Package: TDA (via r-universe)

October 28, 2026

Title Statistical Tools for Topological Data Analysis

Description Tools for Topological Data Analysis. The package focuses on statistical analysis of persistent homology and density clustering. For that, this package provides an R interface for the efficient algorithms of the C++ libraries 'GUDHI' <https://project.inria.fr/gudhi/software/>, 'Dionysus' <https://www.mrzv.org/software/dionysus/>, and 'PHAT' <https://bitbucket.org/phat-code/phat/>. This package also implements methods from Fasy et al. (2014) <doi:10.1214/14-AOS1252> and Chazal et al. (2015) <doi:10.20382/jocg.v6i2a8> for analyzing the statistical significance of persistent homology features.

Version 1.9.4

Depends R (>= 3.1.0)

Imports FNN, Rcpp (>= 0.11.0), igraph, parallel, scales

LinkingTo BH (>= 1.87.0-1), Rcpp, RcppEigen

Suggests testthat, lintr

Date 2025-02-01

Author Brittany T. Fasy [aut], Jisu Kim [aut, cre], Fabrizio Lecci [aut], Clement Maria [aut], David L. Millman [aut], Vincent Rouvreau. [aut]

Maintainer Jisu Kim <jkim82133@snu.ac.kr>

License GPL-3

Copyright See inst/COPYRIGHTS

SystemRequirements gmp, GNU make

Encoding UTF-8

Type Package

NeedsCompilation yes

Config/pak/sysreqs libgmp3-dev make

Repository https://cran.r-universe.dev

Date/Publication 2025-02-02 08:10:05 UTC

RemoteUrl https://github.com/cran/TDA

RemoteRef HEAD

RemoteSha 0e300018d16e8b805d31adf618b64a424166a698

Contents

TDA-package . . . 3
alphaComplexDiag . . . 4
alphaComplexFiltration . . . 6
alphaShapeDiag . . . 8
alphaShapeFiltration . . . 10
bootstrapBand . . . 12
bootstrapDiagram . . . 14
bottleneck . . . 16
circleUnif . . . 18
clusterTree . . . 19
distFct . . . 21
dtm . . . 23
filtrationDiag . . . 25
funFiltration . . . 27
gridDiag . . . 28
gridFiltration . . . 31
hausdInterval . . . 34
kde . . . 35
kernelDist . . . 36
knnDE . . . 38
landscape . . . 39
maxPersistence . . . 40
multipBootstrap . . . 43
plot.clusterTree . . . 45
plot.diagram . . . 46
plot.maxPersistence . . . 48
ripsDiag . . . 49
ripsFiltration . . . 52
silhouette . . . 54
sphereUnif . . . 56
summary.diagram . . . 57
torusUnif . . . 58
wasserstein . . . 59

Index **61**

TDA-package *Statistical Tools for Topological Data Analysis*

Description

Tools for Topological Data Analysis. The package focuses on statistical analysis of persistent homology and density clustering. For that, this package provides an R interface for the efficient algorithms of the C++ libraries GUDHI, Dionysus and PHAT. This package also implements methods from Fasy et al. (2014) and Chazal et al. (2015) for analyzing the statistical significance of persistent homology features.

Details

Package:	TDA
Version:	1.9.4
Date:	2025-02-01
License:	GPL-3

Author(s)

Brittany Terese Fasy, Jisu Kim, Fabrizio Lecci, Clement Maria, David L. Millman, and Vincent Rouvreau

Maintainer: Jisu Kim <jkim82133@snu.ac.kr>

References

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

Fasy BT, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2014). "Confidence Sets for Persistence Diagrams." Annals of Statistics. (arXiv:1303.7117)

Chazal F, Fasy BT, Lecci F, Rinaldo A, Wasserman L (2015). "Stochastic Convergence of Persistence Landscapes and Silhouettes." Journal of Computational Geometry. (arXiv:1312.0308)

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2015a). "Subsampling Methods for Persistent Homology." Proceedings of the 32nd International Conference on Machine Learning (ICML). (arXiv:1406.1901)

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2017). "Robust Topological Inference: Distance-To-a-Measure and Kernel Distance." Journal of Machine Learning Research. (arXiv:1412.7197)

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/.

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology." https://www.mrzv.org/software/dionysus/.

Bauer U, Kerber M, Reininghaus J (2012). "PHAT, a software library for persistent homology". https://bitbucket.org/phat-code/phat/.

`alphaComplexDiag` *Alpha Complex Persistence Diagram*

Description

The function `alphaComplexDiag` computes the persistence diagram of the alpha complex filtration built on top of a point cloud.

Usage

```
alphaComplexDiag(
    X, maxdimension = NCOL(X) - 1, library = "GUDHI",
 location = FALSE, printProgress = 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 of the space.
`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.)
`library`	either a string or a vector of length two. When a vector is given, the first element specifies which library to compute the Alpha Complex filtration, and the second element specifies which library to compute the persistence diagram. If a string is used, then the same library is used. For computing the Alpha Complex filtration, the user can use the library `"GUDHI"`, and is also the default value. For computing the persistence diagram, the user can choose either the library `"GUDHI"`, `"Dionysus"`, or `"PHAT"`. The default value is `"GUDHI"`.
`location`	if `TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram, location of birth point and death point of each homological feature is returned. Additionaly if `library="Dionysus"`, location of representative cycles of each homological feature is also returned. The default value is `FALSE`.
`printProgress`	if `TRUE`, a progress bar is printed. The default value is `FALSE`.

Details

The function `alphaComplexDiag` constructs the Alpha Complex filtration, using the C++ library GUDHI. Then for computing the persistence diagram from the Alpha Complex filtration, the user can use either the C++ library GUDHI, Dionysus, or PHAT. See refereneces.

Value

The function `alphaComplexDiag` returns a list with the following elements:

`diagram` 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 stores 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.

`birthLocation` only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the grid point completing the simplex that gives birth to an homological feature.

`deathLocation` only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the grid point completing the simplex that kills an homological feature.

`cycleLocation` only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: a list of length P, where P is the number of points in the resulting persistence diagram. Each element is a P_i by h_i+1 by d array for h_i dimensional homological feature. It represents location of $h_i + 1$ vertices of P_i simplices, where P_i simplices constitutes the h_i dimensional homological feature.

Author(s)

Jisu Kim and Vincent Rouvreau

References

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

Rouvreau V (2015). "Alpha complex." In GUDHI User and Reference Manual. GUDHI Editorial Board. https://gudhi.inria.fr/doc/latest/group__alpha__complex.html

Edelsbrunner H, Kirkpatrick G, Seidel R (1983). "On the shape of a set of points in the plane." IEEE Trans. Inform. Theory.

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/

See Also

`summary.diagram`, `plot.diagram`, `alphaShapeDiag`, `gridDiag`, `ripsDiag`

Examples

```
# input data generated from a circle
X <- circleUnif(n = 30)

# persistence diagram of alpha complex
DiagAlphaCmplx <- alphaComplexDiag(
    X = X, library = c("GUDHI", "Dionysus"), location = TRUE,
```

```
    printProgress = TRUE)

# plot
par(mfrow = c(1, 2))
plot(DiagAlphaCmplx[["diagram"]])
one <- which(DiagAlphaCmplx[["diagram"]][, 1] == 1)
one <- one[which.max(
    DiagAlphaCmplx[["diagram"]][one, 3] - DiagAlphaCmplx[["diagram"]][one, 2])]
plot(X, col = 2, main = "Representative loop of data points")
for (i in seq(along = one)) {
  for (j in seq_len(dim(DiagAlphaCmplx[["cycleLocation"]][[one[i]]])[1])) {
    lines(
        DiagAlphaCmplx[["cycleLocation"]][[one[i]]][j, , ], pch = 19, cex = 1,
        col = i)
  }
}
par(mfrow = c(1, 1))
```

`alphaComplexFiltration` *Alpha Complex Filtration*

Description

The function `alphaComplexFiltration` computes the alpha complex filtration built on top of a point cloud.

Usage

```
alphaComplexFiltration(
    X, library = "GUDHI", printProgress = 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 of the space.
`library`	a string specifying which library to compute the Alpha Complex filtration. The user can use the library `"GUDHI"`, and is also the default value.
`printProgress`	if `TRUE`, a progress bar is printed. The default value is `FALSE`.

