

Package ‘funHDDC’

March 17, 2021

Type Package

Title Univariate and Multivariate Model-Based Clustering in Group-Specific Functional Subspaces

Version 2.3.1

Date 2021-03-17

Author A Schmutz, J. Jacques & C. Bouveyron

Maintainer Charles Bouveyron <charles.bouveyron@gmail.com>

Depends MASS,fda, R (>= 3.1.0)

Description The funHDDC algorithm allows to cluster functional univariate (Bouveyron and Jacques, 2011, <doi:10.1007/s11634-011-0095-6>) or multivariate data (Schmutz et al., 2018) by modeling each group within a specific functional subspace.

License GPL-2

Encoding UTF-8

LazyData false

RoxygenNote 6.0.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2021-03-17 10:40:02 UTC

R topics documented:

funHDDC	2
mfPCA	6
plot.fD	7
plot.mfPCA	10
predict.funHDDC	11
slopeHeuristic	13
triangle	15
trigo	15

funHDDC	<i>Clustering univariate and multivariate functional data in group-specific functional subspaces</i>
---------	--

Description

The funHDDC algorithm allows to cluster functional univariate (Bouveyron and Jacques, 2011, <doi:10.1007/s11634-011-0095-6>) or multivariate data (Schmutz et al., 2018) by modeling each group within a specific functional subspace.

Usage

```
funHDDC(data,K=1:10,model="AkjBkQkDk",threshold=0.2,itermax=200,eps=1e-6,init="kmeans",
criterion="bic",algo='EM', d_select="Cattell", init.vector=NULL, show=TRUE,
mini.nb=c(5, 10),min.individuals=2, mc.cores=1, nb.rep=1, keepAllRes=TRUE,
kmeans.control = list(), d_max=100)
```

Arguments

data	In the univariate case: a functional data object produced by the fda package, in the multivariate case: a list of functional data objects.
K	The number of clusters, you can test one partition or multiple partitions at the same time, for example K=2:10.
model	The chosen model among 'AkjBkQkDk', 'AkjBQkDk', 'AkBkQkDk', 'ABkQkDk', 'AkBQkDk', 'ABQkDk'. 'AkjBkQkDk' is the default. You can test multiple models at the same time with the command c(). For example c("AkjBkQkDk","AkjBQkDk").
threshold	The threshold of the Cattell' scree-test used for selecting the group-specific intrinsic dimensions.
itermax	The maximum number of iterations.
eps	The threshold of the convergence criterion.
init	A character string. It is the way to initialize the E-M algorithm. There are five ways of initialization: "kmeans" (default), "param", "random", "mini-em" or "vector". See details for more information. It can also be directly initialized with a vector containing the prior classes of the observations.
criterion	The criterion used for model selection: bic (default) or icl. But we recommend using slopeHeuristic function provided in this package.
algo	A character string indicating the algorithm to be used. The available algorithms are the Expectation-Maximisation ("EM"), the Classification E-M ("CEM") and the Stochastic E-M ("SEM"). The default algorithm is the "EM".
d_select	Either "Cattell" (default) or "BIC". This parameter selects which method to use to select the intrinsic dimensions of subgroups.
init.vector	A vector of integers or factors. It is a user-given initialization. It should be of the same length as of the data. Only used when init="vector".

