

Renaudpack2 - tuto

Contents

1	Introduction	3
1.1	Pour l'installer	3
1.2	Pour le mettre à jour	3
1.3	Pour le charger	3
2	Les fonctions graphiques pour les analyses univari?es	4
2.1	Un facteur et des barres d'erreur avec la fonction <code>barres.plot</code>	4
2.2	Pour rajouter les lettres des post-hoc avec la fonction <code>anovLetters</code>	6
2.3	Deux facteurs et des barres d'erreur avec la fonction <code>barres.plot.beside</code>	7
2.4	Faire un tableau r?capitulatif avec <code>Tableau_recap</code>	11
2.5	Des s?ries temporelles avec <code>Time.factor.plot</code>	13
2.6	Des jolies couleurs avec les fonctions <code>Couleur_continue</code> et <code>modif_coul</code>	14
2.7	Des diagrammes en barres mais avec des points avec <code>point.plot</code>	17
3	Fonctions graphiques pour les analyses multivar?es	19
3.1	La fonction <code>multivar.polyg</code>	19
3.2	La fonction <code>MultiDyn</code>	25
3.3	La fonction <code>better_arrows</code>	26
3.4	La fonction <code>label.corV2</code>	27
3.5	La fonction <code>label.corDCA</code>	31
3.6	La fonction <code>label.corNMDS</code>	32
4	Pour de la manipulation de donn?es	33
4.1	La fonction <code>BBtransf</code>	33
4.2	La fonction <code>Classes_def</code>	33
4.3	La fonction <code>combin.tab</code>	33
4.4	La fonction <code>combin.tabV0</code>	33

5	Pour des tâches très spécifiques :	33
5.1	Autour de l'indice de l'intégrité de la structure des communautés	33
5.2	La fonction <code>ComStructIndices</code>	33
5.3	La fonction <code>structure.plot</code>	33
5.4	La fonction <code>structure.plotV2</code>	33
5.5	La fonction “	33

Dernière mise à jour : 27-12-2019

1 Introduction

Il s'agit d'un tuto pour mes fonctions perso, réunies dans le package au doux sobriquet de **Renaudpack2**

Depuis peu l'installation est possible via Github, c'est ultra simple en utilisant une fonction du package **devtools**, qu'il faut donc potentiellement installer si ce n'est déjà fait :

1.1 Pour l'installer

```
install.packages("devtools")  
## ne faire que si vous n'avez jamais installé devtools  
library(devtools)  
devtools::install_github("RenaudJau/Renaudpack2")
```

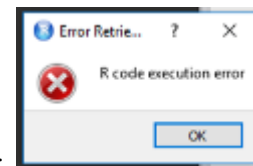
1.2 Pour le mettre à jour

```
install.packages("devtools")  
## ne faire que si vous n'avez jamais installé devtools  
library(devtools)  
devtools::update_packages("Renaudpack2")
```

1.3 Pour le charger

Comme n'importe quel package, avec **library** :

```
library(Renaudpack2)
```



Si jamais il y a un message d'erreur du style de l'impression ?cran ci-apr?s :

Je ne sais pas encore d'où ça vient, mais en redémarrant R **Session > Restart R** ou **Ctrl+Shift+F10**, et en refaisant **library(Renaudpack2)** ?a semble fonctionner.

2 Les fonctions graphiques pour les analyses univariées

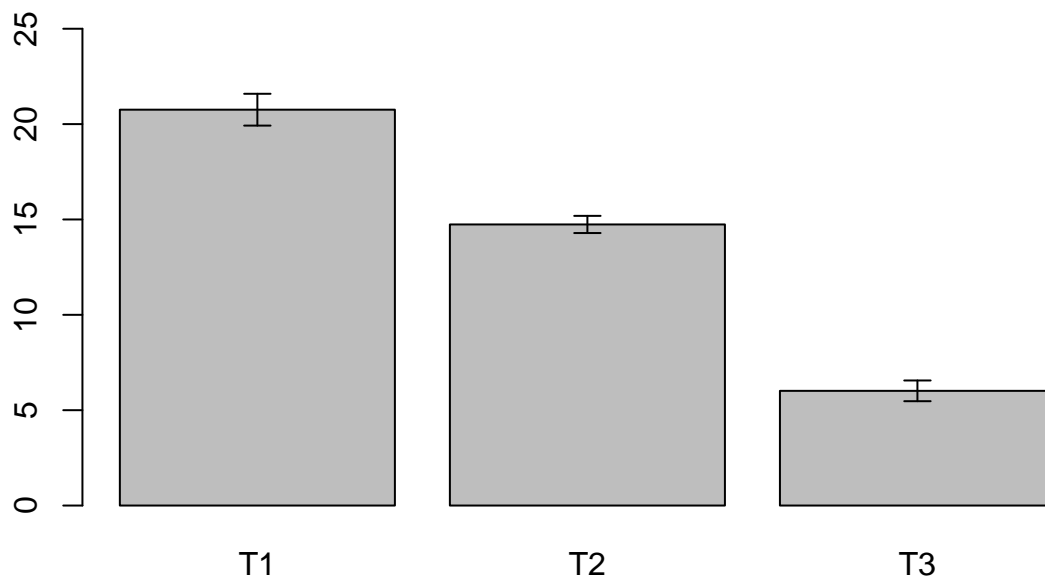
Pour les besoins de ce tuto, on crée des fausses données :

```
biomasse<-c(rnorm(35,20,5),rnorm(35,15,3),rnorm(35,5,3))  
traitement<-factor(rep(c("T1","T2","T3"),each=35))
```

2.1 Un facteur et des barres d'erreur avec la fonction `barres.plot`

Le graphique de base :

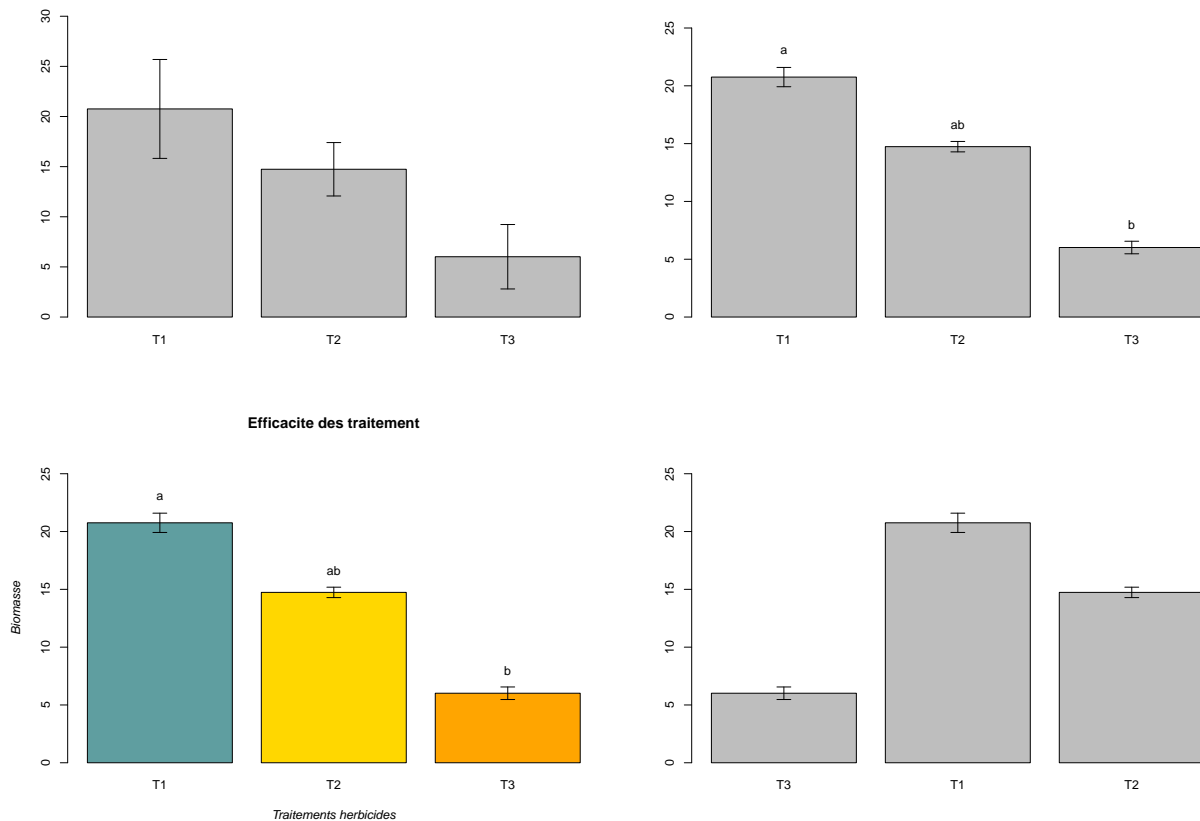
```
barres.plot(biomasse,traitement)
```



