D. Chessel *Biométrie et Biologie Evolutive - Université Lyon1*

Consultations statistiques avec le logiciel 

Comment faire les analyses séparées dans une analyse des correspondances internes ?

Résumé

Une analyse des correspondances internes porte sur un paquet de sous-tableaux. Marie Semon demande une représentation des analyses séparées qui semble légitime.

.

Plan

1. La question

Elle est posée par Marie Semon :

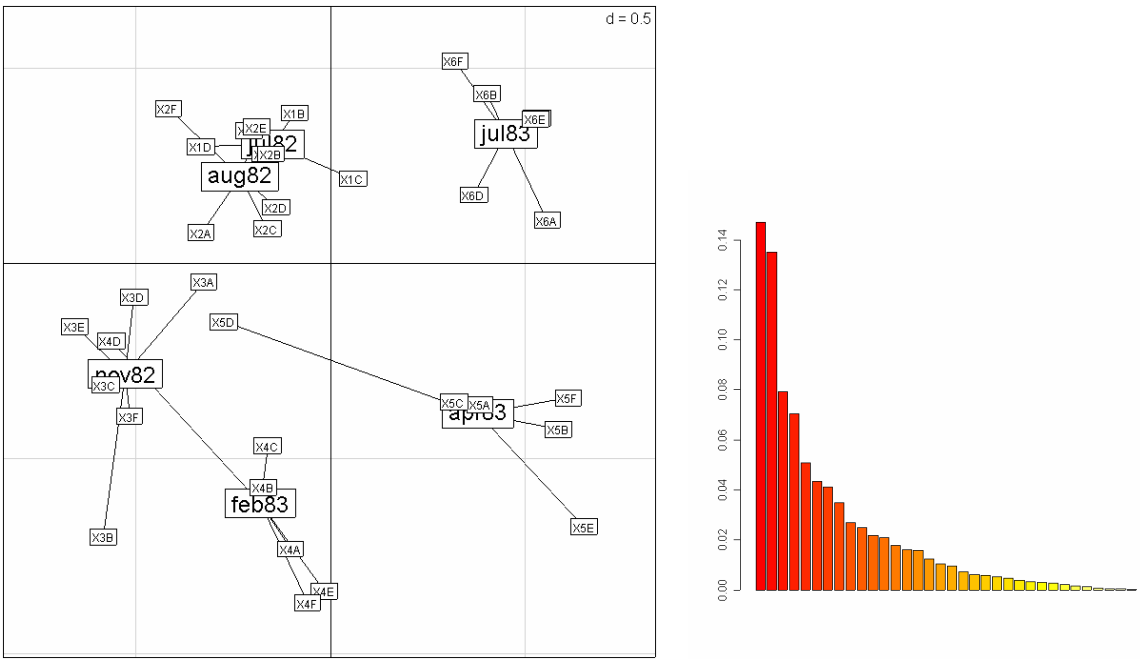
> *Est-ce qu'il y a une fonction dans ADE4 sous R permettant de produire directement les 9 graphes des valeurs propres associés aux 9 analyses de l'analyse descorrespondances internes? (cf Figure 1 de l'article de Lobry et Chessel, 2003 J. Appl. Genet.)*

Plus généralement, une Analyse des correspondances internes est une analyse qui prend en compte un tableau d'analyse des correspondances partagé éventuellement en blocs de lignes et/ou en blocs de colonnes. Comment faire les analyses des blocs élémentaires et représenter simultanément les graphes de valeurs propres.

Il n'y a pas encore de fonctions spécifiques pour faire cela, mais on peut essayer.

2. Approche du problème sur un exemple

```
data(ardeche)
coa1 <- dudi.coa(ardeche$tab,scann=F,nf=4)
s.class(coa1$co,ardeche$dat.fac,clab=1.5,cell=0,axesell=F)
s.label(coa1$co,clab=0.75,add.p=T)
barplot(coa1$eig)
```



Carte des colonnes et valeurs propres de l'AFC globale.