show	Use show = FALSE to settle off the informations that may be printed.
mini.nb	A vector of integers of length two. This parameter is used in the “mini-em” initialization. The first integer sets how many times the algorithm is repeated; the second sets the maximum number of iterations the algorithm will do each time. For example, if init=“mini-em” and mini.nb=c(5,10), the algorithm will be launched 5 times, doing each time 10 iterations; finally the algorithm will begin with the initialization that maximizes the log-likelihood.
min.individuals	This parameter is used to control for the minimum population of a class. If the population of a class becomes strictly inferior to 'min.individuals' then the algorithm stops and gives the message: 'pop<min.indiv.'. Here the meaning of "population of a class" is the sum of its posterior probabilities. The value of 'min.individuals' cannot be lower than 2.
mc.cores	Positive integer, default is 1. If mc.cores>1, then parallel computing is used, using mc.cores cores. Warning for Windows users only: the parallel computing can sometimes be slower than using one single core (due to how parLapply works).
nb.rep	A positive integer (default is 1 for kmeans initialization and 20 for random initialization). Each estimation (i.e. combination of (model, K, threshold)) is repeated nb.rep times and only the estimation with the highest log-likelihood is kept.
keepAllRes	Logical. Should the results of all runs be kept? If so, an argument all_results is created in the results. Default is TRUE.
kmeans.control	A list. The elements of this list should match the parameters of the kmeans initialization (see kmeans help for details). The parameters are “iter.max”, “nstart” and “algorithm”.
d_max	A positive integer. The maximum number of dimensions to be computed. Default is 100. It means that the intrinsic dimension of any cluster cannot be larger than d_max. It quickens a lot the algorithm for datasets with a large number of variables (e.g. thousands).

Details

If you choose init="random", the algorithm is run 20 times with the same model options and the solution which maximises the log-likelihood is printed. This explains why sometimes with this initialization it runs a bit slower than with 'kmeans' initialization.

If the warning message: "In funHDDC(...) : All models diverged" is printed, it means that the algorithm found less classes than the number you choose (parameter K). Because of the EM algorithm, it should be because of a bad initialization of the EM algorithm. So you have to restart the algorithm multiple times in order to check if, with a new initialization of the EM algorithm the model converges, or if there is no solution with the number K you choose.

The different initializations are:

“param”: it is initialized with the parameters, the means being generated by a multivariate normal distribution and the covariance matrix being common to the whole sample.

“mini-em”: it is an initialization strategy, the classes are randomly initialized and the E-M algorithm makes several iterations, this action is repeated a few times (the default is 5 iterations and 10 times), at the end, the initialization chosen is the one which maximises the log-likelihood (see mini.nb for more information about its parametrization).

“random”: the classes are randomly given using a multinomial distribution

“kmeans”: the classes are initialized using the kmeans function (with: algorithm="Hartigan-Wong"; nstart=4; iter.max=50); note that the user can use his own arguments for kmeans using the dot-dot-dot argument

A prior class vector: It can also be directly initialized with a vector containing the prior classes of the observations. To do so use init="vector" and provide the vector in the argument init.vector.

Note that the BIC criterion used in this function is to be maximized and is defined as $2*LL-k*log(n)$ where LL is the log-likelihood, k is the number of parameters and n is the number of observations.

Value

d	The number of dimensions for each cluster.
a	Values of parameter a for each cluster.
b	Values of parameter b for each cluster.
mu	The mean of each cluster in the original space.
prop	The proportion of individuals in each cluster.
loglik	The maximum of log-likelihood.
loglik_all	The log-likelihood at each iteration.
posterior	The posterior probability for each individual to belong to each cluster.
class	The clustering partition.
BIC	The BIC value.
ICL	The ICL value.
complexity	the number of parameters that are estimated.
all_results	if multiple number of clusters are tested or multiple models, results of each models are stored here

References

- C.Bouveyron and J.Jacques, Model-based Clustering of Time Series in Group-specific Functional Subspaces, Advances in Data Analysis and Classification, vol. 5 (4), pp. 281-300, 2011 <doi:10.1007/s11634-011-0095-6>
- C. Bouveyron, L. Cheze, J. Jacques, P. Martin and A. Schmutz, Clustering multivariate functional data in group-specific functional subspaces, Preprint HAL 01652467, 2017.