Il y a plusieurs choses que l'on peut modifier pour personnaliser son graphique :

- changer le type d'erreur représenté
- ajouter des lettres d'un test post-hoc
- changer des paramètres classiques de la fonction `barplot`
- changer l'ordre des modalités (ça n'est pas spécifique à la fonction `barres.plot`)
- on peut aussi modifier l'axe des abscisses (cf les arguments `las.x`, `cex.x` et `labels.x` de la fonction `barres.plot.beside` expliquée plus tard).

```
par(mfrow=c(2,2)) #pour avoir 4 graphiques sur une même fenêtre
barres.plot(biomasse,traitement,ecart=sd)
#erreur standard par défaut, ici cart-type
barres.plot(biomasse,traitement,lettres=c("a","ab","b"))
barres.plot(biomasse,traitement,lettres=c("a","ab","b"),
col=c("cadetblue","gold","orange"),
xlab="Traitements herbicides",ylab="Biomasse",font.lab=3,
main="Efficacite des traitement")
traitement2<-factor(traitement,levels=c("T3","T1","T2"))
barres.plot(biomasse,traitement2)
```



2.2 Pour rajouter les lettres des post-hoc avec la fonction `anovLetters`

Si vous ne voulez pas vous embêter à calculer vous même les lettres des tests post-hoc, et si et uniquement si vous êtes dans les conditions d'utilisation d'une ANOVA à un facteur, alors on peut faire ça :

```
anov<-aov(biomasse~traitement)
summary(anov)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## traitement    2   3845   1922.7   138.2 <2e-16 ***
## Residuals   102   1420    13.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

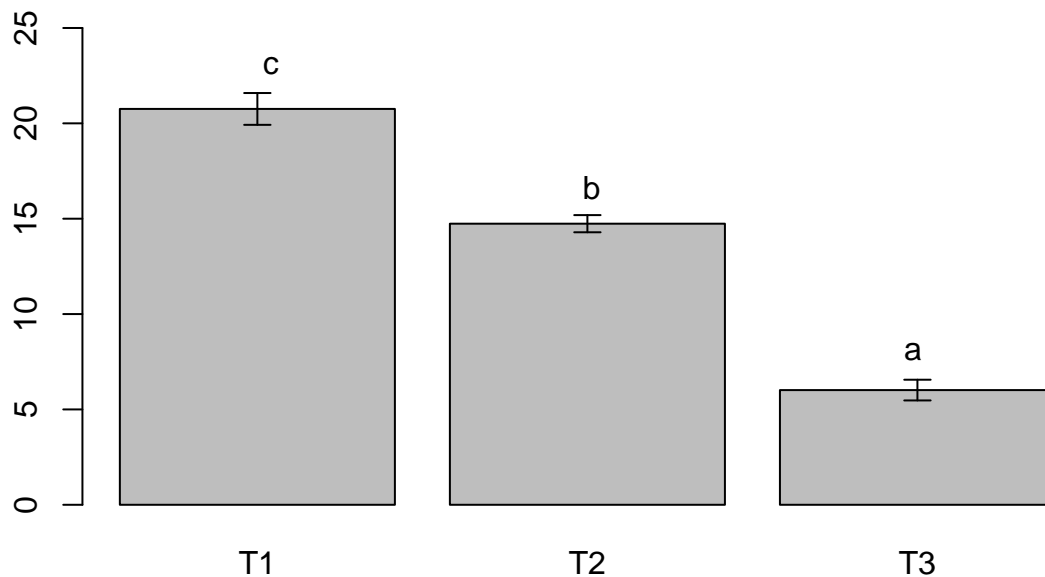
OK, il y a un effet significatif, du coup `anovLetters` renvoie la liste des lettres...

```
lettresPH<-anovLetters(VAR = biomasse,FAC = traitement)
lettresPH
```

```
## [1] " c" " b" " a "
```

...que l'on peut utiliser directement sur `barres.plot` :

```
barres.plot(biomasse,traitement,lettres = lettresPH)
```



*Note : possibilité de changer le seuil alpha en utilisant l'argument `ALPHA`.

2.3 Deux facteurs et des barres d'erreur avec la fonction `barres.plot.beside`

Pour les besoins de ce tuto, on cr?? des fausses donn?es, avec 2 facteurs, l'?ge et le sexe, et une variable, la taille :

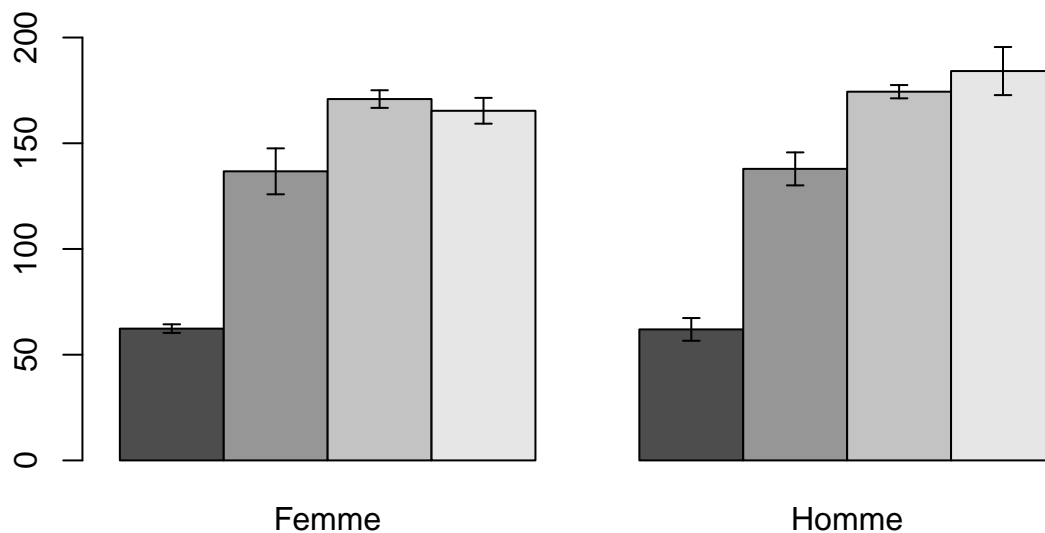
```
Age=factor(rep(c(2,10,20,30),each=10))
Sexe=factor(rep(rep(c("Homme","Femme"),5),4))
Taille=c(rnorm(5,60,7),rnorm(5,55,7),rnorm(5,145,15),rnorm(5,129,15),rnorm(5,175,15),
         rnorm(5,165,15),rnorm(5,175,15),rnorm(5,165,15))
```

En gros ?a donne ?a :

Age	Sexe	Taille
2	Homme	59.16569
2	Femme	69.53185
2	Homme	76.71009
2	Femme	64.04338
2	Homme	62.64141
2	Femme	59.62727

Le graphique de base :

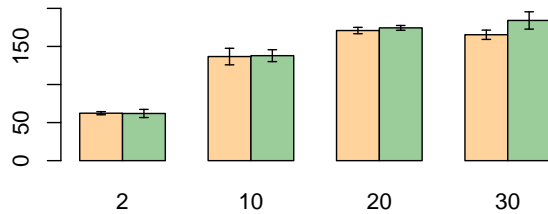
```
barres.plot.beside(Taille,Sexe,Age)
```



On peut aussi modifier quelques petits trucs :

- si on inverse les facteurs, avec des couleurs

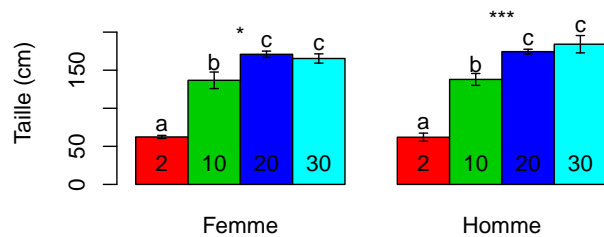
```
barres.plot.beside(Taille, Age, Sexe, col=c("burlywood1", "darkseagreen3"))
```



- avec les annotations qui vont bien

Attention, les lettres et étoiles sont ? d'finir soit m?me, c?-d apr?s tests statistiques, ici pour l'exemple, c'est de l'al?atoire...), et avec des couleurs simples (mais moches...)

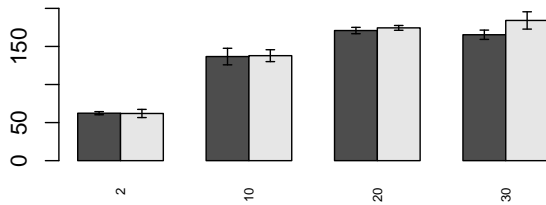
```
barres.plot.beside(Taille, Sexe, Age, POSI="bottom",
lettres=c("a", "b", "c", "c", "a", "b", "c", "c"),
etoiles=c("?", "***"), ylab="Taille (cm)", col=2:5)
```



On peut aussi modifier l'axe des abscisses :

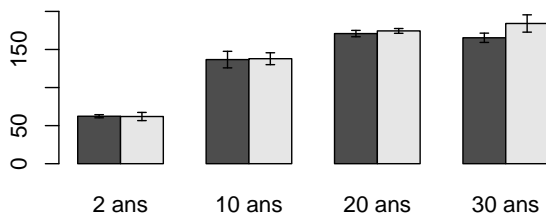
- l'orientation avec `las.x` et la taille avec `cex.x` :