Ce tableau comporte 6 blocs de lignes et 4 blocs de colonnes :

```
ardeche
...
$col.blocks
jul82 aug82 nov82 feb83 apr83 jul83
    5     6     6     6     6     6
```

```
$row.blocks
Eph Ple Col Tri
 11   3  13  16
```

Il est donc formé de 24 sous-tableaux, qui donnent 24 analyses partielles. On peut utiliser les facteurs qui correspondent aux blocs :

```
row.fac=as.factor(rep(1:4,ardeche$row.blocks))
levels(row.fac)=names(ardeche$row.blocks)
col.fac=as.factor(rep(1:6,ardeche$col.blocks))
levels(col.fac)=names(ardeche$col.blocks)

listblocrow = split(ardeche$tab,row.fac)
listbloc = NULL
lapply(listblocrow, function(x)
      listbloc <<- c(listbloc,split(as.data.frame(t(x)),col.fac)))
names(listbloc)
lapply(listbloc,function(x) dudi.coa(x,scann=F)$eig)
```

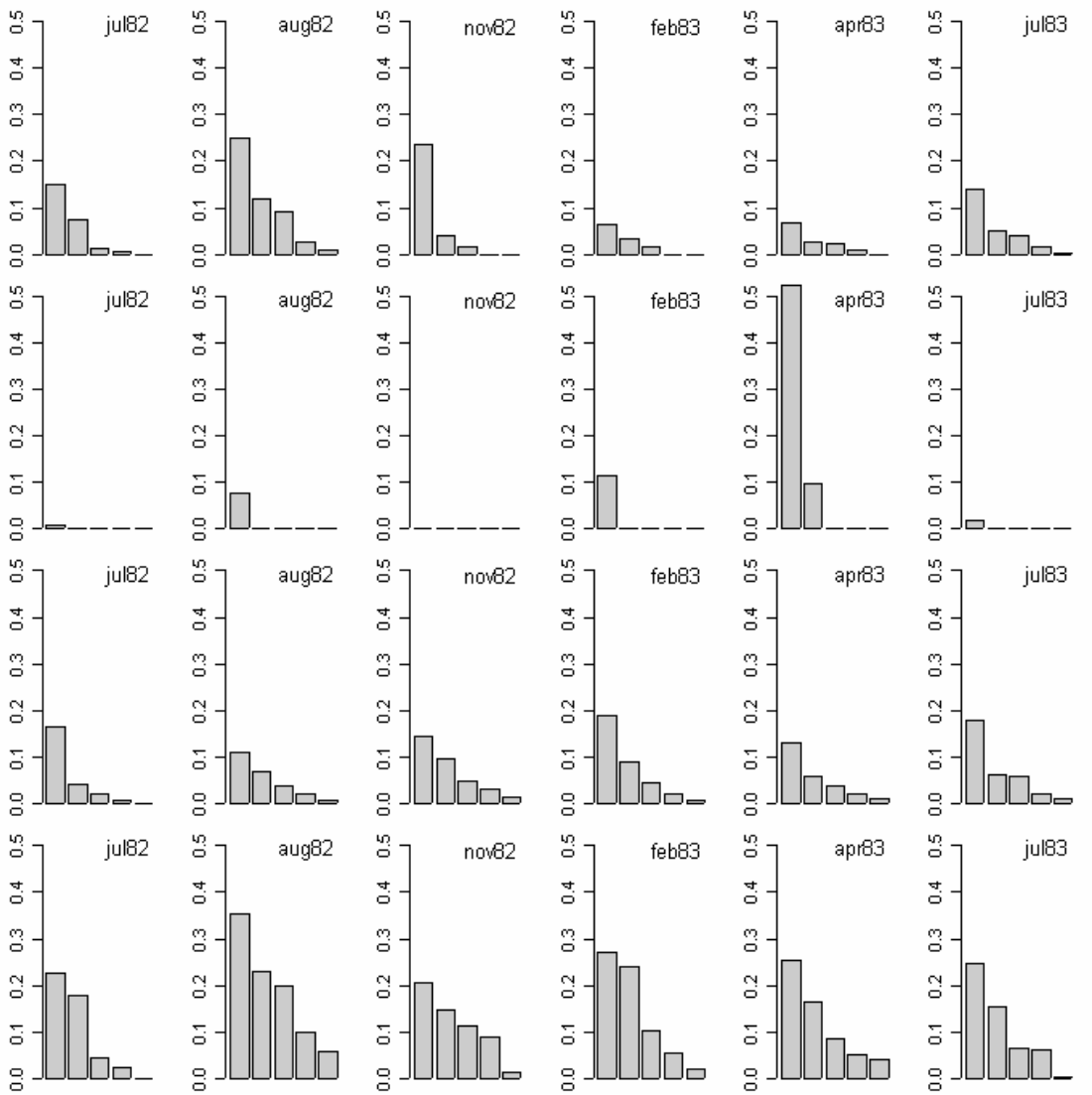
On a une difficulté sur un tableau trop réduit :

```
listbloc[[9]]
   Ple1 Ple2 Ple3
3A    1    0    1
3B    0    0    0
3C    0    0    0
3D    0    0    0
3E    0    0    0
3F    0    0    0
```

```
fun1 <- function(x) {
   x <- data.frame(x)
   if (nrow(x) <2) return (NULL)
   if (ncol(x) <2) return (NULL)
   sumlig <- apply(x,1,sum)
   if (sum(sumlig>0)<2) return (NULL)
   sumcol <- apply(x,2,sum)
   if (sum(sumcol>0)<2) return (NULL)
   return(dudi.coa(x,scann=F)$eig)
 }

lapply(listbloc,function(x) dudi.coa(x,scann=F)$eig)

listeig <- lapply(listbloc,fun1)

$jul82
[1] 0.148998 0.074894 0.012150 0.004734

$aug82
[1] 0.25046 0.11965 0.09266 0.02691 0.01106

$nov82
[1] 0.235540 0.039099 0.016575 0.000479

$feb83
[1] 0.0657711 0.0340359 0.0162237 0.0005215 0.0002836

$apr83
[1] 0.0661189 0.0270016 0.0235896 0.0107589 0.0004928

$jul83
[1] 0.139907 0.051011 0.041939 0.016276 0.003645

...

$feb83
[1] 0.190456 0.090912 0.044866 0.020459 0.008007
```

