# Package 'TDApplied'

January 15, 2023

**Type** Package

**Title** Machine Learning and Inference for Topological Data Analysis

**Version** 2.0.3

**Author** Shael Brown [aut, cre],
Dr. Reza Farivar [aut, fnd]

**Maintainer** Shael Brown <shaelebrown@gmail.com>

**Description** Topological data analysis is a powerful tool for finding non-linear global structure in whole datasets. 'TDApplied' aims to bridge topological data analysis with data, statistical and machine learning practitioners so that more analyses may benefit from the power of topological data analysis. The main tool of topological data analysis is persistent homology, which computes a shape descriptor of a dataset, called a persistence diagram. There are five goals of this package: (1) deliver a fast implementation of persistent homology via a python interface, (2) convert persistence diagrams computed using the two main R packages for topological data analysis into a data frame, (3) implement fast versions of both distance and kernel calculations for pairs of persistence diagrams, (4) contribute tools for the interpretation of persistence diagrams, and (5) provide parallelized methods for machine learning and inference for persistence diagrams.

**Depends** R (>= 3.2.2)

**Imports** parallel, doParallel, foreach, clue, rdist, parallelly,
kernlab, iterators, methods, stats, utils

**License** GPL-3

**URL** https://github.com/shaelebrown/TDApplied

**BugReports** https://github.com/shaelebrown/TDApplied/issues

**Encoding** UTF-8

**NeedsCompilation** yes

**RoxygenNote** 7.1.2

**Suggests** rmarkdown, knitr, testthat (>= 3.0.0), TDA, TDAstats,
reticulate

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Repository** CRAN

**Date/Publication** 2023-01-15 19:00:06 UTC

# R topics documented:



---

`bootstrap_persistence_thresholds` *Estimate persistence threshold(s) for topological features in a data set using bootstrapping.*

---

## Description

Bootstrapping is used to find a conservative estimate of a "confidence interval" around each point in the persistence diagram of the data set, and points whose (open) intervals do not overlap with the diagonal (birth = death) would be considered "significant" or "real". One threshold is computed for each dimension in the diagram.

## Usage

```
bootstrap_persistence_thresholds(
  X,
  FUN = "calculate_homology",
  maxdim = 0,
  thresh,
  distance_mat = FALSE,
  ripser = NULL,
  ignore_infinite_cluster = TRUE,
  calculate_representatives = FALSE,
  num_samples = 30,
  alpha = 0.05,
  return_subsetted = FALSE,
  return_diag = TRUE,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| X | the input dataset, must either be a matrix or data frame. |
| FUN | a string representing the persistent homology function to use, either 'calculate_homology' (the default) or 'ripsDiag'. |
| maxdim | the integer maximum homological dimension for persistent homology, default 0. |
| thresh | the positive numeric maximum radius of the Vietoris-Rips filtration. |
| distance_mat | a boolean representing if 'X' is a distance matrix (TRUE) or not (FALSE, default). dimensions together (TRUE, the default) or if one threshold should be calculated for each dimension separately (FALSE). |
| ripser | the imported ripser module when 'FUN' is 'PyH'. |
| ignore_infinite_cluster | a boolean indicating whether or not to ignore the infinitely lived cluster when 'FUN' is 'PyH'. |
| calculate_representatives | a boolean representing whether to calculate representative (co)cycles, default FALSE. Note that representatives cant be calculated when using the 'calculate_homology' function. |
| num_samples | the positive integer number of bootstrap samples, default 30. |
| alpha | the type-1 error threshold, default 0.05. |
| return_subsetted | a boolean representing whether or not to return the subsetted persistence diagram (with or without representatives), default FALSE. |
| return_diag | a boolean representing whether or not to return the calculated persistence diagram, default TRUE. |
| num_workers | the integer number of cores used for parallelizing (over bootstrap samples), default one less the maximum amount of cores on the machine. |

## Details

The thresholds are determined by calculating the 1-alpha percentile of the bottleneck distance values between the real persistence diagram and other diagrams obtained by bootstrap resampling the data. Note that since calculate_homology can ignore the longest-lived cluster, fewer "real" clusters may be found. To avoid this possibility try setting ‘FUN‘ equal to ’ripsDiag’.

## Value

a numeric vector of threshold values ,with one for each dimension 0..‘maxdim‘ (in that order).

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Chazal F et al (2017). "Robust Topological Inference: Distance to a Measure and Kernel Distance." https://www.jmlr.org/papers/volume18/15-484/15-484.pdf.

## Examples

```
# create a persistence diagram from a sample of the unit circle
df = TDA::circleUnif(n = 50)

# calculate persistence thresholds for alpha = 0.05
# and return the calculated diagram as well as the subsetted diagram
bootstrapped_diagram <- bootstrap_persistence_thresholds(X = df,
FUN = "calculate_homology",maxdim = 1,thresh = 2,num_workers = 2)
```

---

# check_PyH_setup *Make sure that python has been configured correctly for persistent homology calculations.*

---

## Description

Ensures that the reticulate package has been installed, that python is available to be used by reticulate functions, and that the python module "ripser" has been installed.

## Usage

```
check_PyH_setup()
```

## Details

An error message will be thrown if any of the above conditions are not met.

### Author(s)

Shael Brown - <shaelebrown@gmail.com>

---

`check_ripser` *Verify an imported ripser module.*

---

### Description

Verify an imported ripser module.

### Usage

```
check_ripser(ripser)
```

### Arguments

| | |
|---|---|
| `ripser` | the ripser module object. |

### Author(s)

Shael Brown - <shaelebrown@gmail.com>

---

`diagram_distance` *Calculate distance between a pair of persistence diagrams.*

---

### Description

Calculates the distance between a pair of persistence diagrams, either the output from a `diagram_to_df` function call or from a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`, in a particular homological dimension.

### Usage