```
barres.plot.beside(Taille, Age, Sexe, las.x = 2, cex.x = 0.6)
```



- les étiquettes avec `labels.x` :

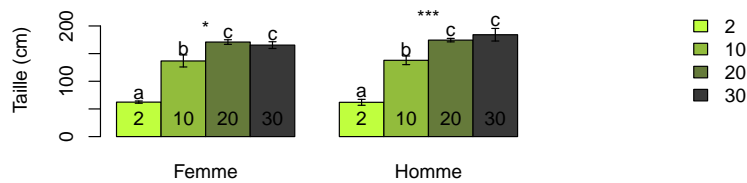
```
barres.plot.beside(Taille, Age, Sexe, labels.x = c("2 ans", "10 ans", "20 ans", "30 ans"))
```



Attention, changer les noms ne veut pas dire qu'on change l'ordre des facteurs.. ne pas écrire n'importe quoi..

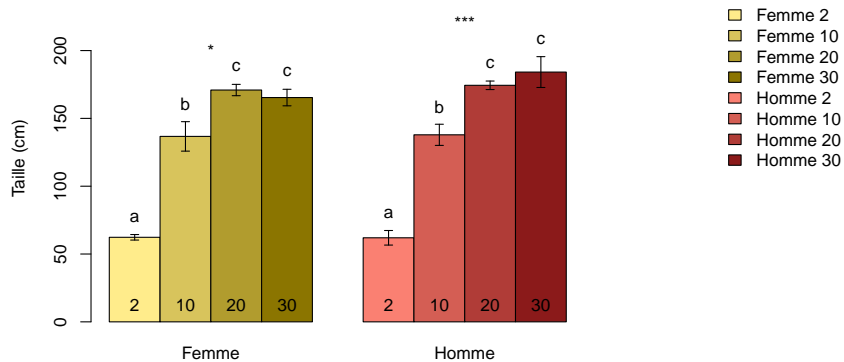
- avec des couleurs choisies avec la fonction `colorRampPalette` et une légende

```
par(mfrow=c(1,2))
coul<-c(colorRampPalette(colors=c("olivedrab1", "grey22"))(4))
barres.plot.beside(Taille,Sexe,Age,POSI="bottom",
  lettres=c("a","b","c","c","a","b","c","c"),
  etoiles=c("*","***"),col=coul,ylab="Taille (cm)")
plot(1,type="n",axes = F,ann = F)
legend("topleft",levels(Age),fill=coul,bty="n")
```



- avec des couleurs pour chaque combinaison de modalités

```
par(mfrow=c(1,2))
coul2<-c(colorRampPalette(colors=c("lightgoldenrod1", "gold4"))(4),
  colorRampPalette(colors=c("salmon", "firebrick4"))(4))
barres.plot.beside(Taille,Sexe,Age,POSI="bottom",
  lettres=c("a","b","c","c","a","b","c","c"),
  etoiles=c("*","***"),col=coul2,ylab="Taille (cm)")
plot(1,type="n",axes = F,ann = F)
legend("topleft",paste(rep(levels(Sexe),each=length(levels(Age))),
  rep(levels(Age),2),sep=" "),fill=coul2,bty="n")
```



2.4 Faire un tableau r?capitulatif avec Tableau_recap

Cette fonction n'est pas graphique, elle donne les informations de `barres.plot` sous forme d'un tableau. Elle a peu d'utilit? en dehors d'un document Rmarkdown.

```
Tableau_recap(VAR = biomasse,FAC = traitement,ROUND = 2)
```

```
##      Modalites Nombre Moyennes Erreur
## T1          T1      35      20.76  0.83
## T2          T2      35      14.74  0.45
## T3          T3      35       6.01  0.54
```

Pour que ce soit plus joli, il est cosneill? d'utiliser la fonction `kabledu` package `knitr` (pour l'installer : `install.packages("knitr")`).

```
kable(Tableau_recap(VAR = biomasse,FAC = traitement,ROUND = 2))
```

	Modalites	Nombre	Moyennes	Erreur
T1	T1	35	20.76	0.83
T2	T2	35	14.74	0.45
T3	T3	35	6.01	0.54

Si on fait un test posthoc, on peut rajouter les lettres dans le tableau :

```
anov<-aov(biomasse~traitemment)
summary(anov)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## traitement    2   3845  1922.7    138.2 <2e-16 ***
## Residuals   102   1420    13.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
TukeyHSD(anov)
```

```
##      Tukey multiple comparisons of means
##      95% family-wise confidence level
##
## Fit: aov(formula = biomasse ~ traitement)
##
## $traitemment
##              diff              lwr              upr p adj
## T2-T1 -6.017650 -8.138682 -3.896618      0
## T3-T1 -14.741086 -16.862118 -12.620054      0
## T3-T2 -8.723436 -10.844468 -6.602404      0
```

#les lettres sont donc a, b et c :

```
kable(Tableau_recap(VAR = biomasse,FAC = traitement,ROUND = 2,
  LETTRES = c("a","b","c")))
```

Modalites		Nombre	Moyennes	Erreur	Lettres
T1	T1	35	20.76	0.83	a
T2	T2	35	14.74	0.45	b
T3	T3	35	6.01	0.54	c

2.5 Des séries temporelles avec `Time.factor.plot`

Pour les besoins de ce tuto, on crée des fausses données :

```
tim<-rep(c(1:4),each=6)
fac<-factor(rep(rep(c("A","B"),each=3),4))
vari<-c(rnorm(3,5,2),rnorm(3,3,2),rnorm(3,3,2),rnorm(3,8,2),
        rnorm(3,8,2),rnorm(3,10,2),rnorm(3,16,2),rnorm(3,12,2))
```

En gros ça donne ça :

tim	fac	vari
1	A	6.4737837
1	A	3.1645833
1	A	5.9696492
1	B	-0.8160700
1	B	1.7446723
1	B	0.9080188

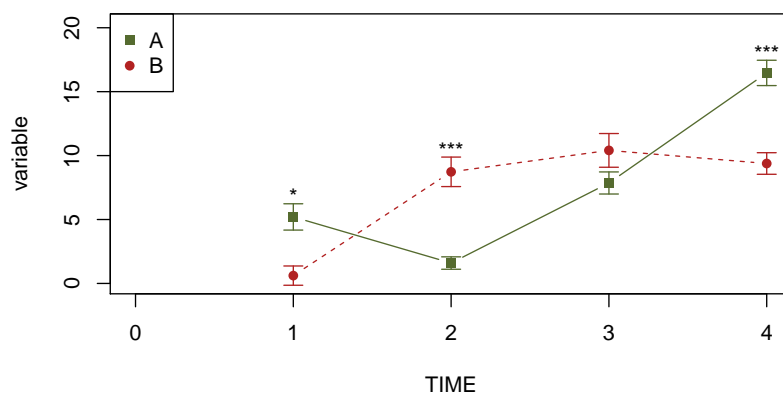
Utilisation de la fonction : *Pourquoi ce message d'erreur? Bonne question.. ? résoudre..*

```
Time.factor.plot(tim,fac,vari,etoiles=c("*","***","","***"),
                 pch = c(15,16),couleur = c("darkolivegreen","firebrick"),
                 lty = c(1,2))
```

```
## Warning in if (couleur == c(0)) rep(1, length(levels(factor(FACTOR)))) else
## couleur: la condition a une longueur > 1 et seul le premier élément est utilisé
```

```
## Warning in if (pch == c(0)) rep(1, length(levels(factor(FACTOR)))) else pch: la
## condition a une longueur > 1 et seul le premier élément est utilisé
```

