```
#########################################################################
###########
#########################################################################
###########
#
#
# PhytoClus.R
#
#
#
# R-Code for implementing functional data analysis and classification
methods to     #
# explore phytoplankton detection by using the raw data of a CytoSub flow
cytometer  #
#
#
# Anthony MALKASSIAN Oct. 2010
#
#########################################################################
###########
#########################################################################
###########



library(fda)
library(cluster)      #the 4 libraries used for data analysis and display
library(evd)
library(rgl)



#########################################################################
###########
#########################################################################
###########
#
#
# 1) the function transfd() ... from raw data to functional data
#
#
#
#input:
#
#  pop is the matrix of sampling points (raw data) with each column
#
#  j corresponding to one of the 5 channels (fws sws flo flr fly)
#
#
#
#output:
#
#   ppnorm  is the decomposition in Fourier basis (functional data) after
#
#   a smoothing step and a normalisation step
#
#########################################################################
###########
```

```
##############################################################################
############

transfd<-function (pop,j)
{
 p=pop#p=by(pop,pop[,1],list)
    xtrans=indiv=vector()
    for ( i in seq(1,length(p))){
    xx1  <- seq(1,length(p[[i]][,j]),len=1001)
    s1<-smooth.spline(p[[i]][,j],spar=0.5)
    xtrans<-cbind(xtrans,predict(s1,xx1)$y)
        }
basis=create.fourier.basis(c(0,1),21)
ppt=data2fd(xtrans,basisobj=basis)
  coefnorm=coefnorms=vector()
  for (i in seq(1,length(p)))
        {
        if (max(p[[i]][,j])>10)
  (coefnorm = ppt$coefs[,i]/ppt$coefs[1,i])
  else coefnorm=rep(0,21)
  coefnorms=cbind(coefnorms,coefnorm)
        }
ppnorm=fd(coefnorms,basis)
return(ppnorm)  }



##############################################################################
############
##############################################################################
############
#
#
#  2) the function bagging ()
#
#
#
#input :
#
#   tab = matrix of data
#
#   flag = vector containing the real classes of  belonging
#
#   nbgroup = number of classes for the partition step
#
#   nbtir = number of bootstrap samples
#
#output :
#
#   resul = matrix of classification success
#
##############################################################################
############
##############################################################################
############

bagging<-function(tab,flag,nbgroup, nbtir)
{
nba1=round(2*(dim(tab)[1])/3)
sel=sort(sample(1:dim(tab)[1],nba1,replace=F))
```

```
sel2=setdiff(seq(1,dim(tab)[1]),sel)
a1= tab[sel,]
deucl<-function(x,y) {sqrt(sum((x-y)^2))}
result=resul=vector()
  for (d in seq(1,nbtir)){
  tirage=sort(sample(1:nba1,nba1,replace=TRUE))
  kme=pam(daisy(a1[tirage,]),nbgroup, diss =TRUE)
  #kme=fanny(daisy(a1[tirage,]),nbgroup, diss =TRUE)
  gp=by(tab[sel[tirage],],kme$cluster,list)
  gpm=lapply(gp,mean)
  gpmtab= t(matrix(unlist(gpm),ncol=nbgroup))
  compare=flag[sel[tirage]]
  tx=by(compare,kme$cluster,list)
