TD4 - Partial Least Squares

Courtenay Rebecca & Ducros Chloé & Lasson Marie

Contents

1 Introduction **1**
1.1 PCA vs PLS 1
1.2 Algorithme 1

2 Questions **2**

1 Introduction

1.1 PCA vs PLS

On se souvient que l'ACP correspondait à écrire une matrice de donnée X dans la base des vecteurs propres V. Si on note $\tilde{X}$ la matrice de données dans la nouvelle base, on avait la décomposition suivante : $X^T = V \cdot \tilde{X}^T$, soit : $X = \tilde{X} \cdot V^T$. Pour être tout à fait exacts, il fallait ajouter un résidu (une erreur) si jamais on ne projetait pas sur la base entière des vecteurs propres, mais sur seulement quelques uns d'entre eux (par exemple quand $V = (v_1, v_2)$). En notant F cette matrice d'erreur on avait alors l'équation de l'ACP qui était:

$$X = \tilde{X} \cdot V^T + F$$

La PLS est similaire: en notant deux jeux de données $X \in \mathbb{R}^{n,p}$ et $Y \in \mathbb{R}^{n,q}$, la PLS consiste à trouver les solutions de:

$$\begin{cases} X = T \cdot C^T + F_X \\ Y = W \cdot E^T + F_Y \end{cases}$$

où F_X et F_Y sont les erreurs résiduelles.

Dans la PLS, on cherche à maximiser la covariance entre T et W là où, dans l'ACP, on cherchait à maximiser la variance dans $\tilde{X}$.

1.2 Algorithme

On va reprendre et implémenter l'algorithme décrit dans la partie 3.5 du cours (page 23), en adaptant quelques notations. L'algorithme permet de résoudre l'équation décrite ci-dessus, la lettre ξ dans le cours correspondant à T dans notre équation. Le critère de convergence (étape 6) est établi sur w, c'est à dire que si le w d'une étape est très proche de celui de l'étape précédente, on considère qu'il y a convergence.

Rappel : pour un vecteur donné w, le produit $w^T w$ peut s'écrire simplement sous R: `sum(w^2)`

Voici le pseudo-code pour une composante:

```
PLS <- function(...) {
  si X et Y ne sont pas centrées-réduites, le faire
  initialiser w à la première colonne de Y
```

```
  initialiser w_old à w
  initialiser t,u,v
  tant que (w proche de w_old) {
    1.a calcul de u
    1.b normaliser u
    2. calcul de t
    3.a calcul de v
    3.b normaliser v
    4.a w_old prend la valeur de w
    4.b calcul du nouveau w
  }
  6. calcul de c et de e
  7. deflation: calcul des matrices résidues de X et Y
  renvoyer w,t,u,v,c,e et les résidus
}
```

Pour faire ce calcul pour plus de composantes, il faut refaire ces opérations sur les données dont la variance n'est pas encore expliquée, i.e. en utilisant les matrices résidues.

Pour pouvoir tester notre algorithme et le comparer, on aura besoin de:

```
library(matlib)
library(plotrix)
#install.packages("devtools")
library(devtools)
#install_github("mixOmicsTeam/mixOmics")
library(mixOmics)
data(nutrimouse)
X <- nutrimouse$gene
Y <- nutrimouse$lipid
```

2 Questions

0. BONUS : Expliquer avec vos mots pourquoi `sum(w^2)` correspond au produit $w^T w$. (1 point)

On sait que : $w^T w = \sum_{i=0} w_i * w_i$. De ce fait,`sum(w^2)` correspond au produit $w^T w$.