Examples

```
#Univariate example
data("trigo")
cls = trigo[,101]
basis<- create.bspline.basis(c(0,1), nbasis=25)
```

```

var1<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[,1:100]),fdParobj=basis)$fd

res.uni<-funHDDC(var1,K=2,model="AkBkQkDk",init="kmeans",threshold=0.2)
table(cls,res.uni$class,dnn=c("True clusters","FunHDDC clusters"))
plot(var1,col=res.uni$class)

#Multivariate example
data("triangle")
cls2 = triangle[,203]
basis<- create.bspline.basis(c(1,21), nbasis=25)
var1<-smooth.basis(argvals=seq(1,21,length.out = 101),y=t(triangle[,1:101]),fdParobj=basis)$fd
var2<-smooth.basis(argvals=seq(1,21,length.out = 101),y=t(triangle[,102:202]),fdParobj=basis)$fd

res.multi<-funHDDC(list(var1,var2),K=3,model="AkjBkQkDk",init="kmeans",threshold=0.2)
table(cls2,res.multi$class,dnn=c("True clusters","FunHDDC clusters"))
par(mfrow=c(1,2))
plot(var1,col=res.multi$class)
plot(var2,col=res.multi$class)

##You can test multiple number of groups at the same time
# data("trigo")
# basis<- create.bspline.basis(c(0,1), nbasis=25)
# var1<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[,1:100]),fdParobj=basis)$fd
# res.uni<-funHDDC(var1,K=2:4,model="AkBkQkDk",init="kmeans",threshold=0.2)
#
# #You can test multiple models at the same time
# data("trigo")
# basis<- create.bspline.basis(c(0,1), nbasis=25)
# var1<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[,1:100]),fdParobj=basis)$fd
# res.uni<-funHDDC(var1,K=3,model=c("AkjBkQkDk","AkBkQkDk"),init="kmeans",threshold=0.2)
#
#
# #another example on Canadian data
# #Clustering the "Canadian temperature" data (Ramsey & Silverman): univariate case
# daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
# daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[,,"Temperature.C"], daybasis65,
# #
# #           fdnames=list("Day", "Station", "Deg C"))$fd
# #
# # res.uni<-funHDDC(daytempfd,K=3,model="AkjBkQkDk",init="random",threshold=0.2)
# #
# # #Graphical representation of groups mean curves
# # select1<-fd(daytempfd$coefs[,which(res.uni$class==1)],daytempfd$basis)
# # select2<-fd(daytempfd$coefs[,which(res.uni$class==2)],daytempfd$basis)
# # select3<-fd(daytempfd$coefs[,which(res.uni$class==3)],daytempfd$basis)
# #
# # plot(mean.fd(select1),col="lightblue",ylim=c(-20,20),lty=1,lwd=3)
# # lines(mean.fd(select2),col="palegreen2",lty=1,lwd=3)
# # lines(mean.fd(select3),col="navy",lty=1,lwd=3)
# #
# #
# # #Clustering the "Canadian temperature" data (Ramsey & Silverman): multivariate case

```

```

# daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
# daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[,,"Temperature.C"], daybasis65,
#                           fdnames=list("Day", "Station", "Deg C"))$fd
# dayprecfd<-smooth.basis(day.5, CanadianWeather$dailyAv[,,"Precipitation.mm"], daybasis65,
#                          fdnames=list("Day", "Station", "Mm"))$fd
#
# res.multi<-funHDDC(list(daytempfd,dayprecfd),K=3,model="AkjBkQkDk",init="random",
#                      threshold=0.2)
#
# #Graphical representation of groups mean curves
# #Temperature
# select1<-fd(daytempfd$coefs[,which(res.multi$class==1)],daytempfd$basis)
# select2<-fd(daytempfd$coefs[,which(res.multi$class==2)],daytempfd$basis)
# select3<-fd(daytempfd$coefs[,which(res.multi$class==3)],daytempfd$basis)
#
# plot(mean.fd(select1),col="lightblue",ylim=c(-20,20),lty=1,lwd=3)
# lines(mean.fd(select2),col="palegreen2",lty=1,lwd=3)
# lines(mean.fd(select3),col="navy",lty=1,lwd=3)
#
# #Precipitation
# select1b<-fd(dayprecfd$coefs[,which(res.multi$class==1)],dayprecfd$basis)
# select2b<-fd(dayprecfd$coefs[,which(res.multi$class==2)],dayprecfd$basis)
# select3b<-fd(dayprecfd$coefs[,which(res.multi$class==3)],dayprecfd$basis)
#
# plot(mean.fd(select1b),col="lightblue",ylim=c(0,5),lty=1,lwd=3)
# lines(mean.fd(select2b),col="palegreen2",lty=1,lwd=3)
# lines(mean.fd(select3b),col="navy",lty=1,lwd=3)

```

mfPCA

Functional principal component analysis for univariate or multivariate functional data

Description

It provides functional principal component analysis for univariate or multivariate functional data.

