P-WASSERSTEIN DISTANCE BETWEEN HIGH-DIMENSIONAL POINT CLOUDS VIA PERSISTENT HOMOLOGY

ZHANG LIU

FINAL REPORT FOR

MA3288 ADVANCED UROPS IN MATHEMATICS I & II

SUPERVISED BY: HAN FEI

AY 2021/2022

Acknowledgements

I would like to thank my mentor Prof. Han Fei for his continual support throughout the project. There have been ups and downs in this project, but Prof. Han Fei has helped me stay flexible and keep a keen spirit for learning and discovery. I am also grateful for support from my family and friends.

NATIONAL UNIVERSITY OF SINGAPORE

Abstract

B.Sc (Hons)

p-Wasserstein Distance between High-dimensional Point Clouds via Persistent Homology

by ZHANG Liu

Key words: applied topology, persistent homology, Wasserstein distance, dimensionality reduction, diffusion map

Topological Data Analysis (TDA) is a recent field that emerged from various works in applied (algebraic) topology and computational geometry. The aim of TDA is to extract and analyze relevant topological and geometric features of real-world data, which are often high-dimensional and have complex underlying geometric structures.

Our project is motivated by the question: how can high-dimensional point clouds be compared based on their topological structures? To this end, we proposed an approach to compare point clouds based on the p-Wasserstein distance of their topological fingerprints. We illustrated the approach with a concrete application in studying the neural population responses to different stimuli, each of which is represented as a high-dimensional point cloud. Our approach allowed us to quantify the pairwise similarity of the neural population responses in terms of their topological structures. Using this approach, we also compared the topological structure of the point cloud associated with each neural population response with that of synthetic data sampled from 2-sphere and torus.

Contents

Chapter 1

Introduction

1.1 Background

Topological Data Analysis (TDA) is a recent field that emerged from various works in applied (algebraic) topology and computational geometry during the first decade of the century. It started as a field with the foundational works of (Edelsbrunner, Letscher, and Zomorodian, 2002) and (Zomorodian, 2005b) and was popularized by (Carlsson, 2009). The aim of TDA is to extract, analyze and make use of the topological and geometric structures underlying data. These data are usually represented as point clouds in Euclidean or more general metric spaces and are often high-dimensional. (Chazal and Michel, 2021)

1.2 Aim and Significance

Our project is motivated by the following question: how can high-dimensional point clouds be compared based on their topological structures? To this

end, we studied the theory of persistent homology, dimensionality reduction, and various distance metrics useful for high-dimensional data analysis. Further, we proposed an approach to compare point clouds based on the p-Wasserstein distance of their topological fingerprints. We illustrated the approach with a concrete application using neural spiking data from lab experiments. The code for this project can be found in the GitHub repository [1] and Appendix A.

The topological methods used in this project provide a new perspective to compare data point clouds as shapes. Topological features capture the global structure of the data, which is especially important in applications where the notion of connectedness and clusters are salient.

1.3 Prior works applying topological methods in neuroscience

The first work that applied topological methods in neuroscience is (Singh et al., 2008). The authors applied computational topology to analyze neural population activity in the primary visual cortex and concluded that the topological structure of neural population activity is align with the topological structure of a 2-sphere both when the cortex is spontaneously active and when evoked by natural images.

Following that, there have been many applications of persistent homology in studying the topological structure of neural population activity across different systems. A notable application is (Chaudhuri et al., 2019), where the authors applied persistent homology to study the

[1] https://github.com/zhang-liu-official/UROPS-2021-Topological-Data-Analysis

the neural population activity from the post-subiculum and anterodorsal thalamus. They showed that the head direction of mice can be decoded from a one-dimensional ring structure. A recent addition in this line of work was by (Beshkov and Tiesinga, 2021). They showed that when using the geodesic distance instead of the Euclidean distance, persistent homology method was able to successfully identify highly non-linear features. They then provided an application using the Neuropixels dataset recorded in mouse visual cortex after stimulation with drifting gratings.

An alternative approach has been taken by several researchers to to use spike train correlation measures to define the “distance" between two neurons by how similar the firing rates of two neurons are given a stimuli. This was computed this from the neural spiking data. With the spike train correlation measures, they then obtained the so-called neural clique topology. Two notable works are (Giusti et al., 2015) and (Bardin, Spreemann, and Hess, 2019).

Chapter 2

Persistent Homology

This chapter includes the necessary theoretical background of persistent homology. The main references for this chapter are (Carlsson, 2009) and (Zomorodian, 2005a).

2.1 Motivation

First of all, there are four major advantages for using topological methods to analyze high-dimensional point clouds.

1. Topology provides qualitative information which is required for data analysis.

2. Metrics are not theoretically justified. Compared to straightforward geometric methods, Topology is less sensitive to the actual choice of metrics.

3. Studying geometric objects using Topology does not depend on the coordinates.

4. Functoriality. This is the most important advantage.

Definition 1 (Functoriality)

For any topological space X, abelian group A, and integer $k \geqslant 0$, there is assigned a group $H_k(X, A)$. For any A and k, and any continuous map $f : X \to Y$, there is an induced homomorphism $H_k(f, A) : H_k(X, A) \to H_k(Y, A)$. Then functoriality refers to the following conditions:

- $H_k(f \circ g, A) : H_k(f, A) \circ H_k(g, A)$
- $H_k(Id_X; A) = Id_{H_k(X,A)}$.