Details

The function `alphaComplexFiltration` constructs the alpha complex filtration, using the C++ library GUDHI. See refereneces.

Value

The function `alphaComplexFiltration` returns a list with the following elements:

`cmplx`	a list representing the complex. Its i-th element represents the vertices of i-th simplex.
`values`	a vector representing the filtration values. Its i-th element represents the filtration value of i-th simplex.
`increasing`	a logical variable indicating if the filtration values are in increasing order (`TRUE`) or in decreasing order (`FALSE`).
`coordinates`	a matrix representing the coordinates of vertices. Its i-th row represents the coordinate of i-th vertex.

Author(s)

Jisu Kim and Vincent Rouvreau

References

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

Rouvreau V (2015). "Alpha complex." In GUDHI User and Reference Manual. GUDHI Editorial Board. https://gudhi.inria.fr/doc/latest/group__alpha__complex.html

Edelsbrunner H, Kirkpatrick G, Seidel R (1983). "On the shape of a set of points in the plane." IEEE Trans. Inform. Theory.

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/

See Also

alphaComplexDiag, filtrationDiag

Examples

```
# input data generated from a circle
X <- circleUnif(n = 10)

# alpha complex filtration
FltAlphaComplex <- alphaComplexFiltration(X = X, printProgress = TRUE)

# plot alpha complex filtration
lim <- rep(c(-1, 1), 2)
plot(NULL, type = "n", xlim = lim[1:2], ylim = lim[3:4],
    main = "Alpha Complex Filtration Plot")
for (idx in seq(along = FltAlphaComplex[["cmplx"]])) {
 polygon(FltAlphaComplex[["coordinates"]][FltAlphaComplex[["cmplx"]][[idx]], , drop = FALSE],
      col = "pink", border = NA, xlim = lim[1:2], ylim = lim[3:4])
}
for (idx in seq(along = FltAlphaComplex[["cmplx"]])) {
```

```
 polygon(FltAlphaComplex[["coordinates"]][FltAlphaComplex[["cmplx"]][[idx]], , drop = FALSE],
      col = NULL, xlim = lim[1:2], ylim = lim[3:4])
}
points(FltAlphaComplex[["coordinates"]], pch = 16)
```

---

alphaShapeDiag *Persistence Diagram of Alpha Shape in 3d*

Description

The function alphaShapeDiag computes the persistence diagram of the alpha shape filtration built on top of a point cloud in 3 dimension.

Usage

```
alphaShapeDiag(
    X, maxdimension = NCOL(X) - 1, library = "GUDHI", location = FALSE,
    printProgress = 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 of the space. Currently d should be 3.
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.)
library	either a string or a vector of length two. When a vector is given, the first element specifies which library to compute the Alpha Shape filtration, and the second element specifies which library to compute the persistence diagram. If a string is used, then the same library is used. For computing the Alpha Shape filtration, the user can use the library "GUDHI", and is also the default value. For computing the persistence diagram, the user can choose either the library "GUDHI", "Dionysus", or "PHAT". The default value is "GUDHI".
location	if TRUE and if "Dionysus" or "PHAT" is used for computing the persistence diagram, location of birth point and death point of each homological feature is returned. Additionaly if library="Dionysus", location of representative cycles of each homological feature is also returned. The default value is FALSE.
printProgress	if TRUE, a progress bar is printed. The default value is FALSE.

Details

The function alphaShapeDiag constructs the Alpha Shape filtration, using the C++ library GUDHI. Then for computing the persistence diagram from the Alpha Shape filtration, the user can use either the C++ library GUDHI, Dionysus, or PHAT. See refereneces.

Value

The function `alphaShapeDiag` returns a list with the following elements:

`diagram`	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 stores 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.
`birthLocation`	only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the grid point completing the simplex that gives birth to an homological feature.
`deathLocation`	only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the grid point completing the simplex that kills an homological feature.
`cycleLocation`	only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: a list of length P, where P is the number of points in the resulting persistence diagram. Each element is a P_i by h_i+1 by d array for h_i dimensional homological feature. It represents location of $h_i + 1$ vertices of P_i simplices, where P_i simplices constitutes the h_i dimensional homological feature.

Author(s)

Jisu Kim and Vincent Rouvreau

References

Fischer K (2005). "Introduction to Alpha Shapes."

Edelsbrunner H, Mucke EP (1994). "Three-dimensional Alpha Shapes." ACM Trans. Graph.

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/

Morozov D (2008). "Homological Illusions of Persistence and Stability."

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

See Also

`summary.diagram`, `plot.diagram`, `alphaComplexDiag`, `gridDiag`, `ripsDiag`

Examples

```
# input data generated from cylinder
n <- 30
X <- cbind(circleUnif(n = n), runif(n = n, min = -0.1, max = 0.1))

# persistence diagram of alpha shape
DiagAlphaShape <- alphaShapeDiag(
```

```
    X = X, maxdimension = 1, library = c("GUDHI", "Dionysus"), location = TRUE,
    printProgress = TRUE)

# plot diagram and first two dimension of data
par(mfrow = c(1, 2))
plot(DiagAlphaShape[["diagram"]])
plot(X[, 1:2], col = 2, main = "Representative loop of alpha shape filtration")
one <- which(DiagAlphaShape[["diagram"]][, 1] == 1)
one <- one[which.max(
    DiagAlphaShape[["diagram"]][one, 3] - DiagAlphaShape[["diagram"]][one, 2])]
for (i in seq(along = one)) {
  for (j in seq_len(dim(DiagAlphaShape[["cycleLocation"]][[one[i]]])[1])) {
    lines(
        DiagAlphaShape[["cycleLocation"]][[one[i]]][j, , 1:2], pch = 19,
        cex = 1, col = i)
  }
}
par(mfrow = c(1, 1))
```

`alphaShapeFiltration` *Alpha Shape Filtration in 3d*

Description

The function `alphaShapeFiltration` computes the alpha shape filtration built on top of a point cloud in 3 dimension.

Usage

```
alphaShapeFiltration(
    X, library = "GUDHI", printProgress = 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 of the space. Currently d should be 3.
`library`	a string specifying which library to compute the Alpha Shape filtration. The user can use the library `"GUDHI"`, and is also the default value.
`printProgress`	if `TRUE`, a progress bar is printed. The default value is `FALSE`.

Details

The function `alphaShapeFiltration` constructs the alpha shape filtration, using the C++ library GUDHI. See refereneces.

Value

The function `alphaShapeFiltration` returns a list with the following elements:

`cmplx`	a list representing the complex. Its i-th element represents the vertices of i-th simplex.
`values`	a vector representing the filtration values. Its i-th element represents the filtration value of i-th simplex.
`increasing`	a logical variable indicating if the filtration values are in increasing order (`TRUE`) or in decreasing order (`FALSE`).
`coordinates`	a matrix representing the coordinates of vertices. Its i-th row represents the coordinate of i-th vertex.

Author(s)

Jisu Kim and Vincent Rouvreau

References

Fischer K (2005). "Introduction to Alpha Shapes."

Edelsbrunner H, Mucke EP (1994). "Three-dimensional Alpha Shapes." ACM Trans. Graph.

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/

Morozov D (2008). "Homological Illusions of Persistence and Stability."

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

See Also

`alphaShapeDiag`, `filtrationDiag`

Examples

```
# input data generated from sphere
X <- sphereUnif(n = 20, d = 2)

# alpha shape filtration
FltAlphaShape <- alphaShapeFiltration(X = X, printProgress = TRUE)
```

bootstrapBand *Bootstrap Confidence Band*

Description

The function 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,
    printProgress = FALSE, weight = NULL, ...)
```

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, at which FUN is evaluated.
B	the number of bootstrap iterations.
alpha	bootstrapBand returns a (1-alpha) confidence band. The default value is 0.05.
parallel	logical: if TRUE the bootstrap iterations are parallelized, using the library parallel. The default value is FALSE.
printProgress	if TRUE, a progress bar is printed. The default value is FALSE.
weight	either NULL, a number, or a vector of length n. If it is NULL, weight is not used. If it is a number, then same weight is applied to each points of X. If it is a vector, weight represents weights of each points of X. The default value is NULL.
...	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

The function bootstrapBand 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)

Jisu Kim and Fabrizio Lecci

References

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

Fasy BT, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology: Confidence Sets for Persistence Diagrams." (arXiv:1303.7117). 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)
```

`bootstrapDiagram` *Bootstrapped Confidence Set for a Persistence Diagram, using the Bottleneck Distance (or the Wasserstein distance).*

Description

The function `bootstrapDiagram` 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. The function returns the `(1-alpha)` quantile of `B` bottleneck distances (or Wasserstein distances), computed in `B` iterations of the bootstrap algorithm.

