Package 'TDA'

August 20, 2014

Type Package

Title Statistical Tools for Topological Data Analysis

Version 1.1

Date 2014-08-20

Author Brittany T. Fasy, Fabrizio Lecci

Maintainer Fabrizio Lecci <lecci@cmu.edu>

Description This package provides tools for the statistical analysis of persistent homology and for density clustering.

Depends FNN, igraph, parallel, scales

License GPL (>= 2.0)

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-20 16:16:21

R topics documented:

`TDA-package` *Statistical Tools for Topological Data Analysis*

Description

This package provides some tools for Topological Data Analysis. In particular it provides functions for the statistical analysis of persistent homology and for density clustering.

Details

Package:	TDA
Type:	Package
Version:	1.1
Date:	2014-08-20
License:	GPL (>= 2.0)

Author(s)

Brittany Terese Fasy and Fabrizio Lecci

Maintainer: Fabrizio Lecci <lecci@cmu.edu>

References

Herbert Edelsbrunner, and John Harer, (2010), "Computational topology: an introduction". American Mathematical Society.

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology: Confidence Sets for Persistence Diagrams", (arXiv:1303.7117). To appear, Annals of Statistics.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." Technical Report.

Frederic Chazal, Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman, (2014), "Stochastic Convergence of Persistence Landscapes and Silhouettes", Proceedings of the 30th Symposium of Computational Geometry (SoCG). (arXiv:1312.0308)

Frederic Chazal, Brittany T. Fasy, Fabrizio Lecci, Bertrand Michel, Alessandro Rinaldo, and Larry Wasserman, (2014), "Subsampling Methods for Persistent Homology". (arXiv:1406.1901)

http://www.mrzv.org/software/dionysus/

`bootstrapBand` *Bootstrap Confidence Band*

Description

`bootstrapBand` computes a uniform symmetric confidence band around a function of the data `X`, evaluated on a `Grid`, using the bootstrap algorithm. See Details and References.

Usage

```
bootstrapBand(X, FUN, Grid, B = 30, alpha = 0.05, parallel = FALSE,
              printStatus=FALSE, ...)
```

Arguments

`X`	an n by d matrix of coordinates of points used by the function `FUN`, where n is the number of points and d is the dimension.
`FUN`	a function whose inputs are an n by d matrix of coordinates `X`, an m by d matrix of coordinates `Grid` and returns a numeric vector of length m. For example see `distFct`, `kde`, and `dtm` which compute the distance function, the kernel density estimator and the distance to measure over a grid of points, using the input `X`.
`Grid`	an m by d matrix of coordinates, where m is the number of points in the grid.
`B`	the number of bootstrap iterations.
`alpha`	`bootstrapBand` returns a (1-`alpha`) confidence band.
`parallel`	logical: if `TRUE` the bootstrap iterations are parallelized, using the library `parallel`.
`printStatus`	if TRUE a progress bar is printed. Default is `FALSE`.
`...`	additional parameters for the function `FUN`.

Details

First, the input function `FUN` is evaluated on the `Grid` using the original data `X`. Then, for `B` times, the bootstrap algorithm subsamples `n` points of `X` (with replacement), evaluates the function `FUN` on the `Grid` using the subsample, and computes the ℓ_∞ distance between the original function and the bootstrapped one. The result is a sequence of `B` values. The (1-`alpha`) confidence band is constructed by taking the (1-`alpha`) quantile of these values.

Value

Returns a list with the following elements:

`width`	number: (1-`alpha`) quantile of the values computed by the bootstrap algorithm. It corresponds to half of the width of the unfiorm confidence band; that is, `width` is the distance of the upper and lower limits of the band from the function evaluated using the original dataset `X`.
`fun`	a numeric vector of length m, storing the values of the input function `FUN`, evaluated on the `Grid` using the original data `X`.
`band`	an m by 2 matrix that stores the values of the lower limit of the confidence band (first column) and upper limit of the confidence band (second column), evaluated over the `Grid`.

Author(s)

Fabrizio Lecci

References

Larry Wasserman (2004), "All of statistics: a concise course in statistical inference", Springer.

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology: Confidence Sets for Persistence Diagrams", (arXiv:1303.7117). To appear, Annals of Statistics.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." Technical Report.

See Also

`kde`, `dtm`

Examples

```
# Generate data from mixture of 2 normals.
n = 2000
X = c(rnorm(n/2), rnorm(n/2, mean=3, sd=1.2))

# Construct a grid of points over which we evaluate the function
by=0.02
Grid=seq(-3, 6, by=by)

## bandwidth for kernel density estimator
h=0.3
## Bootstrap confidence band
band= bootstrapBand(X, kde, Grid, B=80, parallel=FALSE, alpha=0.05, h=h)

plot(Grid,band$fun, type="l", lwd=2, ylim=c(0, max(band$band)),
     main="kde with 0.95 confidence band")
lines(Grid, pmax(band$band[,1],0), col=2, lwd=2)
lines(Grid, band$band[,2], col=2, lwd=2)
```

`bottleneck` *Bottleneck distance between two persistence diagrams*

Description

This function computes the bottleneck distance between two persistence diagrams

Usage

```
bottleneck(Diag1, Diag2, dimension)
```

Arguments

`Diag1`	an object of class `diagram` or a matrix (n by 3) that stores dimension, birth and death of n topological features.
`Diag2`	an object of class `diagram` or a matrix (m by 3) that stores dimension, birth and death of m topological features.
`dimension`	an integer specifying the dimension of the features used to compute the bottleneck distance. 0 for connected components, 1 for loops, 2 for voids and so on.

Details

The bottleneck distance between two diagrams is the cost of the optimal matching between points of the two diagrams. Note that all the diagonal points are included in the persistence diagrams when computing the optimal matching. This function is an R wrapper of the function "bottleneck_distance" in the C++ library Dionysus. See references.

Value

Returns the value of the bottleneck distance between the two persistence diagrams.

Author(s)

Fabrizio Lecci

References

http://www.mrzv.org/software/dionysus/

Herbert Edelsbrunner and John Harer (2010), Computational topology: an introduction. American Mathematical Society.