Functoriality addresses the ambiguities in statistical clustering methods – in particular the arbitrariness of various threshold choices. We now illustrate how exactly functoriality could be used in questions related to clustering.

Let X be the full data set and X_1, X_2 are the subsamples from the data set. If the set of clusterings $C(X_1), C(X_2), C(X_1 \cup X_2)$ correspond well, then we can conclude that the subsample clusterings correspond to clusterings in the full data set X.

2.2 Homotopy

Definition 2 (Homotopy)

Homotopy is a family of maps $f_t : X \to Y$ where $t \in I$ such that $F : X \times I \to Y$ defined by $F(x, t) \mapsto f_t(x)$ is continuous.

Two maps f_0, f_1 are homotopic if there exists a homotopy f_t between f_0 and f_1.

Definition 3 (Homotopy equivalence)

A map $f : X \to Y$ is a homotopy equivalence if there is a map $g : Y \to X$ such that

- *$f \circ g$ is homotopic to the identity map on Y, and*
- *$g \circ f$ is homotopic to f.*

Two spaces X, Y are homotopy equivalent if there exists a homotopy equivalence $f : X \to Y$.

Theorem 4

If f and g are homotopic, then $H_k(f, A) = H_k(g, A)$.

If X and Y are homotopy equivalent, then $H_k(X, A) \cong H_k(Y, A)$.

Note that if two spaces are homotopy equivalent, then all their Betti numbers (defined in subsequent section) are equal.

Definition 5 (Topological space)

Associated to a simplicial complex $(V, \triangle)$ is a topological space $|(V, \triangle)|$. $|(V, \triangle)|$ may be defined using a bijection $\phi : V \to \{1, 2, \ldots, N\}$ as the subspace of $\mathbb{R}^N$ given by the union

$$\bigcup_{\sigma \in \triangle} c(\sigma),$$

where $c(\sigma)$ is the convex hull of the set $\{e_{\phi(s)}\}_{s \in \sigma}$, where e_i denotes the i-th standard basis vector.

We often use abstract simplicial complexes to approximate topological spaces. For simplicial complexes the homology can be computed using only the linear algebra of finitely generated $\mathbb{Z}$-modules. In particular, for simplicial complexes, homology is algorithmically computable.

Definition 6 (Nerve)

Let X be a topological space, and let $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ be any covering of X.

The nerve of $\mathcal{U}$, denoted by $N(\mathcal{U})$, will be the abstract simplicial complex with vertex set A, and where a family $\{\alpha_0, \ldots, \alpha_k\}$ spans a k-simplex if and only if $U_{\alpha_0} \cap U_{\alpha_1} \cap \cdots \cap U_{\alpha_k} \neq \emptyset$.

One reason that this construction is very useful in the following "Nerve Theorem." This theorem gives the criteria for $N(\mathcal{U})$ to be homotopy equivalent to the underlying topological space X.

Theorem 7 (Nerve Theorem)

Suppose that X and U are as above, and suppose that the covering consists of open sets and is numerable. Suppose further that for all $\emptyset \subseteq A$, we have that $\bigcap_{s\in S} U_s$ is either contractible or empty. Then $N(\mathcal{U})$ is homotopy equivalent to X.

The Nerve Theorem is very important in TDA since it provides a way to encode the topology of continuous spaces into abstract combinatorial structures that are more useful for designing computationally efficient data structures and algorithms.

2.3 Simplicial Complexes

Definition 8 (Simplicial complex)

An abstract simplicial complex is a pair $(V, \triangle)$, where V is a finite set, and $\triangle$ is a family of non-empty subsets of V such that

$$\tau \in \triangle \text{ and } \sigma \subseteq \tau \implies \sigma \in \triangle.$$

$\tau \in \triangle$ is face of $\triangle$. The dimension of a face τ is $|\tau| - 1$.

(Intuition) A simplicial complex $\triangle$ in $\mathbb{R}^n$ is a collection of simplices in $\mathbb{R}^n$ such that

1. *Every face of a simplex of $\triangle$ is in $\triangle$.*
2. *The intersection of any two simplicies of $\triangle$ is a face of each.*

Definition 9 (Maximal Face)

A face τ is a maximal face of $\triangle$ if there is no face σ of $\triangle$ such that $\tau \subsetneq \sigma$.

- *0-dimensional faces: vertices*
- *1-dimensional faces: edges*
- *A simplicial complex of dimension $D \leqslant 1$ is a simple and loopless graph.*

Definition 10 (k-skeleton)

A simplicial complex $\triangle_0$ is a subcomplex of $\triangle$ if $\triangle_0 \subseteq \triangle$.

For $k \geqslant -1$, the k-skeleton $\triangle^{(k)}$ of $\triangle$ is the subcomplex of $\triangle$ obtained by removing all faces of dimension higher than k.

Definition 11 (f-vector)

For each $n \geqslant -1$, let $f_n = f_n(\triangle)$ be the number of faces of $\triangle$ of dimension n.

The f-vector of $\triangle$ is the vector $(f_{-1}, f_0, f_1, \ldots, f_d)$ where d is the dimension of $\triangle$.

Example 12

Suppose the family of sets

$$\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a,b\}, \{a,c\}, \{b,c\}, \{b,d\}, \{c,d\}, \{a,b,c\}$$

form a simplicial complex E_1. Then the f-vector of E_1 is $(1,4,5,1)$.