La fonction graphique utile est celle sepan.plot :

```
provi <- function (x, mfrow = c(4,6), csub = 2, ...) {
    opar <- par(ask = par("ask"), mfrow = par("mfrow"), mar = par("mar"))
    on.exit(par(opar))
    par(mar = c(0.6, 2.6, 0.6, 0.6))
    nbloc <- length(x)
    if (is.null(mfrow))
        mfrow <- n2mfrow(nbloc)
    par(mfrow = mfrow)
    if (nbloc > prod(mfrow))
        par(ask = TRUE)
    neig <- max(unlist(lapply(x,length)))
    maxeig <- max(unlist(x))
    for (ianal in 1:nbloc) {
        w <- x[[ianal]]
        scatterutil.eigen(w, xmax = neig, ymax = maxeig, wsel = 0,
            sub = names(x)[ianal], csub = csub, possub = "topright")
    }
}

provi(listeig)
```



Valeurs propres des analyses séparées : les noms des sous-tableaux sont insuffisants.

```
ww0=witwit.coa(coa1, ardeche$row.blocks, ardeche$col.blocks, scann = FALSE)
summary(ww0)
Internal correspondence analysis
class: witwit coa dudi
$call: witwit.coa(dudi = coa1, row.blocks = ardeche$row.blocks, col.blocks =
ardeche$col.blocks,
```

```
    scannf = FALSE)
2 axis-components saved
eigen values: 0.06858 0.06325 0.04253 0.03564 0.02911 ...

Eigen value decomposition among row blocks
    Axis1 Axis2
Eph 215   256
Ple 110   27
Col 110   415
Tri 566   302
sum 1000  1000

Eigen value decomposition among column blocks
      Comp1 Comp2
jul82 30    207
aug82 108   302
nov82 26    1
feb83 381   69
apr83 383   184
jul83 72    236
sum   1000  1000
```

On a une approche cohérente mais non identique entre l'ACI et les analyses séparées. La demande de Marie Semon est parfaitement justifiée.