See Also

`ripsDiag`, `gridDiag`, `plot.diagram`

Examples

```
XX1 = circleUnif(20)
XX2 = circleUnif(20, r=0.2)

DiagLim=5
maxdimension=1

Diag1=ripsDiag(XX1,maxdimension,DiagLim, printStatus=FALSE)
Diag2=ripsDiag(XX2,maxdimension,DiagLim, printStatus=FALSE)

bottleneckDist=bottleneck(Diag1, Diag2, dimension=1)
print(bottleneckDist)
```

`bottleneckInterval` *Bootstrapped Confidence Set for a Persistence Diagram, using the Bottleneck Distance.*

Description

`bottleneckInterval` computes a `(1-alpha)` confidence set for the Persistence Diagram of a filtration of sublevel sets (or superlevel sets) of a function evaluated over a grid of points in dimension d =1, 2 or 3. The function returns the `(1-alpha)` quantile of `B` bottleneck distances, computed in `B` iterations of the bootstrap algorithm. The method is discussed in the 1st reference.

Usage

```
bottleneckInterval(X, FUN, Xlim, Ylim = NA, Zlim = NA, by=(Xlim[2]-Xlim[1])/20,
        sublevel = TRUE, B=30, alpha=0.05, dimension=1, printStatus = FALSE, ...)
```

Arguments

`X`	an n by d matrix of coordinates, used by the function `FUN`, where n is the number of points stored in `X` and d is the dimension (1, 2 or 3).
`FUN`	a function whose inputs are 1) an n by d matrix of coordinates `X`, 2) an m by d matrix of coordinates `Grid`, 3) an optional smoothing parameter, and returns a numeric vector of length m. For example see `distFct`, `kde`, and `dtm` which compute the distance function, the kernel density estimator and the distance to measure, over a grid of points using the input `X`. Note that `Grid` is not an input of `bottleneckInterval`, but is automatically computed by the function using `Xlim`, `Ylim`, `Zlim`, and `by`.
`Xlim`	a numeric vector of length 2, specifying the range of the first dimension of the grid, over which the function `FUN` is evaluated.
`Ylim`	a numeric vector of length 2, specifying the range of the second dimension of the grid, over which the function `FUN` is evaluated. `NA` for a 1 dimensional grid.
`Zlim`	a numeric vector of length 2, specifying the range of the third dimension of the grid, over which the function `FUN` is evaluated. `NA` for a 1 dimensional or 2 dimensional grid.

by	number: space between points of the grid in each dimension.
sublevel	a logical variable indicating if the Persistence Diagram should be computed for sublevel sets (TRUE) or superlevel sets (FALSE) of the function. Default is TRUE.
B	the number of bootstrap iterations.
alpha	bottleneckInterval returns a (1-alpha) quantile.
dimension	an integer specifying the dimension of the features used to compute the bottleneck distance. 0 for connected components, 1 for loops, 2 for voids.
printStatus	if TRUE a progress bar is printed. Default is FALSE.
...	additional parameters for the function FUN.

Details

bottleneckInterval uses gridDiag to compute the persistence diagram of the input function using the entire sample. Then the bootstrap algorithm, for B times, computes the bottleneck distance between the original persistence diagram and the one computed using a subsample. Finally the (1-alpha) quantile of these B values is returned.

Value

Returns the (1-alpha) quantile of the values computed by the bootstrap algorithm. It corresponds to half of width of the confidence set for the persistence diagram.

Note

This function uses the C++ library Dionysus. See references.

Author(s)

Fabrizio Lecci

References

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." Technical Report.

Larry Wasserman (2004), "All of statistics: a concise course in statistical inference", Springer.

http://www.mrzv.org/software/dionysus/

See Also

bottleneck, bootstrapBand, distFct, kde, kernelDist, dtm, summary.diagram, plot.diagram,

Examples

```
## confidence set for the Kernel Density Diagram

# input data
n = 400
XX = circleUnif(n)
```

```
## Ranges of the grid
Xlim=c(-1.8,1.8)
Ylim=c(-1.6,1.6)
by=0.05

h = .3  #bandwidth for the function kde

#Kernel Density Diagram of the superlevel sets
Diag=gridDiag(XX, kde, Xlim, Ylim, by=by, sublevel=FALSE, printStatus=TRUE, h=h)

# confidence set
B=10         ## the number of bootstrap iterations should be higher!
             ## this is just an example
alpha=0.05

cc=bottleneckInterval(XX, kde, Xlim, Ylim, by=by, sublevel=FALSE, B=B, alpha=alpha,
   dimension=1, printStatus=TRUE, h=h)

plot(Diag, band=2*cc)
```

`circleUnif` *Uniform Sample From The Circle*

Description

This function samples `n` points from the circle of radius `r`, uniformly with respect to the circumference length.

Usage

```
circleUnif(n, r = 1)
```

Arguments

`n`	an integer specifying the number of points in the sample.
`r`	a numeric variable specifying the radius of the circle. Default is 1.

Value

`circleUnif` returns an `n` by 2 matrix of coordinates.

Author(s)

Fabrizio Lecci

See Also

`torusUnif`

Examples

```
X=circleUnif(100)
plot(X)
```

`clusterTree` *Density clustering: the cluster tree*

Description

Given a point cloud, or a matrix of distances, this function computes a density estimates and returns an object of class `clusterTree` (lambda tree and kappa tree; see references).

Usage

```
clusterTree(X, k, h = NULL, density="knn", dist="euclidean", d=NULL,
            Nlambda = 100, printStatus = FALSE)
```

Arguments

`X`	If `dist="euclidean"` then `X` is an n by d matrix of coordinates, where n is the number of points stored in `X` and d is the dimension of the space. If `dist="arbitrary"` then `X` is an n by n matrix of distances. Default is `"euclidean"`
`k`	an integer value specifying the parameter of the underlying k-nearest neighbor similarity graph, used to determine connected components. If `density="knn"`, then `k` is also used to compute the k-nearest neighbor density estimator.
`h`	real value: if `density="kde"`, then `h` is used to compute the kernel density estimator with bandwidth `h`. Default is `NULL`.
`density`	string: if `"knn"` then the k-nearest neighbor density estimator is used to compute the cluster tree; if `"kde"` then the kernel density estimator is used to compute the cluster tree. Default is "knn".
`dist`	string: can be `"euclidean"`, when `X` is a point cloud or `"arbitrary"`, when `X` is a matrix of distances.
`d`	integer: if `dist="arbitrary"`, then `d` is the dimension of the underlying space.
`Nlambda`	integer: size of the grid of values of the density estimator, used to compute the cluster tree. High `Nlambda` (i.e. a fine grid) means a more accurate cluster Tree.
`printStatus`	logical: if TRUE a progress bar is printed. Default is `FALSE`.

Details

This function is an implementation of Algorithm 1 in the first reference.