1. Implémenter l'algorithme de PLS. On prendra en entrée X, Y, le nombre de composantes et ε qui sera le seuil pour estimer la convergence (typiquement, on prendra souvent $\varepsilon \approx 10^{-6}$). L'appel que vous pouvez faire pour vérifier le bon fonctionnement est par exemple: PLS(X, Y, 2, 0.01). Il est conseillé de d'abord faire marcher l'algorithme pour une seule composante puis de l'étendre ensuite à un nombre plus grand de composantes à l'aide d'une boucle `for`. (12 points)

```
PLS <- function(X,Y,eps) {
  u <- matrix()
  t <- matrix()
  v <- matrix()
  list_comp <- c()

  w <- Y[,1]
  w <- as.matrix(w)
  w_old <- w

  a <- (t(X)%*%w)
  b <- sum(w^2)
```

```
  u <- a/b
  u <- u/norm(u)
  t <- X %*% u[,1]
  v <- (t(Y)%*%t)/(sum(t^2))
  v <- v/norm(v)
  w <- Y %*% v

  while (abs(norm(w-w_old)) > eps) {
    u <- (t(X)%*%w)/(sum(w^2))
    u <- u/norm(u)
    t <- X %*% u[,1]
    v <- (t(Y)%*%t)/(sum(t^2))
    v <- v/norm(v)
    w_old <- w
    w <- Y %*% v
  }

  c <- (t(X)%*%t)/(sum(t^2))
  e <- (t(Y)%*%t)/(sum(t^2))
  X <- X - t%*%t(c)
  Y <- Y - t%*%t(e)

  list_comp <- append(list_comp,
                      list("w"=w,"t"=t,"u"=u,"v"=v,"c"=c,"e"=e,"X"=X,"Y"=Y))
  return(list_comp)
}

PLS_FOR <- function(X,Y,nbcomp=2,eps) {
  X <- as.matrix(X)
  Y <- as.matrix(Y)
  if (mean(X) != 0) {
    X <- scale(X)
  }
  if (mean(Y) != 0) {
    Y <- scale(Y)
  }

  ws <- c()
  ts <- c()
  us <- c()
  vs <- c()
  cs <- c()
  es <- c()

  for (i in seq.int(1,nbcomp)) {
    res <- PLS(X,Y,0.01)
    X <- res$X
    Y <- res$Y
    ws <- cbind(ws,res$w)
    ts <- cbind(ts, res$t)
    us <- cbind(us,res$u)
    vs <- cbind(vs,res$v)
    cs <- cbind(cs,res$c)
```

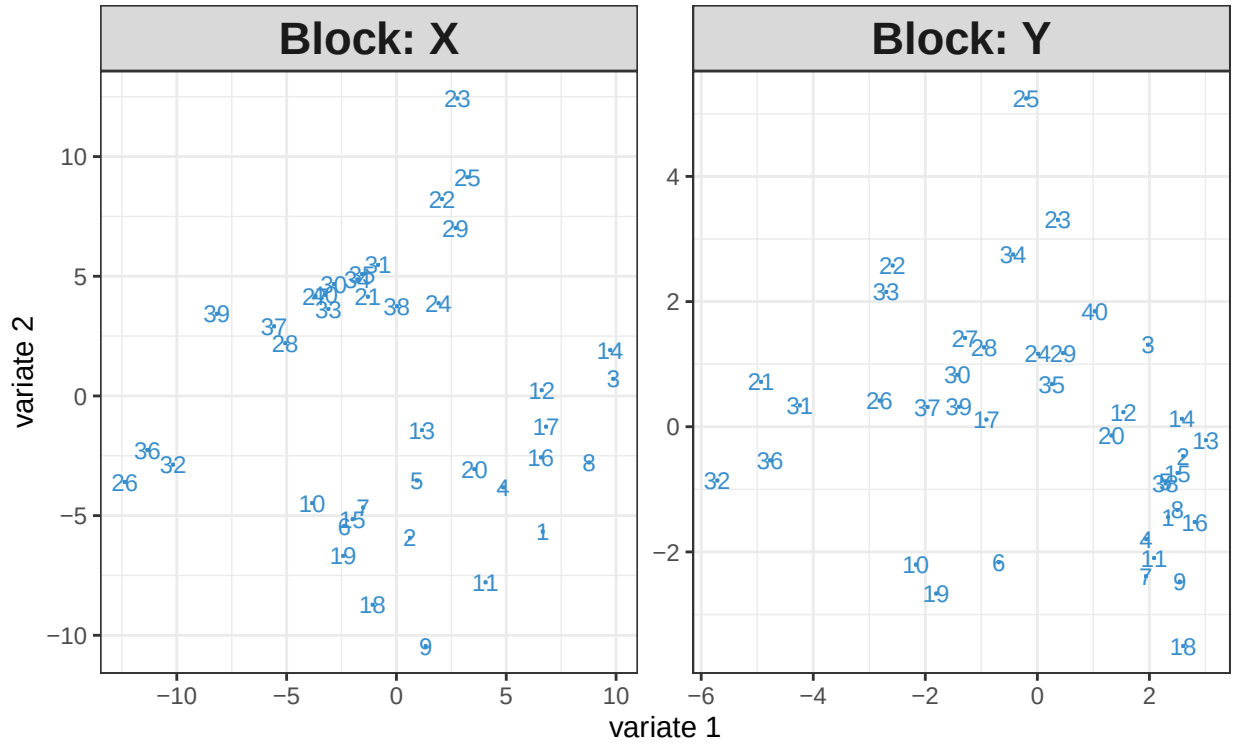
```
    es <- cbind(es,res$e)
  }
  return(list("w"=ws,"t"=ts,"u"=us,"v"=vs,"c"=cs,"e"=es,"X"=X,"Y"=Y))
}

algo <- PLS_FOR(X, Y, 2, 0.01)
```

