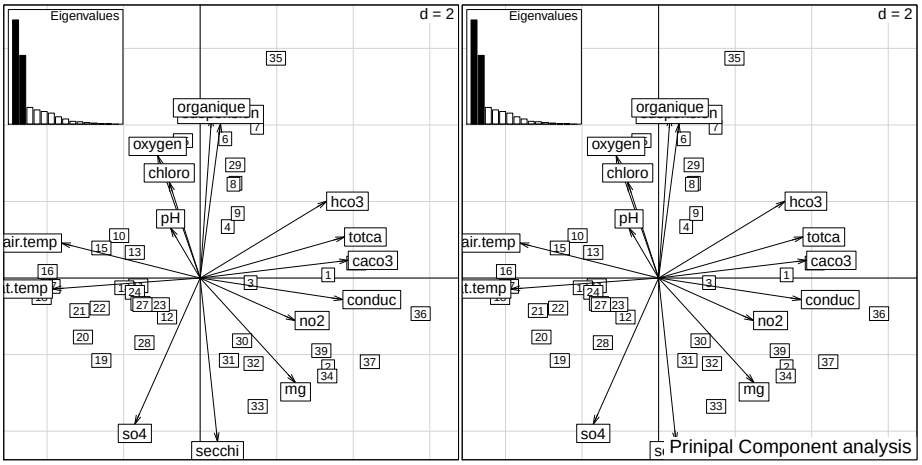


3.3 Quelques paramètres usuels

3.3.1 sub

Ce paramètre permet d'introduire une légende :

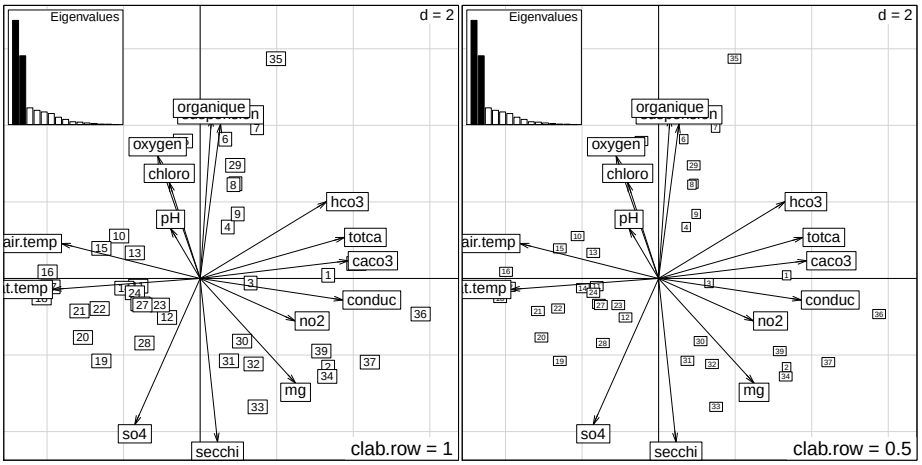
```
par(mfrow = c(1,2))
scatter(pca)
scatter(pca, sub = "Prinipal Component analysis")
```



3.3.2 `clab.row`

Ce paramètre contrôle la taille des étiquettes correspondant aux lignes :

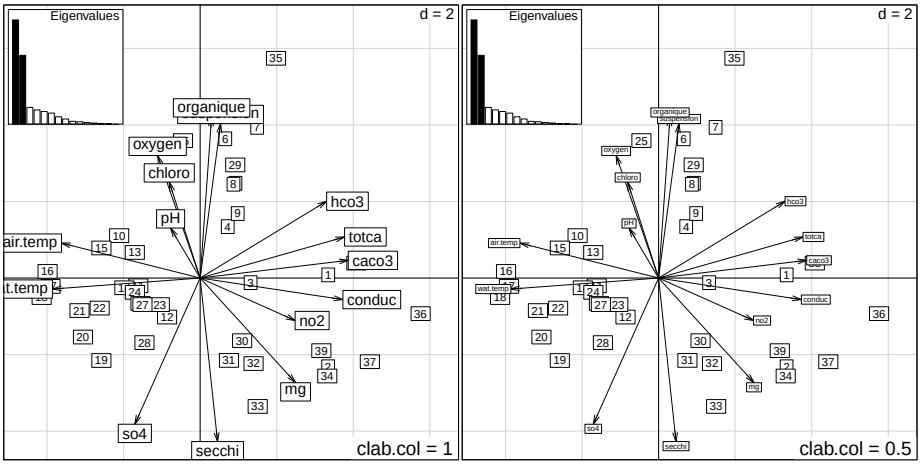
```
par(mfrow = c(1,2))
scatter(pca, sub = "clab.row = 1")
scatter(pca, clab.row = 0.5, sub = "clab.row = 0.5")
```