```
## Warning in if (lty == c(0)) rep(1, length(levels(factor(FACTOR)))) else lty: la
## condition a une longueur > 1 et seul le premier élément est utilisé
```



2.6 Des jolies couleurs avec les fonctions `Couleur_continue` et `modif_coul`

`Couleur_continue` permet de transformer une variable en palette de couleur, utile pour ajouter une 'dimension' ? un plot en 2D

Par exemple, si on a des données (fausses ici..) avec la couverture végétale en fonction de l'altitude, et une information sur la taille des plantes :

taille	recouvrement	altitude
89.2	99	467
18.0	98	1257
40.1	99	1955
90.6	96	2287
86.0	96	1920
83.9	96	1397

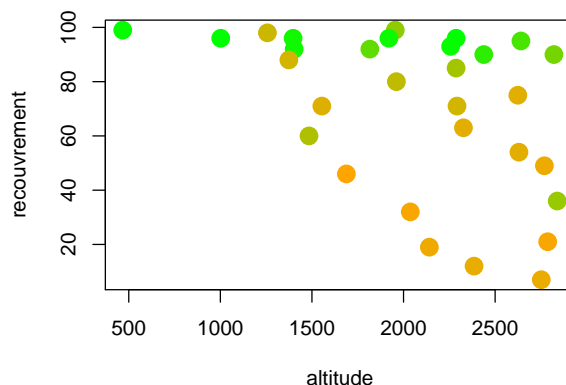
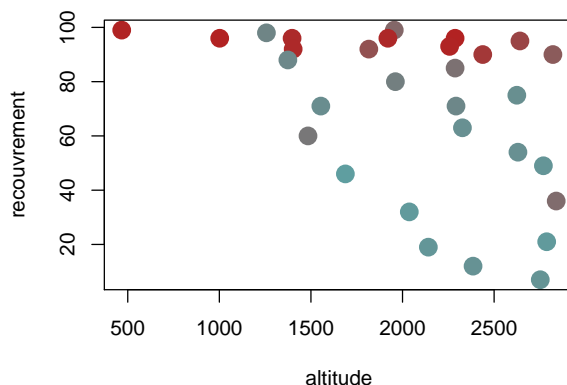
On peut créer une couleur pour chaque valeur de taille (ici avec des couleurs plus froides pour les tailles plus petites) :

```
coul_taille<-Couleur_continue(VAR = taille,COLORS = c("cadetblue","firebrick"))
coul_taille
```

```
## [1] "#B02424" "#6F8586" "#836768" "#B22222" "#AD2828" "#AC2A2A" "#B22222"
## [8] "#AD2929" "#925151" "#B02424" "#A33838" "#9A4445" "#8B5B5B" "#7C7172"
## [15] "#737F80" "#6D8789" "#688F90" "#689092" "#689092" "#649597" "#806C6D"
## [22] "#619A9C" "#659495" "#669294" "#639698" "#619A9C" "#5F9EA0" "#787778"
## [29] "#698D8F" "#6D888A"
```

Ces couleurs (en code hexadecimal) peuvent être utilisées dans un graphique, avec la possibilité de changer les couleurs comme on veut :

```
par(mfrow=c(1,2))
plot(x = altitude, y = recouvrement, pch=16, cex=2, col=coul_taille)
plot(x = altitude, y = recouvrement, pch=16, cex=2,
     col=Couleur_continue(taille,COLORS=c("Orange","Green")))
```

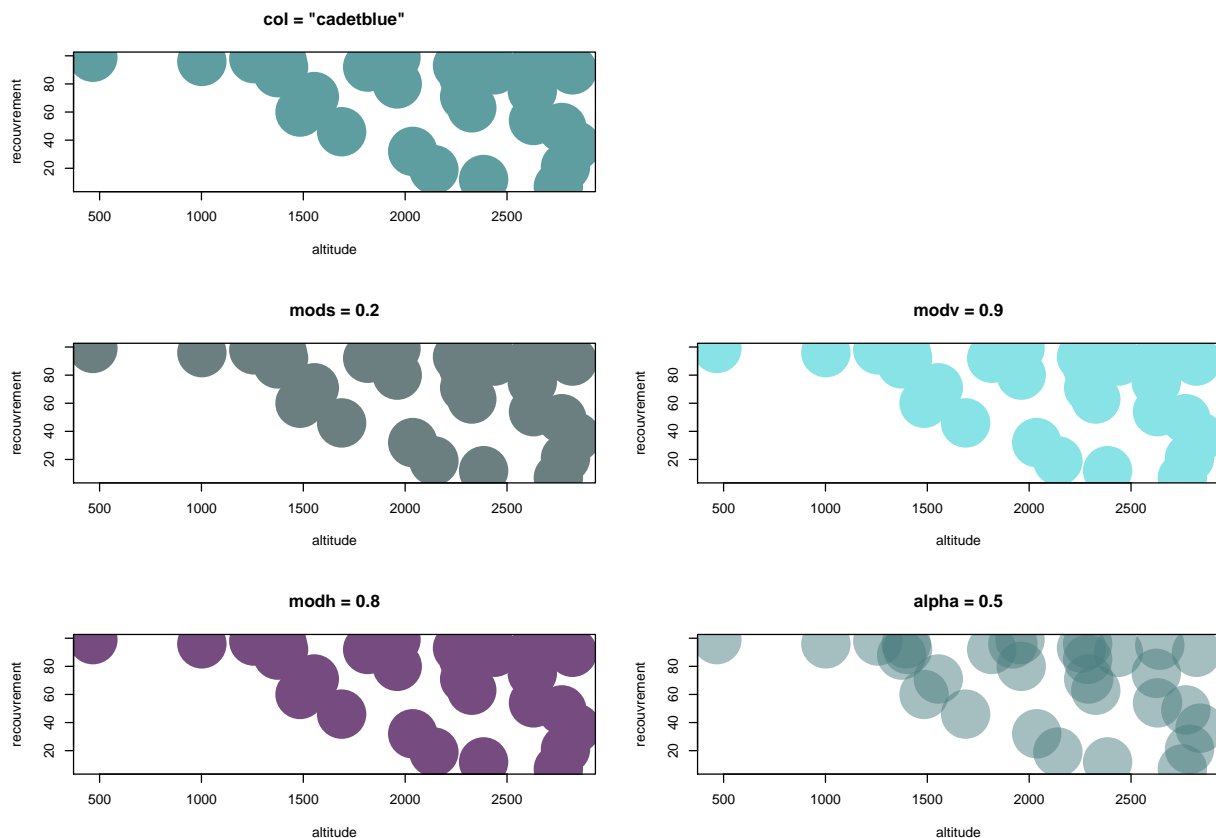


La fonction `modif_coul` permet de modifier une couleur existante :

Par exemple, en partant de la couleur `cadetblue` (qui est un bleu pastel), on peut :

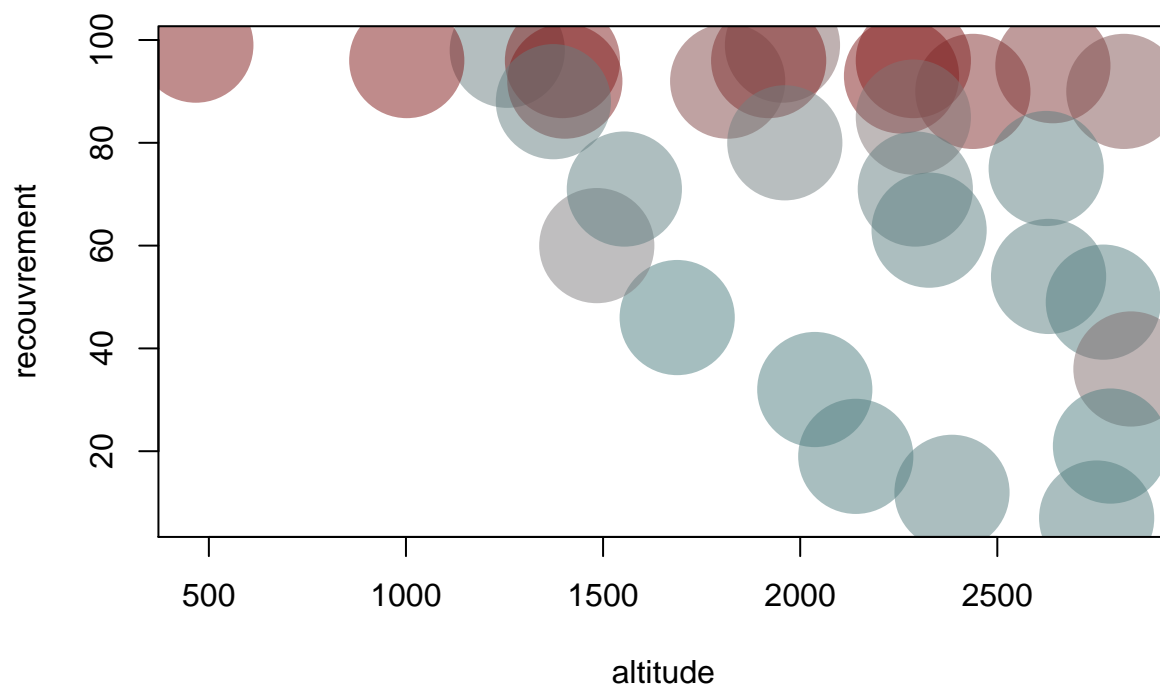
- modifier la saturation, sous 0.5 ?a d?sature, au dessus, ?a sature
- modifier la brillance, sous 0.5 ?a fonce, au dessus, ?a ?claircit
- modifier la teinte, ?a tourne en rond..
- modifier la transparence sous 1, ?a devient transparent

