

# Package ‘phutil’

May 15, 2025

**Title** Persistence Homology Utilities

**Version** 0.0.1

**Description** A low-level package for hosting persistence data. It is part of the 'TDAverse' suite of packages, which is designed to provide a collection of packages for enabling machine learning and data science tasks using persistent homology. Implements a class for hosting persistence data, a number of coercers from and to already existing and used data structures from other packages and functions to compute distances between persistence diagrams. A formal definition and study of bottleneck and Wasserstein distances can be found in Bubenik, Scott and Stanley (2023) [doi:10.1007/s41468-022-00103-8](https://doi.org/10.1007/s41468-022-00103-8). Their implementation in 'phutil' relies on the 'C++' Hera library developed by Kerber, Morozov and Nigmatov (2017) [doi:10.1145/3064175](https://doi.org/10.1145/3064175).

**License** MIT + file LICENSE

**URL** <https://github.com/tdaverse/phutil>,  
<https://tdaverse.github.io/phutil/>

**BugReports** <https://github.com/tdaverse/phutil/issues>

**Depends** R (>= 3.5)

**Imports** cli, rlang

**Suggests** ggplot2, knitr, microbenchmark, quarto, scales, TDA, tdaunif,  
tinysnapshot, tinytest

**LinkingTo** BH

**VignetteBuilder** quarto

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**NeedsCompilation** yes

**Author** Aymeric Stamm [aut, cre] (ORCID:  
<https://orcid.org/0000-0002-8725-3654>),  
Jason Cory Brunson [aut] (ORCID:

<<https://orcid.org/0000-0003-3126-9494>>),  
Michael Kerber [ctb] (HERA C++ code),  
Dmitriy Morozov [ctb] (HERA C++ code),  
Arnur Nigmatov [ctb] (HERA C++ code)

**Maintainer** Aymeric Stamm <aymeric.stamm@cnrs.fr>

**Repository** CRAN

**Date/Publication** 2025-05-15 13:50:08 UTC

Contents

arch_spirals . . . . .	2
distances . . . . .	3
noisy_circle . . . . .	5
pairwise-distances . . . . .	6
persistence . . . . .	8
persistence-set . . . . .	10
persistence_sample . . . . .	11
trefoils . . . . .	12
<b>Index</b>	<b>13</b>

---

arch_spirals	<i>A sample of persistence diagrams from the arch spiral</i>
--------------	--

---

Description

A collection of 24 samples of size 120 on the arch spiral from which a persistence diagram is computed using the `TDA::ripsDiag()` function with `maxdimension = 2` and `maxscale = 6`. Each diagram has been generated using the `tdaunif::sample_arch_spiral()` function with the following parameters: `n = 120`, `arms = 2`` and `sd = 0.05``. The seed was fixed to 28415.

Usage

arch\_spirals

Format

A list of length 24, where each element is an object of class 'persistence'.

---

distances	<i>Distances between two persistence diagrams</i>
-----------	---

---

## Description

This collection of functions computes the distance between two persistence diagrams of the same homology dimension. The diagrams must be represented as 2-column matrices. The first column of the matrix contains the birth times and the second column contains the death times of the points.

## Usage

```
bottleneck_distance(
  x,
  y,
  tol = sqrt(.Machine$double.eps),
  validate = TRUE,
  dimension = 0L
)
```

```
wasserstein_distance(
  x,
  y,
  tol = sqrt(.Machine$double.eps),
  p = 1,
  validate = TRUE,
  dimension = 0L
)
```

```
kantorovich_distance(
  x,
  y,
  tol = sqrt(.Machine$double.eps),
  p = 1,
  validate = TRUE,
  dimension = 0L
)
```

