Package 'kmlShape'

March 5, 2016

Type Package

Title K-Means for Longitudinal Data using Shape-Respecting Distance

Version 0.9.5

Date 2016-03-08

Description
K-means for longitudinal data using shape-respecting distance and shape-respecting means.

License GPL (>= 2)

LazyData yes

Depends methods,class,longitudinalData,kml,lattice

URL http:www.r-project.org

Collate global.R plot.R clds.R reduceTraj.R distanceFrechet.R
meanFrechet.R parKmlShape.R kmlShape.R

Encoding latin1

KeepSource TRUE

NeedsCompilation yes

Author Christophe Genolini [cre, aut],
Elie Guichard [ctb]

Maintainer Christophe Genolini <christophe.genolini@u-paris10.fr>

Repository CRAN

Date/Publication 2016-03-05 00:22:43

R topics documented:

`kmlShape-package` *~ Package: kmlShape ~*

Description

KmlShape is a package design to cluster longitudinal data according to their shape.

Details

Package:	KmlShape
Type:	Package
Version:	0.9.5
Date:	2016-03-04
License:	GPL >2.0

kmlShape cluster longitudinal data according to their shape: instead of merging individual whose trajectories are closed in term of euclidienne distance, it groups the individual that are closed according Frechet's distance.

Since k-means using Frechet has a complexity in O(n^2t^2), KmlShape also provide some function to reduce the size of the data without changing the result:

- `reduceNbId` reduce the number of individual, by merging them using a classical k-means on many centers.
- `reduceNbTimes` reduce the number of measurement, by (optionaly) smoothing the curve then by applying the Douglas-Peuker algorithms.

Author(s)

Christophe Genolini <christophe.genolini@u-paris10.fr>

Examples

```
#########
### Real example, on ictus data

### Preparing the data
set.seed(1)
data(ictusShort)
myClds <- cldsWide(ictusShort)

### Reducing the data size
reduceTraj(myClds,nbSenators=64,nbTimes=5)

### Clustering using shape
kmlShape(myClds,4)

plotMeans(myClds)
```

Clds-class ~ *Class* "Clds" ~

Description

Clds (or ClusterLongDataShape) is a class used to prepared the trajectories that will be cluster by the function kmlShape and to store the result of the clustering. According to the data simplification that the user may perform, it may containt the trajectories in wide format, in long format, the simplified trajectories (called 'senators'), the partition found and the mean's trajectories of the cluster find by kmlShape.

Objects from the Class

Objects can be created by calls of the form new("Clds", ...) or using the constructor cldsWide and cldsLong.

Slots

steps: [vector(logical)] summarizes what data are available and the transformation that the data had already undergone. The first value is TRUE if the data has been generated from a data.frame in a wide format. The second is TRUE if the data are available in wide format. The third is TRUE if the data in long format are available. The fourth is TRUE if the function reduceNbId has been used. The fitfh is TRUE if the function reduceNbTimes has been used. The sixth is TRUE if kmlShape has been used.

id: [vector(factor)] Unique identifier, one for each trajectories.

nbId: [integer] Number of trajectories.

nbCol: [integer] Number of times measurement (if the trajectories are in wide format).

`trajWide`: `[matrix]` Trajectories in wide format. Each line is an individual, each column is a specific time.

`times`: `[vector(numeric)]` Times at which measures are made.

`trajLong`: `[data.frame]` Trajectories in long format. The first column hold the identifiers ; in the second are the times ; the third coutain the values.

`senators`: `[data.frame]` The 'senatorsMeans' are the trajectories get by reducing the number of individual (using `reduceNbId`). The 'senatorShort' are the population after reduction of the number of time (using `reduceNbTimes`). 'senatorsMeansShort' are the trajectories get by using both. The field 'senators' hold either the 'senatorMeans', the 'senatorShort' or the 'senatorMeansShort', according to the reduction that has been used.

`mySenator`: `[data.frame]` In the fisrt column are all the individual indentifier. The second hold the identifier of the senators that represent the individual

`senatorsWeight`: `[integer]` If the procedure `reduceNbId` has been used, each senators is the mean of a clusters. His `senatorsWeight` is the number of individual that are in his clusters. If `reduceNbId` has not been used (and thus, only `reduceNbTimes` has been used), each senators has weight 1.

`clustersSenators`: `[factor]` Clusters of each senators after the used of `kmlShape`.

`clusters`: `[factor]` Clusters of each individual after the used of `kmlShape`. The clusters of an individual is the cluters of its senators.

`trajMeans`: `[data.frame]` Means' trajectories of each clusters after the use of `kmlShape`.

Methods

[: Get the value of the field asked. Possible values are 'step', 'wideAvailable', 'longAvailable', 'senatorsAvailable', 'reduceId', 'reduceTimes', 'kmlShape', 'nbClusters', 'id', 'nbId', 'nbCol', 'trajWide', 'times', 'trajLong', 'senators', 'mySenator', 'senatorsWeight', 'clustersSenators', 'clusters', 'trajMeans'

[<- : Set the selected field to value.

show : Display the object. Since many fields can be empty, it display only the field that ar not empty.

Examples

```
data(ictusShort)
cldsWide(ictusShort)
```

`cldsLong` *~ Function: cldsLong ~*

Description

Turn trajectories in long format into an object of class `Clds`.

Usage

```
cldsLong(trajLong)
```

Arguments

`trajLong`	`[data.frame]`: the trajectories, in long format. The trajectories have to have (no choice) th following format: the first column is the identifier; the second is the time measurement ; the third is the values.