```
par(mfrow=c(3,2))
plot(x = altitude, y = recouvrement, pch = 16, cex = 8, col = "cadetblue",
     main = 'col = "cadetblue"')
plot(1,type="n",axes = F,ann = F) #juste pour un graphique vide..
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",mods = 0.2), main = "mods = 0.2")
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",modv = 0.9), main = "modv = 0.9")
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",modh = 0.8), main = "modh = 0.8")
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",alpha = 0.5), main = "alpha = 0.5")
```



On peut aussi combiner les deux fonctions :

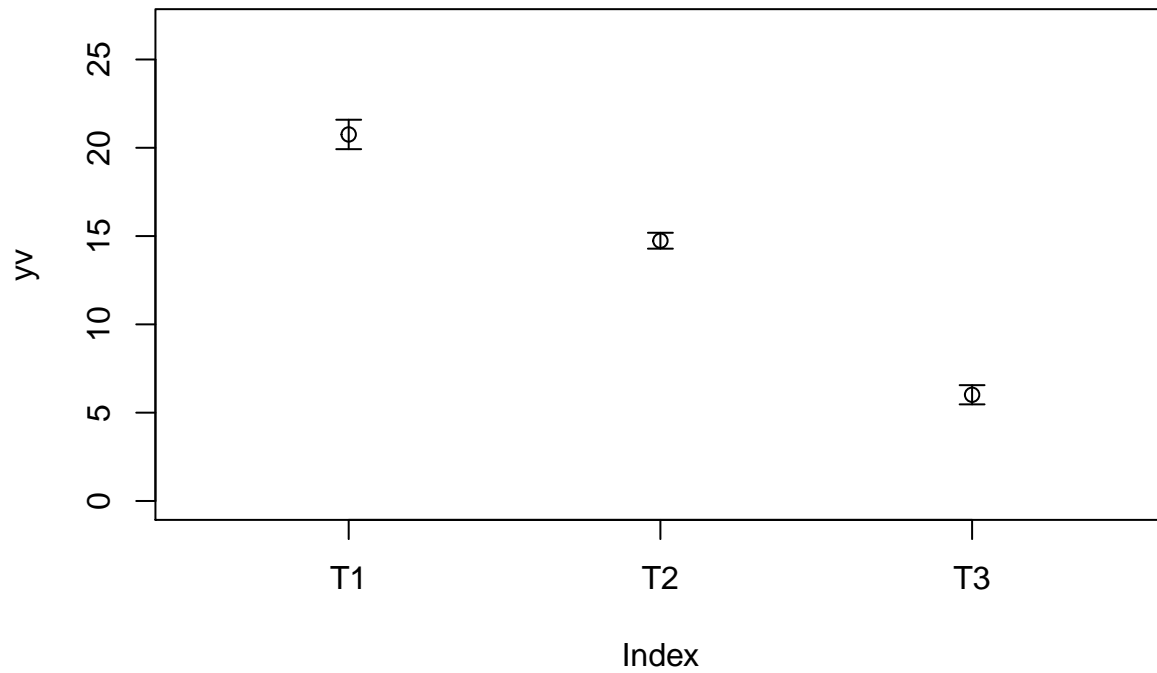
```
coul_taille_transp <- sapply(coul_taille, function(x) modif_coul(x, alpha = 0.5))  
plot(x = altitude, y = recouvrement, pch=16, cex=8, col=coul_taille_transp)
```



2.7 Des diagrammes en barres mais avec des points avec `point.plot`

Quasi identique ? l'utilisation de `barres.plot`.. Le graphique de base :

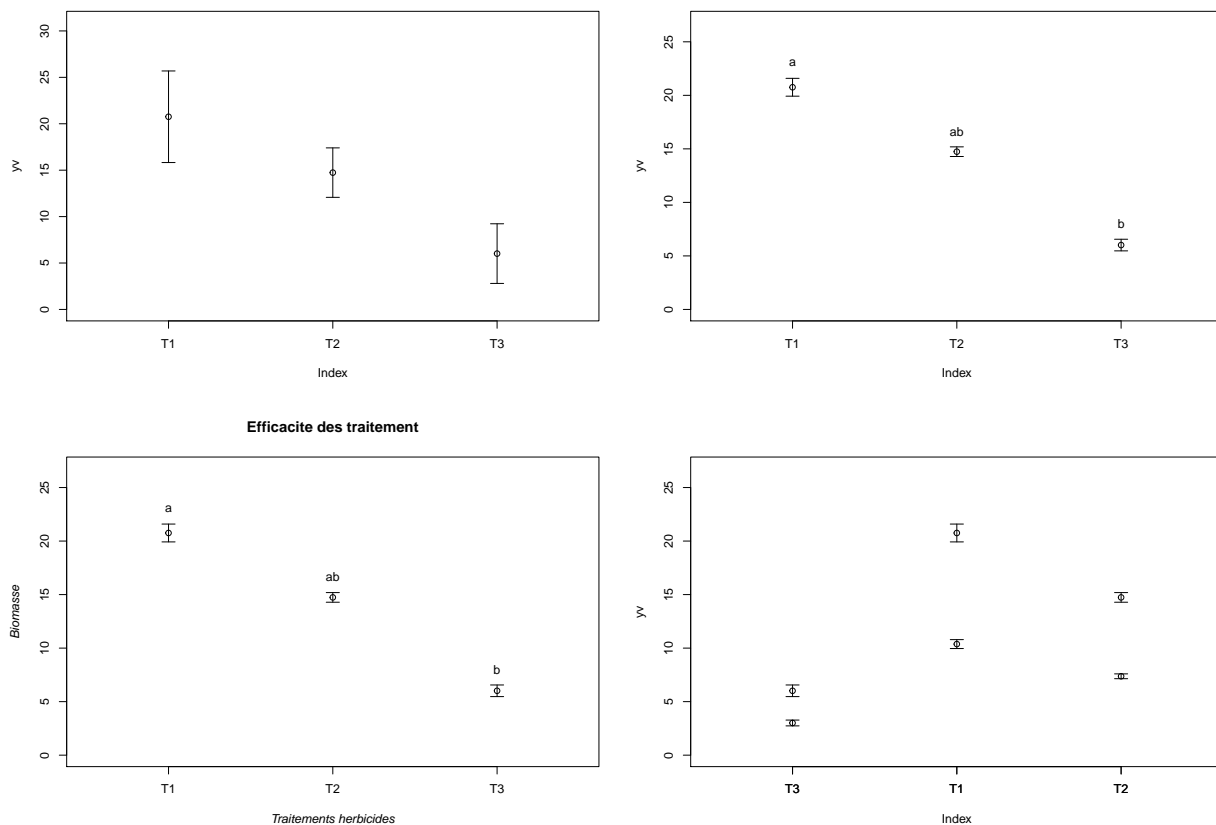
```
point.plot(biomasse,traitement)
```



Il y a plusieurs choses que l'on peut modifier pour personnaliser son graphique :

- changer le type d'erreur représenté
- ajouter des lettres d'un test post-hoc
- changer des paramètres classiques de la fonction `barplot`
- changer l'ordre des modalités (ça n'est pas spécifique à la fonction `point.plot`) et surajouter sur un graphique

```
par(mfrow=c(2,2)) #pour avoir 4 graphiques sur une même fenêtre
point.plot(biomasse,traitement,ecart=sd)
#erreur standard par défaut, ici cart-type
point.plot(biomasse,traitement,lettres=c("a","ab","b"))
point.plot(biomasse,traitement,lettres=c("a","ab","b"),
xlab="Traitements herbicides",ylab="Biomasse",font.lab=3,
main="Efficacite des traitement")
traitement2<-factor(traitement,levels=c("T3","T1","T2"))
point.plot(biomasse,traitement2)
point.plot(biomasse/2,traitement2, add = TRUE)
```



3 Fonctions graphiques pour les analyses multivar?es

3.1 La fonction `multivar.polyg`

La fonction `multivar.polyg` permet de regrouper des points dans une analyse en faisant un polygone autour. On prends

Il faut :

- ANAcoo coordonnées des points d'une analyse multivariées (`__$points` pour une NMDS, `__$li` pour une AFC, etc.)
- FAC facteur qui permettra de séparer les polygones

