Tutorial on the R package TDA

Jisu Kim
Brittany T. Fasy, Jisu Kim, Fabrizio Lecci, Clément Maria, David L. Millman, Vincent Rouvreau

Abstract

This tutorial gives an introduction to the R package **TDA**, which provides some tools for Topological Data Analysis. The salient topological features of data can be quantified with persistent homology. The R package **TDA** provide an R interface for the efficient algorithms of the C++ libraries **GUDHI**, **Dionysus**, and **PHAT**. Specifically, The R package **TDA** includes functions for computing the persistent homology of the Rips complex, alpha complex, and alpha shape complex, and a function for the persistent homology of sublevel sets (or superlevel sets) of arbitrary functions evaluated over a grid of points. The R package **TDA** also provides a function for computing the confidence band that determines the significance of the features in the resulting persistence diagrams.

Keywords: Topological Data Analysis, Persistent Homology.

1. Introduction

R(http://cran.r-project.org/) is a programming language for statistical computing and graphics.

R has several good properties: R has many packages for statistical computing. Also, R is easy to make (interactive) plots. R is a script language, and it is easy to use. But, R is slow. C or C++ stands on the opposite end: C or C++ also has many packages(or libraries). But, C or C++ is difficult to make plots. C or C++ is a compiler language, and is difficult to use. But, C or C++ is fast. In short, R has short development time but long execution time, and C or C++ has long development time but short execution time.

Several libraries are developed for Topological Data Analysis: for example, **GUDHI**(https://project.inria.fr/gudhi/software/), **Dionysus**(http://www.mrzv.org/software/dionysus/), and **PHAT**(https://code.google.com/p/phat/). They are all written in C++, since Topological Data Analysis is computationally heavy and R is not fast enough.

R package **TDA**(http://cran.r-project.org/web/packages/TDA/index.html) bridges between C++ libraries(**GUDHI**, **Dionysus**, **PHAT**) and R. **TDA** package provides an R interface for the efficient algorithms of the C++ libraries **GUDHI**, **Dionysus** and **PHAT**. So by using **TDA** package, short development time and short execution time can be both achieved.

R package **TDA** provides tools for Topological Data Analysis. You can compute several different things with **TDA** package: you can compute common distance functions and density estimators, the persistent homology of the Rips filtration, the persistent homology of sublevel sets of a function over a grid, the confidence band for the persistence diagram, and the cluster density trees for density clustering.

2. Installation

First, you should download R. R of version at least 3.1.0 is required:

http://cran.r-project.org/bin/windows/base/ (for Windows)

http://cran.r-project.org/bin/macosx/ (for (Mac) OS X)

R is part of many Linux distributions, so you should check with your Linux package management system.

You can use whatever IDE that you would like to use(Rstudio, Eclipse, Emacs, Vim...). R itself also provides basic GUI or CUI. I personally use Rstudio:

http://www.rstudio.com/products/rstudio/download/

For Windows and Mac, you can install R package **TDA** as in the following code (or pushing 'Install R packages' button if you use Rstudio).

```
#############################################################################
# installing R package TDA
#############################################################################
if (!require(package = "TDA")) {
  install.packages(pkgs = "TDA")
}

## Loading required package:  TDA
```

If you are using Linux, you should install R package **TDA** from the source. To do this, you need to install two libraries in advance: gmp (https://gmplib.org/) and mpfr (http://www.mpfr.org/). Installation of these packages may differ by your Linux distributions. Once those libraries are installed, you need to install four R packages: **parallel**, **FNN**, **igraph**, and **scales**. **parallel** is included when you install R, so you need to install **FNN**, **igraph**, and **scales** by yourself. You can install them by following code (or pushing 'Install R packages' button if you use Rstudio).

```
#############################################################################
# installing required packages
#############################################################################
if (!require(package = "FNN")) {
  install.packages(pkgs = "FNN")
}

## Loading required package:  FNN

if (!require(package = "igraph")) {
  install.packages(pkgs = "igraph")
}

## Loading required package:  igraph
##
## Attaching package:  'igraph'
## The following object is masked from 'package:FNN':
##
##     knn
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
## The following object is masked from 'package:base':
##
##     union
```

```
if (!require(package = "scales")) {
  install.packages(pkgs = "scales")
}

## Loading required package:  scales
```