Usage

```
bootstrapDiagram(
    X, FUN, lim, by, maxdimension = length(lim) / 2 - 1,
    sublevel = TRUE, library = "GUDHI", B = 30, alpha = 0.05,
    distance = "bottleneck", dimension = min(1, maxdimension),
 p = 1, parallel = FALSE, printProgress = FALSE, weight = 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 of the space.
`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 `bootstrapDiagram`, but is automatically computed by the function using `lim` and `by`.
`lim`	a 2 by d matrix, where each column specifies the range of each dimension of the grid, over which the function `FUN` is evaluated.
`by`	either a number or a vector of length d specifying space between points of the grid in each dimension. If a number is given, then same space is used in each dimension.
`maxdimension`	a number that indicates the maximum dimension to compute persistent homology to. The default value is $d - 1$, which is (dimension of embedding space - 1).
`sublevel`	a logical variable indicating if the Persistence Diagram should be computed for sublevel sets (`TRUE`) or superlevel sets (`FALSE`) of the function. The default value is `TRUE`.
`library`	a string specifying which library to compute the persistence diagram. The user can choose either the library `"GUDHI"`, `"Dionysus"`, or `"PHAT"`. The default value is `"GUDHI"`.

B	the number of bootstrap iterations. The default value is 30.
alpha	The function bootstrapDiagram returns a (1 - alpha) quantile. The default value is 0.05.
distance	a string specifying the distance to be used for persistence diagrams: either "bottleneck" or "wasserstein". The default value is "bottleneck".
dimension	dimension is an integer or a vector 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. The default value is 1 if $maxdimension \geq 1$, and else 0.
p	if distance == "wasserstein", then p is an integer specifying the power to be used in the computation of the Wasserstein distance. The default value is 1.
parallel	logical: if TRUE the bootstrap iterations are parallelized, using the library parallel. The default value is FALSE.
printProgress	if TRUE a progress bar is printed. The default value is FALSE.
weight	either NULL, a number, or a vector of length n. If it is NULL, weight is not used. If it is a number, then same weight is applied to each points of X. If it is a vector, weight represents weights of each points of X. The default value is NULL.
...	additional parameters for the function FUN.

Details

The function bootstrapDiagram 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. See (Chazal, Fasy, Lecci, Michel, Rinaldo, and Wasserman, 2014) for discussion of the method.

Value

The function bootstrapDiagram returns the (1-alpha) quantile of the values computed by the bootstrap algorithm.

Note

The function bootstrapDiagram uses the C++ library Dionysus for the computation of bottleneck and Wasserstein distances. See references.

Author(s)

Jisu Kim and 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.

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

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology." https://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)
lim <- cbind(Xlim, Ylim)
by <- 0.05

h <- .3  #bandwidth for the function kde

#Kernel Density Diagram of the superlevel sets
Diag <- gridDiag(XX, kde, lim = lim, by = by, sublevel = FALSE,
                 printProgress = TRUE, h = h)

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

cc <- bootstrapDiagram(XX, kde, lim = lim, by = by, sublevel = FALSE, B = B,
          alpha = alpha, dimension = 1, printProgress = TRUE, h = h)

plot(Diag[["diagram"]], band = 2 * cc)
```

---

bottleneck *Bottleneck distance between two persistence diagrams*

Description

The function bottleneck computes the bottleneck distance between two persistence diagrams.

Usage

```
bottleneck(Diag1, Diag2, 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.
dimension	an integer or a vector 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. The default value is 1 (loops). The default value is 1.

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. When a vector is given for dimension, then maximum among bottleneck distances using each element in dimension is returned. The function bottleneck is an R wrapper of the function "bottleneck_distance" in the C++ library Dionysus. See references.

Value

The function bottleneck returns the value of the bottleneck distance between the two persistence diagrams.

Author(s)

Jisu Kim and Fabrizio Lecci

References

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology." https://www.mrzv.org/software/dionysus/

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

See Also

wasserstein, alphaComplexDiag, alphaComplexDiag, gridDiag, ripsDiag, plot.diagram

Examples

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

DiagLim <- 5
maxdimension <- 1

Diag1 <- ripsDiag(XX1, maxdimension, DiagLim, printProgress = FALSE)
Diag2 <- ripsDiag(XX2, maxdimension, DiagLim, printProgress = FALSE)

bottleneckDist <- bottleneck(Diag1[["diagram"]], Diag2[["diagram"]],
                             dimension = 1)
print(bottleneckDist)
```

`circleUnif` *Uniform Sample From The Circle*

Description

The function `circleUnif` 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. The default value is 1.

Value

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

Note

Uniform sample from sphere of arbitrary dimension can be generated using `sphereUnif`.

Author(s)

Fabrizio Lecci

See Also

`sphereUnif`, `torusUnif`

Examples

```
X <- circleUnif(100)
plot(X)
```

clusterTree *Density clustering: the cluster tree*

Description

Given a point cloud, or a matrix of distances, the function clusterTree computes a density estimator and returns the corresponding cluster tree of superlevel sets (lambda tree and kappa tree; see references).

Usage

```
clusterTree(
    X, k, h = NULL, density = "knn", dist = "euclidean", d = NULL,
    Nlambda = 100, printProgress = 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.
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. The default value 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. The default value is "knn".
dist	string: can be "euclidean", when X is a point cloud or "arbitrary", when X is a matrix of distances. The default value is "euclidean".
d	integer: if dist="arbitrary", then d is the dimension of the underlying space. The default value is "NULL".
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. The default value is 100.
printProgress	logical: if TRUE, a progress bar is printed. The default value is FALSE.

Details

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

Value

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

`density`	Vector of length `n`: the values of the density estimator evaluated at each of the points stored in `X`
`DataPoints`	A list whose elements are the points of `X` corresponding to each branch, in the same order of `id`
`n`	The number of points stored in the input matrix `X`
`id`	Vector: the IDs associated to the branches of the cluster tree
`children`	A list whose elements are the IDs of the children 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`
`silo`	A list whose elements are the horizontal coordinates of the silo of each branch, in the same order of `id`
`Xbase`	Vector: the horiontal coordinates of the branches of the cluster tree, in the same order of `id`
`lambdaBottom`	Vector: the vertical bottom coordinates of the branches of the lambda tree, in the same order of `id`
`lambdaTop`	Vector: the vertical top coordinates of the branches of the lambda tree, in the same order of `id`
`rBottom`	(only if `density="knn"`) Vector: the vertical bottom coordinates of the branches of the r tree, in the same order of `id`
`rTop`	(only if `density="knn"`) Vector: the vertical top coordinates of the branches of the r tree, in the same order of `id`
`alphaBottom`	Vector: the vertical bottom coordinates of the branches of the alpha tree, in the same order of `id`
`alphaTop`	Vector: the vertical top coordinates of the branches of the alpha 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`

Author(s)

Fabrizio Lecci

References

Kent BP, Rinaldo A, Verstynen T (2013). "DeBaCl: A Python Package for Interactive DEnsity-BAsed CLustering." arXiv:1307.8136

Lecci F, Rinaldo A, Wasserman L (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

The function distFct 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. X is the set of points whose distance is being measured from a base grid.
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. Grid is the base set from which each point is compared to the closest point in X.

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

The function distFct returns a numeric vector of length n, where n is the number of points stored in Grid. Each value in V corresponds to the distance between a point in G and the nearest point in X.

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
interval <- 0.065
Xseq <- seq(-1.6, 1.6, by = interval)
Yseq <- seq(-1.7, 1.7, by = interval)
Grid <- expand.grid(Xseq, Yseq)

## distance fct
distance <- distFct(X, Grid)
```

dtm *Distance to Measure Function*

Description

The function dtm 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

$$d_{m0}(y) = \left(\frac{1}{m0} \int_0^{m0} (G_y^{-1}(u))^r du \right)^{1/r},$$

where $G_y(t) = P(\|X - y\| \leq t)$, and $m0 \in (0, 1)$ and $r \in [1, \infty)$ are tuning parameters. As m0 increases, DTM function becomes smoother, so m0 can be understood as a smoothing parameter. r affects less but also changes DTM function as well. The DTM can be seen as a smoothed version of the distance function. See Details and References.