Definition 13 (Oriented complex)

(Intuition) An oriented simplex is a simplex σ together with an orientation of σ. If $\{a_0, a_1, \ldots, a_p\}$ spans a p-complex σ, then we denote the oriented simplex with $[a_0, a_1, \ldots, a_p]$.

(Formal) Let $\mathbb{F}$ be a commutative ring, $\triangle$ be a simplicial complex. For each $n \geqslant -1$, we form a free $\mathbb{F}$-module $\widetilde{C}_n(\triangle; \mathbb{F})$ with a basis indexed by the n-dimensional faces of $\triangle$. For each n-dimensional face $a_0 a_1 \cdots a_n$, we have a basis element $\vec{e}_{a_0, a_1, \cdots, a_n}$.

We refer to the basis element $\vec{e}_{a_0,a_1,\cdots,a_n}$ as an oriented simplex. The concept of an oriented simplex is analogous to the idea of a unit vector (which gives the direction and has unit length).

Definition 14 (Chain group of degree *n*)

We refer to the free $\mathbb{F}$-module $\widetilde{C}_n(\triangle;\mathbb{F})$ in the previous definition as the chain group of degree n. The rank of $\widetilde{C}_n(\triangle;\mathbb{F})$ is the n-th value $f_n(\triangle)$ in the f-vector of $\triangle$.

Example 15

For $E_1 = \{\emptyset,\{a\},\{b\},\{c\},\{d\},\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{c,d\},\{a,b,c\}\}$, we have the following chain groups of degrees $-1,0,1,2$, respectively:

- $\widetilde{C}_{-1}(E_1) = \{\lambda e_\emptyset : \lambda \in \mathbb{F}\} \cong \mathbb{F}$.
- $\widetilde{C}_0(E_1) = \{\lambda_a e_a + \lambda_b e_b + \lambda_c e_c + \lambda_d e_d : \lambda_a,\lambda_b,\lambda_c,\lambda_d \in \mathbb{F}\} \cong \mathbb{F}^4$.
- $\widetilde{C}_1(E_1) = \{\lambda_{ab} e_{a,b} + \cdots + \lambda_{cd} e_{c,d} : \lambda_{ab},\ldots,\lambda_{cd} \in \mathbb{F}\} \cong \mathbb{F}^5$.
- $\widetilde{C}_2(E_1) = \{\lambda e_{a,b,c} : \lambda \in \mathbb{F}\} \cong \mathbb{F}$.

Definition 16 (Simplicial chain complex)

The simplicial chain complex of a simplicial comlex $\triangle$, denoted with $C(\triangle)$, is defined as:

$$\cdots \xrightarrow{\partial_{n+2}} \widetilde{C}_{n+1}(\triangle) \xrightarrow{\partial_{n+1}} \widetilde{C}_n(\triangle) \xrightarrow{\partial_n} \widetilde{C}_{n-1}(\triangle) \xrightarrow{\partial_{n-1}} \cdots$$

Example 17

The simplicial chain complex of E_1, $C(E_1)$ is

$$\begin{array}{ccccccccccc}
0 & \longrightarrow & \widetilde{C}_2(E_1) & \xrightarrow{\partial_2} & \widetilde{C}_1(E_1) & \xrightarrow{\partial_1} & \widetilde{C}_0(E_1) & \xrightarrow{\partial_0} & \widetilde{C}_{-1}(E_1) & \longrightarrow & 0 \\
 & & \downarrow\cong & & \downarrow\cong & & \downarrow\cong & & \downarrow\cong & & \\
0 & \longrightarrow & \mathbb{F} & \xrightarrow{\partial_2} & \mathbb{F}^5 & \xrightarrow{\partial_1} & \mathbb{F}^4 & \xrightarrow{\partial_0} & \mathbb{F} & \longrightarrow & 0
\end{array}$$

Recall that E_1 has f-vector $(1,4,5,1)$.

Proposition 18 (The double boundary condition)

Boundary maps satisfy the double boundary condition, that is, $\partial_n \circ \partial_{n+1} = 0 \quad \forall n$.

Definition 19 (Simplicial homology of degree k)

(Intuition) The simplicial homology gives an algebraic measure on the amount of cycles that are not the boundaries.

(Formal) Define he $\mathbb{F}$-module $Z_k(\triangle;\mathbb{F})$ of cycles and the $\mathbb{F}$-module of he boundary group $B_k(\triangle;\mathbb{F})$ by the following formulas:

- $Z_k(\triangle;\mathbb{F}) = ker(\partial_k) = \{Z \in \widetilde{C}_k(\triangle;\mathbb{F}) : \partial_k(Z) = 0\}$
- $B_k(\triangle;\mathbb{F}) = im(\partial_{k+1}) = \{Z \in \widetilde{C}_k(\triangle;\mathbb{F}) : \partial_{k+1}(x), \quad x \in \widetilde{C}_{k+1}(\triangle;\mathbb{F})\}$

By 18, $B_k(\triangle;\mathbb{F})$ is a submodule of $Z_k(\triangle;\mathbb{F})$, i.e., $B_k(\triangle;\mathbb{F}) \subseteq Z_k(\triangle;\mathbb{F}) \subseteq C_k(\triangle;\mathbb{F})$.