```
diagram_distance(
  D1,
  D2,
  dim = 0,
  p = 2,
  distance = "wasserstein",
  sigma = NULL
)
```

## Arguments

| | |
|---|---|
| D1 | the first persistence diagram. |
| D2 | the second persistence diagram. |
| dim | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| p | a number representing the wasserstein power parameter, at least 1 and default 2. |
| distance | a string which determines which type of distance calculation to carry out, either "wasserstein" (default) or "fisher". |
| sigma | either NULL (default) or a positive number representing the bandwidth for the Fisher information metric |

## Details

The most common distance calculations between persistence diagrams are the wasserstein and bottleneck distances, both of which "match" points between their two input diagrams and compute the "loss" of the optimal matching (see http://www.geometrie.tugraz.at/kerber/kerber_papers/kmn-ghtcpd_journal.pdf for details). Another method for computing distances, the Fisher information metric, converts the two diagrams into distributions defined on the plane, and calculates a distance between the resulting two distributions (https://proceedings.neurips.cc/paper/2018/file/959ab9a0695c467e7caf75431a872e5c-Paper.pdf). If the ‘distance‘ parameter is "fisher" then ‘sigma‘ must not be NULL.

## Value

the numeric value of the distance calculation.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Kerber M, Morozov D and Nigmetov A (2017). "Geometry Helps to Compare Persistence Diagrams." http://www.geometrie.tugraz.at/kerber/kerber_papers/kmn-ghtcpd_journal.pdf.

Le T, Yamada M (2018). "Persistence fisher kernel: a riemannian manifold kernel for persistence diagrams." https://proceedings.neurips.cc/paper/2018/file/959ab9a0695c467e7caf75431a872e5c-Paper.pdf.

## Examples

```
# create two diagrams
D1 <- TDA::ripsDiag(TDA::circleUnif(n = 20,r = 1),
                    maxdimension = 1,maxscale = 2)
D2 <- TDA::ripsDiag(TDA::sphereUnif(n = 20,d = 2,r = 1),
                    maxdimension = 1,maxscale = 2)
```

```
# calculate 2-wasserstein distance between D1 and D2 in dimension 1
diagram_distance(D1,D2,dim = 1,p = 2,distance = "wasserstein")

# calculate bottleneck distance between D1 and D2 in dimension 0
diagram_distance(D1,D2,dim = 0,p = Inf,distance = "wasserstein")

# Fisher information metric calculation between D1 and D2 for sigma = 1 in dimension 1
diagram_distance(D1,D2,dim = 1,distance = "fisher",sigma = 1)
```

---

`diagram_kernel` *Calculate persistence Fisher kernel value between a pair of persistence diagrams.*

---

## Description

Returns the persistence Fisher kernel value between a pair of persistence diagrams in a particular homological dimension, each of which is either the output from a `diagram_to_df` function call or from a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`.

## Usage

```
diagram_kernel(D1, D2, dim = 0, sigma = 1, t = 1)
```

## Arguments

| | |
|---|---|
| `D1` | the first persistence diagram. |
| `D2` | the second persistence diagram. |
| `dim` | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| `sigma` | a positive number representing the bandwidth for the Fisher information metric, default 1. |
| `t` | a positive number representing the scale for the persistence Fisher kernel, default 1. |

## Details

The persistence Fisher kernel is calculated from the Fisher information metric according to the formula $k_{PF}(D_1, D_2) = exp(-t * d_{FIM}(D_1, D_2))$, resembling a radial basis kernel for standard Euclidean spaces.

## Value

the numeric kernel value.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Le T, Yamada M (2018). "Persistence fisher kernel: a riemannian manifold kernel for persistence diagrams." https://proceedings.neurips.cc/paper/2018/file/959ab9a0695c467e7caf75431a872e5c-Paper.pdf.

Murphy, K. "Machine learning: a probabilistic perspective", MIT press (2012).

## Examples

```
# create two diagrams
D1 <- TDA::ripsDiag(TDA::circleUnif(n = 20,r = 1),
                    maxdimension = 1,maxscale = 2)
D2 <- TDA::ripsDiag(TDA::sphereUnif(n = 20,d = 2,r = 1),
                    maxdimension = 1,maxscale = 2)

# calculate the kernel value between D1 and D2 with sigma = 2, t = 2 in dimension 1
diagram_kernel(D1,D2,dim = 1,sigma = 2,t = 2)
# calculate the kernel value between D1 and D2 with sigma = 2, t = 2 in dimension 0
diagram_kernel(D1,D2,dim = 0,sigma = 2,t = 2)
```

---

# diagram_kkmeans *Cluster a group of persistence diagrams using kernel k-means.*

---

## Description

Finds latent cluster labels for a group of persistence diagrams, using a kernelized version of the popular k-means algorithm. An optimal number of clusters may be determined by analyzing the withinss field of the clustering object over several values of k.

## Usage