Details

Turn trajectories in long format into an object of class `Clds-class`.

Value

Object of class `Clds`.

Examples

```
### Some artificial data
g <- function(x)dnorm(0:100,runif(1,25,75),10)*rnorm(1,5,1)
dn <- data.frame(id=rep(1:200,each=101),
   times=rep((0:100)/10,times=20),
   traj=as.numeric(sapply(1:200,g))
)

### clds format
myClds <- cldsLong(dn)
plotTraj(myClds)
```

`cldsWide` *~ Function: cldsWide ~*

Description

Turn trajectories in wide format into an object of class `Clds`.

Usage

```
cldsWide(trajWide, times, id)
```

Arguments

trajWide	[data.frame] or [matrix]: the trajectories, in wide format (each line is an individual, each column is a specific time measurement).
times	[vector(numeric)] Times at which measures are made.
id	[vector(factor)] Vector of unique identifiers, one for each trajectories. If id is missing, the first column of the trajWide is turn into a factor and is used as id.

Details

Turn trajectories in wide format into an object of class Clds. If id is missing, the first column of the trajWide is turn into a factor and is used as id. Column 2:ncol(trajWide) are the trajectories. If id is not missing, column 1:ncol(trajWide) are the trajectories.

Value

Object of class Clds-class.

Examples

```
data(ictusShort)
myClds <- cldsWide(ictusShort)
myClds
plotTraj(myClds)
```

distFrechet ~ Function: Frechet distance ~

Description

Compute Frechet distance between two trajectories.

Usage

```
distFrechet(Px,Py,Qx, Qy, timeScale=0.1, FrechetSumOrMax = "sum")
distFrechetR(Px,Py,Qx, Qy, timeScale=0.1, FrechetSumOrMax = "sum")
distFrechetRec(Px,Py,Qx, Qy, timeScale=0.1, FrechetSumOrMax = "sum")
```

Arguments

Px	[vector(numeric)] Times (abscisse) of the first trajectories.
Py	[vector(numeric)] Values of the first trajectories.
Qx	[vector(numeric)] Times of the second trajectories.
Qy	[vector(numeric)] Values of the second trajectories.

`timeScale`
: [`numeric`]: allow to modify the time scale, increasing or decreasing the cost of the horizontal shift. If timeScale is very big, then the Frechet's distance is equal to the euclidienne distance. If timeScale is very small, then it is equal to the Dynamic Time Warping.

`FrechetSumOrMax`
: [`character`]: The Frechet's distance can be define using the 'sum' function or the 'max' function. This option let the user to chose one or the other.

Details

Given two curve P and Q, Frechet distance between P and Q is define as `inf_{a,b} max_{t} d(P(a(t)),Q(b(t)))`. It's computation is a NP-complex problem. When P and Q are trajectories (discrete curve), the problem is polynomial.

The Frechet distance can also be define using a sum instead of a max: `inf_{a,b} sum_{t} d(P(a(t)),Q(b(t)))`

The function `distFrechet` is C compiled, the function `distFrechetR` is in R, the function `distFrechetRec` is in recursive (the slowest) in R.

Value

A numeric value.

Author

Christophe Genolini
1. UMR U1027, INSERM, Université Paul Sabatier / Toulouse III / France
2. CeRSM, EA 2931, UFR STAPS, Université de Paris Ouest-Nanterre-La Défense / Nanterre / France

References

[1] Thomas Eiter & Heikki Mannila:
"Computing Discrete Fr´echet Distance"

[2] C. Genolini and B. Falissard
"KmL: k-means for longitudinal data"
Computational Statistics, vol 25(2), pp 317-328, 2010

[3] C. Genolini and B. Falissard
"KmL: A package to cluster longitudinal data"
Computer Methods and Programs in Biomedicine, 104, pp e112-121, 2011

See Also

distTraj

Examples

```
Px <- 1:20
Py <- dnorm(1:20,12,2)
Qx <- 1:20
Qy <- dnorm(1:20,8,2)

### Function from Eiter and Mannila compiled in C
system.time(cat("\n",distFrechet(Px,Py,Qx,Qy)))

### Same thing in R
system.time(cat("\n",distFrechetR(Px,Py,Qx,Qy)))

### Frechet using sum instead of max.
distFrechet(Px,Py,Qx,Qy,FrechetSumOrMax="sum")
```

DouglasPeucker *~ Function: DouglasPeucker ~*

Description

The Ramer-Douglas-Peucker algorithm (RDP) is an algorithm for reducing the number of points in a trajectory.

Usage

```
DouglasPeuckerEpsilon(trajx, trajy, epsilon, spar=NA)
DouglasPeuckerNbPoints(trajx, trajy, nbPoints, spar=NA)
```

Arguments

`trajx`	[`vector(numeric)`]: abscissa of the trajectory.
`trajy`	[`vector(numeric)`]: ordinate of the trajectory.
`epsilon`	[`numeric`]: see details
`nbPoints`	[`numeric`]: see details
`spar`	[`numeric`]: smoothing parameter.

Details

[extract from Wikipedia -begin-]

*** Idea ***

The purpose of the algorithm is, given a curve (trajectory) composed of line segments, to find a similar curve with fewer points. The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the simplified curve. The simplified trajectory consists of a subset of the points that defined the original trajectory.

*** Algorithm with epsilon (function DouglasPeackerEpsilon) ***

The starting curve is an ordered set of points or lines. Let epsilon > 0 be the distance dimension.