We define the simplicial homology in degree k of $\triangle$ to be the quotient group

$$H_k(\triangle;\mathbb{F}) = ker(\partial_k)/im(\partial_{k+1}) \tag{2.1}$$

$$= Z_k(\triangle;\mathbb{F})/B_k(\triangle;\mathbb{F}) \tag{2.2}$$

Definition 20 (k-th Betti number)

The k-th Betti number of the simplicial complex $\triangle$ is

$$\beta_k(\triangle;\mathbb{F}) \coloneqq \dim H_k(\triangle;\mathbb{F}) \tag{2.3}$$

$$= \dim ker(\partial_k) - \dim im(\partial_{k+1}). \tag{2.4}$$

- *elements in $ker(\partial_k)$ are called k-cycles.*
- *elements in $im(\partial_k)$ are called k-boundaries.*
- *k-cycles that are not boundaries represent k-holes, which means that the k-th Betti number $\beta_k(\triangle;\mathbb{F})$ is the number of k-holes.*

Definition 21 (Homology class)

Each member of $H_k(\triangle; \mathbb{F})$ is a homology class. Each such member is an equivalent class under the relation $z \sim z' \iff z - z' \in B_k(\triangle; \mathbb{F})$.

Let $[z]$ denote the homology class containing the cycle z.

Definition 22 (Homology as "holes")

In certain well-behaved cases, homology can be interpreted as an algebraic measure on the amount of "holes" in a simplicial complex $\triangle$.

Formally, given $X \subseteq \mathbb{R}^n$, a "hole" of X is a bounded connected component of $\mathbb{R}^n$ X. For a given $k \geqslant 0$, suppose the $(k+1)$-skeleton $\triangle^{(k+1)}$ of $\triangle$ has a geometric realization $X \subseteq \mathbb{R}^n$. Then $H_k(\triangle; \mathbb{F})$ is a free $\mathbb{F}$-module of rank equal to the number of holes of X.

Homology associates a vector space $H_k(X)$ to a topological space X for each $k \in \mathbb{N}$.

- *$H_0(X)$ is the number of path components in X.*
- *$H_1(X)$ is the number of holes in X.*
- *$H_2(X)$ is the number of voids in X.*

Remark 23

Note, however, that in the general metric spaces, the interpretation that homology measures the amount of "holes" does not hold.

2.4 Persistent homology

The main idea of *persistence* is that instead of selecting a fixed value of the threshold ε, we would like to obtain a useful summary of the homological information for all the different values of ε at once. As ε increases, we add simplicies to the complexes and detect which features "persist."

The main advantages of persistent homology are threefold:

1. It is based on algebraic topology.
2. It is computable via linear algebra.
3. It is robust with regards to perturbations in the input data.

Definition 24 (Filtered simplicial complex)

A subcomplex of $\triangle$ is a subset $\triangle^i \subseteq \triangle$ that is also a simplicial complex. Let $\triangle$ be a finite simplicial complex and let $\triangle^1 \subset \triangle^2 \subset \cdots \subset \triangle^m = \triangle$ be a finite sequence of nested subcomplexes of $\triangle$. The simplicial complex $\triangle$ with such a sequence of subcomplexes, $\emptyset \subseteq \triangle^1 \subseteq \triangle^2 \subseteq \cdots \subseteq \triangle^m = \triangle$, is called filtered simplicial complex.

Definition 25 (p-persistent k-th homology group)

Given a filtered complex, for the i-th subcomplex $\triangle^i$ we can compute the associated

- *the boundary maps ∂_k^i for all dimensions k.*
- *boundary matrices M_k^i for all dimensions k.*
- *groups C_k^i, Z_k^i (cycle group), B_k^i (boundary group), and H_k^i (homology group).*

Then the p-persistent k-th homology group $H_k^{i,p}$ of $\triangle^i$ is

- $H_k^{i,p} = Z_k^i / (B_k^{i+p} \cap Z_k^i)$
- *(equivalently) $H_k i, p \cong Im(\eta_k^{i,p})$, where $\eta_k^{i,p}$ is a bijection $\eta_k^{i,p} : H_k^i \to H_k^{i+p}$ that maps a homology class into another homology class containing it.*

Note that this is simply the definition of homology group of degree k in Definition 19 with the additional notion of persistence.

Definition 26 (p-persistent k-th homology group)

We now have the persistent version of 20:

The p-persistent k-th Betti number of $\triangle^i$ is $\beta_k ip$ = the rank of the free subgroup of $H_k^{i,p}$.

Definition 27 (Persistence complex)

A persistence complex $\mathcal{C}$ is a family of chain complexes $\{C_*^i\}_{i\geqslant 0}$ over R, together with chain map's $f^i : C_*^i \to C_*^{i+1}$, so that we have the following diagram:

$$C_*^0 \xrightarrow{f^0} C_*^1 \xrightarrow{f^1} C_*^2 \xrightarrow{f^2} \cdots$$

The filtered simplicial complex defined in 24 with inclusion maps for the simplices then becomes a persistence complex. To illustrate, below is a portion of a persistence complex with the the chain complexes expanded.

$$\begin{array}{ccccccc}
\downarrow{\scriptstyle \partial_3} & & \downarrow{\scriptstyle \partial_3} & & \downarrow{\scriptstyle \partial_3} & & \\
C_2^0 & \xrightarrow{f^0} & C_2^1 & \xrightarrow{f^1} & C_2^2 & \xrightarrow{f^2} & \cdots \\
\downarrow{\scriptstyle \partial_2} & & \downarrow{\scriptstyle \partial_2} & & \downarrow{\scriptstyle \partial_2} & & \\
C_1^0 & \xrightarrow{f^0} & C_1^1 & \xrightarrow{f^1} & C_1^2 & \xrightarrow{f^2} & \cdots \\
\downarrow{\scriptstyle \partial_1} & & \downarrow{\scriptstyle \partial_1} & & \downarrow{\scriptstyle \partial_1} & & \\
C_0^0 & \xrightarrow{f^0} & C_0^1 & \xrightarrow{f^1} & C_0^2 & \xrightarrow{f^2} & \cdots
\end{array}$$

Definition 28 (Persistence modules)

A persistence module $\mathcal{M}$ is a family of R-modules M^i, together with homomorphisms $\phi^i : M^i \to M^{i+1}$.