2. Tracer le graphe des individus (avec deux composantes) dans les blocs X et Y avec la librairie `mixOmics`, puis avec votre algorithme. (4 points)

Pour les abscisses, nous avons multiplié par (-1) pour optenir le même graphique que la fonction PLS de R.

```
res <- pls(X,Y,ncomp=2)
plotIndiv(res)
```

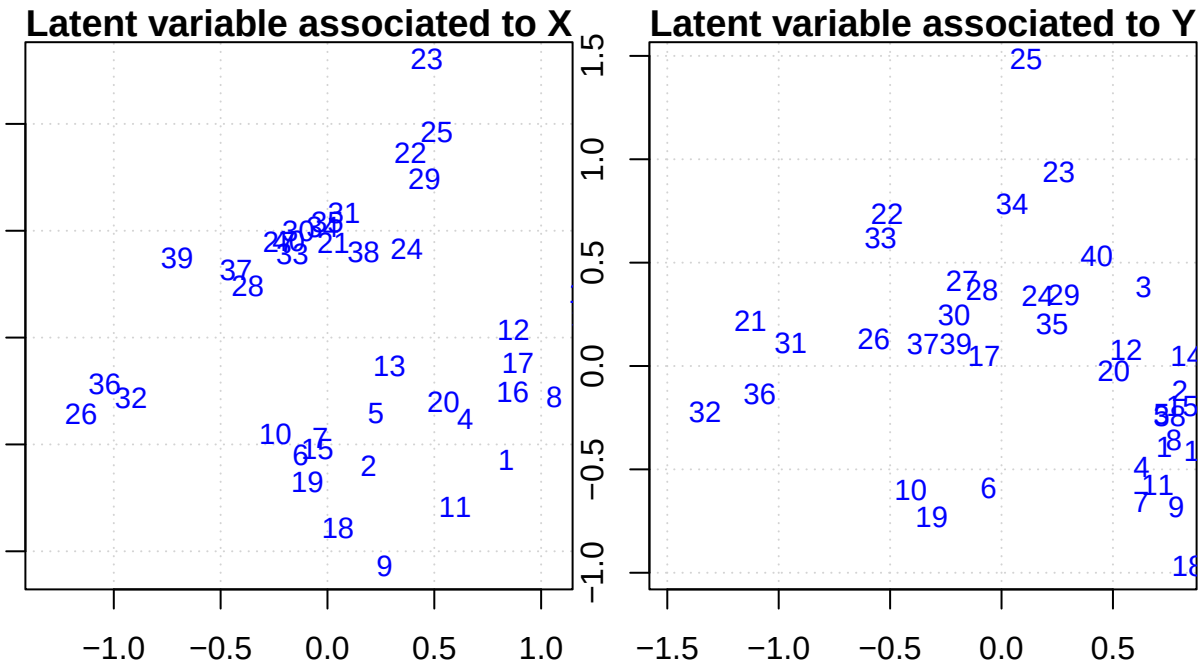



```
par(mfrow=c(1,2))
par(mar=c(1, 1, 1, 1),pty = "s")

ax1 <- algo$t[,1]*(-1)
ax2 <- algo$t[,2]
plot(ax1,ax2,
     xlab="Component 1 - Latent variable associated to X",
     ylab="Component 2 - Latent variable associated to X",
     main="Latent variable associated to X",
     cex=0,
     panel.first=grid())#X
text(ax1,ax2,labels=row.names(algo$X),cex = 0.9, pos = 4, col = "blue")

ax11 <- algo$w[,1]*(-1)
ax21 <- algo$w[,2]
```