## Arguments

- |     |  |
|-----|--|
| x   | Either a matrix of shape $n \times 2$ or an object of class <a href="#">persistence</a> specifying the first persistence diagram.  |
| y   | Either a matrix of shape $m \times 2$ or an object of class <a href="#">persistence</a> specifying the second persistence diagram.   |
| tol | A numeric value specifying the relative error. Defaults to <code>sqrt(.Machine\$double.eps)</code> . For the Bottleneck distance, it can be set to <code>0.0</code> in which case the exact Bottleneck distance is computed, while an approximate Bottleneck distance is computed if <code>tol &gt; 0.0</code> . For the Wasserstein distance, it must be strictly positive. |

validate	A boolean value specifying whether to validate the input persistence diagrams. Defaults to TRUE. If FALSE, the function will not check if the input persistence diagrams are valid. This can be useful for performance reasons, but it is recommended to keep it TRUE for safety.
dimension	An integer value specifying the homology dimension for which to compute the distance. Defaults to 0L. This is only used if x and y are objects of class <a href="#">persistence</a> .
p	A numeric value specifying the power for the Wasserstein distance. Defaults to 1.0.

### Details

A matching  $\varphi : D_1 \rightarrow D_2$  between persistence diagrams is a bijection of multisets, where both diagrams are assumed to have all points on the diagonal with infinite multiplicity. The *p-Wasserstein distance* between  $D_1$  and  $D_2$  is defined as the infimum over all matchings of the expression

$$W_p(D_1, D_2) = \inf_{\varphi: D_1 \rightarrow D_2} \left( \sum_{x \in D_1} \|x - \varphi(x)\|^p \right)^{\frac{1}{p}}$$

that can be thought of as the Minkowski distance between the diagrams viewed as vectors on the shared coordinates defined by the matching  $\varphi$ . The norm  $\|\cdot\|$  can be arbitrary; as implemented here, it is the infinity norm  $\|(x_1, x_2)\|_\infty = \max(x_1, x_2)$ . In the limit  $p \rightarrow \infty$ , the Wasserstein distance becomes the *bottleneck distance*:

$$B(D_1, D_2) = \inf_{\varphi: D_1 \rightarrow D_2} \sup_{x \in D_1} \|x - \varphi(x)\|.$$

The Wasserstein metric is also called the Kantorovich metric in recognition of the originator of the metric.

### Value

A numeric value storing either the Bottleneck or the Wasserstein distance between the two persistence diagrams.

### See Also

[the Hera C++ library](#)

### Examples

```
bottleneck_distance(
  persistence_sample[[1]]$pairs[[1]],
  persistence_sample[[2]]$pairs[[1]]
)
```

```
bottleneck_distance(
  persistence_sample[[1]],
  persistence_sample[[2]]
)
```

```

)

wasserstein_distance(
  persistence_sample[[1]]$pairs[[1]],
  persistence_sample[[2]]$pairs[[1]]
)

wasserstein_distance(
  persistence_sample[[1]],
  persistence_sample[[2]]
)

```

noisy\_circle

*Toy Data: Noisy circle***Description**

A simulated data set consisting of 100 points sampled from a circle with additive Gaussian noise using a standard deviation of 0.05.

**Usage**

```

noisy_circle_points

noisy_circle_ripserr

noisy_circle_tda_rips

```

**Format**

**noisy\_circle\_points:**

A matrix with 100 rows and 2 columns listing the coordinates of the points.

**noisy\_circle\_ripserr:**

An object of class 'PHom' as returned by the `ripserr::vietoris_rips()` function, which is a data frame with 3 variables:

- dimension: the dimension/degree of the feature,
- birth: the birth value of the feature,
- death: the death value of the feature.

**noisy\_circle\_tda\_rips:**

A list of length 1 containing an object of class 'diagram' as returned by the `TDA::ripsDiag()` function, which is a matrix with 3 columns:

- dimension: the dimension/degree of the feature,
- birth: the birth value of the feature,
- death: the death value of the feature.

An object of class PHom (inherits from data.frame) with 101 rows and 3 columns.

An object of class list of length 1.

## Details

The point cloud stored in `noisy_circle_points` has been generated using the **tdaunif** package using the `tdaunif::sample_circle()` function. Specifically, the following parameters were used: `n = 100`, `sd = 0.05` and a seed of 1234.

The persistence diagram stored in `noisy_circle_ripserr` has been computed using the **ripserr** package with the `ripserr::vietoris_rips()` function. Specifically, the following parameters were used: `max_dim = 1L`.