```
library(vegan)
```

```
## Loading required package: permute
```

```
## Loading required package: lattice
```

```
## This is vegan 2.5-6
```

```
data("dune")
```

```
data("dune.env")
```

```
NMDS <- metaMDS(dune)
```

```
## Run 0 stress 0.1192678
```

```
## Run 1 stress 0.1183186
```

```
## ... New best solution
```

```
## ... Procrustes: rmse 0.02026635 max resid 0.06493567
```

```
## Run 2 stress 0.1192681
```

```
## Run 3 stress 0.1809579
```

```
## Run 4 stress 0.1192682
```

```
## Run 5 stress 0.1183187
```

```
## ... Procrustes: rmse 0.0001915233 max resid 0.0005272265
```

```
## ... Similar to previous best
```

```
## Run 6 stress 0.1980521
```

```
## Run 7 stress 0.1922243
```

```
## Run 8 stress 0.1183186
```

```
## ... Procrustes: rmse 5.781651e-05 max resid 0.0001597378
```

```
## ... Similar to previous best
```

```
## Run 9 stress 0.1183186
```

```
## ... New best solution
```

```
## ... Procrustes: rmse 3.945881e-05 max resid 0.0001234327
```

```
## ... Similar to previous best
```

```
## Run 10 stress 0.1192679
```

```
## Run 11 stress 0.1183186
```

```
## ... Procrustes: rmse 2.186933e-05 max resid 4.67649e-05
```

```
## ... Similar to previous best
```

```
## Run 12 stress 0.1183186
```

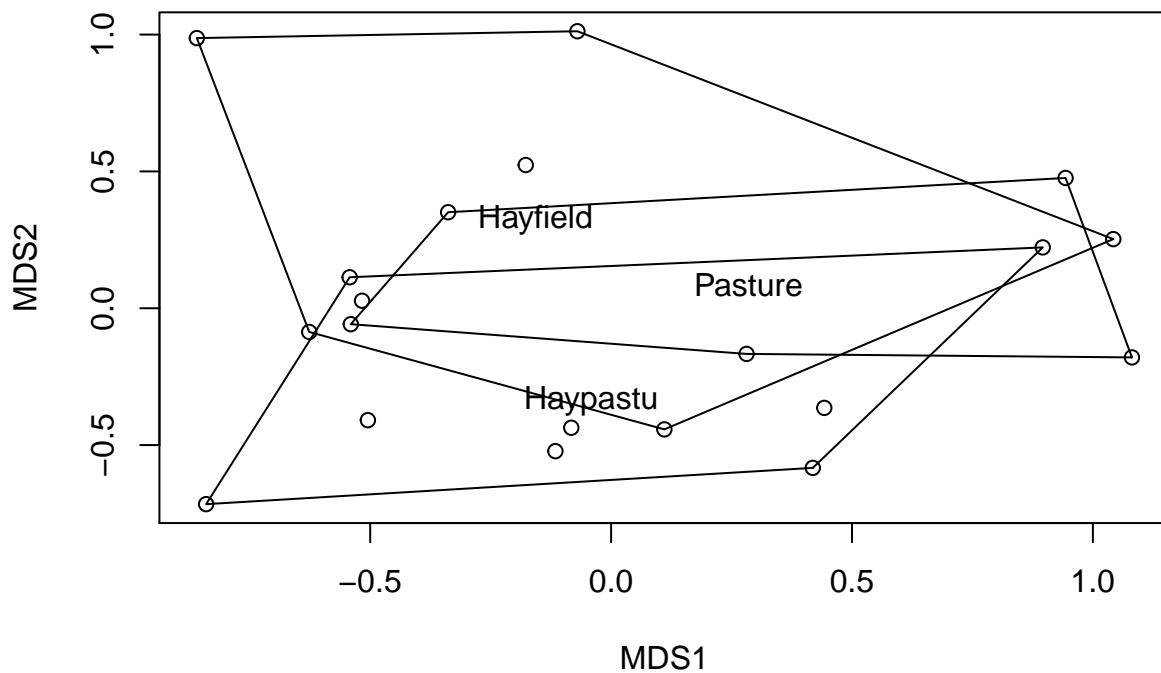
```
## ... Procrustes: rmse 1.209929e-05 max resid 4.214307e-05
```

```
## ... Similar to previous best
```

```
## Run 13 stress 0.1192687
```

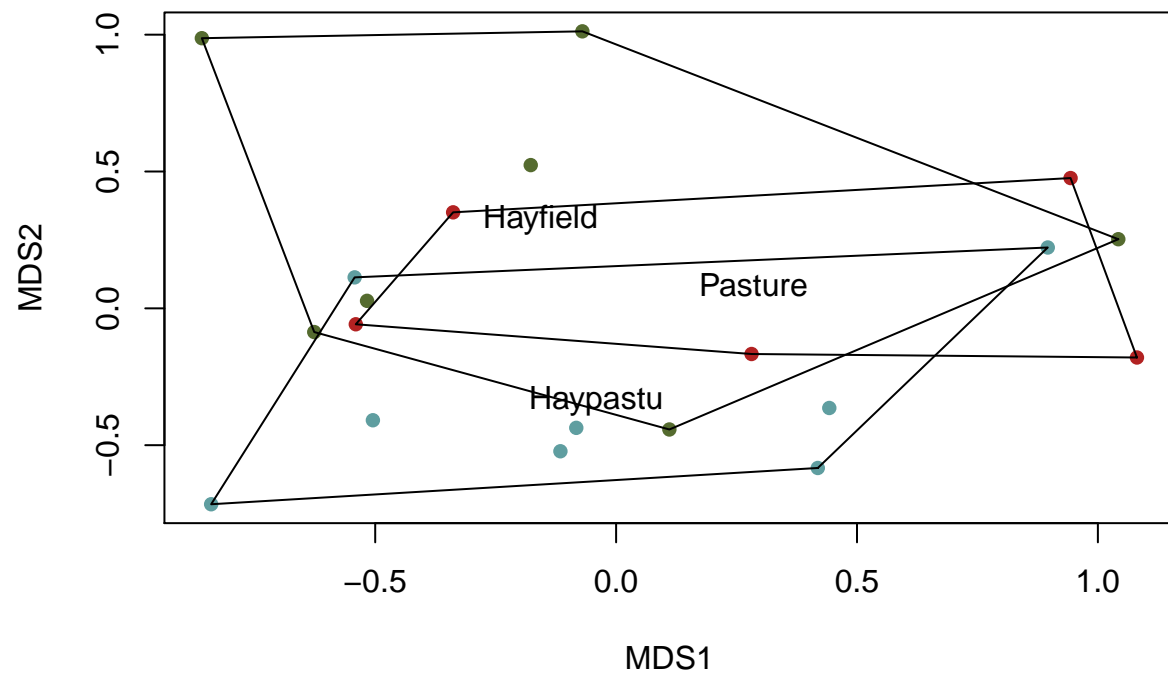
```
## Run 14 stress 0.1183186
## ... Procrustes: rmse 4.029756e-05  max resid 0.0001240249
## ... Similar to previous best
## Run 15 stress 0.1192678
## Run 16 stress 0.1901512
## Run 17 stress 0.1192682
## Run 18 stress 0.1192679
## Run 19 stress 0.1192681
## Run 20 stress 0.1183186
## ... Procrustes: rmse 1.223836e-05  max resid 3.589826e-05
## ... Similar to previous best
## *** Solution reached
```

```
multivar.polyg(ANAcoc = NMDS$points, FAC = dune.env$Use)
```

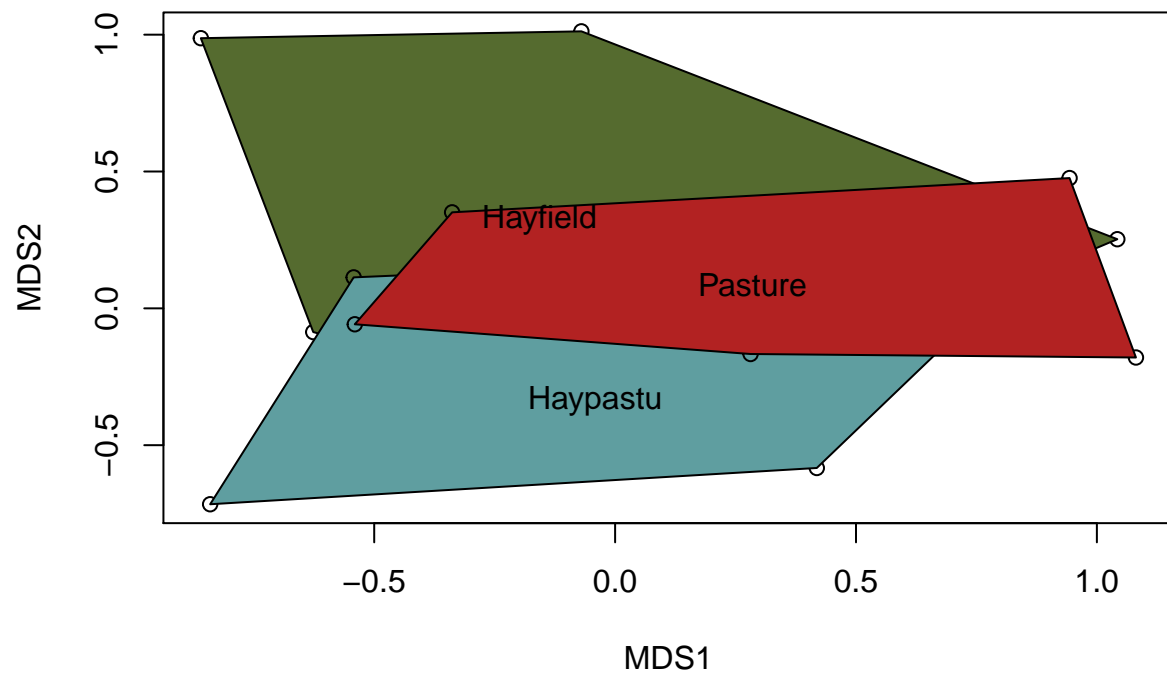


Avec d'autres arguments si besoin :