The algorithm recursively divides the line. Initially it is given all the points between the first and last point. It automatically marks the first and last point to be kept. It then finds the point that is furthest from the line segment with the first and last points as end points (this point is obviously furthest on the curve from the approximating line segment between the end points). If the point is closer than epsilon to the line segment then any points not currently marked to keep can be discarded without the simplified curve being worse than epsilon.

If the point furthest from the line segment is greater than epsilon from the approximation then that point must be kept. The algorithm recursively calls itself with the first point and the worst point and then with the worst point and the last point (which includes marking the worst point being marked as kept).

When the recursion is completed a new output curve can be generated consisting of all (and only) those points that have been marked as kept.

[extract from Wikipedia -end-]

*** Algorithm with a fixed number of point (function DouglasPeackerNbPoints) ***

The previous algorithm stops when the simplified curve and the real curve are at a distance less than epsilon. It gives no control over the number of points which are in the simplified curve.

It is possible to change that by modifying the stopping condition: instead of adding points 'until the curves are close enough to each other', one can choose to add the farest points until a a pre-determined number of points is reach. This is what the function `DouglasPeackerNbPoints` does.

Note that `DouglasPeackerNbPoints` controls the number of points of the simplified curve, but does not bound the distance between the originale curve and the simplified curve.

*** smoothing the curve ***

On unsmooth curves with a lot of small variations, the Douglas-Peucker algorithm gives "strange" results (see example 3). It is therefor preferable to smoothing the curved before simplifying it. The `spar` parameter allows define the degree of smoothing that will be used. If set to `NA`, the curve is not smoothed. Otherwise, it is smoothed using the function `smooth.spline` with parameter `spar`.

*** missings values *** They are removed from the trajectory.

Value

A `data.frame` with the new trajectory. The first (x) hold the abcsissa, the second (y) the ordinate.

Examples

```
Px <- (1:100)/10
Py <- dnorm(Px,3,1)+dnorm(Px,7,1)+Px/10


### Example 1
### Simplification using epsilon

par(mfrow=c(2,2))
plot(Px,Py,type="l")
plot(DouglasPeuckerEpsilon(Px,Py,0.01),type="b",col=4)
plot(DouglasPeuckerEpsilon(Px,Py,0.04),type="b",col=3)
plot(DouglasPeuckerEpsilon(Px,Py,0.1),type="b",col=2)

### Example 2
### Simplification using nbPoints

par(mfrow=c(2,2))
plot(Px,Py,type="l")
plot(DouglasPeuckerNbPoints(Px,Py,20),type="b",col=4)
plot(DouglasPeuckerNbPoints(Px,Py,10),type="b",col=3)
plot(DouglasPeuckerNbPoints(Px,Py,5),type="b",col=2)


### Example 3
### Simplification with and without smoothing

Py <- dnorm(Px,3,1)+dnorm(Px,7,1)+Px/10+rnorm(100,,0.1)

par(mfrow=c(2,2))
plot(Px,Py,type="l")
plot(DouglasPeuckerNbPoints(Px,Py,20),type="b",col=4)
plot(DouglasPeuckerNbPoints(Px,Py,20,spar=0.5),type="b",col=3)
plot(DouglasPeuckerNbPoints(Px,Py,10,spar=0.5),type="b",col=2)
```

`ictusShort` *~ Data: ictusShort ~*

Description

A subset of the longitudinal study ICTUS

Usage

```
data("ictusShort")
```

Format

A data frame with 1374 observations on the following 16 variables.

`id` Unique identifier

‘`MMS-1`’ Mini Mental Score at time 1

‘`MMS-2`’ Mini Mental Score at time 2

‘`MMS-3`’ Mini Mental Score at time 3

‘`MMS-4`’ Mini Mental Score at time 4

‘`MMS-5`’ Mini Mental Score at time 5

‘`MMS-6`’ Mini Mental Score at time 6

‘`MMS-7`’ Mini Mental Score at time 7

‘`MMS-8`’ Mini Mental Score at time 8

`'MMS-9'`	Mini Mental Score at time 9
`'MMS-10'`	Mini Mental Score at time 10
`'MMS-11'`	Mini Mental Score at time 11
`'MMS-12'`	Mini Mental Score at time 12
`'MMS-13'`	Mini Mental Score at time 13
`'MMS-14'`	Mini Mental Score at time 14
`'MMS-15'`	Mini Mental Score at time 15

Details

Ictus [1, 2] is a cohort of 1380 patients with Alzheimer disease followed-up in 12 European countries. These patients were included between February 2003 and July 2005 in 29 centres specialized in neurology, geriatrics, psychiatry, or psycho-geriatrics with a recognized experience in the diagnosis and management of Alzheimer disease. Most of these patients were seen during memory consultations and included consecutively. These patients were examined at six-month intervals over two years. Each examination included (though not exclusively) an Mini Mental Score (MMS) assessment.

The dataset "ictusShort" is a subset of the cohort Ictus. Since the acces to Ictus is submited to conditions, the original data have been transform before inclusion in the package, but the results of the analysis using kmlShape are the same on the real Ictus and ictusShort.

References

[1] Reynish, E., Cortes, F., Andrieu, S., Cantet, C., Olde Rikkert, M., Melis, R., Froelich, L., Frisoni, G., Jonsson, L., Visser, P., et al., 2007. The ictus study: A prospective longitudinal observational study of 1,380 ad patients in europe. Neuroepidemiology 29 (1-2), 29-38

[2] Vellas, B., Hausner, L., Frolich, L., Cantet, C., Gardette, V., Reynish, E., Gillette, S., Aguera-Morales, E., Auriacombe, S., Boada, M., et al., 2012. Progression of alzheimer disease in europe: Data from the european ictus study. Current Alzheimer Research 9 (8), 902-912.