Theorem 29 (Classification Theorem)

For a finite persistence module $\mathcal{M}$ with coefficients in the field F,

$$H_*(\mathcal{M}; F) \cong \underbrace{\bigoplus_i x^{t_i} F(x)}_{\text{free module}} \oplus \underbrace{\left(\bigoplus_j x^{r_j} (F(x)/(x^{s_j} F[x])) \right)}_{\text{torsion module}} \tag{2.5}$$

There exists a bijection between the set of free elements and the set of homology generators with birth at t_i and persist for all future parameter values.

There exists a bijection between the set of torsion elements and the set of homology generators with birth at r_j and death at $r_j + s_j$.

The classification theorem gives the fundamental characterization of persistence barcode.

Theorem 30 (Barcode as the persistence analogue of Betti number)

The rank of $H_k^{i\to j}(\mathcal{C}; F)$ gives the number of intervals in the barcode of $H_k^{i\to j}(\mathcal{C}; F)$ spanning the parameter interval $[i, j]$. In particular, $H_*^{i\to j}(\mathcal{C}_*^i; F)$ gives the number of intervals that contain i.

Remark 31

As with Betti number, the barcode for H_k does not give the actual structure of the homology group, but just a continuously parameterized rank. The barcode is useful in that it can qualitatively filter out topological noise (since they are "short-lived" features) and capture significant topological features (features that persist over increasing values of ε).

2.5 Commonly used complexes

Definition 32 (Čech complex)

For any subset $V \subseteq X$ for which $X = \bigcup_{v\in V} B_\varepsilon(v)$, one can construct the nerve of the covering $\{B_\varepsilon(v)\}_{v\in V}$. This construction is referred to as the "Čech complex" attached to V and is denoted as $\check{\mathrm{C}}(V, \varepsilon)$.

Theorem 33

Let M be a compact Riemannian manifold. Then there is a positive number e so that $\check{\mathrm{C}}(M, \varepsilon)$ is homotopy equivalent to M whenever $\varepsilon \leqslant e$. Moreover, for every $\varepsilon \leqslant e$, there is a finite subset $V \subseteq M$ so that the subcomplex of $\check{\mathrm{C}}(V, \varepsilon) \subseteq \check{\mathrm{C}}(M, \varepsilon)$ is also homotopy equivalent to M.

However, this construction is computationally expensive. A solution is to construct a simplicial complex which can be recovered solely from the edge information, which motivates the following construction known as the "Vietoris-Rips complex."

Definition 34 (Vietoris-Rips complex)

Let X be a metric space with metric d. Then the Vietoris-Rips complex for X, attached to the parameter ε, denoted by $VR(X, \varepsilon)$, will be the simplicial complex whose vertex set is X, and where $\{x_0, \ldots, x_k\}$ spans a k-simplex if and only if $d(x_i, x_j) \leqslant \varepsilon$ for all $0 \leqslant i, j \leqslant k$.

Proposition 35

Comparing the Čech complex and the VR compelx:

$$\check{\mathrm{C}}(X, \varepsilon) \subseteq VR(X, 2\varepsilon) \subseteq \check{\mathrm{C}}(X, 2\varepsilon).$$

However, even the VR complex is computationally expensive. A solution, again, is to use the Voronoi decomposition which studies the subspaces of Euclidean space.

Theorem 36 (Voronoi decomposition)

Let X be any metric space, and let $\mathcal{L} \subseteq X$ be a subset (called the set of landmark points). Given $\lambda \in \mathcal{L}$, we define the Voronoi cell associated to λ, V_λ, by

$$V_\lambda = \{x \in X | d(x, \lambda) \leqslant d(x, \lambda')\} \forall \lambda' \in \mathcal{L}.$$

Definition 37 (Delaunay complex)

Similar to how we define the Čech complex above, we define the Delaunay complex attached to $\mathcal{L}$ to be the nerve of this covering.

However, for finite metric spaces, the Delaunay complex generically produces degenerate (i.e. discrete) complexes with no 1-simplices. To solve

this, we modify the definition to accommodate pairs of points which are "almost" equidistant from a pair of landmark points. We thus have the definition below:

Definition 38 (Strong witness complex)

Let X be any metric space, and suppose we are given a finite set $\mathcal{L}$ of points in X (called the landmark set), and a parameter $\varepsilon > 0$. For every point $x \in X$, we let m_x denote the distance from this point to the set $\mathcal{L}$, i.e., the minimum distance from x to any point in the landmark set.

Then we define the strong witness complex attached to this data to be the complex $W^s(X, \mathcal{L}, \varepsilon)$ whose vertex set is $\mathcal{L}$, and where a collection $\{l_0, \ldots, l_k\}$ spans a k-simplex if and only if there is a point $x \in X$ (the witness) so that $d(x, l_i) \leqslant m_x + \varepsilon$ for all i.