Then you can install the R package **TDA** as in Windows or Mac:

```
#############################################################################
# installing R package TDA
#############################################################################
if (!require(package = "TDA")) {
  install.packages(pkgs = "TDA")
}
```

Once installation is done, R package **TDA** should be loaded as in the following code, before using the package functions.

```
#############################################################################
# loading R package TDA
#############################################################################
library(package = "TDA")
```

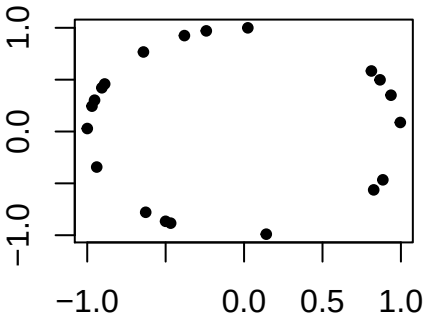
3. Sample on manifolds, Distance Functions, and Density Estimators

3.1. Uniform Sample on manifolds

A set of n points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ has been sampled from some distribution P.

- n sample from the uniform distribution on the circle in $\mathbb{R}^2$ with radius r.

```
#############################################################################
# uniform sample on the circle
#############################################################################
circleSample <- circleUnif(n = 20, r = 1)
plot(circleSample, xlab = "", ylab = "", pch = 20)
```



3.2. Distance Functions and Density Estimators

We compute distance functions and density estimators over a grid of points. Suppose a set of points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ has been sampled from some distribution P. The following code generates a sample of 400 points from the unit circle and constructs a grid of points over which we will evaluate the functions.

```
##############################################################################
# uniform sample on the circle, and grid of points
##############################################################################
X <- circleUnif(n = 400, r = 1)

lim <- c(-1.7, 1.7)
by <- 0.05
margin <- seq(from = lim[1], to = lim[2], by = by)
Grid <- expand.grid(margin, margin)
```

- Given a probability measure P, the distance to measure (DTM) is defined for each $y \in \mathbb{R}^d$ as

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

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 default value of `r` is 2. The DTM can be seen as a smoothed version of the distance function. See (Chazal, Cohen-Steiner, and Mérigot 2011, Definition 3.2) and (Chazal, Massart, and Michel 2015, Equation (2)) for a formal definition of the "distance to measure" function.

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

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

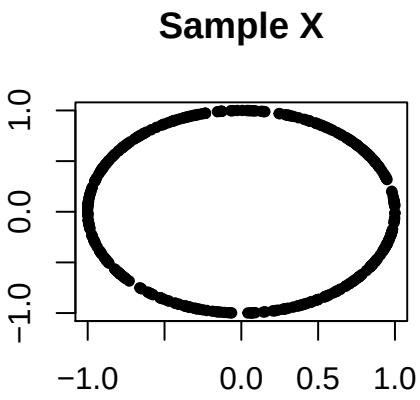
where $k = \lceil m0 * n \rceil$ and $N_k(y)$ is the set containing the k nearest neighbors of y among $x_1, \ldots, x_n$.

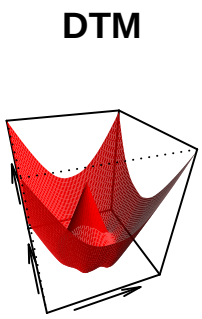
For more details, see (Chazal *et al.* 2011) and (Chazal *et al.* 2015).