Examples

```
data(ictusShort)
summary(ictusShort)
matplot(t(ictusShort),type="l")
```

`kmlShape` *~kmlShape ~*

Description

This function run k-means for longitudinal data using some shape respecting distance and mean.

Usage

```
kmlShape(myClds, nbClusters = 3, timeScale = 0.1, FrechetSumOrMax =
"max", toPlot="both", parAlgo=parKmlShape())
```

Arguments

`myClds`	[`Clds`]: Object that hold the trajectories, the 'senators' resulting from a simplification of the trajectories and, after the use of kmlShape, the clusters.
`nbClusters`	[`numeric`] or [`vector(numeric)`]: either the number of clusters, or a vector of initial (distinct) cluster centers. If a number, a random set of (distinct) trajectories is chosen as the initial centres.
`timeScale`	[`numeric`]: allow to modify the time scale, increasing or decreasing the cost of the horizontal shift. If timeScale is very big, then the Frechet mean tends to the euclidienne distance. If timeScale is very small, then it tends to the Dynamic Time Warping.
`FrechetSumOrMax`	[`character`]: kmlShape uses Frechet's distance and Frechet path. Since both of them can be define using the 'sum' function or the 'max' function, this option let the user to chose one or the other.
`toPlot`	[`character`]: use 'traj' for graphical display during computation, or 'none' for a faster but quiet run.
`parAlgo`	[`ParKmlShape`]: parameters used to run the algorithm. They can be change using the function `parKmlShape`. Option are mainly 'aggregationMethod', 'shuffle', 'sampleSize', 'methodHclust' and 'maxIter'. See `ParKmlShape` for details.

Details

This function run k-means for longitudinal data using a shape respecting distance (`distFrechet`) and a shape respecting mean (`meanFrechet`). See [1] for details.

Value

An object of class `Clds` in which the field 'clustersSenators', 'clusters' and 'trajMeans' are now filled.

Examples

```
###########
### Example

### Generating artificial data
nbLignes <- 20
trajG <- matrix(0,nbLignes,10)
for(i in 1:(nbLignes/2)){
   trajG[i,] <- dnorm(1:10,runif(1,3,8),1)*rnorm(1,10,0.1)
}
for(i in (nbLignes/2+1):nbLignes){
   trajG[i,] <- dnorm(1:10,runif(1,3,8),1)*rnorm(1,5,0.1)
}

myClds <- cldsWide(data.frame(1:20,trajG))
plot(myClds)

### kmlshape
par(ask=FALSE)
kmlShape(myClds,2)
par(ask=TRUE)
plot(myClds)



###########
### Example 2

### Generating artificial data
nbLignes <- 12
trajH <- matrix(0,nbLignes,10)

 for(i in 1:(nbLignes/3)){
   trajH[i,] <- pnorm(1:10,runif(1,3,8),1)*rnorm(1,10,1)
}
for(i in (nbLignes/3+1):(2*nbLignes/3)){
   trajH[i,] <- dnorm(1:10,runif(1,3,8),1)*rnorm(1,13,1)
}

for(i in (2*nbLignes/3+1):nbLignes){
   trajH[i,] <- pnorm(1:10,runif(1,3,8),1)*rnorm(1,5,0.1)
}

myClds2 <- cldsWide(data.frame(1:60,trajH))
plot(myClds2)

### kmlshape
par(ask=FALSE)
kmlShape(myClds2,3)
par(ask=TRUE)
plot(myClds2)
```

matplotLong *~ Function: matplotLong ~*

Description

Plot some longitudinal data in long format.

Usage

```
matplotLong(trajLong, col = 1:6, lty = 1:5, lwd=1, add = FALSE,
   main="", xlab="Times",ylab="",pourcent=NA)
```

Arguments

`trajLong`	[`data.frame`]: trajectories in long format. The `data.frame` has to be (no choice!) in the following format: the first column should be the individual indentifiant. The second should be the times at which the measurement are made. The third one should be the measurement.
`col`	[`vector(numeric)`] or [`vector(character)`]: vector that define the trajectories' colors. If the length of the vector is one, `col` is duplicated.
`lty`	[`numeric`]: lines type.
`lwd`	[`numeric`]: lines width.
`add`	[`logical`]: shall the function start a new graph (add=FALSE) or add the lines to the current graph (add=TRUE) ?
`main`	[`character`]: main title.
`xlab`	[`character`]: x label.
`ylab`	[`character`]: y label.
`pourcent`	[`numeric`]: if pourcent is not NA, then a legend is added on the top of the graph. The legend takes the values given by the vecteur `pourcent`.

Details

Plot some longitudinal data in long format. Only the color and the lines width can be modifid by the user.

Value

A graph.