The persistence diagram stored in `noisy_circle_tda_rips` has been computed using the **TDA** package with the `TDA::ripsDiag()` function. Specifically, the following parameters were used: `maxdimension = 1L` and `maxscale = 1.6322`.

## Source

<https://tdaverse.github.io/tdaunif/reference/circles.html>, [https://tdaverse.github.io/ripserr/reference/vietoris\\_rips.html](https://tdaverse.github.io/ripserr/reference/vietoris_rips.html), <https://www.rdocumentation.org/packages/TDA/versions/1.9.1/topics/ripsDiag>

---

pairwise-distances	<i>Pairwise distances within a set of persistence diagrams</i>
--------------------	--

---

## Description

This collection of functions computes the pairwise distance matrix between all pairs in a set of persistence diagrams of the same homology dimension. The diagrams must be represented as 2-column matrices. The first column of the matrix contains the birth times and the second column contains the death times of the points.

## Usage

```
bottleneck_pairwise_distances(
  x,
  tol = sqrt(.Machine$double.eps),
  validate = TRUE,
  dimension = 0L,
  ncores = 1L
)

wasserstein_pairwise_distances(
  x,
  tol = sqrt(.Machine$double.eps),
  p = 1,
  validate = TRUE,
  dimension = 0L,
  ncores = 1L
)
```

```
kantorovich_pairwise_distances(
  x,
  tol = sqrt(.Machine$double.eps),
  p = 1,
  validate = TRUE,
  dimension = 0L,
  ncores = 1L
)
```

### Arguments

x	A list of either 2-column matrices or objects of class <code>persistence</code> specifying the set of persistence diagrams.
tol	A numeric value specifying the relative error. Defaults to <code>sqrt(.Machine\$double.eps)</code> . For the Bottleneck distance, it can be set to <code>0.0</code> in which case the exact Bottleneck distance is computed, while an approximate Bottleneck distance is computed if <code>tol &gt; 0.0</code> . For the Wasserstein distance, it must be strictly positive.
validate	A boolean value specifying whether to validate the input persistence diagrams. Defaults to <code>TRUE</code> . If <code>FALSE</code> , the function will not check if the input persistence diagrams are valid. This can be useful for performance reasons, but it is recommended to keep it <code>TRUE</code> for safety.
dimension	An integer value specifying the homology dimension for which to compute the distance. Defaults to <code>0L</code> . This is only used if <code>x</code> and <code>y</code> are objects of class <code>persistence</code> .
ncores	An integer value specifying the number of cores to use for parallel computation. Defaults to <code>1L</code> .
p	A numeric value specifying the power for the Wasserstein distance. Defaults to <code>1.0</code> .

### Value

An object of class 'dist' containing the pairwise distance matrix between the persistence diagrams.

### Examples

```
spl <- persistence_sample[1:10]

# Extract the list of 2-column matrices for dimension 0 in the sample
x <- lapply(spl[1:10], function(x) x$pairs[[1]])

# Compute the pairwise Bottleneck distances
Db <- bottleneck_pairwise_distances(spl)
Db <- bottleneck_pairwise_distances(x)

# Compute the pairwise Wasserstein distances
Dw <- wasserstein_pairwise_distances(spl)
Dw <- wasserstein_pairwise_distances(x)
```

---

persistence

An S3 class for storing persistence data

---

## Description

A collection of functions to coerce persistence data into objects of class `persistence` (See **Value** section for more details on this class). It is currently possible to coerce persistence data from the following sources:

- a matrix with at least 3 columns (dimension/degree, start/birth, end/death) as returned by `ripserr::vietoris_rips()` in the form of the 'PHom' class,
- a list as returned by any `*Diag()` function in the **TDA** package.