Given $X = \{x_1, \ldots, x_n\}$, the empirical version of the distance to measure is

$$\hat{d}_{m0}(y) = \left(\frac{1}{k} \sum_{x_i \in N_k(y)} \|x_i - y\|^r \right)^{1/r},$$

where $k = \lceil m0 * n \rceil$ and $N_k(y)$ is the set containing the k nearest neighbors of y among $x_1, \ldots, x_n$.

Usage

```
dtm(X, Grid, m0, r = 2, weight = 1)
```

Arguments

X	an n by d matrix of coordinates of points used to construct the uniform empirical measure for the distance to measure, where n is the number of points and d is the dimension.
Grid	an m by d matrix of coordinates of points where the distance to measure is computed, where m is the number of points in Grid and d is the dimension.
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. The value of m0 should be in $(0, 1)$.
r	a numeric variable for the tuning parameter of the distance to measure. The value of r should be in $[1, \infty)$, and the default value is 2.
weight	either a number, or a vector of length n. If it is a number, then same weight is applied to each points of X. If it is a vector, weight represents weights of each points of X. The default value is 1.

Details

See (Chazal, Cohen-Steiner, and Merigot, 2011, Definition 3.2) and (Chazal, Massart, and Michel, 2015, Equation (2)) for a formal definition of the "distance to measure" function.

Value

The function `dtm` returns a vector of length m (the number of points stored in `Grid`) containing the value of the distance to measure function evaluated at each point of `Grid`.

Author(s)

Jisu Kim and Fabrizio Lecci

References

Chazal F, Cohen-Steiner D, Merigot Q (2011). "Geometric inference for probability measures." Foundations of Computational Mathematics 11.6, 733-751.

Chazal F, Massart P, Michel B (2015). "Rates of convergence for robust geometric inference."

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)
```

`filtrationDiag` *Persistence Diagram of Filtration*

Description

The function `filtrationDiag` computes the persistence diagram of the filtration.

Usage

```
filtrationDiag(
    filtration, maxdimension, library = "GUDHI", location = FALSE,
    printProgress = FALSE, diagLimit = NULL)
```

Arguments

`filtration`	a list representing the input filtration. This list consists of three components: `"cmplx"`, a list representing the complex, `"values"`, a vector representing the filtration values, and `"increasing"`, a logical variable indicating if the filtration values are in increasing order or in decreasing order.
`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.)
`library`	a string specifying which library to compute the persistence diagram. The user can choose either the library `"GUDHI"` or `"Dionysus"`. The default value is `"GUDHI"`.
`location`	if `TRUE` and if `"Dionysus"` is used for computing the persistence diagram, location of birth point, death point, and representative cycles, of each homological feature is returned.
`printProgress`	logical: if `TRUE`, a progress bar is printed. The default value is `FALSE`.
`diagLimit`	a number that replaces `Inf` in the persistence diagram. The default value is `NULL` and `Inf` value in the persistence diagram will not be replaced.

Details

The user can decide to use either the C++ library GUDHI or Dionysus. See refereneces.

Value

The function `filtrationDiag` returns a list with the following elements:

`diagram`	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.

`birthLocation` only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: a vector of length P. Each row represents the index of the vertex completing the simplex that gives birth to an homological feature.

`deathLocation` only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: a vector of length P. Each row represents the index of the vertex completing the simplex that kills an homological feature.

`cycleLocation` only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: a P_i by $h_i + 1$ matrix for h_i dimensional homological feature. It represents index of $h_i + 1$ vertices of P_i simplices on a representative cycle of the h_i dimensional homological feature.

Author(s)

Jisu Kim

References

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/.

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology". https://www.mrzv.org/software/dionysus/

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

Fasy B, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology." (arXiv:1303.7117). Annals of Statistics.

See Also

summary.diagram, plot.diagram

Examples

```
n <- 5
X <- cbind(cos(2*pi*seq_len(n)/n), sin(2*pi*seq_len(n)/n))
maxdimension <- 1
maxscale <- 1.5
dist <- "euclidean"
library <- "Dionysus"

FltRips <- ripsFiltration(X = X, maxdimension = maxdimension,
               maxscale = maxscale, dist = "euclidean", library = "Dionysus",
               printProgress = TRUE)

DiagFltRips <- filtrationDiag(filtration = FltRips, maxdimension = maxdimension,
                   library = "Dionysus", location = TRUE, printProgress = TRUE)

plot(DiagFltRips[["diagram"]])
```

```
FUNvalues <- X[, 1] + X[, 2]

FltFun <- funFiltration(FUNvalues = FUNvalues, cmplx = FltRips[["cmplx"]])

DiagFltFun <- filtrationDiag(filtration = FltFun, maxdimension = maxdimension,
                             library = "Dionysus", location = TRUE, printProgress = TRUE)

plot(DiagFltFun[["diagram"]], diagLim = c(-2, 5))
```

`funFiltration` *Filtration from function values*

Description

The function `funFiltration` computes the filtration from the complex and the function values.

Usage

```
funFiltration(FUNvalues, cmplx, sublevel = TRUE)
```

Arguments

`FUNvalues`	The function values on the vertices of the complex.
`cmplx`	the complex.
`sublevel`	a logical variable indicating if the Persistence Diagram should be computed for sublevel sets (`TRUE`) or superlevel sets (`FALSE`) of the function. The default value is `TRUE`.

Details

See references.

Value

The function `funFiltration` returns a list with the following elements:

`cmplx`	a list representing the complex. Its i-th element represents the vertices of i-th simplex.
`values`	a vector representing the filtration values. Its i-th element represents the filtration value of i-th simplex.
`increasing`	a logical variable indicating if the filtration values are in increasing order (`TRUE`) or in decreasing order (`FALSE`).

Author(s)

Jisu Kim

References

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

See Also

filtrationDiag

Examples

```
n <- 5
X <- cbind(cos(2*pi*seq_len(n)/n), sin(2*pi*seq_len(n)/n))
maxdimension <- 1
maxscale <- 1.5
dist <- "euclidean"
library <- "Dionysus"

FltRips <- ripsFiltration(X = X, maxdimension = maxdimension,
               maxscale = maxscale, dist = "euclidean", library = "Dionysus",
               printProgress = TRUE)

FUNvalues <- X[, 1] + X[, 2]

FltFun <- funFiltration(FUNvalues = FUNvalues, cmplx = FltRips[["cmplx"]])
```

---

gridDiag *Persistence Diagram of a function over a Grid*

Description

The function gridDiag computes the Persistence Diagram of a filtration of sublevel sets (or superlevel sets) of a function evaluated over a grid of points in arbitrary dimension d.

Usage

```
gridDiag(
    X = NULL, FUN = NULL, lim = NULL, by = NULL, FUNvalues = NULL,
    maxdimension = max(NCOL(X), length(dim(FUNvalues))) - 1,
    sublevel = TRUE, library = "GUDHI", location = FALSE,
    printProgress = 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 of the space. NULL if this option is not used. The default value is NULL.