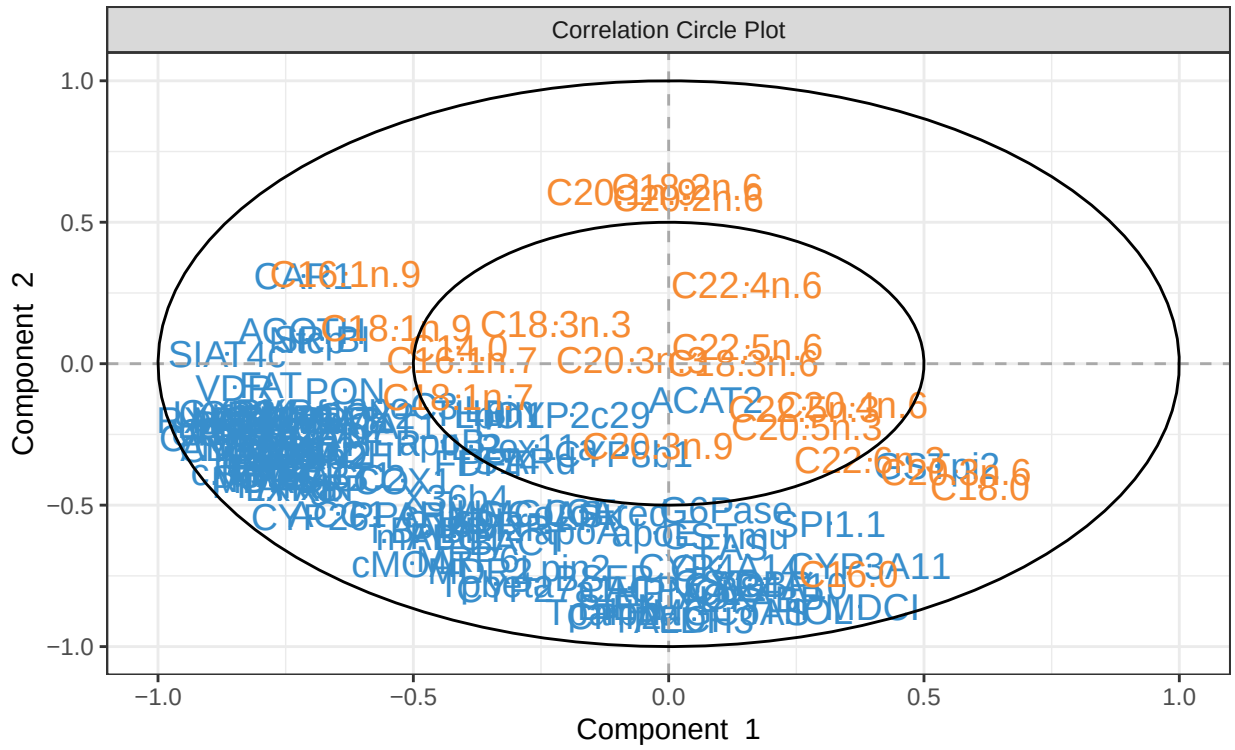
```
plot(ax11, ax21,
     xlab="Component 1 - Latent variable associated to Y",
     ylab="Component 2 - Latent variable associated to Y",
     main = "Latent variable associated to Y",
     panel.first=grid(),
     cex=0)#Y
text(ax11,ax21,labels=row.names(algo$X),cex = 0.9, pos = 4, col = "blue")
```