Value

This function returns an object of class `clusterTree`, a list with the following components

`n`	The number of points stored in the input matrix `X`
`id`	Vector: the IDs associated to the branches of the cluster tree
`Xbase`	Vector: the horiontal coordinates of the branches of the cluster tree, in the same order of `id`
`Ybottom`	Vector: the vertical bottom coordinates of the branches of the lambda tree, in the same order of `id`
`Ytop`	Vector: the vertical top coordinates of the branches of the lambda tree, in the same order of `id`
`Kbottom`	Vector: the vertical bottom coordinates of the branches of the kappa tree, in the same order of `id`
`Ktop`	Vector: the vertical top coordinates of the branches of the kappa tree, in the same order of `id`
`silo`	A list whose elements are the horizontal coordinates of the silo of each branch, in the same order of `id`
`sons`	A list whose elements are the IDs of the sons of each branch, in the same order of `id`
`parent`	Vector: the IDs of the parents of each branch, in the same order of `id`
`DataPoints`	A list whose elements are the points of `X` corresponding to each branch, in the same order of `id`
`density`	Vector of length `n`: the values of the density estimator evaluated at each of the points stored in `X`

Author(s)

Fabrizio Lecci

References

Brian P. Kent, Alessandro Rinaldo, and Timothy Verstynen, (2013), "DeBaCl: A Python Package for Interactive DEnsity-BAsed CLustering."arXiv:1307.8136

Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman, (2014), "Metric Embeddings for Cluster Trees"

See Also

`plot.clusterTree`

Examples

```
## Generate data: 3 clusters
n=1200    #sample size
Neach=floor(n/4)
X1=cbind(rnorm(Neach,1,.8),rnorm(Neach,5,0.8))
X2=cbind(rnorm(Neach,3.5,.8),rnorm(Neach,5,0.8))
X3=cbind(rnorm(Neach,6,1),rnorm(Neach,1,1))
X=rbind(X1,X2,X3)

k=100     #parameter of knn

## Density clustering using knn and kde
Tree=clusterTree(X,k, density="knn")
TreeKDE=clusterTree(X,k,h=0.3, density="kde")

par(mfrow=c(2,3))
plot(X, pch=19, cex=0.6)
# plot lambda trees
plot(Tree, type="lambda", main="lambda Tree (knn)")
plot(TreeKDE, type="lambda", main="lambda Tree (kde)")
# plot clusters
plot(X, pch=19, cex=0.6, main="cluster labels")
for (i in Tree$id){
points(matrix(X[Tree$DataPoints[[i]],],ncol=2), col=i, pch=19, cex=0.6)
}
#plot kappa trees
plot(Tree, type="kappa", main="kappa Tree (knn)")
plot(TreeKDE, type="kappa", main="kappa Tree (kde)")
```

`distFct` *Distance function*

Description

This function computes the distance between each point of a set `Grid` and the corresponding closest point of another set `X`.

Usage

```
distFct(X, Grid)
```

Arguments

`X`	a numeric m by d matrix of coordinates in the space, where m is the number of points in `X` and d is the dimension of the space.
`Grid`	a numeric n by d matrix of coordinates in the space, where n is the number of points in `Grid` and d is the dimension of the space.

Details

Given a set of points X, the distance function computed at g is defined as

$$d(g) = \inf_{x \in X} \|x - g\|_2$$

Value

Returns a numeric vector of length n, where n is the number of points stored in Grid.

Author(s)

Fabrizio Lecci

See Also

kde,kernelDist, dtm

Examples

```
## Generate Data from the unit circle
n = 300
X = circleUnif(n)

## Construct a grid of points over which we evaluate the function
by=0.065
Xseq=seq(-1.6, 1.6, by=by)
Yseq=seq(-1.7, 1.7, by=by)
Grid=expand.grid(Xseq,Yseq)

## distance fct
distance= distFct(X, Grid)
```

---

dtm *Distance to Measure Function*

Description

This function computes the "distance to measure function" on a set of points Grid, using the uniform empirical measure on a set of points X. Given a probability measure P, The distance to measure function, for each $y \in R^d$, is defined by

$$d_m = \sqrt{\frac{1}{m} \int_0^m (G_y^{-1}(u))^2 du},$$

where $G_y(t) = P(\|X - y\| \leq t)$ and $0 < m < 1$ is a smoothing parameter. See Details and References.

Usage

```
dtm(X, Grid, m0)
```

Arguments

X	a matrix of coordinates of points, that are used to construct the uniform empirical measure for the distance to measure.
Grid	an n by d matrix of coordinates, where n is the number of points in Grid.
m0	a numeric variable for the smoothing parameter of the distance to measure. Roughly, m0 is the the percentage of points of X that are considered when the distance to measure is computed for each point of Grid.

Details

See Definition 3.2 of the reference for a formal definition of the "distance to measure" function.

Value

dtm returns a vector of length n (the number of points stored in Grid) containing the value of the distance to measure function for each point of Grid.

Author(s)

Fabrizio Lecci

References

Frederic Chazal, David Cohen-Steiner, and Quentin Merigot. "Geometric inference for probability measures." Foundations of Computational Mathematics 11.6 (2011): 733-751.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." Technical Report.

See Also

kde, kernelDist, distFct

Examples

```
## Generate Data from the unit circle
n = 300
X = circleUnif(n)

## Construct a grid of points over which we evaluate the function
by=0.065
Xseq=seq(-1.6, 1.6, by=by)
Yseq=seq(-1.7, 1.7, by=by)
Grid=expand.grid(Xseq,Yseq)

## distance to measure
```

```
m0=0.1
DTM=dtm(X, Grid, m0)
```

---

gridDiag *Persistence Diagram of a function over a Grid*

Description

gridDiag computes the Persistence Diagram of a filtration of sublevel sets (or superlevel sets) of a function evaluated over a grid of points in dimension d =1, 2 or 3.