3.3.3 `clab.col`

Ce paramètre contrôle la taille des étiquettes correspondant aux colonnes :

```
par(mfrow = c(1,2))
scatter(pca, sub = "clab.col = 1")
scatter(pca, clab.col = 0.5, sub = "clab.col = 0.5")
```

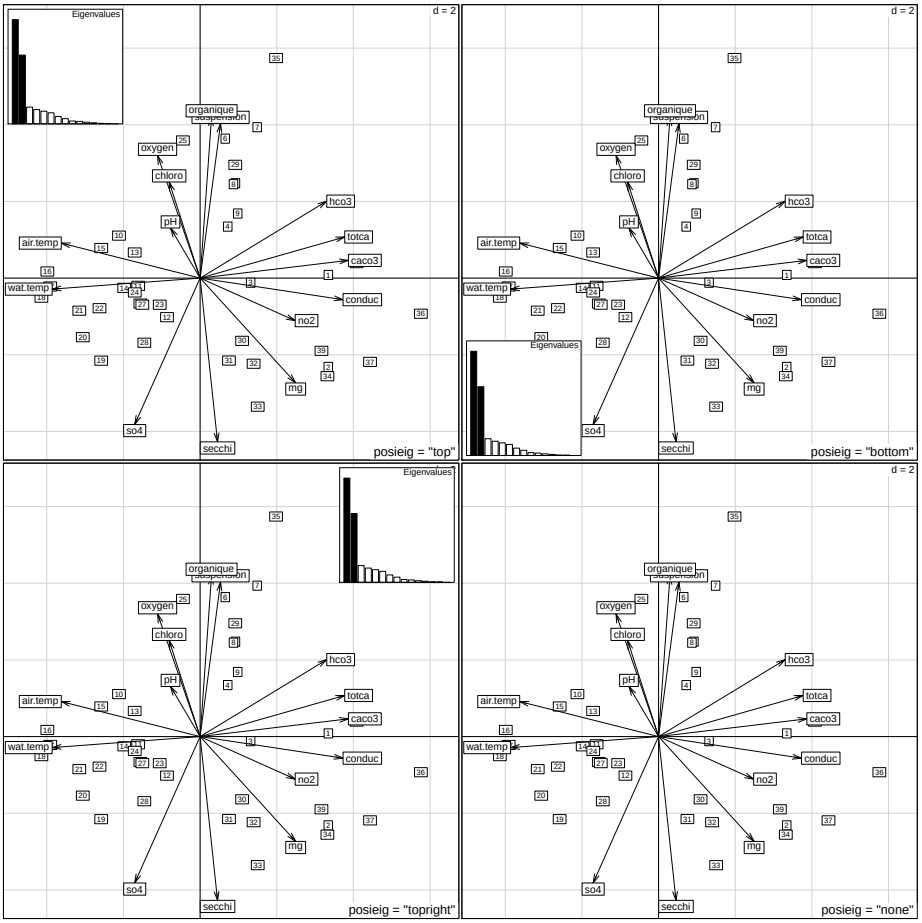


3.3.4 `posieig`

Ce paramètre contrôle la position du graphe des valeurs propres :

```
par(mfrow = c(2,2))
scatter(pca, sub = "posieig = \"top\"")
scatter(pca, posieig = "bottom", sub = "posieig = \"bottom\"")
scatter(pca, posieig = "topright", sub = "posieig = \"topright\"")
scatter(pca, posieig = "none", sub = "posieig = \"none\"")
```

```
NULL
```



Références

[1] D. Chessel, A.-.B. Dufour, and J. Thioulouse. The ade4 package-I- One-table methods. *R News*, 4 :5–10, 2004.

[2] S. Dray and A. Siberchicot. *adegraphics : An S4 Lattice-Based Package for the Representation of Multivariate Data*, 2016. R package version 1.0-6.