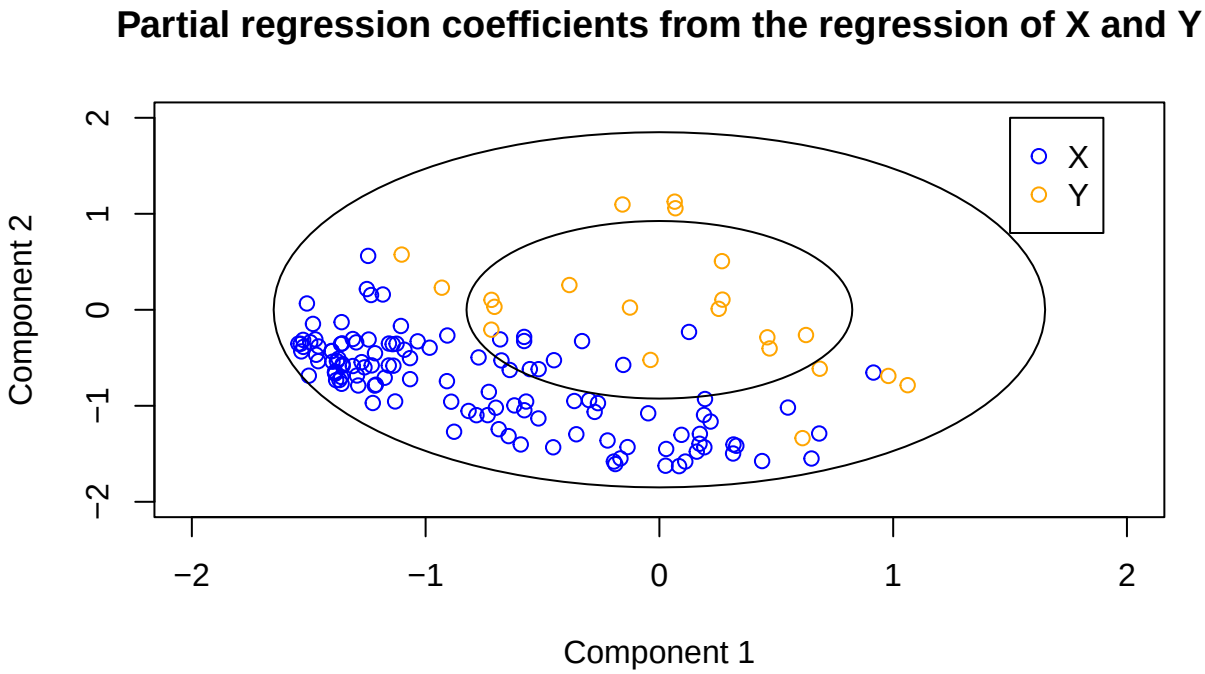
3. Tracer le cercle des variables (avec deux composantes) avec la librairie `mixOmics`, puis avec votre algorithme.

Ici, nous n'avons pas besoin de multiplier par (-1), les données des variables sont déjà correctement ordonnées.

```
plotVar(res)
```




```
plot(-algo$c[,1],algo$c[,2],col = "blue",
     xlab="Component 1",ylab="Component 2",xlim=c(-2,2), ylim=c(-2,2),
     main = "Partial regression coefficients from the regression of X and Y")
points(-algo$e[,1],algo$e[,2], col = "orange")
draw.ellipse(0,0,1.65,1.85,lty=1,lwd=1)
draw.ellipse(0,0,0.825,0.925)
legend(1.5,2,legend=c("X", "Y"),
       col=c("blue","orange"), pch=1)
```


Partial regression coefficients from the regression of X and Y
Component 2
Component 1
X
Y
−2
−1
0
1
2