Usage

```
gridDiag(X, FUN, Xlim, Ylim = NA, Zlim = NA, by=(Xlim[2]-Xlim[1])/20,
         sublevel = TRUE, printStatus = FALSE, diagLimit=NULL, ...)
```

Arguments

X	an n by d matrix of coordinates, used by the function FUN, where n is the number of points stored in X and d is the dimension (1, 2 or 3).
FUN	a function whose inputs are 1) an n by d matrix of coordinates X, 2) an m by d matrix of coordinates Grid, 3) an optional smoothing parameter, and returns a numeric vector of length m. For example see distFct, kde, and dtm which compute the distance function, the kernel density estimator and the distance to measure, over a grid of points using the input X. Note that Grid is not an input of gridDiag, but is automatically computed by the function using Xlim, Ylim, Zlim, and by.
Xlim	a numeric vector of length 2, specifying the range of the first dimension of the grid, over which the function FUN is evaluated.
Ylim	a numeric vector of length 2, specifying the range of the second dimension of the grid, over which the function FUN is evaluated. NA for a 1 dimensional grid.
Zlim	a numeric vector of length 2, specifying the range of the third dimension of the grid, over which the function FUN is evaluated. NA for a 1 dimensional or 2 dimensional grid.
by	number: space between points of the grid in each dimension.
sublevel	a logical variable indicating if the Persistence Diagram should be computed for sublevel sets (TRUE) or superlevel sets (FALSE) of the function. Default is TRUE.
printStatus	if TRUE a progress bar is printed. Default is FALSE.
diagLimit	a number that replaces Inf (if sublevel is TRUE) or -Inf (if sublevel is FALSE) in the Death value of the most persistent connected component. Deafult is NULL and the min/max of the function is used.
...	additional parameters for the function FUN.

Details

The function evaluates the function FUN over a grid. Then it constructs a filtration by triangulating the grid and considering the simplices determined by the values of the function.

Value

gridDiag returns an object of class diagram, a P by 3 matrix, where P is the number of points in the resulting persistence diagram. The first column contains the dimension of each feature (0 for components, 1 for loops, 2 for voids). Second and third columns are Birth and Death of features, in case of a filtration constructed using sublevel sets, or Death and Birth of features, in case of a filtration constructed using superlevel sets.

Note

This function uses the C++ library Dionysus. See references.

Author(s)

Brittany T. Fasy and Fabrizio Lecci

References

Brittany Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology", (arXiv:1303.7117). To appear, Annals of Statistics.

http://www.mrzv.org/software/dionysus/

See Also

summary.diagram, plot.diagram, distFct, kde, kernelDist, dtm, ripsDiag

Examples

```
## Distance Function Diagram and Kernel Density Diagram

# input data
n = 300
XX = circleUnif(n)

## Ranges of the grid
Xlim=c(-1.8,1.8)
Ylim=c(-1.6,1.6)
by=0.05

h = .3  #bandwidth for the function kde

#Distance Function Diagram of the sublevel sets
Diag1=gridDiag(XX,distFct, Xlim, Ylim, by=by, sublevel=TRUE, printStatus=TRUE)

#Kernel Density Diagram of the superlevel sets
```

```
Diag2=gridDiag(XX, kde, Xlim, Ylim, by=by, sublevel=FALSE, printStatus=TRUE, h=h)

#plot
par(mfrow=c(1,3))
plot(XX,cex=0.5, pch=19)
title(main="Data")
plot(Diag1)
title(main="Distance Function Diagram")
plot(Diag2)
title(main="Density Persistence Diagram")
```

`hausdInterval` *Subsampling Confidence Interval for the Hausdorff Distance between a Manifold and a Sample*

Description

`hausdInterval` computes a confidence interval for the Hausdorff distance between a point cloud `X` and the underlying manifold from which `X` was sampled. See Details. The validity of the method is proved in the 1st Reference.

Usage

```
hausdInterval(X, m, B = 30, alpha = 0.05, parallel = FALSE,
              printStatus=FALSE)
```

Arguments

`X`	an n by d matrix of coordinates of sampled points.
`m`	the size of the subsamples.
`B`	the number of subsampling iterations.
`alpha`	`hausdInterval` returns a (1-`alpha`) confidence interval.
`parallel`	logical: if `TRUE` the iterations are parallelized, using the library `parallel`.
`printStatus`	if TRUE a progress bar is printed. Default is `FALSE`.

Details

For `B` times, the subsampling algorithm subsamples `m` points of `X` (without replacement) and computes the Hausdorff distance between the original sample `X` and the subsample. The result is a sequence of `B` values. Let q be the (1-`alpha`) quantile of these values and let $c = 2 * q$. The interval $[0, c]$ is a valid (1-`alpha`) confidence interval for the Hausdorff distance between `X` and the underlying manifold, as proven in Theorem 3 of the first reference.

Value

Returns a number c. The confidence interval is $[0, c]$.

Author(s)

Fabrizio Lecci

References

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology: Confidence Sets for Persistence Diagrams", (arXiv:1303.7117). To appear, Annals of Statistics.

See Also

bootstrapBand

Examples

```
print(0)
```

---

kde *Kernel Density Estimator over a Grid of Points*

Description

Given a point cloud `X` (n points), this function computes the Kernel Density Estimator over a grid of points. The kernel is a Gaussian Kernel with smoothing parameter `h`. For each $x \in R^d$, the Kernel Density estimator is defined as

$$p_X(x) = \frac{1}{n(\sqrt{2\pi}h)^d} \sum_{i=1}^{n} \exp\left(\frac{-\|x - X_i\|_2^2}{2h^2}\right).$$

Usage

```
kde(X, Grid, h)
```

Arguments

`X`	an n by d matrix of coordinates of points used in the kernel density estimation process, where n is the number of points and d is the dimension.
`Grid`	an m by d matrix of coordinates, where m is the number of points in the grid.
`h`	number: the smoothing paramter of the Gaussian Kernel.

Value

`kde` returns a vector of length m (the number of points in the grid) containing the value of the kernel density estimator for each point in the grid.

Author(s)

Fabrizio Lecci

References

Larry Wasserman (2004), "All of statistics: a concise course in statistical inference", Springer.

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology: Confidence Sets for Persistence Diagrams", (arXiv:1303.7117). To appear, Annals of Statistics.

See Also

kernelDist, distFct, dtm

Examples

```
## Generate Data from the unit circle
n = 300
X = circleUnif(n)

## Construct a grid of points over which we evaluate the function
by=0.065
Xseq=seq(-1.6, 1.6, by=by)
Yseq=seq(-1.7, 1.7, by=by)
Grid=expand.grid(Xseq,Yseq)

## kernel density estimator
h=0.3
KDE= kde(X, Grid, h)
```

---

kernelDist *Kernel distance over a Grid of Points*

Description

Given a point cloud X, this function computes the kernel distance over a grid of points. The kernel is a Gaussian Kernel with smoothing parameter h:

$$K_h(x,y) = \exp\left(\frac{-\|x-y\|_2^2}{2h^2}\right).$$

For each $x \in R^d$ the Kernel distance is defined by

$$\kappa_X(x) = \sqrt{\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} K_h(X_i,X_j) + K_h(x,x) - 2\frac{1}{n}\sum_{i=1}^{n} K_h(x,X_i)}.$$

Usage

```
kernelDist(X, Grid, h)
```

Arguments

X	an n by d matrix of coordinates of points, where n is the number of points and d is the dimension.
Grid	an m by d matrix of coordinates, where m is the number of points in the grid.
h	number: the smoothing paramter of the Gaussian Kernel.