We can also consider the version of this complex in which the 1-simplices are identical to those of $W(X, \mathcal{L}, \varepsilon)$, but where the family $\{l_0, \ldots, l_k\}$ spans a k-simplex if and only if all the pairs (l_i, l_j) are 1-simplices. We will denote this by W^s_{VR}.

A modified version of the strong witness complex is also useful:

Definition 39 (Weak witness complex)

We construct the weak witness complex, $W^w(X, \mathcal{L}, \varepsilon)$, attached to the given data by declaring that a family $\Lambda = \{l_0, \ldots, l_k\}$ spans a k-simplex if and only if Λ and all its faces admit ε weak witnesses.

Similar to the definition for strong witness complex, we can also consider the version of the weak witness complex in which the 1-simplices are identical to those of $W(X, \mathcal{L}, \varepsilon)$, but where the family $\{l_0, \ldots, l_k\}$ spans a k-simplex if and only if all the pairs (l_i, l_j) are 1-simplices. We will denote this by W^w_{VR}.

Chapter 3

Distances

3.1 Introduction

In (Mémoli and Sapiro, 2004), four main categories of data analysis technique for high-dimensional data are compared, which are dimension reduction, clustering, regression analysis, and topological data analysis (TDA). Regarding choosing the appropriate distance in these methods, the paper (Aggarwal, Hinneburg, and Keim, 2001) examined the commonly used L_p norm and showed that in high dimensional space, whether the notion of distance is qualitatively meaningful depends on the value of p.

In TDA, the relevant distance metric include the Gromov-Hausdorff distance, and on the space of persitence diagrams, the Bottleneck distance and the p-Wasserstein distance. In our application, we followed closely the algorithm proposed in (Kerber, Morozov, and Nigmetov, 2016) to compute the p-Wasserstein distance between persistence diagrams.

In this chapter, we will explain in detail the definitions and use cases of the above-mentioned distance metrics. The main references for this chapter are (Chazal and Michel, 2021) and (Phillips, 2013).

Definition 40 (Metric Spaces)

Given a set X and a function $d : X \times X \to \mathbb{R}$, we say that the pair $\mathcal{M} = (X, d)$ is a metric space on X if and only if d satisfies the following properties:

1. *(Non-negativeness) For all $x, y \in X, d(x, y) \geqslant 0$.*
2. *(Identity) For all $x, y \in X, d(x, y) = 0 \iff x = y$.*
3. *(Symmetry) For all $x, y \in X, d(x, y) = d(y, x)$.*
4. *(Triangle Inequality) For all $x, y, z \in X, d(x, z) \leqslant d(x, y) + d(y, z)$.*

A point cloud is essentially a metric space X. Elements of X are called points of the metric space. $d(x, y)$ refers to the distance between points x and y.

3.2 Distance between points in $\mathbb{R}^d$

3.2.1 L_p distance (Minkowski distance)

Definition 41 (L_p distance)

Suppose $x, y \in \mathbb{R}^d$. Then the L_p distance is defined as

$$d_p(x, y) = \|x - y\|_p = \left(\sum_{i=1}^{d} (|a_i - b_i|^p) \right)^{1/p}. \tag{3.1}$$

The following distance metrics are special instances of the L_p distance.

Definition 42 (L_2 distance (Euclidean distance))

The most common distance metric is the L_2 distance (or Euclidean distance).

$$d_2(x, y) = \|x - y\|_2 = \left(\sum_{i=1}^{d} (|a_i - b_i|^2) \right)^{1/2}. \tag{3.2}$$

Definition 43 (L_1 distance (Manhattan distance))

$$d_1(x, y) = \|x - y\|_1 = \sum_{i=1}^{d} (|a_i - b_i|). \tag{3.3}$$

Based on the paper (Aggarwal, Hinneburg, and Keim, 2001), in metric spaces with a high dimension, the L_1 distance and L_p distance with fractional p are more useful than the common L_2 distance. For this reason, in many computer vision tasks, the L_1 distance is usually preferred.

Definition 44 (L_∞ distance)

$$d_\infty(x, y) = \|x - y\|_\infty = \max_{i=1}^{d} |a_i - b_i|. \tag{3.4}$$

The following figure compares the above instances of L_p distance:

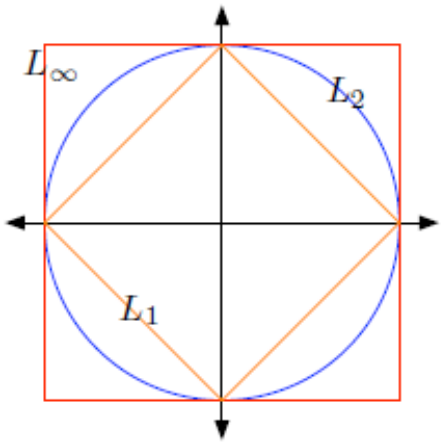


Figure 3.1: Comparing the different L_p distances. Adapted from (Phillips, 2013).

3.2.2 Cosine distance

The cosine distance measures the cosine of the angle between vectors $x, y \in \mathbb{R}^d$

$$d_{cos}(x, y) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3.5}$$

3.3 Distance between closed sets of points

Definition 45 (ε-thickening)

Given a metric space X, the set of closed sets of X supports a metric, the Hausdorff metric.