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 lim, and by. NULL if this option is not used. The default value is NULL.
lim	a 2 by d matrix, where each column specifying the range of each dimension of the grid, over which the function FUN is evaluated. NULL if this option is not used. The default value is NULL.
by	either a number or a vector of length d specifying space between points of the grid in each dimension. If a number is given, then same space is used in each dimension. NULL if this option is not used. The default value is NULL.
FUNvalues	an $m1 * m2 * ... * md$ array of function values over $m1 * m2 * ... * md$ grid, where mi is the number of scales of grid on ith dimension. NULL if this option is not used. The default value is NULL.
maxdimension	a number that indicates the maximum dimension of the homological features to compute: 0 for connected components, 1 for loops, 2 for voids and so on. The default value is $d - 1$, which is (dimension of embedding space - 1).
sublevel	a logical variable indicating if the Persistence Diagram should be computed for sublevel sets (TRUE) or superlevel sets (FALSE) of the function. The default value is TRUE.
library	a string specifying which library to compute the persistence diagram. The user can choose either the library "GUDHI", "Dionysus", or "PHAT". The default value is "GUDHI".
location	if TRUE and if "Dionysus" or "PHAT" is used for computing the persistence diagram, location of birth point and death point of each homological feature is returned. Additionaly if library="Dionysus", location of representative cycles of each homological feature is also returned. The default value is FALSE.
printProgress	if TRUE a progress bar is printed. The default value 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. The default value is NULL and the max/min of the function is used.
...	additional parameters for the function FUN.

Details

If the values of X, FUN are set, then FUNvalues should be NULL. In this case, gridDiag evaluates the function FUN over a grid. If the value of FUNvalues is set, then X, FUN should be NULL. In this case, FUNvalues is used as function values over the grid. If location=TRUE, then lim, and by should be set.

Once function values are either computed or given, gridDiag constructs a filtration by triangulating the grid and considering the simplices determined by the values of the function of dimension up to maxdimension+1.

Value

The function `gridDiag` returns a list with the following components:

`diagram`	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 stores 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, in case of a filtration constructed using sublevel sets (from -Inf to Inf), or Death and Birth of features, in case of a filtration constructed using superlevel sets (from Inf to -Inf).
`birthLocation`	only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the grid point completing the simplex that gives birth to an homological feature.
`deathLocation`	only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the grid point completing the simplex that kills an homological feature.
`cycleLocation`	only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: a list of length P, where P is the number of points in the resulting persistence diagram. Each element is a P_i by h_i+1 by d array for h_i dimensional homological feature. It represents location of $h_i + 1$ vertices of P_i simplices, where P_i simplices constitutes the h_i dimensional homological feature.

Note

The user can decide to use either the C++ library GUDHI, Dionysus, or PHAT. See references.

Since dimension of simplicial complex from grid points in R^d is up to d, homology of dimension $\geq d$ is trivial. Hence setting `maxdimension` with values $\geq d$ is equivalent to `maxdimension=d-1`.

Author(s)

Brittany T. Fasy, Jisu Kim, and Fabrizio Lecci

References

Fasy B, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology." (arXiv:1303.7117). Annals of Statistics.

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology." https://www.mrzv.org/software/dionysus/

Bauer U, Kerber M, Reininghaus J (2012). "PHAT, a software library for persistent homology." https://bitbucket.org/phat-code/phat/

See Also

`summary.diagram`, `plot.diagram`, `distFct`, `kde`, `kernelDist`, `dtm`, `alphaComplexDiag`, `alphaComplexDiag`, `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)
lim <- cbind(Xlim, Ylim)
by <- 0.05

h <- .3  #bandwidth for the function kde

#Distance Function Diagram of the sublevel sets
Diag1 <- gridDiag(XX, distFct, lim = lim, by = by, sublevel = TRUE,
                  printProgress = TRUE)

#Kernel Density Diagram of the superlevel sets
Diag2 <- gridDiag(XX, kde, lim = lim, by = by, sublevel = FALSE,
    library = "Dionysus", location = TRUE, printProgress = TRUE, h = h)
#plot
par(mfrow = c(2, 2))
plot(XX, cex = 0.5, pch = 19)
title(main = "Data")
plot(Diag1[["diagram"]])
title(main = "Distance Function Diagram")
plot(Diag2[["diagram"]])
title(main = "Density Persistence Diagram")
one <- which(Diag2[["diagram"]][, 1] == 1)
plot(XX, col = 2, main = "Representative loop of grid points")
for (i in seq(along = one)) {
  points(Diag2[["birthLocation"]][one[i], , drop = FALSE], pch = 15, cex = 3,
      col = i)
  points(Diag2[["deathLocation"]][one[i], , drop = FALSE], pch = 17, cex = 3,
      col = i)
  for (j in seq_len(dim(Diag2[["cycleLocation"]][[one[i]]])[1])) {
    lines(Diag2[["cycleLocation"]][[one[i]]][j, , ], pch = 19, cex = 1, col = i)
  }
}
```

---

gridFiltration *Persistence Diagram of a function over a Grid*

Description

The function gridFiltration computes the Persistence Diagram of a filtration of sublevel sets (or superlevel sets) of a function evaluated over a grid of points in arbitrary dimension d.

Usage

```
gridFiltration(
    X = NULL, FUN = NULL, lim = NULL, by = NULL, FUNvalues = NULL,
    maxdimension = max(NCOL(X), length(dim(FUNvalues))) - 1,
    sublevel = TRUE, printProgress = 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 of the space. NULL if this option is not used. The default value is NULL.
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 gridFiltration, but is automatically computed by the function using lim, and by. NULL if this option is not used. The default value is NULL.
lim	a 2 by d matrix, where each column specifying the range of each dimension of the grid, over which the function FUN is evaluated. NULL if this option is not used. The default value is NULL.
by	either a number or a vector of length d specifying space between points of the grid in each dimension. If a number is given, then same space is used in each dimension. NULL if this option is not used. The default value is NULL.
FUNvalues	an $m1 * m2 * ... * md$ array of function values over $m1 * m2 * ... * md$ grid, where mi is the number of scales of grid on ith dimension. NULL if this option is not used. The default value is NULL.
maxdimension	a number that indicates the maximum dimension of the homological features to compute: 0 for connected components, 1 for loops, 2 for voids and so on. The default value is $d - 1$, which is (dimension of embedding space - 1).
sublevel	a logical variable indicating if the Persistence Diagram should be computed for sublevel sets (TRUE) or superlevel sets (FALSE) of the function. The default value is TRUE.
printProgress	if TRUE a progress bar is printed. The default value is FALSE.
...	additional parameters for the function FUN.

Details

If the values of X, FUN are set, then FUNvalues should be NULL. In this case, gridFiltration evaluates the function FUN over a grid. If the value of FUNvalues is set, then X, FUN should be NULL. In this case, FUNvalues is used as function values over the grid.

Once function values are either computed or given, gridFiltration constructs a filtration by triangulating the grid and considering the simplices determined by the values of the function of dimension up to maxdimension+1.

Value

The function `gridFiltration` returns a list with the following elements:

`cmplx`	a list representing the complex. Its i-th element represents the vertices of i-th simplex.
`values`	a vector representing the filtration values. Its i-th element represents the filtration value of i-th simplex.
`increasing`	a logical variable indicating if the filtration values are in increasing order (`TRUE`) or in decreasing order (`FALSE`).
`coordinates`	only if both `lim` and `by` are not `NULL`: a matrix representing the coordinates of vertices. Its i-th row represents the coordinate of i-th vertex.

Note

The user can decide to use either the C++ library GUDHI, Dionysus, or PHAT. See references.

Since dimension of simplicial complex from grid points in R^d is up to d, homology of dimension $\geq d$ is trivial. Hence setting `maxdimension` with values $\geq d$ is equivalent to `maxdimension=d-1`.

Author(s)

Brittany T. Fasy, Jisu Kim, and Fabrizio Lecci

References

Fasy B, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology." (arXiv:1303.7117). Annals of Statistics.

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology." https://www.mrzv.org/software/dionysus/

Bauer U, Kerber M, Reininghaus J (2012). "PHAT, a software library for persistent homology." https://bitbucket.org/phat-code/phat/

See Also

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

Examples

```
# input data
n <- 10
XX <- circleUnif(n)

## Ranges of the grid
Xlim <- c(-1, 1)
Ylim <- c(-1, 1)
lim <- cbind(Xlim, Ylim)
by <- 1
```

```
#Distance Function Diagram of the sublevel sets
FltGrid <- gridFiltration(
  XX, distFct, lim = lim, by = by, sublevel = TRUE, printProgress = TRUE)
```

`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 and References.