Value

`kernelDist` returns a vector of lenght m (the number of points in the grid) containing the value of the Kernel distance for each point in the grid.

Author(s)

Fabrizio Lecci

References

Jeff M. Phillips, Bei Wang, and Yan Zheng (2013), "Geometric Inference on Kernel Density Estimates," arXiv:1307.7760.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." Technical Report.

See Also

`kde`, `dtm`, `distFct`

Examples

```
## Generate Data from the unit circle
n = 300
X = circleUnif(n)

## Construct a grid of points over which we evaluate the functions
by=0.065
Xseq=seq(-1.6, 1.6, by=by)
Yseq=seq(-1.7, 1.7, by=by)
Grid=expand.grid(Xseq,Yseq)

## kernel distance estimator
h=0.3
Kdist= kernelDist(X, Grid, h)
```

knnDE *k Nearest Neighbors Density Estimator over a Grid of Points*

Description

Given a point cloud X (n points), this function computes the k Nearest Neighbors Density Estimator over a grid of points. For each $x \in R^d$, the knn Density Estimator is defined by

$$p_X(x) = \frac{k}{n\ v_d\ r_k^d(x)},$$

where v_n is the volume of the Euclidean d dimensional unit ball and $r_k^d(x)$ is the Euclidean distance form point x to its k'th closest neighbor.

Usage

```
knnDE(X, Grid, k)
```

Arguments

X	an n by d matrix of coordinates of points used in the density estimation process, where n is the number of points and d is the dimension.
Grid	an m by d matrix of coordinates, where m is the number of points in the grid.
k	number: the smoothing paramter of the k Nearest Neighbors Density Estimator.

Value

knnDE returns a vector of length m (the number of points in the grid) containing the value of the knn Density Estimator for each point in the grid.

Author(s)

Fabrizio Lecci

See Also

kde,kernelDist, distFct, dtm

Examples

```
## Generate Data from the unit circle
n = 300
X = circleUnif(n)

## Construct a grid of points over which we evaluate the function
by=0.065
Xseq=seq(-1.6, 1.6, by=by)
Yseq=seq(-1.7, 1.7, by=by)
```

```
Grid=expand.grid(Xseq,Yseq)

## kernel density estimator
k=50
KNN= knnDE(X, Grid, k)
```

`landscape` *The Persistence Landscape Function*

Description

This function computes the landscape function corresponding to a given persistence diagram.

Usage

```
landscape(Diag, dimension = 1, KK = 1,
          tseq=seq( min(Diag[,2:3]), max(Diag[,2:3]), length=500) )
```

Arguments

`Diag`	an object of class `diagram` or a P by 3 matrix, storing a persistence diagram with columns: "dimension", "Birth", "Death".
`dimension`	the dimension of the topological features under consideration. Default is 1 (loops).
`KK`	the order of the landscape function. Default is 1. (First Landscape function).
`tseq`	a vector of values at which the landscape function is evaluated.

Value

Returns a numeric vector of the same length of `tseq`, with the values of the landscape function evaluated at each point of `tseq`.

Author(s)

Fabrizio Lecci

References

Peter Bubenik, (2012), "Statistical topology using persistence landscapes", arXiv1207.6437.

Frederic Chazal, Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman, (2014), "Stochastic Convergence of Persistence Landscapes and Silhouettes", Proceedings of the 30th Symposium of Computational Geometry (SoCG). (arXiv:1312.0308)

See Also

`silhouette`

Examples

```
Diag=matrix(c(0,0,10,1,0,3,1,3,8), ncol=3, byrow=TRUE)
DiagLim=10
colnames(Diag)=c("dim", "Birth", "Death")

#persistence landscape
tseq=seq(0,DiagLim, length=1000)
Land= landscape(Diag, dimension=1, KK=1, tseq)

par(mfrow=c(1,2))
plot.diagram(Diag)
plot(tseq, Land, type="l", xlab="t", ylab="landscape", asp=1)
```

`maxPersistence` *Maximal Persistence Method*

Description

Given a point cloud and a function built on top of the data, we are interested in studying the evolution of the sublevel sets (or superlevel sets) of the function, using persistent homology. The Maximal Persistence Method selects the optimal smoothing parameter of the function, by maximizing the number of significant topological features, or by maximizing the total significant persistence of the features. For each value of the smoothing parameter, this function computes a persistence diagram using `gridDiag` and returns the values of the two criteria, the dimension of detected features, their persistence, and a bootstrapped confidence band. The features that fall outside of the band are statistically significant. See References.

Usage

```
maxPersistence(FUN, parameters, X, Xlim, Ylim=NA, Zlim=NA, by, sublevel = TRUE,
             B = 30, alpha = 0.05, parallel = FALSE, printProgress = FALSE)
```

Arguments

`FUN`	the name of a function whose inputs are: 1) `X`, a n by d matrix of coordinates of the input point cloud, where d is the dimension of the space; 2) a matrix of coordinates of points forming a grid at which the function can be evaluated (note that this grid is not passed as an input, but is automatically computed by `maxPersistence`); 3) a real valued smoothing parameter. For example, see `kde`, `dtm`, `kernelDist`.
`parameters`	a numerical vector, storing a sequence of values for the smoothing paramter of `FUN` among which `maxPersistence` will select the optimal ones.
`X`	a n by d matrix of coordinates of the input point cloud, where d is the dimension of the space.
`Xlim`	a numeric vector of length 2, specifying the range of the first dimension of the grid, over which the function `FUN` is evaluated.

Ylim	a numeric vector of length 2, specifying the range of the second dimension of the grid, over which the function FUN is evaluated. NA for a 1 dimensional grid.
Zlim	a numeric vector of length 2, specifying the range of the third dimension of the grid, over which the function FUN is evaluated. NA for a 1 dimensional or 2 dimensional grid.
by	number: space between points of the grid in each dimension.
sublevel	a logical variable indicating if the persistent homology should be computed for sublevel sets of FUN (TRUE) or superlevel sets (FALSE). Default is TRUE.
B	the number of bootstrap iterations.
alpha	for each value store in parameters, maxPersistence computes a (1-alpha) confidence band.
parallel	logical: if TRUE the bootstrap iterations are parallelized, using the library parallel.
printProgress	if TRUE a progress bar is printed. Default is FALSE.