The DTM is computed for each point of the `Grid` with the following code:

```
###############################################################################
# distance to measure
###############################################################################
m0 <- 0.1
DTM <- dtm(X = X, Grid = Grid, m0 = m0)

par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X", pch = 20)
persp(x = margin, y = margin,
      z = matrix(DTM, nrow = length(margin), ncol = length(margin)),
      xlab = "", ylab = "", zlab = "", theta = -20, phi = 35, scale = FALSE,
      expand = 3, col = "red", border = NA, ltheta = 50, shade = 0.5,
      main = "DTM")
```





- The Gaussian Kernel Density Estimator (KDE), for each $y \in \mathbb{R}^d$, is defined as

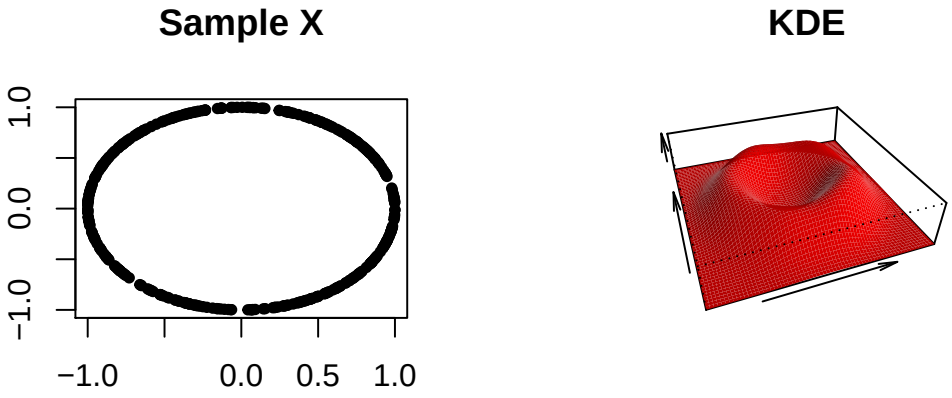
$$\hat{p}_h(y) = \frac{1}{n(\sqrt{2\pi}h)^d} \sum_{i=1}^{n} \exp\left(\frac{-\|y - x_i\|_2^2}{2h^2}\right).$$

where h is a smoothing parameter.

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

par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X", pch = 20)
```

```
persp(x = margin, y = margin,
      z = matrix(KDE, nrow = length(margin), ncol = length(margin)),
      xlab = "", ylab = "", zlab = "", theta = -20, phi = 35, scale = FALSE,
      expand = 3, col = "red", border = NA, ltheta = 50, shade = 0.5,
      main = "KDE")
```



4. Persistent Homology

4.1. Persistent Homology Over a Grid

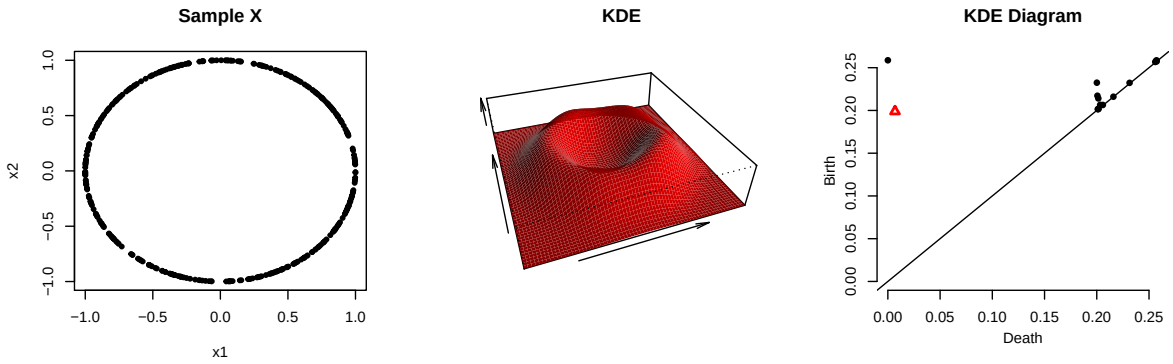
`gridDiag` function computes the persistent homology of sublevel (and superlevel) sets of the functions. The function `gridDiag` evaluates a given real valued function over a triangulated grid (in arbitrary dimension), constructs a filtration of simplices using the values of the function, and computes the persistent homology of the filtration. The user can choose to compute persistence diagrams using either the C++ library **GUDHI** (`library = "GUDHI"`), **Dionysus** (`library = "Dionysus"`), or **PHAT** (`library = "PHAT"`) .

The following code computes the persistent homology of the superlevel sets (`sublevel = FALSE`) of the kernel density estimator (`FUN = kde`, `h = 0.3`) using the point cloud stored in the matrix `X` from the previous example. The other inputs are the features of the grid over which the `kde` is evaluated (`lim` and `by`), and a logical variable that indicates whether a progress bar should be printed (`printProgress`).