Usage

```
mfPCA(fdoj, center)
```

Arguments

fdoj	For univariate FPCA: a functional data object produced by fd() function of fda package, for multivariate FPCA: a list of functional data objects.
center	If TRUE (default), it centers each lines of data coefficients by the mean before calculating the FPCA.

Value

eigval	A list of eigen values.
harmonics	A functional data object for the harmonics or eigenfunctions.
scores	A matrix of scores on the harmonics.
varprop	A vector giving the proportion of variance explained by each harmonic.
meanfd	A functional data object giving the mean function after centering (default) or the mean function of raw data.

Examples

```
####Univariate case: "Canadian temperature" data (Ramsey & Silverman)
daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[, "Temperature.C"], daybasis65,
                          fdnames=list("Day", "Station", "Deg C"))$fd

res.pca<-mfpca(daytempfd)
plot.mfpca(res.pca)

####Multivariate case: "Canadian temperature" data (Ramsey & Silverman)
daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[, "Temperature.C"], daybasis65,
                          fdnames=list("Day", "Station", "Deg C"))$fd
dayprecfd<-smooth.basis(day.5, CanadianWeather$dailyAv[, "Precipitation.mm"], daybasis65,
                        fdnames=list("Day", "Station", "Mm"))$fd

res.pca<-mfpca(list(daytempfd, dayprecfd))
plot.mfpca(res.pca)
```

plot.fd

Plot a Functional Data Object

Description

Functional data observations, or a derivative of them, are plotted. These may be either plotted simultaneously, as `matplot` does for multivariate data, or one by one with a mouse click to move from one plot to another. The function also accepts the other plot specification arguments that the regular plot does. Calling `plot` with an `fdSmooth` or an `fdPar` object plots its `fd` component.

Usage

```
## S3 method for class 'fd'
plot(x, y, Lfdobj=0, href=TRUE, titles=NULL,
      xlim=NULL, ylim=NULL, xlab=NULL,
      ylab=NULL, ask=FALSE, nx=NULL, axes=NULL, col=1, ...)
```

Arguments

x	functional data object(s) to be plotted.
y	sequence of points at which to evaluate the functions 'x' and plot on the horizontal axis. Defaults to seq(rangex[1], rangex[2], length = nx). NOTE: This will be the values on the horizontal axis, NOT the vertical axis.
Lfdobj	either a nonnegative integer or a linear differential operator object. If present, the derivative or the value of applying the operator is plotted rather than the functions themselves.
href	a logical variable: If TRUE, add a horizontal reference line at 0.
titles	a vector of strings for identifying curves
xlab	a label for the horizontal axis.
ylab	a label for the vertical axis.
xlim	a vector of length 2 containing axis limits for the horizontal axis.
ylim	a vector of length 2 containing axis limits for the vertical axis.
ask	a logical value: If TRUE, each curve is shown separately, and the plot advances with a mouse click
nx	the number of points to use to define the plot. The default is usually enough, but for a highly variable function more may be required.
axes	Either a logical or a list or NULL. <ul style="list-style-type: none"> • logical whether axes should be drawn on the plot • list a list used to create custom axes used to create axes via x\$axes[[1]] and x\$axes[-1]. The primary example of this uses list("axesIntervals", ...), e.g., with Fourier bases to create CanadianWeather plots
col	line colors
...	additional plotting arguments that can be used with function plot

Details

Note that for multivariate data, a suitable array must first be defined using the par function.