  txtab=lapply(tx,table)
  txgrp=lapply(txtab,which.max)
  txgp=as.numeric(lapply(txgrp,names))
  la2= length(sel2)
  a2=tab[sel2,]
  test=flag[sel2]
  res=vector()
    for (n in seq(1,la2)){
    qmin=vector()
    qmin=apply(gpmtab,1,y=a2[n,],deucl)
    res=c(res,is.element(test[n],unlist(txgp)[which.min(qmin)]) )
    }
  resul=rbind(resul,res)
  }

return(resul)
}

#  the function baggingbis() is a modified version of bagging()
#  allowing to keep the classification success for each
#  individual taken separately  and compute the classificaton
#  success for each strain

baggingbis<-function(tab,flag,nbgroup, nbtir)
{
nba1=round(2*(dim(tab)[1])/3)
sel=sort(sample(1:dim(tab)[1],nba1,replace=F))
sel2=setdiff(seq(1,dim(tab)[1]),sel)
a1= tab[sel,]
deucl<-function(x,y) {sqrt(sum((x-y)^2))}
result=resul=vector()
  for (d in seq(1,nbtir)){
  tirage=sort(sample(1:nba1,nba1,replace=TRUE))
  kme=pam(daisy(a1[tirage,]),nbgroup, diss =TRUE)
  #kme=fanny(daisy(a1[tirage,]),nbgroup, diss =TRUE)
  gp=by(tab[sel[tirage],],kme$cluster,list)
  gpm=lapply(gp,mean)
  gpmtab= t(matrix(unlist(gpm),ncol=nbgroup))
  compare=flag[sel[tirage]]
  tx=by(compare,kme$cluster,list)
  txtab=lapply(tx,table)
  txgrp=lapply(txtab,which.max)
  txgp=as.numeric(lapply(txgrp,names))
  la2= length(sel2)
  a2=tab[sel2,]
  test=flag[sel2]
  res=vector()
    for (n in seq(1,la2)){
```

```
    qmin=vector()
    qmin=apply(gpmtab,1,y=a2[n,],deucl)
    res=c(res,is.element(test[n],unlist(txgp)[which.min(qmin)]) )
    }
  resul=rbind(resul,res)
  }
return(list(resul,flag[sel2]) )
}



##############################################################################
############
##############################################################################
############
#
#
#  3) the function dist_inv ()
#
#
#
#
#
# input :
#
#   cotir = matrix containing the Fourier coefficients in column
#
#   nbbase = number of basis functions used for the decomposition
#
#           (always an odd number)
#
# output :
#
#   dis = matrix of pairwise distances invariant to orientation
#
##############################################################################
############
##############################################################################
############

dist_inv<-function (cotir,nbbase)
{
inv=c(rep(c(1,-1),(nbbase-1)/2),1)
coefinv=cotir*inv
xtx=t(cotir)%*%cotir
xinvtx=t(coefinv)%*%cotir
dxtx=diag(xtx)
nbind=dim(cotir)[2]
    ppp=pppinv=vector()
    for (j in 1:nbind)
    {
    pp=ppinv=vector()
        for ( k in 1:nbind )
        {
        pp=c(pp,dxtx[k]+dxtx[j]-2*xtx[k,j])
        ppinv=c(ppinv,dxtx[k]+dxtx[j]-2*xinvtx[k,j])
        }
    ppp=cbind(ppp,pp)
    pppinv=cbind(pppinv,ppinv)
    }
```

```
disn=sqrt(ppp)
disv=sqrt(pppinv)
dis=pmin(disn,disv)
return(dis)  }


#
#   fast version without loop
#

dist_inv2<-function (cotir,nbbase)
{

inv=c(rep(c(1,-1),(nbbase-1)/2),1)
coefinv=cotir*inv
xtx=t(cotir)%*%cotir
xinvtx=t(coefinv)%*%cotir
nbind=dim(cotir)[2]

eucl<-function(x,y) {y-2*x}
aa=apply(xtx,1,eucl,y=diag(xtx))
dxtx=t(matrix(nrow=dim(xtx)[1],ncol=dim(xtx)[1],data=diag(xtx)))
eucli=sqrt(aa+dxtx)
diag(eucli)

aa2=apply(xinvtx,1,eucl,y=diag(xtx))
dxtx2=t(matrix(nrow=dim(xtx)[1],ncol=dim(xtx)[1],data=diag(xtx)))
eucli2=sqrt(aa2+dxtx2)

dis=pmin(eucli,eucli2)
return(dis)  }
```