```
multivar.polyg(ANAcoc = NMDS$points, FAC = dune.env$Use,
  pch = 16, # forme des points des sites
  col_dot = c("darkolivegreen", "cadetblue", "firebrick")
  #couleur des points des sites
)
```

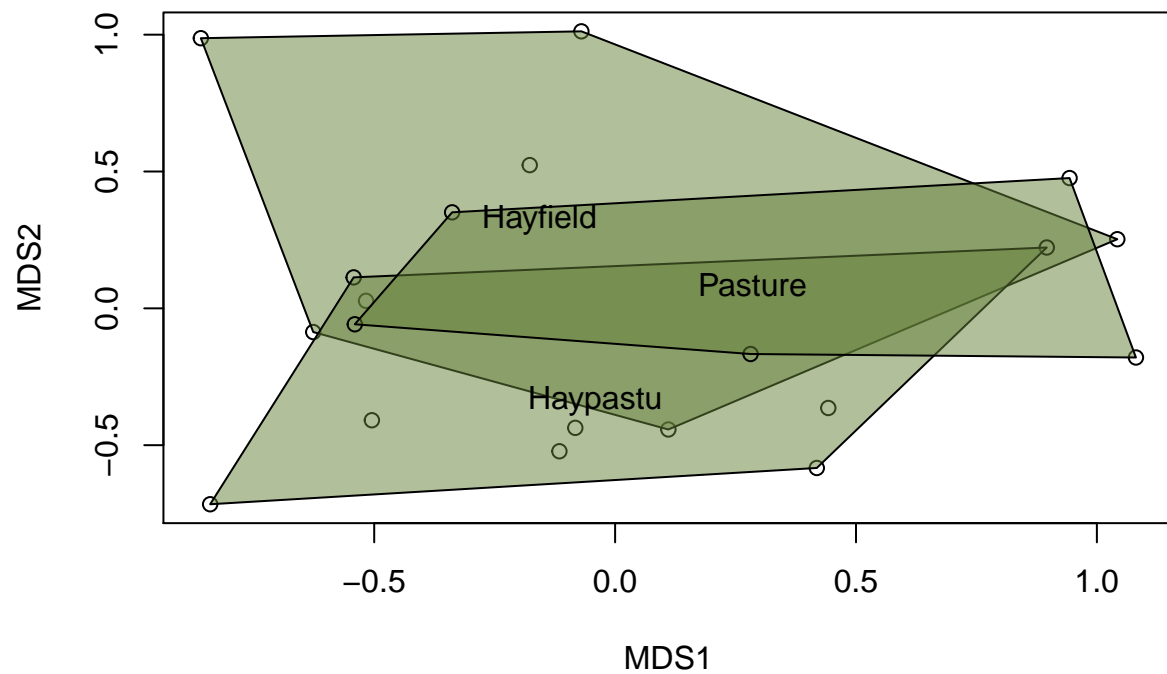


```
multivar.polyg(ANAcOO = NMDS$points, FAC = dune.env$Use,
  col_fill = c("darkolivegreen", "cadetblue", "firebrick")
  #couleur des polygones
)
```



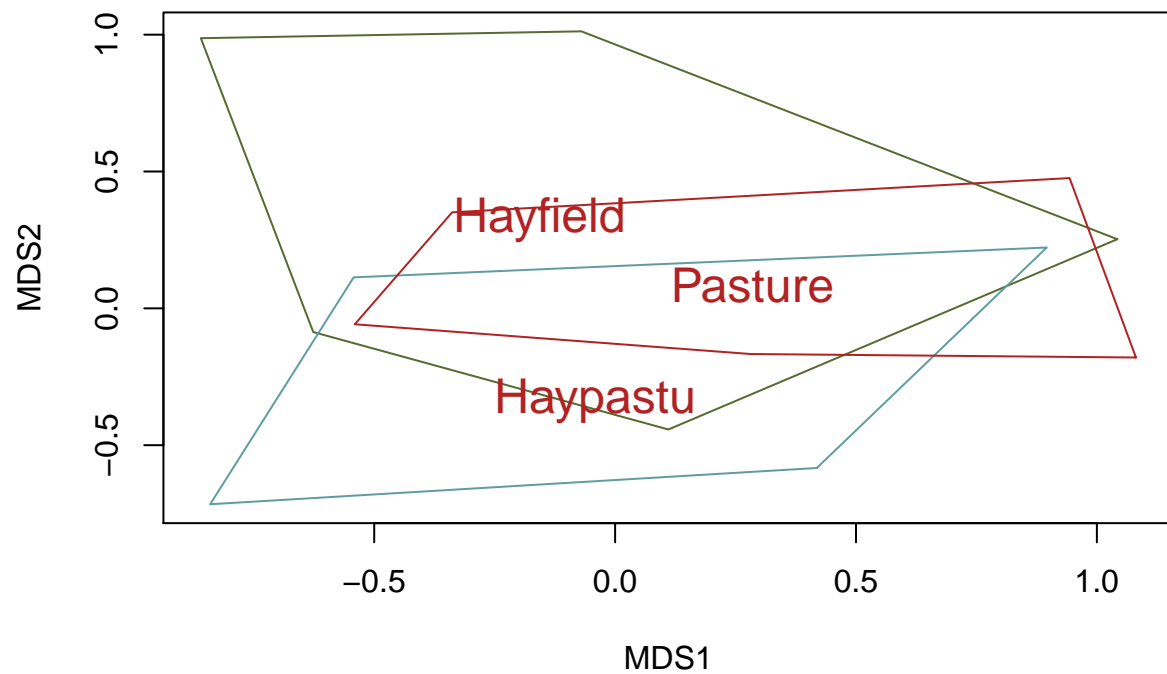
Mieux vaut utiliser la fonction `modif_coul` pour ajouter de la transparence :

```
multivar.polyg(ANAcOO = NMDS$points, FAC = dune.env$Use,
  col_fill = modif_coul(c("darkolivegreen", "cadetblue", "firebrick"), alpha = 0.5))
```



Et encore :

```
multivar.polyg(ANAcco = NMDS$points, FAC = dune.env$Use,
  col_bord = c("darkolivegreen","cadetblue","firebrick"),
  #couleur des bordures des polygones
  col_text = c("darkolivegreen","cadetblue","firebrick"),
  #couleur du texte
  cex_lab = 1.5, # taille des étiquettes
  dot = "no" # présence ou non des points des sites
)
```



Possibilité aussi avec les arguments `lab` de ne pas afficher les étiquettes des modalités, `new` de ne pas faire ça sur un nouveau graphe mais sur un graphe existant, et `sep` de faire une nouvelle fenêtre pour chaque modalité du facteur.

3.2 La fonction `MultiDyn`

3.3 La fonction `better_arrows`

3.4 La fonction `label.corV2`

La fonction `label.corV2` permet de calculer les corrélations des espèces avec les axes et d'afficher seulement les plus corrélées.

Il y a plusieurs arguments obligatoires :

- `COO_ESP` Coordonnées des espèces suites à une ordination, pour les deux premiers axes, après `dudi.pca` et `dudi.coa` c'est `ANALYSE$co[,1:2]`, après `decorana` c'est `ANALYSE$cproj[,1:2]`, après `metaMDS` c'est `ANALYSE$species[,1:2]`, etc.
- `COO_REL` Coordonnées des relevés suites à une ordination, pour les deux premiers axes, après `dudi.pca` et `dudi.coa` c'est `ANALYSE$li[,1:2]`, après `decorana` c'est `ANALYSE$rproj[,1:2]`, après `metaMDS` c'est `ANALYSE$points[,1:2]`, etc.
- `RELEVES` les relevés ayant servi à faire l'analyse multivariée
- `METHOD` Méthode de choix des espèces à afficher, soit `"P"` (défaut) - les espèces dont le p du test de spearman sont inférieures à la valeur de P sur au moins un axe sont affichées, soit `"RHO"` - les espèces dont le Rho du test de spearman sont supérieur à RHO sur au moins un axe sont affichées, soit `"N_base_P"` - les N espèces dont le P sont les plus faibles sont affichées, soit `"N_base_RHO"` - les N espèces dont les RHO sont les plus fort sont affichées.

Selon la méthode choisie, vous devrez aussi renseigner :

- `P` la p value maximum pour être affichée
- `RHO` le Rho minimum pour être affichée
- `N` le nombre d'espèces à afficher, attention, si des ex-aequo, pour ne pas surcharger, il y a moins d'espèces affichées que le N choisi

Et éventuellement :

- `COEF` un coef multiplicateur (pour exploser ou non la dispersion dans le plan)
- `ADD` par défaut, créer un graphe seul, si `add=T`, les noms d'especes sont ajouté au graphe déjà existant
- ... d'autres arguments de la fonction `text`

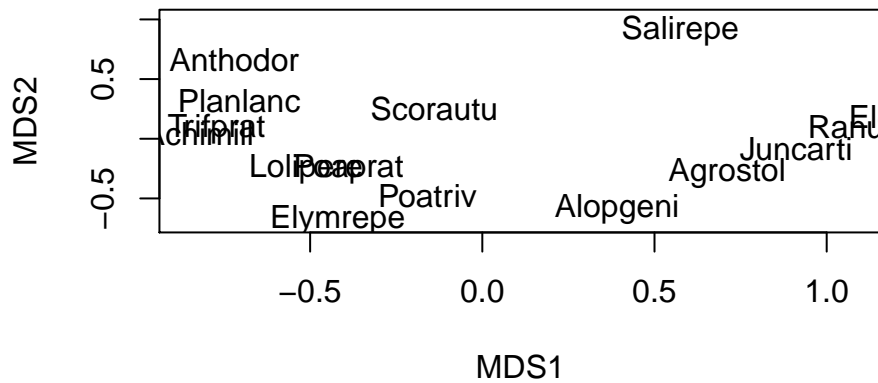
Pour l'exemple, on récupère les données `dune` du package `vegan` et on fait une NMDS :