```
diagram_kkmeans(
  diagrams,
  centers,
  dim = 0,
  t = 1,
  sigma = 1,
  num_workers = parallelly::availableCores(omit = 1),
  ...
)
```

## Arguments

| | |
|---|---|
| diagrams | a list of n>=2 persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or the diagram_to_df function. |

| | |
|---|---|
| centers | number of clusters to initialize, no more than the number of diagrams although smaller values are recommended. |
| dim | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| t | a positive number representing the scale for the persistence Fisher kernel, default 1. |
| sigma | a positive number representing the bandwidth for the Fisher information metric, default 1 |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |
| ... | additional parameters for the kkmeans kernlab function. |

## Details

Returns the output of kkmeans on the desired Gram matrix of a group of persistence diagrams in a particular dimension. The additional list elements stored in the output are needed to estimate cluster labels for new persistence diagrams in the 'predict_diagram_kkmeans' function.

## Value

a 'diagram_kkmeans' S3 object containing the output of kkmeans on the Gram matrix, i.e. a list containing the elements

**clustering** an S4 object of class specc, the output of a kkmeans function call. The '.Data' slot of this object contains cluster memberships, 'withinss' contains the within-cluster sum of squares for each cluster, etc.

**diagrams** the input 'diagrams' argument.

**dim** the input 'dim' argument.

**t** the input 't' argument.

**sigma** the input 'sigma' argument.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Dhillon, I and Guan, Y and Kulis, B (2004). "A Unified View of Kernel k-means , Spectral Clustering and Graph Cuts." https://people.bu.edu/bkulis/pubs/spectral_techreport.pdf.

## See Also

predict_diagram_kkmeans for predicting cluster labels of new diagrams.

## Examples

```
# create two diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g <- list(D1,D1,D2,D2)

# calculate kmeans clusters with centers = 2, and sigma = t = 2 in dimension 0
clust <- diagram_kkmeans(diagrams = g,centers = 2,dim = 0,t = 2,sigma = 2,num_workers = 2)
```

---

# diagram_kpca *Calculate the kernel PCA embedding of a group of persistence diagrams.*

---

## Description

Project a group of persistence diagrams into a low-dimensional embedding space using a kernelized version of the popular PCA algorithm.

## Usage

```
diagram_kpca(
  diagrams,
  dim = 0,
  t = 1,
  sigma = 1,
  features = 1,
  num_workers = parallelly::availableCores(omit = 1),
  th = 1e-04
)
```

## Arguments

| | |
|---|---|
| diagrams | a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df. |
| dim | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| t | a positive number representing the scale for the persistence Fisher kernel, default 1. |
| sigma | a positive number representing the bandwidth for the Fisher information metric, default 1 |
| features | number of features (principal components) to return, default 1. |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |
| th | the threshold value under which principal components are ignored (default 0.0001). |

## Details

Returns the output of kernlab's kpca function on the desired Gram matrix of a group of persistence diagrams in a particular dimension. The prediction function predict_diagram_kpca can be used to project new persistence diagrams using an old embedding, and this could be one practical advantage of using diagram_kpca over diagram_mds. The embedding coordinates can also be used for further analysis, or simply as a data visualization tool for persistence diagrams.

## Value

a list containing the elements

**pca** the output of kernlab's kpca function on the Gram matrix: an S4 object containing the slots 'pcv' (a matrix containing the principal component vectors (column wise)), 'eig' (the corresponding eigenvalues), 'rotated' (the original data projected (rotated) on the principal components) and 'xmatrix' (the original data matrix).

**diagrams** the input 'diagrams' argument.

**t** the input 't' argument.

**sigma** the input 'sigma' argument.

**dim** the input 'dim' argument.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Scholkopf, B and Smola, A and Muller, K (1998). "Nonlinear Component Analysis as a Kernel Eigenvalue Problem." https://www.mlpack.org/papers/kpca.pdf.

## See Also

predict_diagram_kpca for predicting embedding coordinates of new diagrams.

## Examples

```
# create six diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 50,r = 1),
                                   dim = 1,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::sphereUnif(n = 50,d = 2,r = 1),
                                   dim = 1,threshold = 2)
D3 <- TDAstats::calculate_homology(TDA::torusUnif(n = 50,a = 0.25,c = 0.75),
                                   dim = 1,threshold = 2)
D4 <- TDAstats::calculate_homology(TDA::circleUnif(n = 50,r = 1),
                                   dim = 1,threshold = 2)
D5 <- TDAstats::calculate_homology(TDA::sphereUnif(n = 50,d = 2,r = 1),
                                   dim = 1,threshold = 2)
D6 <- TDAstats::calculate_homology(TDA::torusUnif(n = 50,a = 0.25,c = 0.75),
                                   dim = 1,threshold = 2)
```

```
g <- list(D1,D2,D3,D4,D5,D6)

# calculate their 2D PCA embedding with sigma = t = 2 in dimension 1
pca <- diagram_kpca(diagrams = g,dim = 1,t = 2,sigma = 2,features = 2,num_workers = 2)
```

---

`diagram_ksvm` *Fit a support vector machine model where each training set instance is a persistence diagram.*

---

## Description

Returns the output of kernlab's `ksvm` function on the Gram matrix of the list of persistence diagrams in a particular dimension.

## Usage

```
diagram_ksvm(
  diagrams,
  cv = 1,
  dim,
  t = 1,
  sigma = 1,
  y,
  type = NULL,
  C = 1,
  nu = 0.2,
  epsilon = 0.1,
  prob.model = FALSE,
  class.weights = NULL,
  fit = TRUE,
  cache = 40,
  tol = 0.001,
  shrinking = TRUE,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| `diagrams` | a list of persistence diagrams which are either the output of a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`, or `diagram_to_df`. |
| `cv` | a positive number at most the length of 'diagrams' which determines the number of cross validation splits to be performed (default 1, aka no cross-validation). |
| `dim` | a non-negative integer vector of homological dimensions in which the model is to be fit. |
| `t` | a vector of positive numbers representing the grid of values for the scale of the persistence Fisher kernel, default 1. |