Usage

```
hausdInterval(
    X, m, B = 30, alpha = 0.05, parallel = FALSE,
    printProgress = 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. The default value is `30`.
`alpha`	`hausdInterval` returns a (`1-alpha`) confidence interval. The default value is `0.05`.
`parallel`	logical: if `TRUE`, the iterations are parallelized, using the library `parallel`. The default value is `FALSE`.
`printProgress`	if `TRUE`, a progress bar is printed. The default value 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 (Fasy, Lecci, Rinaldo, Wasserman, Balakrishnan, and Singh, 2013, Theorem 3).

Value

The function `hausdInterval` returns a number c. The confidence interval is $[0, c]$.

Author(s)

Fabrizio Lecci

References

Fasy BT, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology: Confidence Sets for Persistence Diagrams." (arXiv:1303.7117). Annals of Statistics.

See Also

bootstrapBand

Examples

```
X <- circleUnif(1000)
interval <- hausdInterval(X, m = 800)
print(interval)
```

---

kde *Kernel Density Estimator over a Grid of Points*

Description

Given a point cloud `X` (n points), the function `kde` 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, kertype = "Gaussian", weight = 1,
    printProgress = FALSE)
```

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.
`kertype`	string: if `kertype = "Gaussian"`, Gaussian kernel is used, and if `kertype = "Epanechnikov"`, Epanechnikov kernel is used. Defaults to `"Gaussian"`.
`weight`	either a number, or a vector of length n. If it is a number, then same weight is applied to each points of `X`. If it is a vector, `weight` represents weights of each points of `X`. The default value is `1`.
`printProgress`	if `TRUE`, a progress bar is printed. The default value is `FALSE`.

Value

The function 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)

Jisu Kim and 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, the function kernelDist 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, weight = 1, printProgress = FALSE)
```

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.
weight	either a number, or a vector of length n. If it is a number, then same weight is applied to each points of X. If it is a vector, weight represents weights of each points of X. The default value is 1.
printProgress	if TRUE, a progress bar is printed. The default value is FALSE.

Value

The function 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)

Jisu Kim and Fabrizio Lecci

References

Phillips JM, Wang B, Zheng Y (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), The function knnDE 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 from 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

The function 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

The function `landscape` 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 colnames: "dimension", "Birth", "Death".
`dimension`	the dimension of the topological features under consideration. The default value is 1 (loops).
`KK`	a vector: the order of the landscape function. The default value is `1`. (First Landscape function).
`tseq`	a vector of values at which the landscape function is evaluated.

Value

The function `landscape` returns a numeric matrix with the number of row as the length of `tseq` and the number of column as the length of `KK`. The value at ith row and jth column represents the value of the `KK[j]`-th landscape function evaluated at `tseq[i]`.

Author(s)

Fabrizio Lecci

References

Bubenik P (2012). "Statistical topology using persistence landscapes." arXiv:1207.6437.

Chazal F, Fasy BT, Lecci F, Rinaldo A, Wasserman L (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("dimension", "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, the function `maxPersistence` 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, lim, by,
    maxdimension = length(lim) / 2 - 1, sublevel = TRUE,
    library = "GUDHI", B = 30, alpha = 0.05,
    bandFUN = "bootstrapBand", distance = "bottleneck",
    dimension = min(1, maxdimension), p = 1, parallel = FALSE,
    printProgress = FALSE, weight = NULL)
```

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.
lim	a 2 by d matrix, where each column specifying the range of each dimension of the grid, over which the function FUN is evaluated.
by	either a number or a vector of length d specifying space between points of the grid in each dimension. If a number is given, then same space is used in each dimension.
maxdimension	a number that indicates the maximum dimension to compute persistent homology to. The default value is $d - 1$, which is (dimension of embedding space - 1).
sublevel	a logical variable indicating if the persistent homology should be computed for sublevel sets of FUN (TRUE) or superlevel sets (FALSE). The default value is TRUE.
library	a string specifying which library to compute the persistence diagram. The user can choose either the library "GUDHI", "Dionysus", or "PHAT". The default value is "GUDHI".
bandFUN	the function to be used in the computation of the confidence band. Either "bootstrapDiagram" or "bootstrapBand".
B	the number of bootstrap iterations.
alpha	for each value store in parameters, maxPersistence computes a (1-alpha) confidence band.
distance	optional (if bandFUN == bootstrapDiagram): a string specifying the distance to be used for persistence diagrams: either "bottleneck" or "wasserstein"
dimension	optional (if bandFUN == bootstrapDiagram): an integer or a vector specifying the dimension of the features used to compute the bottleneck distance. 0 for connected components, 1 for loops, 2 for voids. The default value is 1.
p	optional (if bandFUN == bootstrapDiagram AND distance == "wasserstein"): integer specifying the power to be used in the computation of the Wasserstein distance. The default value is 1.
parallel	logical: if TRUE, the bootstrap iterations are parallelized, using the library parallel.
printProgress	if TRUE, a progress bar is printed. The default value is FALSE.
weight	either NULL, a number, or a vector of length n. If it is NULL, weight is not used. If it is a number, then same weight is applied to each points of X. If it is a vector, weight represents weights of each points of X.

Details

The function maxPersistence calls the gridDiag function, which computes the persistence diagram of sublevel (or superlevel) sets of a function, evaluated over a grid of points.

Value

The function maxPersistence 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)

Jisu Kim and Fabrizio Lecci

References

Chazal F, Cisewski J, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: distance-to-a-measure and kernel distance."

Fasy BT, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology", (arXiv:1303.7117). 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)
lim <- cbind(Xlim, Ylim)
by <- 0.2

B <- 80
alpha <- 0.05

## candidates
```

```
parametersKDE <- seq(0.1, 0.5, by = 0.2)

maxKDE <- maxPersistence(kde, parametersKDE, X, lim = lim, by = by,
                         bandFUN = "bootstrapBand", 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

The function `multipBootstrap` 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,
    printProgress = 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`.
`printProgress`	logical: if `TRUE` a progress bar is printed. The default value is `FALSE`.

Details

See Algorithm 1 in the reference.

Value

The function `multipBootstrap` 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 BT, Lecci F, Rinaldo A, 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
#here we'll store the landscapes
Lands <- matrix(0, nrow = NN, ncol = length(tseq))

for (i in seq_len(NN)){
  subXX <- XX[sample(seq_len(nn), mm), ]
  Diags[[i]] <- ripsDiag(subXX, maxdimension, DiagLim)
  Lands[i, ] <- landscape(Diags[[i]][["diagram"]], 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

The function `plot.clusterTree` 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 "r", then the r Tree is plotted. if "alpha", then the alpha Tree is plotted. if "kappa", then the kappa Tree is plotted.
`color`	number: the color of the branches of the Cluster Tree. The default value is `NULL` and a different color is assigned to each branch.
`add`	logical: if `TRUE`, the Tree is added to an existing plot.
`...`	additional graphical parameters.

Author(s)

Fabrizio Lecci

References

Kent BP, Rinaldo A, Verstynen T (2013). "DeBaCl: A Python Package for Interactive DEnsity-BAsed CLustering." arXiv:1307.8136

Lecci F, Rinaldo A, Wasserman L (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

The function `plot.diagram` 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, lab.line = 2.2,
    colorBand = "pink", colorBorder = NA, add = FALSE, ...)
```

Arguments

x	an object of class diagram (as returned by the functions alphaComplexDiag, alphaComplexDiag, gridDiag, or 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). The default value 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. The default value is NULL.
lab.line	number of lines from the plot edge, where the labels will be placed. The default value is 2.2.
colorBand	the color for filling the confidence band. The default value is "pink". (NA leaves the band unfilled)
colorBorder	the color to draw the border of the confidence band. The default value is NA and omits the border.
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

alphaComplexDiag, alphaComplexDiag, 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, printProgress = TRUE)

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

---

plot.maxPersistence *Summary plot for the maxPersistence function*

Description

The function `plot.maxPersistence` 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, features = "dimension", colorBand = "pink",
    colorBorder = NA, ...)
```

Arguments

`x`	an object of class `maxPersistence`, as returned by the functions `maxPersistence`
`features`	string: if "all" then all the features are plotted; if "dimension" then only the features of the dimension used to compute the confidence band are plotted.
`colorBand`	the color for filling the confidence band. The default is "pink". (NA leaves the band unfilled)
`colorBorder`	the color to draw the border of the confidence band. The default is NA and omits the border.
`...`	additional graphical parameters.