## Usage

```
as_persistence(x, warn = TRUE, ...)

## S3 method for class 'list'
as_persistence(x, warn = TRUE, ...)

## S3 method for class 'persistence'
as_persistence(x, warn = TRUE, ...)

## S3 method for class 'data.frame'
as_persistence(x, warn = TRUE, ...)

## S3 method for class 'matrix'
as_persistence(x, warn = TRUE, ...)

## S3 method for class 'diagram'
as_persistence(x, warn = TRUE, ...)

## S3 method for class 'PHom'
as_persistence(x, ...)

## S3 method for class 'hclust'
as_persistence(x, warn = TRUE, birth = NULL, ...)

## S3 method for class 'persistence'
print(x, ...)

## S3 method for class 'persistence'
format(x, ...)

get_pairs(x, dimension, ...)

## S3 method for class 'persistence'
```



```
as.matrix(x, ...)

## S3 method for class 'persistence'
as.data.frame(x, row.names = NULL, optional = TRUE, ...)
```

## Arguments

x	<p>An R object containing the persistence data to be coerced into an object of class <code>persistence</code>. Currently supported forms are:</p> <ul style="list-style-type: none"> <li>• a <math>\geq 2</math>-column matrix (or object coercible to one) with dimension/degree, start/birth and end/death columns; if it has only 2 columns, we assume that the dimension column is missing and we set it to 0 (i.e. we assume that the data is in the form birth and death),</li> <li>• a <code>base::data.frame</code> (or object coercible to one) with at least 3 columns containing the persistence data; if it has only 2 columns, we assume that the dimension column is missing and we set it to 0,</li> <li>• a list of 2-column matrices (or objects coercible to one) with the first column being the birth and the second column being the death of homological features; indexed by dimension, i.e. the <math>i</math>-th element of the list corresponds to the <math>(i - 1)</math>-th homology dimension,</li> <li>• an object of class 'PHom' as returned by <code>ripserr::vietoris_rips()</code>,</li> <li>• (a list as returned by a <code>*Diag()</code> function in <b>TDA</b> (e.g. <code>TDA::ripsDiag()</code>) whose first element is) an object of class 'diagram',</li> <li>• an object of class <code>stats::hclust</code> in which case we use the entry height as the death of homological features and 0 as the birth of all features.</li> </ul>
warn	A boolean specifying whether to issue a warning if the input persistence data contained unordered pairs. Defaults to TRUE.
...	Parameters passed to methods.
birth	A numeric value specifying the height at which to declare all leaves were born. Defaults to 0 if all heights are non-negative and $-\text{Inf}$ otherwise.
dimension	A non-negative integer specifying the homology dimension for which to recover a matrix of persistence pairs.
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If TRUE, setting row names and converting column names (to syntactic names: see <code>make.names</code> ) is optional. Note that all of R's <b>base</b> package <code>as.data.frame()</code> methods use <code>optional</code> only for column names treatment, basically with the meaning of <code>data.frame(*, check.names = !optional)</code> . See also the <code>make.names</code> argument of the <code>matrix</code> method.

## Details

**Caution.** When providing an *unnamed* input matrix, the matrix coercer assumes that it has at least 3 columns, with the first column being the dimension/degree, the second column being the start/birth and the third column being the end/death.

**Value**

An object of class `persistence` which is a list of 2 elements:

- `pairs`: A list of 2-column matrices containing birth-death pairs. The  $i$ -th element of the list corresponds to the  $(i - 1)$ -th homology dimension. If there is no pairs for a given dimension but there are pairs in higher dimensions, the corresponding element(s) is/are filled with a  $0 \times 2$  numeric matrix.
- `metadata`: A list of length 6 containing information about how the data was computed:
  - `ordered_pairs`: A boolean indicating whether the pairs in the `pairs` list are ordered (i.e. the first column is strictly less than the second column).
  - `data`: The name of the object containing the original data on which the persistence data was computed.
  - `engine`: The name of the package and the function of this package that computed the persistence data in the form "package\_name::package\_function".
  - `filtration`: The filtration used in the computation in a human-readable format (i.e. full names, capitals where need, etc.).
  - `parameters`: A list of parameters used in the computation.
  - `call`: The exact call that generated the persistence data.