| | |
|---|---|
| sigma | a vector of positive numbers representing the grid of values for the bandwidth of the Fisher information metric, default 1 |
| y | a response vector with one label for each persistence diagram. Must be either numeric or factor. |
| type | a string representing the type of task to be performed. |
| C | a number representing the cost of constraints violation (default 1) this is the 'C'-constant of the regularization term in the Lagrange formulation. |
| nu | numeric parameter needed for nu-svc, one-svc and nu-svr. The 'nu' parameter sets the upper bound on the training error and the lower bound on the fraction of data points to become Support Vector (default 0.2). |
| epsilon | epsilon in the insensitive-loss function used for eps-svr, nu-svr and eps-bsvm (default 0.1). |
| prob.model | if set to TRUE builds a model for calculating class probabilities or in case of regression, calculates the scaling parameter of the Laplacian distribution fitted on the residuals. Fitting is done on output data created by performing a 3-fold cross-validation on the training data. For details see references (default FALSE). |
| class.weights | a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named. |
| fit | indicates whether the fitted values should be computed and included in the model or not (default TRUE). |
| cache | cache memory in MB (default 40). |
| tol | tolerance of termination criteria (default 0.001). |
| shrinking | option whether to use the shrinking-heuristics (default TRUE). |
| num_workers | the number of cores used for parallel computation, default is one less the number of cores on the machine. |

## Details

Cross validation is carried out in parallel, using a trick noted in doi: 10.1007/s4146801700087 - since the persistence Fisher kernel can be written as $d_{PF}(D_1, D_2) = exp(t * d_{FIM}(D_1, D_2)) = exp(d_{FIM}(D_1, D_2))^t$, we can store the Fisher information metric distance matrix for each sigma value in the parameter grid to avoid recomputing distances, and cross validation is therefore performed in parallel. Note that the response parameter 'y' must be a factor for classification - a character vector for instance will throw an error.

## Value

a list containing the elements

**models** the cross-validation results - a matrix storing the parameters for each model in the tuning grid and its mean cross-validation error over all splits.

**best_model** the output of `ksvm` run on the whole dataset with the optimal model parameters found during cross-validation. See the help page for `ksvm` for more details about this object.

**diagrams** the diagrams which were support vectors in the ‘best_model‘. These are used for downstream prediction.

**dim** the input ‘dim‘ argument.

**t** the input ‘t‘ argument.

**sigma** the input ‘sigma‘ argument.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Murphy, K. "Machine learning: a probabilistic perspective." MIT press (2012).

## See Also

predict_diagram_ksvm for predicting labels of new diagrams.

## Examples

```
# create four diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D3 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D4 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g <- list(D1,D2,D3,D4)

# create response vector
y <- as.factor(c("circle","sphere","circle","sphere"))

# fit model without cross validation
model_svm <- diagram_ksvm(diagrams = g,cv = 1,dim = c(0),
                          y = y,sigma = c(1),t = c(1),
                          num_workers = 2)
```

---

# diagram_mds *Dimension reduction of a group of persistence diagrams via metric multidimensional scaling.*

---

## Description

Projects a group of persistence diagrams into a low-dimensional embedding space via metric multidimensional scaling. Such a projection can be used for visualization of data, or a static analysis of the embedding dimensions.

## Usage

```
diagram_mds(
  diagrams,
  k = 2,
  distance = "wasserstein",
  dim = 0,
  p = 2,
  sigma = NULL,
  eig = FALSE,
  add = FALSE,
  x.ret = FALSE,
  list. = eig || add || x.ret,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| diagrams | a list of n>=2 persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df. |
| k | the dimension of the space which the data are to be represented in; must be in 1,2,...,n-1. |
| distance | a string representing the desired distance metric to be used, either 'wasserstein' (default) or 'fisher'. |
| dim | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| p | a positive number representing the wasserstein power, a number at least 1 (infinity for the bottleneck distance), default 2. |
| sigma | a positive number representing the bandwidth for the Fisher information metric, default NULL. |
| eig | a boolean indicating whether the eigenvalues should be returned. |
| add | a boolean indicating if an additive constant c* should be computed, and added to the non-diagonal dissimilarities such that the modified dissimilarities are Euclidean. |
| x.ret | a boolean indicating whether the doubly centered symmetric distance matrix should be returned. |
| list. | a boolean indicating if a list should be returned or just the n*k matrix. |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |

## Details

Returns the output of cmdscale on the desired distance matrix of a group of persistence diagrams in a particular dimension. If ‘distance‘ is "fisher" then ‘sigma‘ must not be NULL.

## Value

the output of cmdscale on the diagram distance matrix. If 'list.' is false (as per default), a matrix with 'k' columns whose rows give the coordinates of the points chosen to represent the dissimilarities.

Otherwise, a list containing the following components.

**points** a matrix with 'k' columns whose rows give the coordinates of the points chosen to represent the dissimilarities.

**eig** the $n$ eigenvalues computed during the scaling process if 'eig' is true.

**x** the doubly centered distance matrix if 'x.ret' is true.

**ac** the additive constant $c*$, 0 if 'add' = FALSE.

**GOF** the numeric vector of length 2, representing the sum of all the eigenvalues divided by the sum of their absolute values (first vector element) or by the sum of the max of each eigenvalue and 0 (second vector element).

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Cox M and Cox F (2008). "Multidimensional Scaling." doi: 10.1007/9783540330370_14.

## Examples

```
# create two diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g <- list(D1,D2)