Author(s)

Fabrizio Lecci

References

Chazal F, Cisewski J, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014). "Robust Topological Inference: distance-to-a-measure and kernel distance."

Fasy BT, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology." (arXiv:1303.7117). 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)
lim <- cbind(Xlim, Ylim)
by <- 0.2

B <- 80
alpha <- 0.05

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

maxKDE <- maxPersistence(kde, parametersKDE, X, lim = lim, by = by,
                         bandFUN = "bootstrapBand", 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)
```

---

ripsDiag *Rips Persistence Diagram*

Description

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

Usage

```
ripsDiag(
    X, maxdimension, maxscale, dist = "euclidean",
    library = "GUDHI", location = FALSE, printProgress = 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.) Currently there is a bug for computing homological features of dimension higher than 1 when the distance is arbitrary (dist = "arbitrary") and library 'GUDHI' is used (library = "GUDHI").
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. Currently there is a bug for the arbitrary distance (dist = "arbitrary") when computing homological features of dimension higher than 1 and library 'GUDHI' is used (library = "GUDHI").
library	either a string or a vector of length two. When a vector is given, the first element specifies which library to compute the Rips filtration, and the second element specifies which library to compute the persistence diagram. If a string is used, then the same library is used. For computing the Rips filtration, if dist = "euclidean", the user can use either the library "GUDHI" or "Dionysus". If dist = "arbitrary", the user can use either the library "Dionysus". The default value is "GUDHI" if dist = "euclidean", and "Dionysus" if dist == "arbitrary". When "GUDHI" is used for dist = "arbitrary", "Dionysus" is implicitly used. For computing the persistence diagram, the user can choose either the library "GUDHI", "Dionysus", or "PHAT". The default value is "GUDHI". Currently there is a bug for 'GUDHI' (library = "GUDHI") when computing homological features of dimension higher than 1 and the distance is arbitrary (dist = "arbitrary").
location	if TRUE and if "Dionysus" or "PHAT" is used for computing the persistence diagram, location of birth point and death point of each homological feature is returned. Additionaly if library="Dionysus", location of representative cycles of each homological feature is also returned.
printProgress	logical: if TRUE, a progress bar is printed. The default value is FALSE.

Details

For Rips filtration based on Euclidean distance of the input point cloud, the user can decide to use either the C++ library GUDHI or Dionysus. For Rips filtration based on arbitrary distance, the user can decide to the C++ library Dionysus. Then for computing the persistence diagram from the Rips filtration, the user can use either the C++ library GUDHI, Dionysus, or PHAT. Currently there is a bug for computing homological features of dimension higher than 1 when the distance is arbitrary (dist = "arbitrary") and library 'GUDHI' is used (library = "GUDHI"). See refereneces.

Value

The function `ripsDiag` returns a list with the following elements:

`diagram` 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.

`birthLocation` only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: if `dist="euclidean"`, then `birthLocation` is a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the data point completing the simplex that gives birth to an homological feature. If `dist="arbitrary"`, then `birthLocation` is a vector of length P. Each row represents the index of the data point completing the simplex that gives birth to an homological feature.

`deathLocation` only if `location=TRUE` and if `"Dionysus"` or `"PHAT"` is used for computing the persistence diagram: if `dist="euclidean"`, then `deathLocation` is a P by d matrix, where P is the number of points in the resulting persistence diagram. Each row represents the location of the data point completing the simplex that kills an homological feature. If `dist="arbitrary"`, then `deathLocation` is a vector of length P. Each row represents the index of the data point completing the simplex that kills an homological feature.

`cycleLocation` only if `location=TRUE` and if `"Dionysus"` is used for computing the persistence diagram: if `dist="euclidean"`, then `cycleLocation` is a list of length P, where P is the number of points in the resulting persistence diagram. Each element is a P_i by $h_i + 1$ by d array for h_i dimensional homological feature. It represents location of $h_i + 1$ vertices of P_i simplices, where P_i simplices constitutes the h_i dimensional homological feature. If `dist = "arbitrary"`, then each element is a P_i by $h_i + 1$ matrix for for h_i dimensional homological feature. It represents index of $h_i + 1$ vertices of P_i simplices on a representative cycle of the h_i dimensional homological feature.

Author(s)

Brittany T. Fasy, Jisu Kim, Fabrizio Lecci, and Clement Maria

References

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/.

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology". https://www.mrzv.org/software/dionysus/

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

Fasy B, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2013). "Statistical Inference For Persistent Homology." (arXiv:1303.7117). Annals of Statistics.

See Also

summary.diagram, plot.diagram, gridDiag

Examples

```
## EXAMPLE 1: rips diagram for circles (euclidean distance)
X <- circleUnif(30)
maxscale <- 5
maxdimension <- 1
## note that the input X is a point cloud
DiagRips <- ripsDiag(
    X = X, maxdimension = maxdimension, maxscale = maxscale,
    library = "Dionysus", location = TRUE, printProgress = TRUE)

# plot
layout(matrix(c(1, 3, 2, 2), 2, 2))
plot(X, cex = 0.5, pch = 19)
title(main = "Data")
plot(DiagRips[["diagram"]])
title(main = "rips Diagram")
one <- which(
    DiagRips[["diagram"]][, 1] == 1 &
    DiagRips[["diagram"]][, 3] - DiagRips[["diagram"]][, 2] > 0.5)
plot(X, col = 2, main = "Representative loop of data points")
for (i in seq(along = one)) {
  for (j in seq_len(dim(DiagRips[["cycleLocation"]][[one[i]]])[1])) {
    lines(
     DiagRips[["cycleLocation"]][[one[i]]][j, , ], pch = 19, cex = 1,
        col = i)
  }
}


## 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)
maxscale <- 5
maxdimension <- 1
## note that the input distXX is a distance matrix
DiagTri <- ripsDiag(distX, maxdimension, maxscale, dist = "arbitrary",
                    printProgress = TRUE)
#points with lifetime = 0 are not shown. e.g. the loop of the triangle.
print(DiagTri[["diagram"]])
```

---

ripsFiltration *Rips Filtration*

Description

The function ripsFiltration computes the Rips filtration built on top of a point cloud.

Usage

```
ripsFiltration(
    X, maxdimension, maxscale, dist = "euclidean",
    library = "GUDHI", printProgress = 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.
library	a string specifying which library to compute the Rips filtration. If dist = "euclidean", the user can use either the library "GUDHI" or "Dionysus". If dist = "arbitrary", the user can use the library "Dionysus". The default value is "GUDHI" if dist = "euclidean", and "Dionysus" if dist == "arbitrary". When "GUDHI" is used for dist = "arbitrary", "Dionysus" is implicitly used.
printProgress	logical: if TRUE, a progress bar is printed. The default value is FALSE.

Details

For Rips filtration based on Euclidean distance of the input point cloud, the user can decide to use either the C++ library GUDHI or Dionysus. For Rips filtration based on arbitrary distance, the user can use the C++ library Dionysus. See refereneces.

Value

The function ripsFiltration returns a list with the following elements:

cmplx	a list representing the complex. Its i-th element represents the vertices of i-th simplex.
values	a vector representing the filtration values. Its i-th element represents the filtration value of i-th simplex.
increasing	a logical variable indicating if the filtration values are in increasing order (TRUE) or in decreasing order (FALSE).
coordinates	only if dist = "euclidean": a matrix representing the coordinates of vertices. Its i-th row represents the coordinate of i-th vertex.

Author(s)

Jisu Kim

References

Maria C (2014). "GUDHI, Simplicial Complexes and Persistent Homology Packages." https://project.inria.fr/gudhi/software/.

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology". https://www.mrzv.org/software/dionysus/

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

See Also

ripsDiag, filtrationDiag

Examples