Examples

```
### Preparing data
g <- function(x)dnorm(x,3)+dnorm(x,7)+x/10
dn <- data.frame(id=rep(1:20,each=101),
    times=rep((0:100)/10,times=20),
    traj=rep(g((0:100)/10),20)+rep(runif(20),each=101)+rnorm(20*101,,0.1))

### matplotLong
matplotLong(dn)

### matplotLong with a legend
matplotLong(dn,col=2:3,pourcent=c(0.50,0.50))
```

meanFrechet ~ *Function: meanFrechet* ~

Description

Compute the Frechet mean

Usage

```
meanFrechet(trajLong, timeScale = 0.1, FrechetSumOrMax = "sum",
  aggregationMethod = "all", shuffle = TRUE, sampleSize = NA, methodHclust = "average")
```

Arguments

`trajLong`	[`data.frame`]: trajectories in long format. The data.frame has to be (no choice!) in the following format: the first column should be the individual indentifiant. The second should be the times at which the measurement are made. The third one should be the measurements.
`timeScale`	[`numeric`]: allow to modify the time scale, increasing or decreasing the cost of the horizontal shift. If timeScale is very big, then the Frechet mean tends to the euclidienne distance. If timeScale is very small, then it tends to the Dynamic Time Warping.
`FrechetSumOrMax`	[`character`]: Like Frechet's distance, the Frechet Mean can be define using the 'sum' function or the 'max' function. This option let the user to chose one or the other.
`aggregationMethod`	[`character`]: define the agglomerative method used to compute the mean. Three methods are curently available: "all", "sample" and "hierarchical". See detail.
`shuffle`	[`logical`]: shall the order of the agglomeration should be randomly chosen? (only for methods "all" and "sample")
`sampleSize`	[`integer`]: define the size of the sample (for method 'sample' only).
`methodHclust`	[`character`]: define the distance between two clusters used by the hierarchical clustering. The methods available are the one usable by the function `hclust`

Details

Compute the Frechet mean, as define in [1]. The main idea of the algorithm is the following:

The Frechet mean of two trajectories can be easely define as the middle of the leash that joint the two trajectories (see `meanFrechet2`). Then the mean of n individual can be obtain by merging the individual trajectories two by two, then merging the resulting trajectories and so on until there is only one trajectory left. This last trajectory is the Frechet mean. Theoriticaly, the final result depend of the order of agglomeration. In practice, on large sample, this order has little impact on the final result (see [1] for detail).

So far, three agglomeration methods are availables:

- `all`: the n individuals are scattered (randomly if `shuffle=TRUE`) on the leaves of a complete binary tree (all the knots have zero or two leaves) having depth h with 2^h <= n <2^h+1. The value of each non-terminal leaf is the Frechet mean for two trajectories of the two children leaves. Frechet mean is thus the value of the tree root. (Informally, this structure is close to that of a tennis tournament). The complexity of this method is O(nt^2).
- `sample`: This method is the method `all` applied only to a sample of `sampleSize` trajectories. The complexity of the method is $O(n^0t^2)$, $n^0$ being the size of the random sample.
- `hierarchical`: the combination order between individuals is fixed in a deterministic way through an ascending hierarchical classification; the closest individuals being combined first. The complexity of this method is $O(n^2t^2)$.

Value

A `data.frame` holding a trajectory.

See Also

`meanFrechet2`, `pathFrechet`

Examples

```
require(lattice)

### Define artificial data
g <- function(x)dnorm(0:20,runif(1,5,15),2)*rnorm(1,5,1)
dn <- data.frame(id=rep(1:20,each=21),
   times=rep((0:20),times=20),
   traj=as.numeric(sapply(1:20,g)),
   weight=1
)

xyplot(traj ~ times, data=dn, groups=id,type="l",ylim=c(0,1.4))
plot(meanFrechet(dn),ylim=c(0,1.4))
plot(meanFrechet(dn,0.001),ylim=c(0,1.4))
plot(meanFrechet(dn,10),ylim=c(0,1.4))
```

`meanFrechet2` *~ Function: meanFrechet2 ~*

Description

Compute the Frechet mean between two curves.

Usage

```
meanFrechet2(Px, Py, Qx, Qy, timeScale = 0.1, FrechetSumOrMax = "sum", weightPQ = c(1,1))
```

Arguments

`Px`	[`vector(numeric)`] Times (abscisse) of the first trajectories.
`Py`	[`vector(numeric)`] Values of the first trajectories.
`Qx`	[`vector(numeric)`] Times of the second trajectories.
`Qy`	[`vector(numeric)`] Values of the second trajectories.
`timeScale`	[`numeric`]: allow to modify the time scale, increasing or decreasing the cost of the horizontal shift. If timeScale is very big, then the Frechet's mean is equal to the euclidienne distance. If timeScale is very slow, then it is equal to the Dynamic Time Warping.
`FrechetSumOrMax`	[`character`]: Like Frechet's distance, Frechet's mean can be define using the 'sum' function or the 'max' function. This option let the user to chose one or the other.
`weightPQ`	[`couple(numeric)`]: respective weight of the two trajectories (for a weighted mean).

Details

Given two curve P and Q

- The Frechet distance between P and Q is define as `distFrechet(P,Q)=inf_{a,b} max_{t} d(P(a(t)),Q(b(t)))`.
- The Frechet path is the couple of function `(a(t),b(t))` that realize the equality of the Frechet distance: `distFrechet(P,Q)=max_{t} d(P(a(t)),Q(b(t)))`
- Frechet mean is the curve define by the sequence of all the center of the segments define by the Frechet path `[a(t),b(t)]`. If P and Q have respectively weight p and q, the center is the weighted mean of the segments : $c(t)=(p.a(t)+q.b(t))/(p+q)$.

The Frechet distance, path and means can also be define using a sum instead of a max.

Value

A numeric value.