# calculate their 1D MDS embedding in dimension 0 with the bottleneck distance
mds <- diagram_mds(diagrams = g,k = 1,dim = 0,p = Inf,num_workers = 2)
```

---

# diagram_to_df *Convert a TDA/TDAstats persistence diagram to a data frame.*

---

## Description

The output of homology calculations from the R packages TDA and TDAstats are not dataframes. This function converts these outputs into a data frame either for further usage in this package or for personalized analyses.

## Usage

```
diagram_to_df(d)
```

## Arguments

| | |
|---|---|
| d | the output of a TDA/TDAstats homology calculation, like ripsDiag or calculate_homology. |

## Details

If a diagram is constructed using a TDA function like ripsDiag with the 'location' parameter set to true then the return value will ignore the location information.

## Value

a 3-column data frame, with each row representing a topological feature. The first column is the feature dimension (a non-negative integer), the second column is the birth radius of the feature and the third column is the death radius.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## Examples

```
# create a persistence diagram from a 2D Gaussian
df = data.frame(x = rnorm(n = 20,mean = 0,sd = 1),y = rnorm(n = 20,mean = 0,sd = 1))

# compute persistence diagram with ripsDiag from package TDA
phom_TDA = TDA::ripsDiag(X = df,maxdimension = 0,maxscale = 1)

# convert to data frame
phom_TDA_df = diagram_to_df(d = phom_TDA)

# compute persistence diagram with calculate_homology from package TDAstats
phom_TDAstats = TDAstats::calculate_homology(mat = df,dim = 0,threshold = 1)

# convert to data frame
phom_TDAstats_df = diagram_to_df(d = phom_TDAstats)
```

---

# distance_matrix *Compute a distance matrix from a list of persistence diagrams.*

---

## Description

Calculate the distance matrix $d$ for either a single list of persistence diagrams $(D_1, D_2, \ldots, D_n)$, i.e. $d[i,j] = d(D_i, D_j)$, or between two lists, $(D_1, D_2, \ldots, D_n)$ and $(D'_1, D'_2, \ldots, D'_n)$, $d[i,j] = d(D_i, D'_j)$, in parallel.

## Usage

```
distance_matrix(
  diagrams,
  other_diagrams = NULL,
  dim = 0,
  distance = "wasserstein",
  p = 2,
  sigma = NULL,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| diagrams | a list of persistence diagrams, either the output of persistent homology calculations like ripsDiag/calculate_homology/PyH, or diagram_to_df. |
| other_diagrams | either NULL (default) or another list of persistence diagrams to compute a cross-distance matrix. |
| dim | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| distance | a character determining which metric to use, either "wasserstein" (default) or "fisher". |
| p | a number representing the wasserstein power parameter, at least 1 and default 2. |
| sigma | a positive number representing the bandwidth of the Fisher information metric, default NULL. |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |

## Details

Distance matrices of persistence diagrams are used in downstream analyses, like in the diagram_mds, permutation_test and diagram_ksvm functions. If ‘distance‘ is "fisher" then ‘sigma‘ must not be NULL.

## Value

the numeric distance matrix.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## Examples

```
# create two diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
```

```
                                    dim = 0,threshold = 2)
g <- list(D1,D2)

# calculate their distance matrix in dimension 0 with the 2-wasserstein metric
# using 2 cores in dimension 1
D <- distance_matrix(diagrams = g,dim = 0,distance = "wasserstein",p = 2,num_workers = 2)

# now do the cross distance matrix, which is the same as the original
D_cross <- distance_matrix(diagrams = g,other_diagrams = g,
                           dim = 0,distance = "wasserstein",
                           p = 2,num_workers = 2)
```

---

`gram_matrix` *Compute the gram matrix for a group of persistence diagrams.*

---

## Description

Calculate the Gram matrix $K$ for either a single list of persistence diagrams $(D_1, D_2, \ldots, D_n)$, i.e. $K[i,j] = k_{PF}(D_i, D_j)$, or between two lists of persistence diagrams, $(D_1, D_2, \ldots, D_n)$ and $(D'_1, D'_2, \ldots, D'_n)$, $K[i,j] = k_{PF}(D_i, D'_j)$, in parallel.

## Usage

```
gram_matrix(
  diagrams,
  other_diagrams = NULL,
  dim = 0,
  sigma = 1,
  t = 1,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| `diagrams` | a list of persistence diagrams, where each diagram is either the output of a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`, or `diagram_to_df`. |
| `other_diagrams` | either NULL (default) or another list of persistence diagrams to compute a cross-Gram matrix. |
| `dim` | the non-negative integer homological dimension in which the distance is to be computed, default 0. |
| `sigma` | a positive number representing the bandwidth for the Fisher information metric, default 1. |
| `t` | a positive number representing the scale for the kernel, default 1. |
| `num_workers` | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |

### Details

Gram matrices are used in downstream analyses, like in the ‘diagram_kkmeans‘, ‘diagram_nearest_cluster‘,‘diagram_kpca‘, ‘predict_diagram_kpca‘, ‘predict_diagram_ksvm‘ and ‘independence_test‘ functions.

### Value

the numeric (cross) Gram matrix of class ’kernelMatrix’.

### Author(s)

Shael Brown - <shaelebrown@gmail.com>

### Examples

```
# create two diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g <- list(D1,D2)

# calculate the Gram matrix in dimension 0 with sigma = 2, t = 2
G <- gram_matrix(diagrams = g,dim = 0,sigma = 2,t = 2,num_workers = 2)

# calculate cross-Gram matrix, which is the same as G
G_cross <- gram_matrix(diagrams = g,other_diagrams = g,dim = 0,sigma = 2,
                       t = 2,num_workers = 2)
```

---

## import_ripser *Import the python module ripser.*

---

### Description

The ripser module is needed for fast persistent cohomology calculations with the PyH function.

### Usage