```
n <- 5
X <- cbind(cos(2*pi*seq_len(n)/n), sin(2*pi*seq_len(n)/n))
maxdimension <- 1
maxscale <- 1.5

FltRips <- ripsFiltration(X = X, maxdimension = maxdimension,
               maxscale = maxscale, dist = "euclidean", library = "GUDHI",
               printProgress = TRUE)

# plot rips filtration
lim <- rep(c(-1, 1), 2)
plot(NULL, type = "n", xlim = lim[1:2], ylim = lim[3:4],
    main = "Rips Filtration Plot")
for (idx in seq(along = FltRips[["cmplx"]])) {
  polygon(FltRips[["coordinates"]][FltRips[["cmplx"]][[idx]], , drop = FALSE],
      col = "pink", border = NA, xlim = lim[1:2], ylim = lim[3:4])
}
for (idx in seq(along = FltRips[["cmplx"]])) {
  polygon(FltRips[["coordinates"]][FltRips[["cmplx"]][[idx]], , drop = FALSE],
      col = NULL, xlim = lim[1:2], ylim = lim[3:4])
}
points(FltRips[["coordinates"]], pch = 16)
```

---

silhouette *The Persistence Silhouette Function*

Description

The function silhouette 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 colnames: "dimension", "Birth", "Death".
p	a vector: 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. The default value is 1 (loops).
tseq	a vector of values at which the silhouette function is evaluated.

Value

The function silhouette returns a numeric matrix of with the number of row as the length of tseq and the number of column as the length of p. The value at ith row and jth column represents the value of the p[j]-th power silhouette function evaluated at tseq[i].

Author(s)

Fabrizio Lecci

References

Chazal F, Fasy BT, Lecci F, Rinaldo A, 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

Examples

```
Diag <- matrix(c(0, 0, 10, 1, 0, 3, 1, 3, 8), ncol = 3, byrow = TRUE)
DiagLim <- 10
colnames(Diag) <- c("dimension", "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)
```

sphereUnif *Uniform Sample From The Sphere S^d*

Description

The function sphereUnif samples n points from the sphere S^d of radius r embedded in R^{d+1}, uniformly with respect to the volume measure of the sphere.

Usage

```
sphereUnif(n, d, r = 1)
```

Arguments

n	an integer specifying the number of points in the sample.
d	an integer specifying the dimension of the sphere S^d
r	a numeric variable specifying the radius of the sphere. The default value is 1.

Value

The function sphereUnif returns an n by 2 matrix of coordinates.

Note

When d = 1, this function is same as using circleUnif.

Author(s)

Jisu Kim

See Also

circleUnif, torusUnif

Examples

```
X <- sphereUnif(n = 100, d = 1, r = 1)
plot(X)
```

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

Description

The function 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.

The function 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, alphaComplexDiag, alphaComplexDiag, 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, printProgress = TRUE)

print(Diag[["diagram"]])
print(summary(Diag[["diagram"]]))
```

torusUnif *Uniform Sample From The 3D Torus*

Description

The function `torusUnif` 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 `torusUnif` is an implementation of Algorithm 1 in the reference.

Value

The function `torusUnif` returns an `n` by 3 matrix of coordinates.

Author(s)

Fabrizio Lecci

References

Diaconis P, Holmes S, and Shahshahani M (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`,`sphereUnif`

Examples

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

`wasserstein` *Wasserstein distance between two persistence diagrams*

Description

The function `wasserstein` 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. The default value is 1.
`dimension`	an integer or a vector 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. The default value is 1 (loops).

Details

The Wasserstein distance between two diagrams is the cost of the optimal matching between points of the two diagrams. When a vector is given for `dimension`, then maximum among bottleneck distances using each element in `dimension` is returned. This function is an R wrapper of the function "wasserstein_distance" in the C++ library Dionysus. See references.

Value

The function `wasserstein` returns the value of the Wasserstein distance between the two persistence diagrams.

Author(s)

Jisu Kim and Fabrizio Lecci

References

Morozov D (2007). "Dionysus, a C++ library for computing persistent homology". https://www.mrzv.org/software/dionysus/.

Edelsbrunner H, Harer J (2010). "Computational topology: an introduction." American Mathematical Society.

See Also

bottleneck, alphaComplexDiag, alphaComplexDiag, gridDiag, ripsDiag, plot.diagram

Examples

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

DiagLim <- 5
maxdimension <- 1

Diag1 <- ripsDiag(XX1, maxdimension, DiagLim, printProgress = FALSE)
Diag2 <- ripsDiag(XX2, maxdimension, DiagLim, printProgress = FALSE)

wassersteinDist <- wasserstein(Diag1[["diagram"]], Diag2[["diagram"]], p = 1,
                               dimension = 1)
print(wassersteinDist)
```

Index

∗ **datagen**
circleUnif, 18
sphereUnif, 56
torusUnif, 58
∗ **hplot**
plot.clusterTree, 45
plot.diagram, 46
plot.maxPersistence, 48
∗ **htest**
bootstrapBand, 12
bootstrapDiagram, 14
hausdInterval, 34
multipBootstrap, 43
∗ **methods**
alphaComplexDiag, 4
alphaComplexFiltration, 6
alphaShapeDiag, 8
alphaShapeFiltration, 10
bottleneck, 16
filtrationDiag, 25
funFiltration, 27
gridDiag, 28
gridFiltration, 31
landscape, 39
maxPersistence, 40
ripsDiag, 49
ripsFiltration, 52
silhouette, 54
wasserstein, 59
∗ **nonparametric**
bootstrapBand, 12
bootstrapDiagram, 14
clusterTree, 19
distFct, 21
dtm, 23
hausdInterval, 34
kde, 35
kernelDist, 36
knnDE, 38
multipBootstrap, 43
∗ **optimize**
bottleneck, 16
wasserstein, 59
∗ **package**
TDA-package, 3

alphaComplexDiag, 4, *7*, *9*, *17*, *30*, *33*, *47*, *57*, *60*
alphaComplexFiltration, 6
alphaShapeDiag, *5*, 8, *11*
alphaShapeFiltration, 10

bootstrapBand, 12, *16*, *35*, *42*
bootstrapDiagram, 14
bottleneck, 16, *16*, *60*

circleUnif, 18, *56*, *58*
clusterTree, 19, *45*

distFct, *12*, *14*, *16*, 21, *24*, *29, 30*, *32, 33*, *36–38*
dtm, *12–14*, *16*, *22*, 23, *29, 30*, *32, 33*, *36–38*, *40*, *42*

filtrationDiag, *7*, *11*, 25, *28*, *54*
funFiltration, 27

gridDiag, *5*, *9*, *15*, *17*, 28, *41, 42*, *47*, *52*, *57*, *60*
gridFiltration, 31

hausdInterval, 34

kde, *12–14*, *16*, *22*, *24*, *29, 30*, *32, 33*, 35, *37, 38*, *40*, *42*
kernelDist, *16*, *22*, *24*, *30*, *33*, 36, *36*, *38*, *40*, *42*
knnDE, 38

landscape, 39, *44*, *55*

maxPersistence, 40, *48, 49*
multipBootstrap, 43

plot.clusterTree, *21*, 45
plot.diagram, *5*, *9*, *16, 17*, *26*, *30*, *33*, 46, *52*, *57*, *60*
plot.maxPersistence, 48
print.clusterTree, *45*
print.clusterTree *(*clusterTree*)*, 19
print.diagram *(*summary.diagram*)*, 57
print.maxPersistence *(*maxPersistence*)*, 40
print.summary.diagram *(*summary.diagram*)*, 57
print.summary.maxPersistence *(*maxPersistence*)*, 40

ripsDiag, *5*, *9*, *17*, *30*, *33*, *47*, 49, *54*, *57*, *60*
ripsFiltration, 52

silhouette, *40*, *44*, 54
sphereUnif, *18*, 56, *58*
summary.diagram, *5*, *9*, *16*, *26*, *30*, *33*, *52*, 57
summary.maxPersistence *(*maxPersistence*)*, 40

TDA *(*TDA-package*)*, 3
TDA-package, 3
torusUnif, *18*, *56*, 58

wasserstein, *17*, 59