Value

'done'

Side Effects

a plot of the functional observations

See Also

[lines.fd](#), [plotfit.fd](#)

Examples

```

##
## plot.fd
##

daybasis65 <- create.fourier.basis(c(0, 365), 65,
                                axes=list("axesIntervals"))
harmacelLfd <- vec2Lfd(c(0,(2*pi/365)^2,0), c(0, 365))

harmfdPar    <- fdPar(daybasis65, harmacelLfd, lambda=1e5)

daytempfd <- with(CanadianWeather, smooth.basis(day.5,
        dailyAv[,,"Temperature.C"], daybasis65)$fd)

# plot all the temperature functions for the monthly weather data
plot(daytempfd, main="Temperature Functions")

## Not run:
# To plot one at a time:
# The following pauses to request page changes.
\dontshow{
# (Without 'dontrun', the package build process
# might encounter problems with the par(ask=TRUE)
# feature.)
}
plot(daytempfd, ask=TRUE)

## End(Not run)

##
## plot.fdSmooth
##
b3.4 <- create.bspline.basis(norder=3, breaks=c(0, .5, 1))
# 4 bases, order 3 = degree 2 =
# continuous, bounded, locally quadratic
fdPar3 <- fdPar(b3.4, lambda=1)

# Penalize excessive slope Lfdobj=1;
# (Can not smooth on second derivative Lfdobj=2 at it is discontinuous.)
fd3.4s0 <- smooth.basis(0:1, 0:1, fdPar3)

# using plot.fd directly
plot(fd3.4s0$fd)

##
## with Date and POSIXct argvals
##

```

```

# Date
invasion1 <- as.Date('1775-09-04')
invasion2 <- as.Date('1812-07-12')
earlyUS.Canada <- as.numeric(c(invasion1, invasion2))
BspInvasion <- create.bspline.basis(earlyUS.Canada)

earlyUSyears <- seq(invasion1, invasion2, length.out=7)
earlyUScubic <- (as.numeric(earlyUSyears-invasion1)/365.24)^3
earlyUSyears <- as.numeric(earlyUSyears)
fitCubic <- smooth.basis(earlyUSyears, earlyUScubic, BspInvasion)$fd
plot(fitCubic)

# POSIXct
AmRev.ct <- as.POSIXct1970(c('1776-07-04', '1789-04-30'))
AmRevYrs.ct <- seq(AmRev.ct[1], AmRev.ct[2], length.out=14)
AmRevLin.ct <- as.numeric(AmRevYrs.ct-AmRev.ct[2])
AmRevYrs.ct <- as.numeric(AmRevYrs.ct)
BspRev.ct <- create.bspline.basis(AmRev.ct)
fitLin.ct <- smooth.basis(AmRevYrs.ct, AmRevLin.ct, BspRev.ct)$fd
plot(fitLin.ct)

```

plot.mfpca

Graphical representation for MFPCA

Description

It provides graphical representations for MFPCA: smoothed data plots, scores plots, variation of the mean curve and eigenfunction plots.

Usage

```

## S3 method for class 'mfpca'
plot(x, nharm, threshold, ...)

```

Arguments

x	An object produced by mfpca function.
nharm	The number of harmonics for which you want graphical representations, default value is 3.
threshold	The threshold of proportion of variance that stop plotting, default value is 0.05.
...	Some additional parameters.

Value

Data plot	Plot of all smooth curves for each functional variable.
Scores plots	Plot of curves coordinates on the number of eigenfunctions/harmonics selected and depending on the threshold choosen.

Mean curve plots

Plot of variation of the mean curve. Variations are estimated based on the mean values with addition (red) and subtraction (blue) of each eigenfunction.

Eigenfunction plots

visualization of each eigenfunction/harmonic selected and depending on the threshold chosen.

Examples

```
####Univariate case: "Canadian temperature" data (Ramsey & Silverman)
daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[, "Temperature.C"], daybasis65,
                          fdnames=list("Day", "Station", "Deg C"))$fd

res.pca<-mfpca(daytempfd)
plot(res.pca, nharm=4)

####Multivariate case: "Canadian temperature" data (Ramsey & Silverman)
daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[, "Temperature.C"], daybasis65,
                          fdnames=list("Day", "Station", "Deg C"))$fd
dayprecfd<-smooth.basis(day.5, CanadianWeather$dailyAv[, "Precipitation.mm"], daybasis65,
                        fdnames=list("Day", "Station", "Mm"))$fd

res.pca<-mfpca(list(daytempfd, dayprecfd))
plot(res.pca, nharm=4)
```

predict.funHDDC

Predict new results from a funHDDC model

Description

It provides predictions for a new functional dataset using the funHDDC model you train.