```
import_ripser()
```

### Details

Same as "reticulate::import("ripser")", just with additional checks.

### Value

the python ripser module.

### Author(s)

Shael Brown - <shaelebrown@gmail.com>

### Examples

```
## Not run:
# import ripser
ripser <- import_ripser()

## End(Not run)
```

---

`independence_test` *Independence test for two groups of persistence diagrams.*

---

### Description

Carries out inference to determine if two groups of persistence diagrams are independent or not based on kernel calculations (see (https://proceedings.neurips.cc/paper/2007/file/d5cfead94f5350c12c322b5b6
pdf) for details). A small p-value in a certain dimension suggests that the groups are not independent in that dimension.

### Usage

```
independence_test(
  g1,
  g2,
  dims = c(0, 1),
  sigma = 1,
  t = 1,
  num_workers = parallelly::availableCores(omit = 1),
  verbose = FALSE
)
```

### Arguments

| | |
|---|---|
| `g1` | the first group of persistence diagrams, where each diagram was either the output from a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`, or `diagram_to_df`. |
| `g2` | the second group of persistence diagrams, where each diagram was either the output from a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`, or `diagram_to_df`. |
| `dims` | a non-negative integer vector of the homological dimensions in which the test is to be carried out, default c(0,1). |
| `sigma` | a positive number representing the bandwidth for the Fisher information metric, default 1. |

| | |
|---|---|
| t | a positive number representing the scale for the persistence Fisher kernel, default 1. |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |
| verbose | a boolean flag for if the time duration of the function call should be printed, default FALSE |

## Details

The test is carried out with a parametric null distribution, making it much faster than non-parametric approaches. If all of the diagrams in either g1 or g2 are the same in some dimension, then some p-values may be NaN.

## Value

a list with the following elements:

**dimensions** the input 'dims' argument.

**test_statisics** a numeric vector of the test statistic value in each dimension.

**p_values** a numeric vector of the p-values in each dimension.

**run_time** the run time of the function call, containing time units.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Gretton A et al. (2007). "A Kernel Statistical Test of Independence." https://proceedings.neurips.cc/paper/2007/file/d5cfead94f5350c12c322b5b664544c1-Paper.pdf.

## Examples

```
# create two independent groups of diagrams of length 6, which
# is the minimum length
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g1 <- list(D1,D2,D2,D2,D2,D2)
g2 <- list(D2,D1,D1,D1,D1,D1)

# do independence test with sigma = t = 1 in dimension 0
indep_test <- independence_test(g1,g2,dims = c(0),num_workers = 2)
```

permutation_test *Permutation test for finding group differences between persistence diagrams.*

## Description

A non-parametric ANOVA-like test for persistence diagrams (see https://link.springer.com/article/10.1007/s41468-017-0008-7 for details). In each desired dimension a test statistic (loss) is calculated, then the group labels are shuffled for some number of iterations and the loss is recomputed each time thereby generating a null distribution for the test statistic. This test generates a p-value in each desired dimension.

## Usage

```
permutation_test(
  ...,
  iterations = 20,
  p = 2,
  q = 2,
  dims = c(0, 1),
  paired = FALSE,
  distance = "wasserstein",
  sigma = NULL,
  num_workers = parallelly::availableCores(omit = 1),
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| ... | lists of persistence diagrams which are either the output of persistent homology calculations like ripsDiag/calculate_homology/PyH, or diagram_to_df. Each list must contain at least 2 diagrams. |
| iterations | the number of iterations for permuting group labels, default 20. |
| p | a positive number representing the wasserstein power parameter, a number at least 1 (and Inf if using the bottleneck distance) and default 2. |
| q | a finite number at least 1 for exponentiation in the Turner loss function, default 2. |
| dims | a non-negative integer vector of the homological dimensions in which the test is to be carried out, default c(0,1). |
| paired | a boolean flag for if there is a second-order pairing between diagrams at the same index in different groups, default FALSE |
| distance | a string which determines which type of distance calculation to carry out, either "wasserstein" (default) or "fisher". |
| sigma | the positive bandwidth for the Fisher information metric, default NULL. |

| | |
|---|---|
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |
| verbose | a boolean flag for if the time duration of the function call should be printed, default FALSE |

## Details

The test is carried out in parallel and optimized in order to not recompute already-calculated distances. As such, memory issues may occur when the number of persistence diagrams is very large. Like in (https://github.com/hassan-abdallah/Statistical_Inference_PH_fMRI/blob/main/Abdallah_et_al_Statistical_Inference_PH_fMRI.pdf) an option is provided for pairing diagrams between groups to reduce variance (in order to boost statistical power), and like it was suggested in the original paper functionality is provided for an arbitrary number of groups (not just 2). A small p-value in a dimension suggests that the groups are different (separated) in that dimension. If 'distance' is "fisher" then 'sigma' must not be NULL. TDAstats also has a 'permutation_test' function so care should be taken to use the desired function when using TDApplied with TDAstats.

## Value

a list with the following elements:

**dimensions** the input 'dims' argument.

**permvals** a numeric vector of length 'iterations' with the permuted loss value for each iteration (permutation)

**test_statisics** a numeric vector of the test statistic value in each dimension.

**p_values** a numeric vector of the p-values in each dimension.

**run_time** the run time of the function call, containing time units.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## References

Robinson T, Turner K (2017). "Hypothesis testing for topological data analysis." https://link.springer.com/article/10.1007/s41468-017-0008-7.

Abdallah H et al. (2021). "Statistical Inference for Persistent Homology applied to fMRI." https://github.com/hassan-abdallah/Statistical_Inference_PH_fMRI/blob/main/Abdallah_et_al_Statistical_Inference_PH_fMRI.pdf.