```
#########################################################################
# persistent homology of a function over a grid
#########################################################################
DiagGrid <- gridDiag(X = X, FUN = kde, lim = cbind(lim, lim), by = by,
    sublevel = FALSE, library = "Dionysus", printProgress = FALSE, h = 0.3)
```

The function `plot` plots persistence diagram for objects of the class `"diagram"`.

```
#########################################################################
# plotting persistence diagram
#########################################################################
par(mfrow = c(1,3))
plot(X, main = "Sample X", pch = 20)
persp(x = margin, y = margin,
      z = matrix(KDE, nrow = length(margin), ncol = length(margin)),
      xlab = "", ylab = "", zlab = "", theta = -20, phi = 35, scale = FALSE,
      expand = 3, col = "red", border = NA, ltheta = 50, shade = 0.9,
      main = "KDE")
plot(x = DiagGrid[["diagram"]], main = "KDE Diagram")
```



4.2. Rips Persistent Homology

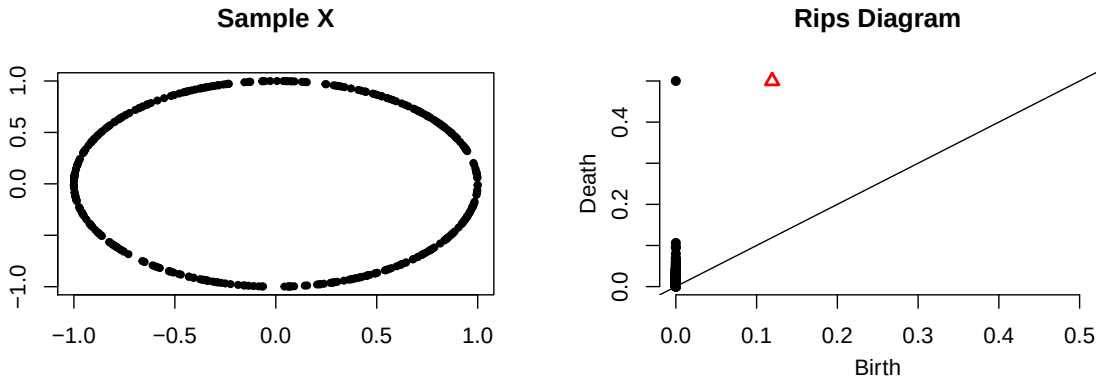
The *Vietoris-Rips complex* $R(X, \varepsilon)$ consists of simplices with vertices in $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and diameter at most ε. In other words, a simplex σ is included in the complex if each pair of vertices in σ is at most ε apart. The sequence of Rips complexes obtained by gradually increasing the radius ε creates a filtration.

The `ripsDiag` function computes the persistence diagram of the Rips filtration built on top of a point cloud. The user can choose to compute the Rips filtration using either the C++ library **GUDHI** or **Dionysus**. Then for computing the persistence diagram from the Rips filtration, the user can use either the C++ library **GUDHI**, **Dionysus**, or **PHAT**.

The following code computes the persistent homology of the Rips filtration using the point cloud stored in the matrix `X` from the previous example, and the plot the data and the diagram.