3. Assemblage d'une solution

Les éléments sont assemblés dans la fonction **witwitsepan** (en annexe, coller dans un fichier texte et sourcer dans R, sera intégrée dans ade4 à la prochaine version).

Titre A ajouter à la fiche de witwit

Description **witwitsepan** donne le calcul et la représentation des valeurs propres des analyses séparées dans une Analyse des correspondances internes

Usage

```
witwitsepan <- function (ww, mfrow = NULL, csub = 2, plot = TRUE)
```

Arguments

ww: an object of class 'witwit'

mfrow: a vector of the form "c(nr,nc)", otherwise computed by a special own function 'n2mfrow'

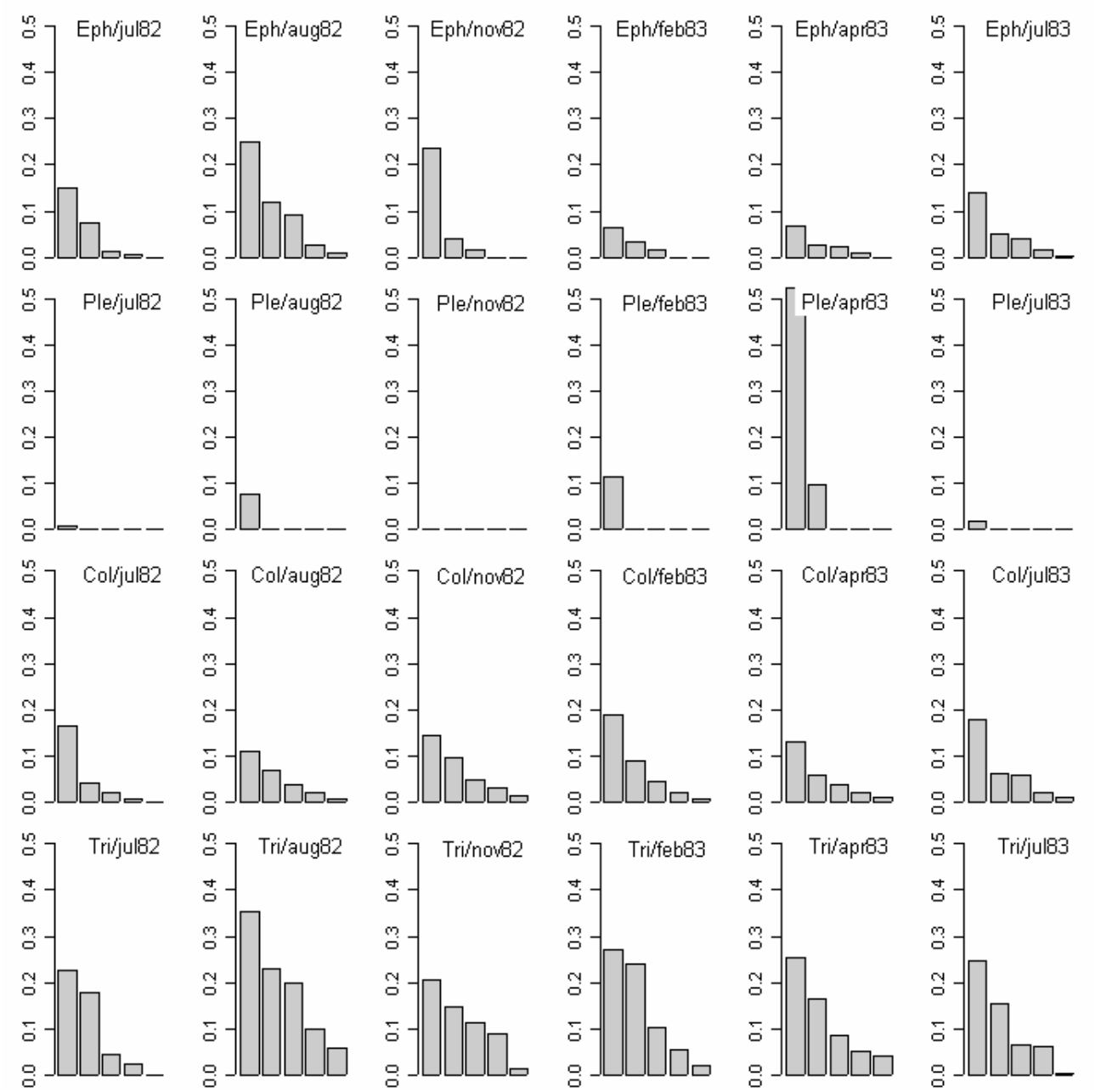
csub: a character size for the sub-titles, used with 'par("cex")*csub'