Examples

```
traj <- matrix(0,4,5)
traj[1,2] <- 10
traj[2,3] <- 11
traj[3,4] <- 10
traj[4,2] <- 8

matplot(x=1:5,y=t(traj),type="l",col=2:5,lty=1)
m12 <- meanFrechet2(Px=1:5,Py=traj[1,],Qx=1:5,Qy=traj[2,])
m34 <- meanFrechet2(Px=1:5,Py=traj[3,],Qx=1:5,Qy=traj[4,])
lines(m12,col=2,lwd=3)
lines(m34,col=2,lwd=3)

m1234 <- meanFrechet2(Px=m12$times,Py=m12$traj,Qx=m34$times,Qy=m34$traj)
lines(m1234,col=1,lwd=5)
```

parKmlShape *~ Function: parKmlShape ~*

Description

parKmlShape is a constructor for the object ParKml.

Usage

```
parKmlShape(aggregationMethod="all", shuffle=TRUE, sampleSize=128,
   methodHclust="average", maxIter=100)
```

Arguments

aggregationMethod	[character]: define the aglomerative method used to compute the mean. Three methods are curently available: "all", "sample" and "hierarchical". See meanFrechet and [1] for details.
shuffle	[logical]: if the agglomerationMethod is "all" or "sample", this variable is use to decide if the trajectories will be agglomerate in a random order (shuffle=TRUE) or not. If not, the lexical order based on the individual identifiant is used.
sampleSize	[integer]: Define the number of trajectories that will be use to compute the meanFrechet when the aggregationMethod is "sample".
methodHclust	[character]: define the distance between two clusters used by the hierarchical clustering when the aggregationMethod is "hierarchical". The methods available are the one for hclust
maxIter	[numeric]: the maximum number of iteration allowed.

Details

parKmlShape is the constructor of object ParKml.

Value

An object ParKmlShape.

Examples

```
parKmlShape()
parKmlShape(aggregationMethod="hierarchical",methodHclust="single")
```

`ParKmlShape-class` *~ Class: "ParKmlShape" ~*

Description

`ParKmlShape` is an object containing some parameter used by `kmlShape`.

Slots

aggregationMethod `[character]`: define the aglomerative method used to compute the mean. Three methods are curently available: "all", "sample" and "hierarchical". See `meanFrechet` and [1] for details.

shuffle `[logical]`: if the agglomerationMethod is "all" or "sample", this variable is use to decide if the trajectories will be agglomerate in a random order (`shuffle=TRUE`) or not. If not, the lexical order based on the individual identifiant is used.

sampleSize `[integer]`: Define the number of trajectories that will be use to compute the `meanFrechet` when the aggregationMethod is "sample".

methodHclust `[character]`: define the distance between two clusters used by the hierarchical clustering when the aggregationMethod is "hierarchical". The methods available are the one for `hclust`

maxIter `[numeric]`: the maximum number of iteration allowed.

Methods

`object['xxx']` Get the value of the field `xxx`.

Examples

```
parAlgo <- parKmlShape()
parAlgo["aggregationMethod"]
parAlgo["aggregationMethod"] <- "hierarchical"
```

`pathFrechet` *~ Function: Frechet distance ~*

Description

Compute Frechet distance and Frechet path between two trajectories.

Usage

```
pathFrechet(Px,Py,Qx,Qy,timeScale=0.1,FrechetSumOrMax = "sum")
```

Arguments

`Px`	[vector(numeric)] Times (abscisse) of the first trajectories.
`Py`	[vector(numeric)] Values of the first trajectories.
`Qx`	[vector(numeric)] Times of the second trajectories.
`Qy`	[vector(numeric)] Values of the second trajectories.
`timeScale`	[`numeric`]: allow to modify the time scale, increasing or decreasing the cost of the horizontal shift. If timeScale is very big, then the Frechet path is only a set of vertical segment. If timeScale is very small, then it is equal to the path find by Dynamic Time Warping.
`FrechetSumOrMax`	[`character`]: Like Frechet's distance, the Frechet's path can be define using the 'sum' function or the 'max' function. This option let the user to chose one or the other.

Details

Given two curve P and Q, Frechet distance between P and Q is define as `distFrechet(P,Q)=inf_{a,b} max_{t} d(P(a(t)`

The Frechet path is the couple of function `(a(t),b(t))` that realize the previous equality : `distFrechet(P,Q)=max_{t} d(P`

It's computation is a NP-complex problem. When P and Q are trajectories (discrete curve), the problem is polynomial.

The Frechet path can also be define using a sum instead of a max: `inf_{a,b} sum_{t} d(P(a(t)),Q(b(t)))`

The function `pathFrechet` is C compiled whereas the function `pathFrechetR` is in R.

Value

A numeric value and the Frechet path in a matrix.

Author

Christophe Genolini \& Elie Guichard
1. UMR U1027, INSERM, Université Paul Sabatier / Toulouse III / France
2. CeRSME, EA 2931, UFR STAPS, Université de Paris Ouest-Nanterre-La Défense / Nanterre / France

References

Thomas Eiter & Heikki Mannila: "Computing Discrete Fr´echet Distance"

[1] C. Genolini and B. Falissard
"KmL: k-means for longitudinal data"
Computational Statistics, vol 25(2), pp 317-328, 2010

[2] C. Genolini and B. Falissard
"KmL: A package to cluster longitudinal data"
Computer Methods and Programs in Biomedicine, 104, pp e112-121, 2011

See Also

distFrechet

Examples

```
Px <- 1:20
Py <- dnorm(1:20,12,2)
Qx <- 1:20
Qy <- dnorm(1:20,8,2)

### Function from Eiter and Mannila compiled in C
system.time(pathFrechet(Px,Py,Qx,Qy))

### Same thing in R
system.time(pathFrechet(Px,Py,Qx,Qy))

### Frechet using sum instead of max.
pathFrechet(Px,Py,Qx,Qy,FrechetSumOrMax="sum")
```

`plot` *~ Function: plot for Clds ~*

Description

`plot` plot both the trajectories the the clusters' means of an object `Clds`.