```
#########################################################################
# rips persistence diagram
#########################################################################
DiagRips <- ripsDiag(X = X, maxdimension = 1, maxscale = 0.5,
    library = c("GUDHI", "Dionysus"), location = TRUE)

#########################################################################
# plotting persistence diagram
#########################################################################
par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X", pch = 20)
plot(x = DiagRips[["diagram"]], main = "Rips Diagram")
```



4.3. Landscapes

The Persistence landscape is a real-valued function that further summarizes the information contained in a persistence diagram. It has been introduced and studied in Bubenik (2012), Chazal, Fasy, Lecci, Rinaldo, and Wasserman (2014b), and Chazal, Fasy, Lecci, Michel, Rinaldo, and Wasserman (2014a). The persistence landscape is a collection of continuous, piecewise linear functions $\lambda\colon \mathbb{Z}^+ \times \mathbb{R} \to \mathbb{R}$ that summarizes a persistence diagram. To define the landscape, consider the set of functions created by tenting each each point $p = (x, y) = \left(\frac{b+d}{2}, \frac{d-b}{2}\right)$ representing a birth-death pair (b, d) in the persistence diagram D as follows:

$$\Lambda_p(t) = \begin{cases} t - x + y & t \in [x - y, x] \\ x + y - t & t \in (x, x + y] \\ 0 & \text{otherwise} \end{cases} = \begin{cases} t - b & t \in [b, \frac{b+d}{2}] \\ d - t & t \in (\frac{b+d}{2}, d] \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We obtain an arrangement of piecewise linear curves by overlaying the graphs of the functions $\{\Lambda_p\}_p$; see Figure 1 (left). The persistence landscape of D is a summary of this arrangement. Formally, the persistence landscape of D is the collection of functions

$$\lambda(k, t) = \underset{p}{k\text{max}}\ \Lambda_p(t), \quad t \in [0, T], k \in \mathbb{N}, \tag{2}$$

where $k$max is the kth largest value in the set; in particular, 1max is the usual maximum function. see Figure 1 (right).

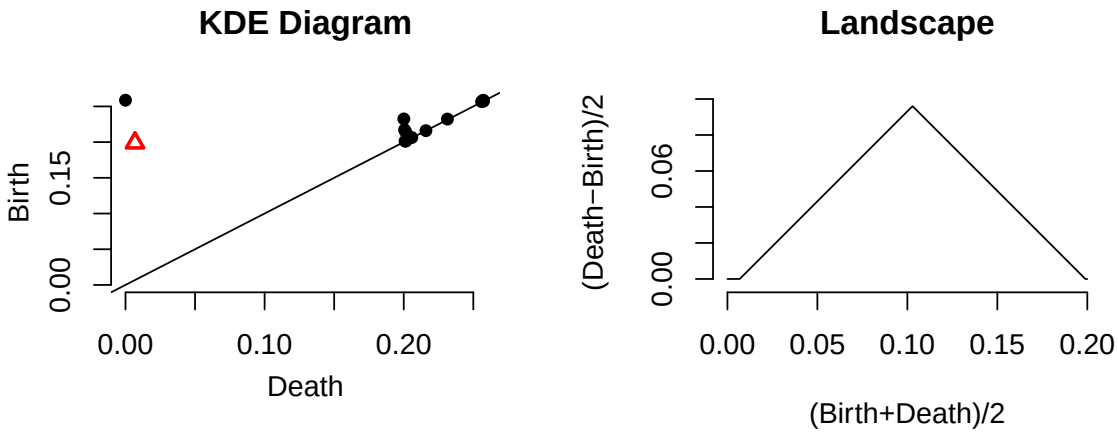
The landscape function can be evaluated over a one-dimensional grid of points `tseq` using the function `landscape`. In the following code, we use the persistence diagram of KDE to construct the corresponding landscape for one-dimensional features (`dimension = 1`). The option (`KK = 1`) specifies that we are interested in the 1st landscape function. The functions `landscape` return real valued vectors, which can be simply plotted with `plot(tseq, Land, type = "l")`.

```
#############################################################################
# computing landscape function
#############################################################################
tseq <- seq(0, 0.2, length = 1000)
Land <- landscape(DiagGrid[["diagram"]], dimension = 1, KK = 1, tseq = tseq)

#############################################################################
```

Figure 1: Left: we use the rotated axes to represent a persistence diagram D. A feature $(b, d) \in D$ is represented by the point $(\frac{b+d}{2}, \frac{d-b}{2})$ (pink). In words, the x-coordinate is the average parameter value over which the feature exists, and the y-coordinate is the half-life of the feature. Right: the blue curve is the landscape $\lambda(1, \cdot)$.