**Examples**

```
as_persistence(noisy_circle_ripserr)

x <- as_persistence(noisy_circle_tda_rips)
x

as_persistence(x)

get_pairs(x, dimension = 1)

as.data.frame(x)

# distances between cities
euroclust <- hclust(eurodist, method = "ward.D")
as_persistence(euroclust)

# `hclust()` can accommodate negative distances
d <- as.dist(rbind(c(0, 3, -4), c(3, 0, 5), c(-4, 5, 0)))
hc <- hclust(d, method = "single")
ph <- as_persistence(hc, birth = -10)
get_pairs(ph, 0)
```

---

persistence-set

An 'S3' class object for storing sets of persistence diagrams

---

**Description**

An 'S3' class object for storing sets of persistence diagrams

**Usage**

```
as_persistence_set(x)

## S3 method for class 'persistence_set'
format(x, ...)

## S3 method for class 'persistence_set'
print(x, ...)
```

**Arguments**

`x`                      A list of objects of class [persistence](#).

`...`                    Additional arguments passed to the function.

**Value**

An object of class 'persistence\_set' containing the set of persistence diagrams.

**Examples**

```
# Create a persistence set from a list of persistence diagrams
as_persistence_set(persistence_sample[1:10])
```

---

persistence_sample	<i>Toy Data: A sample of persistence diagrams</i>
--------------------	---

---

**Description**

A collection of 100 samples of size 100 on the sphere from which a persistence diagram is computed using the [TDA::ripsDiag\(\)](#) function with parameters `maxdimension = 1L` and `maxscale = 1.6322`. Each diagram has been generated using the [tdaunif::sample\\_2sphere\(\)](#) function with the following parameters: `n = 100` and `sd = 0.05`. The seed was fixed to 1234.

**Usage**

```
persistence_sample
```

**Format**

A list of length 100, where each element is an object of class 'persistence'.

---

`trefoils`*A sample of persistence diagrams from the trefoil*

---

**Description**

A collection of 24 samples of size 120 on the trefoil from which a persistence diagram is computed using the `TDA::ripsDiag()` function with `maxdimension = 2` and `maxscale = 6`. Each diagram has been generated using the `tdaunif::sample_trefoil()` function with the following parameters: `n = 120` and `sd = 0.05`. The seed was fixed to 28415.

**Usage**`trefoils`**Format**

A list of length 24, where each element is an object of class 'persistence'.

# Index

- \* **datasets**
  - arch\_spirals, [2](#)
  - noisy\_circle, [5](#)
  - persistence\_sample, [11](#)
  - trefoils, [12](#)
- arch\_spirals, [2](#)
- as.data.frame.persistence
  - (persistence), [8](#)
- as.matrix.persistence (persistence), [8](#)
- as\_persistence (persistence), [8](#)
- as\_persistence\_set (persistence-set), [10](#)
- base::data.frame, [9](#)
- bottleneck\_distance (distances), [3](#)
- bottleneck\_pairwise\_distances
  - (pairwise-distances), [6](#)
- data.frame, [9](#)
- distances, [3](#)
- format.persistence (persistence), [8](#)
- format.persistence\_set
  - (persistence-set), [10](#)
- get\_pairs (persistence), [8](#)
- kantorovich\_distance (distances), [3](#)
- kantorovich\_pairwise\_distances
  - (pairwise-distances), [6](#)
- make.names, [9](#)
- noisy\_circle, [5](#)
- noisy\_circle\_points (noisy\_circle), [5](#)
- noisy\_circle\_ripserr (noisy\_circle), [5](#)
- noisy\_circle\_tda\_rips (noisy\_circle), [5](#)
- pairwise-distances, [6](#)
- persistence, [3](#), [4](#), [7](#), [8](#), [8](#), [9–11](#)
- persistence-set, [10](#)
- persistence\_sample, [11](#)
- print.persistence (persistence), [8](#)
- print.persistence\_set
  - (persistence-set), [10](#)
- stats::hclust, [9](#)
- TDA::ripsDiag(), [2](#), [11](#), [12](#)
- tdaunif::sample\_2sphere(), [11](#)
- tdaunif::sample\_arch\_spiral(), [2](#)
- tdaunif::sample\_trefoil(), [12](#)
- trefoils, [12](#)
- wasserstein\_distance (distances), [3](#)
- wasserstein\_pairwise\_distances
  - (pairwise-distances), [6](#)