## Examples

```
# create two groups of diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g1 <- list(D1,D2)
```

```
g2 <- list(D1,D2)

# run test in dimension 0 with 1 iteration
perm_test <- permutation_test(g1,g2,iterations = 1,
                              num_workers = 2,
                              dims = c(0))
```

---

plot_diagram *Plot persistence diagrams*

---

## Description

Plots a persistence diagram outputted from either a persistent homology calculation or from diagram_to_df, with maximum homological dimension no more than 12 (otherwise the legend doesn't fit in the plot). Each homological dimension has its own color and point type (with colors chosen to be clear and distinct from each other), and the main plot title can be altered via the 'title' parameter.

## Usage

```
plot_diagram(
  D,
  title = NULL,
  max_radius = NULL,
  legend = TRUE,
  thresholds = NULL
)
```

## Arguments

| | |
|---|---|
| D | a persistence diagram, either outputted from either a persistent homology homology calculation like ripsDiag/calculate_homology/PyH or from diagram_to_df, with maximum dimension at most 12. |
| title | the character string plot title, default NULL. |
| max_radius | the x and y limits of the plot are defined as 'c(0,max_radius)', and the default value of 'max_radius' is the maximum death value in 'D'. |
| legend | a logical indicating whether to include a legend of feature dimensions, default TRUE. |
| thresholds | either a numeric vector with one persistence threshold for each dimension in 'D' or the output of a bootstrap_persistence_thresholds function call, default NULL. |

## Details

The 'thresholds' parameter, if not NULL, can either be a user-defined numeric vector, with one entry (persistence threshold) for each dimension in 'D', or the output of bootstrap_persistence_thresholds. Points whose persistence are greater than or equal to their dimension's threshold will be plotted in their dimension's color, and in gray otherwise.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## Examples

```
# create a sample diagram from the unit circle
df <- TDA::circleUnif(n = 50)
diag <- TDAstats::calculate_homology(df,threshold = 2)

# plot without title
plot_diagram(diag)

# plot with title
plot_diagram(diag,title = "Example diagram")

# determine persistence thresholds
thresholds <- bootstrap_persistence_thresholds(X = df,maxdim = 1,
thresh = 2,num_samples = 3,
num_workers = 2)

# plot with bootstrap persistence thresholds
plot_diagram(diag,title = "Example diagram with thresholds",thresholds = thresholds)

#' # plot with personalized persistence thresholds
plot_diagram(diag,title = "Example diagram with personalized thresholds",thresholds = c(0.5,1))
```

---

# predict_diagram_kkmeans

*Predict the cluster labels for new persistence diagrams using a pre-computed clustering.*

---

## Description

Returns the nearest (highest kernel value) kkmeans cluster center label for new persistence diagrams. This allows for reusing old cluster models for new tasks, or to perform cross validation.

## Usage

```
predict_diagram_kkmeans(
  new_diagrams,
  clustering,
  num_workers = parallelly::availableCores(omit = 1)
)
```

### Arguments

| | |
|---|---|
| new_diagrams | a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df. |
| clustering | the output of a diagram_kkmeans function call. |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |

### Value

a vector of the predicted cluster labels for the new diagrams.

### Author(s)

Shael Brown - <shaelebrown@gmail.com>

### See Also

diagram_kkmeans for clustering persistence diagrams.

### Examples

```
# create two diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g <- list(D1,D1,D2,D2)

# calculate kmeans clusters with centers = 2, and sigma = t = 2 in dimension 0
clust <- diagram_kkmeans(diagrams = g,centers = 2,dim = 0,t = 2,sigma = 2,num_workers = 2)

# create two new diagrams
D4 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D5 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g_new <- list(D4,D5)

# predict cluster labels
predict_diagram_kkmeans(new_diagrams = g_new,clustering = clust,num_workers = 2)
```

---

## predict_diagram_kpca *Project persistence diagrams into a low-dimensional space via a pre-computed kernel PCA embedding.*

---

## Description

Compute the location in low-dimensional space of each element of a list of new persistence diagrams using a previously-computed kernel PCA embedding (from the diagram_kpca function).

## Usage

```
predict_diagram_kpca(
  new_diagrams,
  embedding,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| new_diagrams | a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df. |
| embedding | the output of a diagram_kpca function call. |
| num_workers | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |

## Value

the data projection (rotation), stored as a numeric matrix. Each row corresponds to the same-index diagram in 'new_diagrams'.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## See Also

diagram_kpca for embedding persistence diagrams into a low-dimensional space.

## Examples

```
# create six diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 50,r = 1),
                                   dim = 1,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::sphereUnif(n = 50,d = 2,r = 1),
                                   dim = 1,threshold = 2)
D3 <- TDAstats::calculate_homology(TDA::torusUnif(n = 50,a = 0.25,c = 0.75),
                                   dim = 1,threshold = 2)
D4 <- TDAstats::calculate_homology(TDA::circleUnif(n = 50,r = 1),
                                   dim = 1,threshold = 2)
D5 <- TDAstats::calculate_homology(TDA::sphereUnif(n = 50,d = 2,r = 1),
                                   dim = 1,threshold = 2)
D6 <- TDAstats::calculate_homology(TDA::torusUnif(n = 50,a = 0.25,c = 0.75),
                                   dim = 1,threshold = 2)
g <- list(D1,D2,D3,D4,D5,D6)
```

```
# calculate their 2D PCA embedding with sigma = t = 2 in dimension 0
pca <- diagram_kpca(diagrams = g,dim = 1,t = 2,sigma = 2,features = 2,num_workers = 2)