plot : if FALSE, numeric results are returned

Examples

Après ceux de witwit :

```
data(ardeche)
coa1 <- dudi.coa(ardeche$tab, scann = FALSE, nf = 4)
ww <- witwit.coa(coa1, ardeche$row.blocks, ardeche$col.blocks, scann = FALSE)

witwitsepan(ww,c(4,6))
```



Remarque : on pourrait améliorer le multifenêtrage.

```
witwitsepan <- function (ww, mfrow = NULL, csub = 2, plot = TRUE, ...) {
    if (!inherits(ww, "witwit")) stop ("witwit object expected")
    appel <- as.list(ww$call)
    rowblo <- eval(appel[[3]], sys.frame(0))
    colblo <- eval(appel[[4]], sys.frame(0))
    anal <- eval(appel[[2]], sys.frame(0))
    tab <- eval(as.list(anal$call)[[2]], sys.frame(0))

    rowfac=as.factor(rep(1:length(rowblo),rowblo))
    if (is.null(names(rowblo))) names(rowblo) <- as.character(1:length(rowblo))
    levels(rowfac)=names(rowblo)

    colfac=as.factor(rep(1:length(colblo),colblo))
    if (is.null(names(colblo))) names(colblo) <- as.character(1:length(colblo))
    levels(colfac)=names(colblo)

    listblocrow = split(tab,rowfac)
    listbloc = NULL
    lapply(listblocrow, function(x)
         listbloc <<- c(listbloc,split(as.data.frame(t(x)),col.fac)))

    fun1 <- function(x) {
        x <- data.frame(x)
        if (nrow(x) <2) return (NULL)
        if (ncol(x) <2) return (NULL)
        sumlig <- apply(x,1,sum)
        if (sum(sumlig>0)<2) return (NULL)
        sumcol <- apply(x,2,sum)
        if (sum(sumcol>0)<2) return (NULL)
        return(dudi.coa(x,scann=F)$eig)
    }

    names(listbloc) <- t(outer(names(rowblo),names(colblo),function(x,y)
paste(x,y,sep="/")))

    result <- lapply(listbloc,fun1)
    if (!plot) return(result)

    opar <- par(ask = par("ask"), mfrow = par("mfrow"), mar = par("mar"))
    on.exit(par(opar))
    par(mar = c(0.6, 2.6, 0.6, 0.6))
    nbloc <- length(result)
    if (is.null(mfrow))
        mfrow <- n2mfrow(nbloc)
    par(mfrow = mfrow)
    if (nbloc > prod(mfrow))
        par(ask = TRUE)
    neig <- max(unlist(lapply(result,length)))
    maxeig <- max(unlist(result))
    for (ianal in 1:nbloc) {
        w <- result[[ianal]]
        su0 <- names(result)[ianal]
        scatterutil.eigen(w, xmax = neig, ymax = maxeig, wsel = 0,
            sub = su0, csub = csub, possub = "topright")
    }
    return(invisible(result))

}
```