Usage

```
## S3 method for class 'funHDDC'
predict(object, newdata, ...)
```

Arguments

object	An object produced by funHDDC function
newdata	In the univariate case: a functional data object produced by the fda package, in the multivariate case: a list of functional data objects, for which you want to predict the group membership from the model.
...	Some additional parameters.

Value

class	The clustering partition.
t	The probability of each individual to belong to each cluster.
L	The loglikelihood.

Examples

```
##Univariate example
data("trigo")
basis<- create.bspline.basis(c(0,1), nbasis=25)
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
z<-sample(1:100,0.8*100,replace=FALSE)
var1_train<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[z,1:100]),
                        fdParobj=basis)$fd
var1_test<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[-z,1:100]),
                       fdParobj=basis)$fd

model<-funHDDC(var1_train,K=2)
pred<-predict(model,var1_test)

##Multivariate example
data("triangle")
basis<- create.bspline.basis(c(1,21), nbasis=25)
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
z<-sample(1:100,0.8*100,replace=FALSE)
var1_train<-smooth.basis(argvals=seq(1,21,length.out = 101),y=t(triangle[z,1:101]),
                        fdParobj=basis)$fd
var1_test<-smooth.basis(argvals=seq(1,21,length.out = 101),y=t(triangle[-z,1:101]),
                       fdParobj=basis)$fd
var2_train<-smooth.basis(argvals=seq(1,21,length.out = 101),y=t(triangle[z,102:202]),
                        fdParobj=basis)$fd
var2_test<-smooth.basis(argvals=seq(1,21,length.out = 101),y=t(triangle[-z,102:202]),
                       fdParobj=basis)$fd

model<-funHDDC(list(var1_train,var2_train),K=3)
pred<-predict(model,list(var1_test,var2_test))

##NOT RUN: another example on Canadian data
##Clustering the "Canadian temperature" data (Ramsey & Silverman): univariate case
daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
#z<-sample(1:35,0.9*35,replace=FALSE)
#daytempfd_train <- smooth.basis(day.5, CanadianWeather$dailyAv[,z,"Temperature.C"],
#daybasis65,fdnames=list("Day", "Station", "Deg C"))$fd
#daytempfd_test <- smooth.basis(day.5, CanadianWeather$dailyAv[-z,"Temperature.C"],
#daybasis65,fdnames=list("Day", "Station", "Deg C"))$fd

#model<-funHDDC(daytempfd_train,K=3)
#pred<-predict.funHDDC(model,daytempfd_test)
```

```

##Clustering the "Canadian temperature" data (Ramsey & Silverman): multivariate case
#daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
#z<-sample(1:35,0.9*35,replace=FALSE)
#daytempfd_train <- smooth.basis(day.5, CanadianWeather$dailyAv[,z,"Temperature.C"],
#daybasis65,fdnames=list("Day", "Station", "Deg C"))$fd
#daytempfd_test <- smooth.basis(day.5, CanadianWeather$dailyAv[-z,"Temperature.C"],
#daybasis65,fdnames=list("Day", "Station", "Deg C"))$fd
#dayprecf_train<-smooth.basis(day.5, CanadianWeather$dailyAv[,z,"Precipitation.mm"],
#daybasis65,fdnames=list("Day", "Station", "Mm"))$fd
#dayprecf_test<-smooth.basis(day.5, CanadianWeather$dailyAv[-z,"Precipitation.mm"],
#daybasis65,fdnames=list("Day", "Station", "Mm"))$fd

#model<-funHDDC(list(daytempfd_train,dayprecf_train),K=3)
#pred<-predict.funHDDC(model,list(daytempfd_test,dayprecf_test))

```

slopeHeuristic

Calculate slope heuristic

Description

It provides slope heuristic for number of groups selection or model selection (see Schmutz et al, 2018).

Usage

```
slopeHeuristic(mod)
```

Arguments

mod A funHDDC object with different number of groups tested or different models.