If A is a set in X and $\varepsilon > 0$, we define the ε-thickening of A to be the set $A^{(\varepsilon)}$ defined by

$$A^{(\varepsilon)} = \bigcup_{x \in A} B_x(\varepsilon), \tag{3.6}$$

where $B_x(\varepsilon)$ is the open ball of radius ε centered at x.

Definition 46 (Hausdorff distance)

Suppose $A, B \subseteq X$ are closed sets, define their Hausdorff distance $d_H(A, B)$:

$$d_H(A, B) = \inf\{\varepsilon > 0 \mid A \subseteq B^{\varepsilon} \text{ and } B \subseteq A^{\varepsilon}\}. \tag{3.7}$$

Definition 47 (Gromov-Hausdorff distance)

The Gromov-Hausdorff distance extends the Hausdorff distance from subsets of the same metric space to subsets of distinct metric spaces.

Suppose A, B are two closed metric spaces. Then the Gromov-Hausdorff distance is

$$d_{GH}(A, B) = \inf_{f,g}\{d_H(f_{A\to X}(A), g_{B\to X}(B))\}, \tag{3.8}$$

where $f_{A\to X}$ denotes an isometric embedding of A into some metric space X and $f_{B\to X}$ denotes an isometric embedding of B into some metric space X. The infimum is taken over all possible such embeddings.

3.4 Distance between persistence diagrams

To make use of the topological features obtained from persistent homology, we need a distance metric to compare persistence diagrams. (Kerber, Morozov, and Nigmetov, 2016)

Definition 48 (Matching between persistent diagrams)

A matching between a pair of persistence diagrams dgm_1 and dgm_2 is a subset $M \subseteq dgm_1 \times dgm_2$ such that every point in dgm_1 and dgm_2 appears exactly once in M. That is, for any $x \in dgm_1$ and $y \in dgm_2$, $(\{x\} \times dgm_2) \cap M$ and $(dgm_1 \times \{y\}) \cap M$ each contains a single pair. A matching is perfect if every vertex is matched, that is, the matching is a bijection $\eta : dgm_1 \to dgm_2$. A perfect matching is illustrated in Figure 3.2 below.

The above definition has an equivalent formulation as an assignment problem:

Definition 49 (Matching as assignment problem)

Given a weighted bipartite graph $G = (A \sqcup B, E, w)$, with $|A| = n = |B|$ and a weight function $w : E \to \mathbb{R}_+$, a matching is a subset $M \subseteq E$ such that every vertex of A and of B is incident to at most one edge in M.

Definition 50 (Bottleneck distance between persistent diagrams)

The Bottleneck distance between a pair of persistence diagrams dgm_1 and dgm_2 is defined by

$$d_B(dgm_1, dgm_2) = \inf_M \{ \max_{(x,y)\in M} \|x - y\|_\infty \}, \tag{3.9}$$

where the infimum is taken over all possible matchings M.

If the matching is perfect, then the Bottleneck distance can also be expressed as

$$d_B(dgm_1, dgm2) = \inf_{\eta: dgm_1 \to dgm_2} \{ \sup_{x\in dgm_1} \|x - \eta(x)\|_\infty \}, \tag{3.10}$$

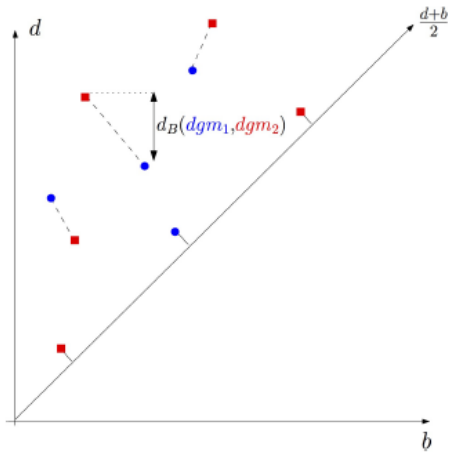


Figure 3.2: A perfect matching and the Bottleneck distance between the blue and red persistent diagrams. Some points are matched to points on the diagonal. Adapted from (Chazal and Michel, 2021).

Definition 51 (*p*-Wasserstein distance between persistent diagrams)

Given $p \geqslant 1$, the p-Wasserstein distance between a pair of persistence diagrams dgm_1 and dgm_2 is defined by

$$W_p(dgm_1, dgm_2) = \left(\inf_M \sum_{(x,y)\in M} \|x - y\|_\infty^p \right)^{1/p}, \tag{3.11}$$

where the infimum is taken over all possible matchings M. If the matching perfect, then the p-Wasserstein distance can also be expressed as

$$W_p(dgm_1, dgm_2) = \left(\inf_{\eta: dgm_1 \to dgm_2} \sum_{x\in dgm_1} \|x - \eta(x)\|_\infty^p \right)^{1/p}, \tag{3.12}$$

As p tends to infinity, the Wasserstein distance approaches the bottleneck distance.

Note that in this report, when we use the term "Wasserstein distance" interchangably with "p-Wasserstein distance."

Definition 52 (*q*-tame persistence module)

A persistence module $\mathbb{V}$ indexed by $T \subseteq \mathbb{R}$ is q-tame if for any $r < s$ in T, the rank of the linear map $v_s^r : V_r \to V_s$ is finite.

Theorem 53 (Chazal et al., 2009)

If $\mathbb{V}$ is a q-tame persistence module, then it has a well-defined persistence diagram.