Usage

```
## S4 method for signature 'Clds,missing'
plot(x,y,col="darkgrey",lty=1,legend=TRUE,...)
```

Arguments

`x`	[`Clds`]: Object containing the trajectories to plot.
`y`	Useless. For compatibility only.
`col`	[`character`], [`integer`] or [`vector(integer)`]: color use for the trajectories. If the special value "clusters" is used, the trajectories will be colored according to their clusters.
`lty`	[`integer`] or [`vector(integer)`]: line type of the trajectories
`legend`	[`logical`]: shall the pourcent of individual in each cluster be printed as a legend ?
`...`	Arguments to be passed to methods. These arguments need to be compatible with `plotTraj`.

Details

`plot` plot both the trajectories the the clusters' means of an object `Clds`. If the option `col="clusters"` is used, the trajectories will be colored according to their clusters.

Examples

```
### Generating artificial data
nbLignes <- 12
trajH <- matrix(0,nbLignes,10)

 for(i in 1:(nbLignes/3)){
   trajH[i,] <- pnorm(1:10,runif(1,3,8),1)*rnorm(1,10,1)
}
for(i in (nbLignes/3+1):(2*nbLignes/3)){
   trajH[i,] <- dnorm(1:10,runif(1,3,8),1)*rnorm(1,13,1)
}

for(i in (2*nbLignes/3+1):nbLignes){
   trajH[i,] <- pnorm(1:10,runif(1,3,8),1)*rnorm(1,5,0.1)
}

myClds <- cldsWide(data.frame(1:60,trajH))

par(mfrow=c(1,2))
plotTraj(myClds)
kmlShape(myClds,toPlot="none")
plotTraj(myClds)
```

---

plotSenators *~ Function: plotSenators ~*

Description

Plot the senators hold a an `linkS4class{Clds}` object.

Usage

```
plotSenators(x, col = 2:7, lty = 1:5, lwd=1, add = FALSE,main="", xlab="Times",ylab="")
```

Arguments

x	[`Clds`] Object holding the senators.
col	[`vector(numeric)`] or [`vector(character)`]: vector that define the trajectories' colors. If the length of the vector is one, then its value is duplicated.
lty	[`numeric`]: lines type.
lwd	[`numeric`]: lines width.
add	[`logical`]: shall the function start a new graph (add=FALSE) or add the lines to the current graph (add=TRUE) ?
main	[`character`]: main title of the graphical display.
xlab	[`character`]: x label of the graphical display.
ylab	[`character`]: y label of the graphical display.

Details

Plot the senators hold a an `linkS4class{Clds}` object.

Value

A graph.

Examples

```
data(ictusShort)
myClds <- cldsWide(ictusShort)
reduceTraj(myClds,nbSenators=4)
plotSenators(myClds)
reduceTraj(myClds,nbSenators=32)
plotSenators(myClds)
```

`plotTraj` *~ Function: plotTraj for Clds ~*

Description

`plotTraj` plot the trajectories of an object `Clds`.

Usage

```
## S4 method for signature 'Clds,missing'
plotTraj(x, y, col="clusters", pourcent=NA, ...)
```

Arguments

`x`	[`Clds`]: Object containing the trajectories to plot.
`y`	Useless. For compatibility only.
`col`	[`character`], [`integer`] or [`vector(integer)`]: color use for the trajectories. If the special value "clusters" is used, the trajectories will be colored according to their clusters.
`pourcent`	[`vector(numeric)`]: if pourcent is not NA, then the vector pourcent is used as value for the legend. If pourcent is NA, the legend is not printed.
`...`	Arguments to be passed to methods. These arguments need to be compatible with `matplot`.

Details

`plotTraj` the trajectories of an object `Clds`. If the option `col="clusters"` is used, the trajectories will be colored according to their clusters.

Examples

```
data(ictusShort)
myClds <- cldsWide(ictusShort)
plot(myClds)
```

---

reduceNbId *~ Function: reduceNbId ~*

Description

This function 'summerize' a (big) population in a smaller groups of individual. Hopefully, the smaller groups will have the same properties than the whole population. The trajectories of the smaller groups are called the 'senator' (since they are representing the whole population). The 'election' is done using the classical k-means algorithm. The trajectories should be in 'wide' format.

Usage

```
reduceNbId(id, trajWide, nbSenators = 64, imputationMethod = "linearInterpol")
```

Arguments

`id`	`[vector(factor)]`: unique identifier for each trajectories.
`trajWide`	`[data.frame]`: `data.frame` that hold the trajectories (in wide format).
`nbSenators`	`[integer]` number of trajectories that will be use to represent the population (i.e., number of clusters used by k-means).
`imputationMethod`	`[character]`: Method that will be used to impute the missing values.

Details

This function 'summerize' a (big) population in a smaller groups of individual. Hopefully, the smaller groups will have the same properties than the whole population. The trajectories of the smaller groups are called the 'senator' (since they are representing the whole population). The 'election' is done using the classical k-means algorithm. The trajectories should be in 'wide' format.