```
# plotting landscape function
################################################################################
par(mfrow = c(1,2))
plot(x = DiagGrid[["diagram"]], main = "KDE Diagram")
plot(tseq, Land, type = "l", xlab = "(Birth+Death)/2",
     ylab = "(Death-Birth)/2", asp = 1, axes = FALSE, main = "Landscape")
axis(1); axis(2)
```



5. Statistical Inference on Persistent Homology

$(1 - \alpha)$ confidence band can be computed for a function using the bootstrap algorithm, which we briefly describe using the kernel density estimator:

1. Given a sample $X = \{x_1, \ldots, x_n\}$, compute the kernel density estimator $\hat{p}_h$;
2. Draw $X^* = \{x_1^*, \ldots, x_n^*\}$ from $X = \{x_1, \ldots, x_n\}$ (with replacement), and compute $\theta^* = \sqrt{n}\|\hat{p}_h^*(x) - \hat{p}_h(x)\|_\infty$, where $\hat{p}_h^*$ is the density estimator computed using X^*;

3. Repeat the previous step B times to obtain $\theta_1^*, \ldots, \theta_B^*$;
4. Compute $q_\alpha = \inf\left\{q : \frac{1}{B}\sum_{j=1}^{B} I(\theta_j^* \geq q) \leq \alpha\right\}$;
5. The $(1-\alpha)$ confidence band for $\mathbb{E}[\hat{p}_h]$ is $\left[\hat{p}_h - \frac{q_\alpha}{\sqrt{n}}, \hat{p}_h + \frac{q_\alpha}{\sqrt{n}}\right]$.

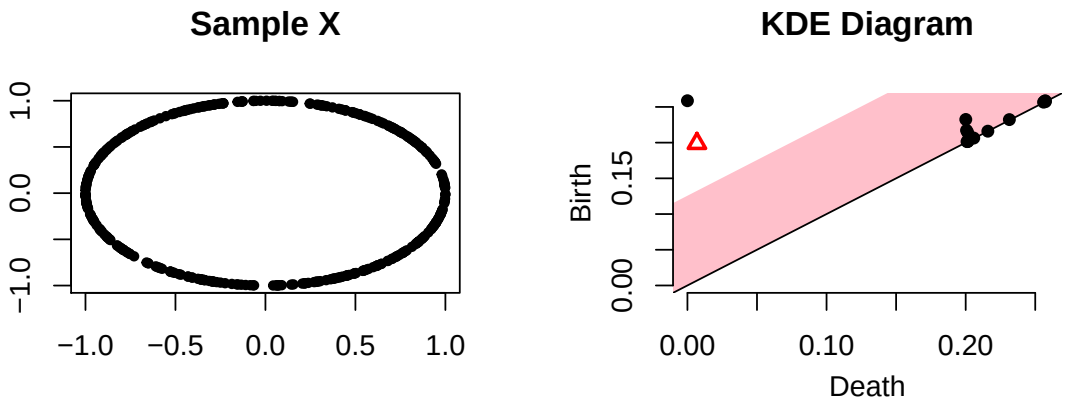
`bootstrapBand` computes $(1-\alpha)$ bootstrap confidence band, with the option of parallelizing the algorithm (`parallel=TRUE`). The following code computes a 90% confidence band for $\mathbb{E}[\hat{p}_h]$.

```
##########################################################################
# bootstrap confidence band for kde function
##########################################################################
bandKDE <- bootstrapBand(X = X, FUN = kde, Grid = Grid, B = 100,
                         parallel = FALSE, alpha = 0.1, h = h)
print(bandKDE[["width"]])