Details

This function works for model selection. If you test one model (for example AkjBQkDk) and multiple clusters, the function returns the number of clusters selected by slope heuristic. If you test multiple models (for example all 6 models) and one or multiple values of clusters, the function returns the rank of the best model selected by slope heuristic. The rank is the first column of the output provided by funHDDC function.

Value

The best partition if the function is used for the selection of number of clusters. The index of the best model, among all estimated models if the function is used for model selection.

Examples

```

##Univariate example: multiple clusters, one model tested
data("trigo")
basis<- create.bspline.basis(c(0,1), nbasis=25)
var1<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[,1:100]),fdParobj=basis)$fd
res.uni<-funHDDC(var1,K=2:10,model="AkBkQkDk")
slopeHeuristic(res.uni)

# ##Univariate example: one cluster, 6 models tested
# data("trigo")
# basis<- create.bspline.basis(c(0,1), nbasis=25)
# var1<-smooth.basis(argvals=seq(0,1,length.out = 100),y=t(trigo[,1:100]),fdParobj=basis)$fd
# res.uni<-funHDDC(var1,K=2,model=c("AkjBkQkDk", "AkjBQkDk", "AkBkQkDk",
#                                   "AkBQkDk", "ABkQkDk", "ABQkDk"))
# slopeHeuristic(res.uni)
#
# ##Multivariate example
# data("triangle")
# basis <- create.bspline.basis(c(1,21), nbasis=25)
# var1<-smooth.basis(argvals=seq(from=1,to=21,length.out = 101),y=t(triangle[,1:101]),
#                   fdParobj=basis)$fd
# var2<-smooth.basis(argvals=seq(from=1,to=21,length.out = 101),y=t(triangle[,102:202]),
#                   fdParobj=basis)$fd
# res.multi<-funHDDC(list(var1,var2),K=2:10,model="AkjBQkDk")
# slopeHeuristic(res.multi)
#
# ##An other example on Canada dataset
# #Clustering the "Canadian temperature" data (Ramsey & Silverman): univariate case
# daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
# daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[,,"Temperature.C"], daybasis65,
#                          fdnames=list("Day", "Station", "Deg C"))$fd
#
# res.uni<-funHDDC(daytempfd,K=2:10,model="AkjBkQkDk", threshold=0.2)
#
# slopeHeuristic(res.uni)
#
# #Clustering the "Canadian temperature" data (Ramsey & Silverman): multivariate case
# daybasis65 <- create.fourier.basis(c(0, 365), nbasis=65, period=365)
# daytempfd <- smooth.basis(day.5, CanadianWeather$dailyAv[,,"Temperature.C"], daybasis65,
#                          fdnames=list("Day", "Station", "Deg C"))$fd
# dayprecfd<-smooth.basis(day.5, CanadianWeather$dailyAv[,,"Precipitation.mm"], daybasis65,
#                        fdnames=list("Day", "Station", "Mm"))$fd
#
# res.multi<-funHDDC(list(daytempfd,dayprecfd),K=2:8,model="AkjBkQkDk",
#                      init="random", threshold=0.2)
#
# slopeHeuristic(res.multi)

```

triangle	<i>Simulated functional bivariate dataset</i>
----------	---

Description

A simulated dataset of bivariate functional data from 3 groups.

Usage

```
data(triangle)
```

Format

A dataframe with 100 lines and 203 columns. A row represents a curve and a column a sampling point. Columns from 1 to 101 are sampling points of the first functional variable. Columns from 102 to 202 are sampling points of the second functional variable. Column 203 is the variable which indicates the true clustering partition.

trigo	<i>Simulated functional univariate dataset</i>
-------	--

Description

A simulated dataset of univariate functional data from 2 groups.

Usage

```
data(trigo)
```

Format

A dataframe with 100 lines and 101 columns. A row represents a curve and a column a sampling point. Columns from 1 to 100 are sampling points of the functional variable. Column 101 is the variable which indicates the true clustering partition.

Index

* **hplot**

plot.fd, 7

* **smooth**

plot.fd, 7

funHDDC, 2

lines.fd, 8

mf pca, 6

plot.fd, 7

plot.mfpca, 10

plotfit.fd, 8

predict.funHDDC, 11

slopeHeuristic, 13

triangle, 15

trigo, 15