Value

A list with three fields:

- mySenator: `[data.frame]` whose first column is the individual identifier and whose second column is the 'senator' that represent the individual of the first column.
- senatorsWide `[matrix]` containing the trajectories of the senators, in wide format. The first column is an unique identifier for each senators.
- senatorsWeight`[vector(numeric)]` Number of individual that a senator is representing (i.e. number of individual that are in the cluster whose senator is the mean.)

Examples

```
par(mfrow=c(1,3))
### Some artificial data
myTraj <- t(sapply(1:1000,function(x)dnorm(1:200,runif(1,50,150),20)*rnorm(1,10,2)))
matplot(t(myTraj),type="l",ylim=c(0,0.33))

### Election of 64 senator
### All individual is closed to one senators. Senators are representatives.
election64 <- reduceNbId(id=1:1000,myTraj,nbSenators=64)
matplot(t(election64$senatorsWide[,-1]),type="l",ylim=c(0,0.33))

### Election of 4 senators. They are not representatives.
election4 <- reduceNbId(id=1:1000,myTraj,nbSenators=4)
matplot(t(election4$senatorsWide[,-1]),type="l",ylim=c(0,0.33))
```

`reduceNbTimes` *~ Function: reduceNbTimes ~*

Description

`reduceNbTimes` simplify some trajectories (in long format) by reducing their number of points.

Usage

```
reduceNbTimes(trajLong, nbPoints, spar=NA)
```

Arguments

`trajLong`	[`data.frame`]: `data.frame` that hold the trajectories in long format. The data.frame has to be (no choice!) in the following format: the first column should be the individual indentifiant. The second should be the times at which the measurement are made. The third one should be the measurement.
`nbPoints`	[`numeric`]: fixe the number of that the simplified trajectories should have.
`spar`	[`numeric`]: smoothing parameter that is used if the trajectories shall be smoothed before being simplified.

Details

`reduceNbTimes` simplify some trajectories by reducing their number of points. The trajectories should be in long format. If a value is given to `spar` (different from NA), trajectories are smoothed using `smooth.spline`.

The reduction of the number of point is done using a variation of Douglas-Peucker algorithme based on the number of points instead of an epsilon.

Value

A `data.frame` holding the simplified trajectories, in long format.

Author(s)

Christophe Genolini

See Also

reduceNbTimes, DouglasPeuckerEpsilon, DouglasPeuckerNbPoints

Examples

```
require(lattice)

### Some artificial data
g <- function(x)dnorm(x,3)+dnorm(x,7)+x/10
dn <- data.frame(id=rep(1:20,each=101),
  times=rep((0:100)/10,times=20),
  traj=rep(g((0:100)/10),20)+rep(runif(20),each=101)+rnorm(20*101,,0.1))

xyplot(traj ~ times, data=dn, groups=id,type="l")

### Reduction to 50 points
dn2 <- reduceNbTimes(trajLong=dn,nbPoints=50)
xyplot(traj ~ times, data=dn2, groups=id,type="l")

### Reduction to 20 points
dn3 <- reduceNbTimes(trajLong=dn,nbPoints=20)
xyplot(traj ~ times, data=dn3, groups=id,type="l")

### Smoothing then reduction to 20 points
dn4 <- reduceNbTimes(trajLong=dn,nbPoints=20,spar=0.5)
xyplot(traj ~ times, data=dn4, groups=id,type="l")
```

---

reduceTraj *~ Function: reduceTraj ~*

Description

This function 'summerize' a (big) population in a smaller groups of individual, then simplify the trajectories by reducing their number of points. It use reduceNbId and reduceNbTimes. Main difference with these two function, its applies on a Clds object.

Usage

```
reduceTraj(myClds, nbSenators = NA, nbTimes = NA, spar = 0.5,
   imputationMethod = "linearInterpol")
```

Arguments

`myClds`	[`Clds`]: object holding the trajectories that should be simplified.
`nbSenators`	[`integer`] number of trajectories that will be use to represent the population (i.e., number of clusters used by k-means).
`nbTimes`	[`numeric`]: fixe the number of that the simplified trajectories should have.
`spar`	[`numeric`]: smoothing parameter that is used if the trajectories shall be smoothed before being simplified.
`imputationMethod`	[`character`]: Method that will be used to impute the missing values.

Details

This function 'summerize' a (big) population in a smaller groups of individual, then simplify the trajectories by reducing their number of points. If 'nbSenators' is not NA, then `reduceNbId` is called. If 'nbTimes' is not NA, then `reduceNbTimes` is called. Note that 'nbSenators' and 'nbTimes' should not be both missing.

If both are non-missing, `reduceNbId` is called first. The results is store in the field 'senators' of the `Clds` object.

Value

A `Clds` object in which the fields 'senators', 'mySenators' and 'senatorsWeight' are now filled.

Examples

```
### Generating artificial data
nbLignes <- 200
trajG <- matrix(0,nbLignes,51)
for(i in 1:(nbLignes/2)){
   trajG[i,] <- dnorm(0:50,runif(1,15,35),5)*rnorm(1,10,0.1)
}
for(i in (nbLignes/2+1):nbLignes){
   trajG[i,] <- dnorm(0:50,runif(1,15,35),5)*rnorm(1,5,0.1)
}
myClds <- cldsWide(data.frame(1:200,trajG))
plot(myClds)

### Reducing the number of time measurement
reduceTraj(myClds,nbTimes=7)
plotSenators(myClds)

### Reducing the number of individual
reduceTraj(myClds,nbSenators=32)
plotSenators(myClds)

### Reducing both
reduceTraj(myClds,nbSenators=32,nbTimes=7)
plotSenators(myClds)
```

Index