Details

maxPersistence calls the gridDiag function, which computes the persistence diagram of sublevel (or superlevel) sets of a function, evaluated over a grid of points in dimension 1,2, or 3.

Value

The function returns an object of the class "maxPersistence", a list with the following components

parameters	the same vector parameters given in input
sigNumber	a numeric vector storing the number of significant features in the persistence diagrams computed using each value in parameters
sigPersistence	a numeric vector storing the sum of significant persistence of the features in the persistence diagrams, computed using each value in parameters
bands	a numeric vector storing the bootstrap band's width, for each value in parameters
Persistence	a list of the same lenght of parameters. Each element of the list is a P_i by 2 matrix, where P_i is the number of features found using the parameter i: the first column stores the dimension of each feature and the second column the persistence abs(death-birth|).

Author(s)

Fabrizio Lecci

References

Frederic Chazal, Jessi Cisewski, Brittany T. Fasy, Fabrizio Lecci, Bertrand Michel, Alessandro Rinaldo, and Larry Wasserman, (2014), "Robust Topological Inference: distance-to-a-measure and kernel distance"

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology", (arXiv:1303.7117). To appear, Annals of Statistics.

See Also

gridDiag, kde, kernelDist, dtm, bootstrapBand

Examples

```
## input data: circle with clutter noise
n=600
percNoise=0.1
XX1 = circleUnif(n)
noise=cbind(runif(percNoise*n, -2,2),runif(percNoise*n, -2,2))
X=rbind(XX1,noise)

## limits of the Gird at which the density estimator is evaluated
Xlim=c(-2,2)
Ylim=c(-2,2)
by=0.2

B=80
alpha=0.05

## candidates
parametersKDE=seq(0.1,0.5, by=0.2)

maxKDE=maxPersistence(kde, parametersKDE, X, Xlim, Ylim, Zlim=NA, by=by, B=B,
                  alpha=alpha, parallel=FALSE, printProgress = TRUE)
print(summary(maxKDE))

par(mfrow=c(1,2))
plot(X, pch=16, cex=0.5, main="Circle")
plot(maxKDE)
```

---

multipBootstrap *Multiplier Bootstrap for Persistence Landscapes and Silhouettes*

Description

This function computes a confidence band for the average landscape (or the average silhouette) using the multiplier bootstrap.

Usage

```
multipBootstrap(Y, B=30, alpha=0.05, parallel=FALSE, printStatus=FALSE)
```

Arguments

Y	an N by m matrix of values of N persistence landscapes (or silhouettes) evaluated over a 1 dimensional grid of length m.
B	the number of bootstrap iterations.

alpha	multipBootstrap returns a 1-alpha confidence band for the mean landscape (or silhouette).
parallel	logical: if TRUE the bootstrap iterations are parallelized, using the library parallel.
printStatus	logical: if TRUE a progress bar is printed. Default is FALSE.

Details

See Algorithm 1 in the reference.

Value

Returns a list with the following elements:

width	number: half of the width of the unfiorm confidence band; that is, the distance of the upper and lower limits of the band from the empirical average landscape (or silhouette).
mean	a numeric vector of length m, storing the values of the empirical average landscape (or silhouette) over a 1 dimensional grid of length m.
band	an m by 2 matrix that stores the values of the lower limit of the confidence band (first column) and upper limit of the confidence band (second column), evaluated over a 1 dimensional grid of length m.

Author(s)

Fabrizio Lecci

References

Chazal, F., Fasy, B.T., Lecci, F., Rinaldo, A., and Wasserman, L., (2014), "Stochastic Convergence of Persistence Landscapes and Silhouettes", Proceedings of the 30th Symposium of Computational Geometry (SoCG). (arXiv:1312.0308)

See Also

landscape, silhouette

Examples

```
nn=3000  #large sample size
mm=50   #small subsample size
NN=5    #we will compute NN diagrams using subsamples of size mm

XX = circleUnif(nn)  ## large sample from the unit circle

DiagLim=2
maxdimension=1
tseq=seq(0,DiagLim, length=1000)

Diags=list()    #here we will store the NN rips diagrams
                #constructed using different subsamples of mm points
```

```
Lands=matrix(0,nrow=NN, ncol=length(tseq))  #here we'll store the landscapes

for (i in 1:NN){
subXX=XX[sample(1:nn,mm),]
Diags[[i]]=ripsDiag(subXX,maxdimension,DiagLim)
Lands[i,]=landscape(Diags[[i]], dimension=1, KK=1, tseq )
}

## now we use the NN landscapes to construct a confidence band
B=50
alpha=0.05
boot=multipBootstrap(Lands,B,alpha)

LOWband=boot$band[,1]
UPband=boot$band[,2]
MeanLand=boot$mean

plot(tseq, MeanLand, type="l", lwd=2, xlab=" ", ylab="" ,
     main="Mean Landscape with band", ylim=c(0,1.2))
polygon(c(tseq, rev(tseq)), c(LOWband,rev(UPband)), col="pink")
lines(tseq, MeanLand, lwd=1, col=2)
```

`plot.clusterTree` *Plots the Cluster Tree*

Description

This function plots the Cluster Tree stored in an object of class `clusterTree`.

Usage

```
## S3 method for class 'clusterTree'
plot(x, type = "lambda", color = NULL, add = FALSE, ...)
```

Arguments

`x`	an object of class `clusterTree`. (see `clusterTree`)
`type`	string: if "lambda", then the lambda Tree is plotted. if "kappa", then the kappa Tree is plotted.
`color`	the color of the branches of the Cluster Tree
`add`	logical: if `TRUE` the Tree is added to an existing plot.
`...`	additional graphical parameters.