##         90%
## 0.06283067
```

Then such confidence band for $\mathbb{E}[\hat{p}_h]$ can be used as the confidence band for the persistent homology.

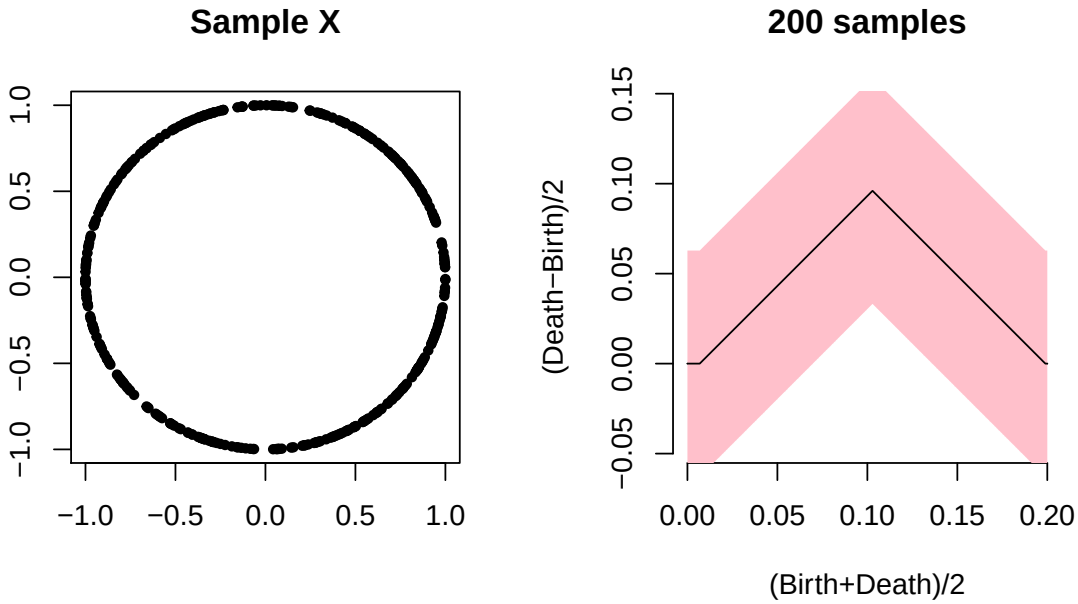
```
##########################################################################
# bootstrap confidence band for persistent homology over a grid
##########################################################################
par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X", pch = 20)
plot(x = DiagGrid[["diagram"]], band = 2 * bandKDE[["width"]],
     main = "KDE Diagram")
```



And the same confidence band for $\mathbb{E}[\hat{p}_h]$ can be used as the confidence band for the landscape function as well.

```
##########################################################################
# bootstrap confidence band for landscape
##########################################################################
```

```
par(mfrow = c(1,2))
plot(X, xlab = "", ylab = "", main = "Sample X", pch = 20)
plot(tseq, Land, type = "l", xlab = "(Birth+Death)/2",
     ylab = "(Death-Birth)/2", asp = 1, axes = FALSE, main = "200 samples")
axis(1); axis(2)
polygon(c(tseq, rev(tseq)), c(Land - bandKDE[["width"]],
        rev(Land + bandKDE[["width"]])), col = "pink", lwd = 1.5,
        border = NA)
lines(tseq, Land)
```



References

Bubenik P (2012). "Statistical topological data analysis using persistence landscapes." *arXiv preprint arXiv:1207.6437*.

Chazal F, Cohen-Steiner D, Mérigot Q (2011). "Geometric inference for probability measures." *Foundations of Computational Mathematics*, **11**(6), 733–751.

Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014a). "Subsampling Methods for Persistent Homology." *arXiv preprint arXiv:1406.1901*.

Chazal F, Fasy BT, Lecci F, Rinaldo A, Wasserman L (2014b). "Stochastic Convergence of Persistence Landscapes and Silhouettes." In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pp. 474:474–474:483. ACM, New York, NY, USA. ISBN 978-1-4503-2594-3. doi:10.1145/2582112.2582128. URL http://doi.acm.org/10.1145/2582112.2582128.

Chazal F, Massart P, Michel B (2015). "Rates of convergence for robust geometric inference." *CoRR*, **abs/1505.07602**. URL http://arxiv.org/abs/1505.07602.

Affiliation:

Firstname Lastname
Affiliation
Address, Country
E-mail: name@address
URL: http://link/to/webpage/