```
library(vegan)
data("dune")
NMDS <- metaMDS(dune)
```

Usage de la fonction :

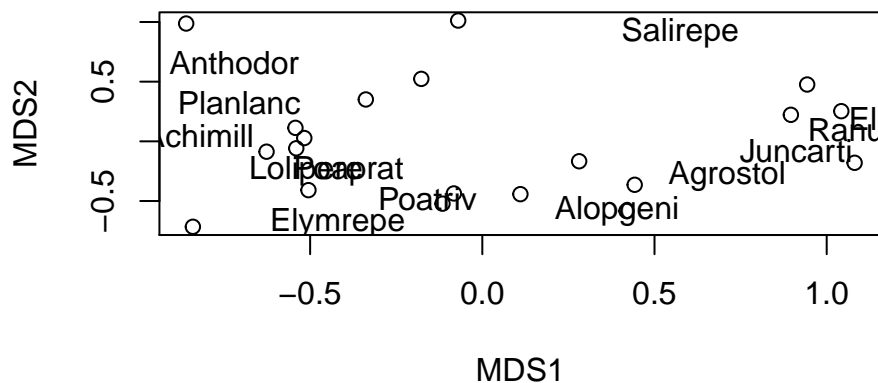
- En fonction du p, et sans graphe existant :

```
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
            RELEVES = dune, METHOD = "P", P = 0.05)
```



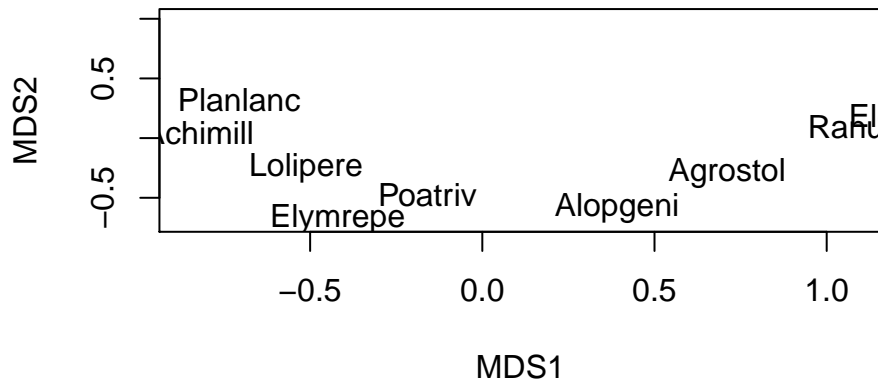
- En fonction du rho et avec graphe existant :

```
plot(NMDS$points[,1:2])
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
            RELEVES = dune, METHOD = "RHO", RHO = 0.5, ADD = T)
```



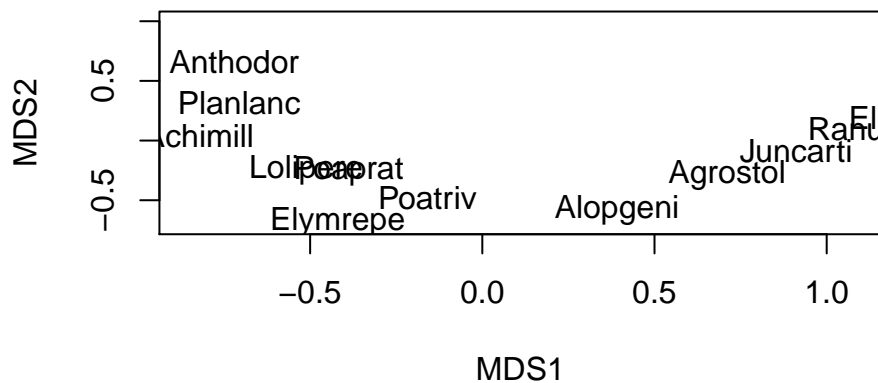
- Sans graphe précédent, par un nombre d'espèces en fonction du p :

```
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
            RELEVES = dune, METHOD = "N_base_P", N = 10)
```



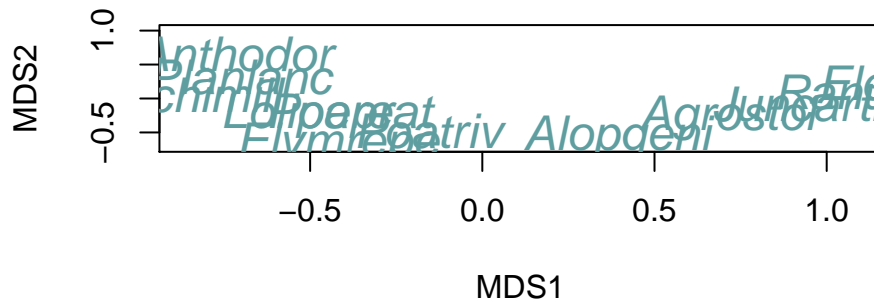
- Sans graphe précédent, par un nombre d'espèces en fonction du rho :

```
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
            RELEVES = dune, METHOD = "N_base_RHO", N = 15)
```



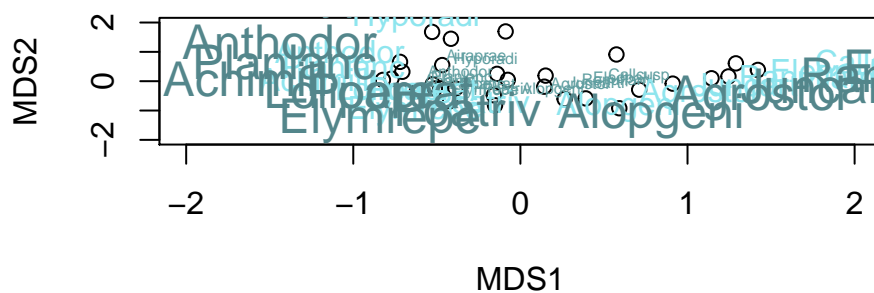
- Sans graphe précédent, par un nombre d'espèces en fonction du rho et en utilisant tout un tas d'autres arguments de la fonction `text`:

```
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
  RELEVES = dune, METHOD = "N_base_RHO", N = 15,
  col="cadetblue", cex=1.5, font = 3)
```



- Et pour voir les effets de l'argument `COEF` :

```
plot(NMDS$species, ylim = c(-2, 2), xlim = c(-2,2))
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
  RELEVES = dune, METHOD = "N_base_RHO", N = 15,
  col="cadetblue", cex=0.5, COEF = 0.5, ADD = T)
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
  RELEVES = dune, METHOD = "N_base_RHO", N = 15,
  col="cadetblue2", cex=1, COEF = 1.5, ADD = T)
label.corV2(COO_ESP = NMDS$species[,1:2], COO_REL = NMDS$points[,1:2],
  RELEVES = dune, METHOD = "N_base_RHO", N = 15,
  col="cadetblue4", cex=1.5, COEF = 2, ADD = T)
```



3.5 La fonction `label.corDCA`

3.6 La fonction `label.corNMDS`

4 Pour de la manipulation de donn es

4.1 La fonction `BBtransf`

4.2 La fonction `Classes_def`

4.3 La fonction `combin.tab`

4.4 La fonction `combin.tabV0`

5 Pour des t ches tr s sp cifiques :

5.1 Autour de l'indice de l'int grit  de la structure des communaut s

5.2 La fonction `ComStructIndices`

5.3 La fonction `structure.plot`

5.4 La fonction `structure.plotV2`

5.5 La fonction “

“

“

“

“

“

“