Author(s)

Fabrizio Lecci

References

Brian P. Kent, Alessandro Rinaldo, and Timothy Verstynen, (2013), "DeBaCl: A Python Package for Interactive DEnsity-BAsed CLustering."arXiv:1307.8136

Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman, (2014), "Metric Embeddings for Cluster Trees"

See Also

clusterTree, print.clusterTree

Examples

```
## Generate data: 3 clusters
n=1200    #sample size
Neach=floor(n/4)
X1=cbind(rnorm(Neach,1,.8),rnorm(Neach,5,0.8))
X2=cbind(rnorm(Neach,3.5,.8),rnorm(Neach,5,0.8))
X3=cbind(rnorm(Neach,6,1),rnorm(Neach,1,1))
XX=rbind(X1,X2,X3)

k=100     #parameter of knn

## Density clustering using knn and kde
Tree=clusterTree(XX,k, density="knn")
TreeKDE=clusterTree(XX,k,h=0.3, density="kde")

par(mfrow=c(2,3))
plot(XX, pch=19, cex=0.6)
# plot lambda trees
plot(Tree, type="lambda", main="lambda Tree (knn)")
plot(TreeKDE, type="lambda", main="lambda Tree (kde)")
# plot clusters
plot(XX, pch=19, cex=0.6, main="cluster labels")
for (i in Tree$id){
points(matrix(XX[Tree$DataPoints[[i]],],ncol=2), col=i, pch=19, cex=0.6)
}
#plot kappa trees
plot(Tree, type="kappa", main="kappa Tree (knn)")
plot(TreeKDE, type="kappa", main="kappa Tree (kde)")
```

`plot.diagram` *Plot the Persistence Diagram*

Description

This function plots the Persistence Diagram stored in an object of class `diagram`. Optionally, it can also represent the diagram as a persistence barcode.

Usage

```
## S3 method for class 'diagram'
plot(x, diagLim=NULL, dimension=NULL, col=NULL, rotated=FALSE, barcode=FALSE,
            band=NULL, add = FALSE, ...)
```

Arguments

x	an object of class diagram (as returned by the functions gridDiag and ripsDiag) or an n by 3 matrix, where n is the number of features to be plotted.
diagLim	numeric vector of length 2, specifying the limits of the plot. If NULL then it is automatically computed using the lifetimes of the features.
dimension	number specifying the dimension of the features to be plotted. If NULL all the features are plotted.
col	an optional vector of length P that stores the colors of the topological features to be plotted, where P is the number of topological features stored in x.
rotated	logical: if FALSE the plotted diagram has axes (birth, death), if TRUE the plotted diagram has axes ((birth+death)/2,(death-birth)/2). Default is FALSE.
barcode	logical: if TRUE the persistence barcode is plotted, in place of the diagram.
band	numeric: if band!=NULL, a pink band of size band is added around the diagonal. If also barcode is TRUE, then bars shorter than band are dotted. Default is NULL.
add	logical: if TRUE, the points of x are added to an existing plot.
...	additional graphical parameters.

Author(s)

Fabrizio Lecci

References

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology", (arXiv:1303.7117). To appear, Annals of Statistics.

Frederic Chazal, Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman, (2014), "Stochastic Convergence of Persistence Landscapes and Silhouettes", Proceedings of the 30th Symposium of Computational Geometry (SoCG). (arXiv:1312.0308)

See Also

gridDiag, ripsDiag

Examples

```
XX1 = circleUnif(30)
XX2 = circleUnif(30, r=2) +3
XX=rbind(XX1,XX2)

DiagLim=5
```

```
maxdimension=1

## rips diagram
Diag=ripsDiag(XX,maxdimension,DiagLim, printStatus=TRUE)

#plot
par(mfrow=c(1,3))
plot(Diag)
plot(Diag, rotated=TRUE)
plot(Diag, barcode=TRUE)
```

`plot.maxPersistence` *Summary plot for the maxPersistence function*

Description

This function plots an object of class `maxPersistence`, for the selection of the optimal smoothing parameter for persistent homology. For each value of the smoothing parameter, the plot shows the number of detected features, their persistence, and a bootstrap confidence band.

Usage

```
## S3 method for class 'maxPersistence'
plot(x, ...)
```

Arguments

`x`	an object of class `maxPersistence`, as returned by the functions `maxPersistence`
`...`	additional graphical parameters.

Author(s)

Fabrizio Lecci

References

Frederic Chazal, Jessi Cisewski, Brittany T. Fasy, Fabrizio Lecci, Bertrand Michel, Alessandro Rinaldo, and Larry Wasserman, (2014), "Robust Topological Inference: distance-to-a-measure and kernel distance"

Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology", (arXiv:1303.7117). To appear, Annals of Statistics.

See Also

`maxPersistence`

Examples

```
## input data: circle with clutter noise
n=600
percNoise=0.1
XX1 = circleUnif(n)
noise=cbind(runif(percNoise*n, -2,2),runif(percNoise*n, -2,2))
X=rbind(XX1,noise)

## limits of the Gird at which the density estimator is evaluated
Xlim=c(-2,2)
Ylim=c(-2,2)
by=0.2

B=80
alpha=0.05

## candidates
parametersKDE=seq(0.1,0.5, by=0.2)

maxKDE=maxPersistence(kde, parametersKDE, X, Xlim, Ylim, Zlim=NA, by=by, B=B,
                  alpha=alpha, parallel=FALSE, printProgress = TRUE)

par(mfrow=c(1,2))
plot(X, pch=16, cex=0.5, main="Circle")
plot(maxKDE)
```

---

ripsDiag *Rips Persistence Diagram*

Description

This function computes the persistence diagram of the Rips filtration built on top of a point cloud.

Usage

```
ripsDiag(X, maxdimension, maxscale, dist = "euclidean", printStatus = FALSE)
```

Arguments

X	if dist="euclidean", X is an n by d matrix of coordinates, where n is the number of points in the d-dimensional euclidean space. if dist="arbitrary", X is an n by n matrix of distances of n points.
maxdimension	integer: max dimension of the homological features to be computed. (e.g. 0 for connected components, 1 for connected components and loops, 2 for connected components, loops, voids, etc.)
maxscale	number: maximum value of the rips filtration.
dist	"euclidean" for Euclidean distance, "arbitrary" for an arbitrary distance given in input as a distance matrix.
printStatus	logical: if TRUE, a progress bar is printed. Default is FALSE.

Details

This function is an R wrapper of the function "rips-pairwise" of the C++ library Dionysus. See refereneces.

Value

`ripsDiag` returns an object of class `diagram`, a P by 3 matrix, where P is the number of points in the resulting persistence diagram. The first column contains the dimension of each feature (0 for components, 1 for loops, 2 for voids, etc.). Second and third columns are Birth and Death of the features.