# project two new diagrams onto old model
D7 <- TDAstats::calculate_homology(TDA::circleUnif(n = 50,r = 1),
                                   dim = 0,threshold = 2)
D8 <- TDAstats::calculate_homology(TDA::circleUnif(n = 50,r = 1),
                                   dim = 0,threshold = 2)
g_new <- list(D4,D5)

# calculate new embedding coordinates
new_pca <- predict_diagram_kpca(new_diagrams = g_new,embedding = pca,num_workers = 2)
```

---

`predict_diagram_ksvm` *Predict the outcome labels for a list of persistence diagrams using a pre-trained diagram ksvm model.*

---

## Description

Returns the predicted response vector of the model on the new diagrams.

## Usage

```
predict_diagram_ksvm(
  new_diagrams,
  model,
  num_workers = parallelly::availableCores(omit = 1)
)
```

## Arguments

| | |
|---|---|
| `new_diagrams` | a list of persistence diagrams which are either the output of a persistent homology calculation like `ripsDiag`/`calculate_homology`/`PyH`, or `diagram_to_df`. |
| `model` | the output of a `diagram_ksvm` function call. |
| `num_workers` | the number of cores used for parallel computation, default is one less than the number of cores on the machine. |

## Details

This function is a wrapper of the kernlab `predict` function.

## Value

a vector containing the output of `predict.ksvm` on the cross Gram matrix of the new diagrams and the support vector diagrams stored in the model.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## See Also

diagram_ksvm for training a SVM model on a training set of persistence diagrams.

## Examples

```
# create four diagrams
D1 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D2 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D3 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D4 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g <- list(D1,D2,D3,D4)

# create response vector
y <- as.factor(c("circle","sphere","circle","sphere"))

# fit model without cross validation
model_svm <- diagram_ksvm(diagrams = g,cv = 1,dim = c(0),
                          y = y,sigma = c(1),t = c(1),
                          num_workers = 2)

# create two new diagrams
D5 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
D6 <- TDAstats::calculate_homology(TDA::circleUnif(n = 10,r = 1),
                                   dim = 0,threshold = 2)
g_new <- list(D5,D6)

# predict
predict_diagram_ksvm(new_diagrams = g_new,model = model_svm,num_workers = 2)
```

---

PyH *Fast persistent homology calculations with python.*

---

## Description

This function is a wrapper of the python wrapper of the ripser engine for persistent cohomology, but is still faster than using the R package TDAstats (see the TDApplied package vignette for details).

## Usage

```
PyH(
  X,
  maxdim = 1,
  thresh,
  distance_mat = FALSE,
  ripser,
  ignore_infinite_cluster = TRUE,
  calculate_representatives = FALSE
)
```

## Arguments

| | |
|---|---|
| X | either a matrix or dataframe, representing either point cloud data or a distance matrix. In either case there must be at least two rows and 1 column. |
| maxdim | the non-negative integer maximum dimension for persistent homology, default 1. |
| thresh | the non-negative numeric radius threshold for the Vietoris-Rips filtration. |
| distance_mat | a boolean representing whether the input X is a distance matrix or not, default FALSE. |
| ripser | the ripser python module. |
| ignore_infinite_cluster | a boolean representing whether to remove clusters (0 dimensional cycles) which die at the threshold value. Default is TRUE as this is the default for TDAstats homology calculations, but can be set to FALSE which is the default for python ripser. |
| calculate_representatives | a boolean representing whether to return a list of representative cocycles for the topological features found in the persistence diagram, default FALSE. |

## Details

If ‘distance_mat‘ is ‘TRUE‘ then ‘X‘ must be a square matrix. The ‘ripser‘ parameter should be the result of an ‘import_ripser‘ function call, but since that function is slow the ripser object should be explicitly created before a PyH function call (see examples). Cohomology is computed over Z2, as is the case for the TDAstats function `calculate_homology` (this is also the default for ripser in c++). If representative cocycles are returned, then they are stored in a list with one element for each point in the persistence diagram, ignoring dimension 0 points. Each representative of a dimension d cocycle (1 for loops, 2 for voids, etc.) is a kxd dimension matrix/array containing the row number-labelled edges, triangles etc. in the cocycle.

## Value

Either a dataframe containing the persistence diagram if ‘calculate_representatives‘ is ‘FALSE‘ (the default), otherwise a list with two elements: diagram of class diagram, containing the persistence diagram, and representatives, a list containing the edges, triangles etc. contained in each representative cocycle.

## Author(s)

Shael Brown - <shaelebrown@gmail.com>

## Examples

```
## Not run:
# create sample data
df <- data.frame(x = 1:10,y = 1:10)

# import the ripser module
ripser <- import_ripser()

# calculate persistence diagram up to dimension 1 with a maximum
# radius of 5
phom <- PyH(X = df,thresh = 5,ripser = ripser)

## End(Not run)
```

---

TDApplied *Machine learning and inference for persistence diagrams*

---

## Description

This package aims to bridge topological data analysis (TDA) with data, statistical and machine learning practitioners so that more analyses may benefit from the power of TDA. The main tool of TDA is persistent homology, which computes a shape descriptor of a dataset, called a persistence diagram. There are five goals of this package: (1) convert the output from the persistent homology calculations in the two main R package for TDA into a data frame which can easily be used in personalized analyses, (2) provide a fast method for computing distances between persistence diagrams, (3) implement kernel machine learning and statistical methods for analyzing groups of persistence diagrams, (4) supply a fast persistent homology calculation via python and reticulate, and (5) provide tools for interpreting persistence diagrams.

# Index