Author(s)

Brittany T. Fasy and Fabrizio Lecci

References

http://www.mrzv.org/software/dionysus/

Herbert Edelsbrunner and John Harer (2010), Computational topology: an introduction. American Mathematical Society.

Brittany Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, and Aarti Singh. (2013), "Statistical Inference For Persistent Homology", (arXiv:1303.7117). To appear, Annals of Statistics.

See Also

`summary.diagram`, `plot.diagram`, `gridDiag`

Examples

```
## EXAMPLE 1: rips diagram for circles (euclidean distance)
XX = circleUnif(30)
DiagLim=5
maxdimension=1
## note that the input XX is a point cloud
Diag=ripsDiag(XX,maxdimension,DiagLim, printStatus=TRUE)


## EXAMPLE 2: rips diagram with arbitrary distance
## distance matrix for triangle with edges of length: 1,2,4
distX=matrix(c(0,1,2,1,0,4,2,4,0), ncol=3)

#rips diagram using the distance matrix as input
DiagLim=5
maxdimension=1
## note that the input distXX is a distance matrix
DiagTri=ripsDiag(distX,maxdimension,DiagLim, dist="arbitrary", printStatus=TRUE)
#points with lifetime=0 are not shown. e.g. the loop of the triangle.
print(DiagTri)
```

`silhouette` *The Persistence Silhouette Function*

Description

This function computes the silhouette function corresponding to a given persistence diagram.

Usage

```
silhouette(Diag, p = 1, dimension = 1,
           tseq = seq(min(Diag[, 2:3]), max(Diag[, 2:3]), length = 500))
```

Arguments

`Diag`	an object of class `diagram` or a P by 3 matrix, storing a persistence diagram with columns: "dimension", "Birth", "Death".
`p`	number: the power of the weights of the silhouette function. See the definition of silhouette function, Section 5 in the reference.
`dimension`	the dimension of the topological features under consideration. Default is 1 (loops).
`tseq`	a vector of values at which the silhouette function is evaluated.

Value

Returns a numeric vector of the same length of `tseq`, with the values of the silhouette function evaluated at the points of `tseq`.

Author(s)

Fabrizio Lecci

References

Frederic Chazal, Brittany T. Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman, (2014), "Stochastic Convergence of Persistence Landscapes and Silhouettes", Proceedings of the 30th Symposium of Computational Geometry (SoCG). (arXiv:1312.0308)

See Also

`landscape`

Examples

```
Diag=matrix(c(0,0,10,1,0,3,1,3,8), ncol=3, byrow=TRUE)
DiagLim=10
colnames(Diag)=c("dim", "Birth", "Death")

#persistence silhouette
tseq=seq(0,DiagLim, length=1000)
Sil=silhouette(Diag, p=1,  dimension=1, tseq)

par(mfrow=c(1,2))
plot.diagram(Diag)
plot(tseq, Sil, type="l", xlab="t", ylab="silhouette", asp=1)
```

---

summary.diagram print *and* summary *for* diagram

Description

print.diagram prints a persistence diagram, a P by 3 matrix, where P is the number of points in the diagram. The first column contains the dimension of each feature (0 for components, 1 for loops, 2 for voids, etc.). Second and third columns are Birth and Death of the features.

summary.diagram produces basic summaries of a persistence diagrams.

Usage

```
## S3 method for class 'diagram'
print(x, ...)
## S3 method for class 'diagram'
summary(object, ...)
```

Arguments

x	an object of class diagram
object	an object of class diagram
...	additional arguments affecting the summary produced.

Author(s)

Fabrizio Lecci

See Also

plot.diagram, gridDiag, ripsDiag,

Examples

```
# Generate data from 2 circles
XX1 = circleUnif(30)
XX2 = circleUnif(30, r=2) +3
XX=rbind(XX1,XX2)

DiagLim=5            # limit of the filtration
maxdimension=1       # computes betti0 and betti1

Diag=ripsDiag(XX,maxdimension,DiagLim, printStatus=TRUE)

print(Diag)
print(summary(Diag))
```

---

torusUnif *Uniform Sample From The 3D Torus*

Description

This function samples n points from the 3D torus, uniformly with respect to its surface.

Usage

```
torusUnif(n, a, c)
```

Arguments

n	an integer specifying the number of points in the sample.
a	the radius of the torus tube.
c	the radius from the center of the hole to the center of the torus tube.

Details

This function is an implementation of Algorithm 1 in the reference.

Value

torusUnif returns an n by 3 matrix of coordinates.

Author(s)

Fabrizio Lecci

References

Persi Diaconis, Susan Holmes, and Mehrdad Shahshahani, (2013), "Sampling from a manifold." Advances in Modern Statistical Theory and Applications: A Festschrift in honor of Morris L. Eaton. Institute of Mathematical Statistics, 102-125.

See Also

circleUnif

Examples

```
X=torusUnif(300, a=1.8, c=5)
plot(X)
```

---

wasserstein *Wasserstein distance between two persistence diagrams*

Description

This function computes the Wasserstein distance between two persistence diagrams

Usage

```
wasserstein(Diag1, Diag2, p=1, dimension=1)
```

Arguments

Diag1	an object of class diagram or a matrix (n by 3) that stores dimension, birth and death of n topological features.
Diag2	an object of class diagram or a matrix (m by 3) that stores dimension, birth and death of m topological features.
p	integer specifying the power to be used in the computation of the Wasserstein distance. Default is 1.
dimension	an integer specifying the dimension of the features used to compute the wasserstein distance. 0 for connected components, 1 for loops, 2 for voids and so on. Default is 1.

Details

The Wasserstein distance between two diagrams is the cost of the optimal matching between points of the two diagrams. This function is an R wrapper of the function "wasserstein_distance" in the C++ library Dionysus. See references.

Value

Returns the value of the Wasserstein distance between the two persistence diagrams.

Author(s)

Fabrizio Lecci

References

http://www.mrzv.org/software/dionysus/

Herbert Edelsbrunner and John Harer (2010), Computational topology: an introduction. American Mathematical Society.

See Also

ripsDiag, gridDiag, plot.diagram

Examples

```
XX1 = circleUnif(20)
XX2 = circleUnif(20, r=0.2)

DiagLim=5
maxdimension=1

Diag1=ripsDiag(XX1,maxdimension,DiagLim, printStatus=FALSE)
Diag2=ripsDiag(XX2,maxdimension,DiagLim, printStatus=FALSE)

wassersteinDist=wasserstein(Diag1, Diag2, p=1, dimension=1)
print(wassersteinDist)
```

Index