Theorem 54 (Stability of persistence diagrams)

Let $\mathbb{V}$ and $\mathbb{W}$ be two q-tame persistence modules. If $\mathbb{V}$ and $\mathbb{W}$ are δ-interleaved for some $\delta \geqslant 0$, then

$$d_B(dgm(\mathbb{V}), dgm(\mathbb{W})) \leqslant \delta, \tag{3.13}$$

$$W_p(dgm(\mathbb{V}), dgm(\mathbb{W})) \leqslant \delta. \tag{3.14}$$

Intuitively, the stability of persistence diagrams means that small perturbations in the persistence module will result in small perturbations in the Bottleneck distance and the p-Wasserstein distance between their respective persistence diagrams.

Chapter 4

Dimensionality Reduction

Diffusion map is one of the common dimensionality reduction methods. This chapter gives an overview of the main ideas and steps in diffusion map. The main references for this chapter are (Coifman et al., 2005) and (Coifman and Lafon, 2006).

4.1 Key ideas of diffusion map

- The similarity kernel gives us the *local* geometry. As the time steps move forward, we integrate the local geometry and thus reveal the geometric structures at different scales.
- A cluster from a random walk is a region where the probability of escaping this region is low.

4.2 Main steps of diffusion map

Now, we summarize the main steps of diffusion maps:

1. Construct weighted graph using Gaussian similarity kernel. Each entry in the similarity matrix, W_{ij}, is the pairwise similarity value

between inputs i and j and is taken as the weight of the edge between nodes i and j.

$$W_{ij} = \exp\{-\frac{\|x_i - x_j\|^2}{2\sigma^2}\}. \tag{4.1}$$

2. We can construct a lazy random walk with the probability of reaching node j from i, $p(j \mid i)$, proportional to their pairwise similarity value W_{ij}. (The random walk is lazy if we allow $p(i \mid i) > 0$, i.e., to stay at some point i as the time step moves forward).

$$p(j \mid i) = \frac{W_{ij}}{\sum_k W_{ik}}. \tag{4.2}$$

The probabilities can be represented by a Markov matrix M, which is essentially the similarity matrix normalized to have row sums equal to 1:

$$M = D^{-1}W, \text{where } D_{ii} = \sum_k W_{ik}. \tag{4.3}$$

The probability of reaching node j from i after t steps is then

$$p(t, j \mid i) = e_i^T M^t e_j. \tag{4.4}$$

3. Eigendecomposition of the Markov matrix:

The eigendecomposition of M is derived from the eigendecomposition of $M_s = D^{1/2}MD^{-1/2} = \Omega\Lambda\Omega^T$:

$$M = D^{-1/2}\Omega\Lambda\Omega^T D^{-1/2} := \Psi\Lambda\Phi^T, \tag{4.5}$$

Note that since Ψ and Φ are mutually orthogonal, Ψ contains the right eigenvectors, as shown below:

$$M\Psi = \Psi\Lambda\Phi^T\Psi = \Psi\Lambda = \Lambda\Psi. \tag{4.6}$$

The eigendecomposition of M after t steps is then

$$M = \Psi\Lambda^t\Phi^T. \tag{4.7}$$

- Diffusion coordinate functions: right eigenvectors of Markov matrix scaled by their corresponding eigenvalues:

$$\Upsilon := \Psi\Lambda. \tag{4.8}$$

- Diffusion distance after t steps:

$$\|e_i^T\Upsilon - e_j^T\Upsilon\|^2 = \sum_k (p(t,k \mid i) - p(t,k \mid j))^2 (D_{kk}^{-1}). \tag{4.9}$$

- Growth of eigenvalues leads to geometric properties (more on spectral geometry).

  Two extreme situations:

(a) If disconnected graph (none of the nodes are connected), then:

$$P = I, \lambda_i = \lambda_j \quad \forall i, j,$$

which implies a flat spectrum with zero decay rate.

(b) If fully connected graph (each of the node is connected to all the rest of the nodes), assuming weights of all edges are 1, then:

$$\lambda_1 = 1, \lambda_i = 0 \quad \forall i \neq 1.$$

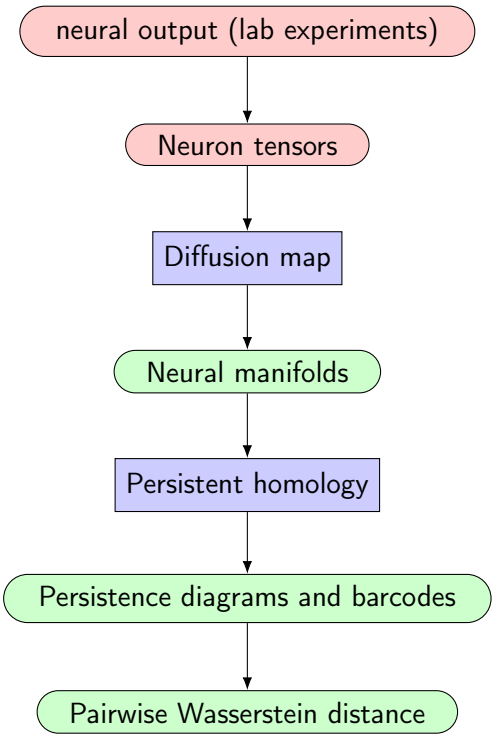
Chapter 5

Application to Neural Spiking Data

5.1 Outline of Methods

We applied the theory outlined in the previous chapters to neural spiking data in which we quantified the pairwise similarity of the neural population responses to different stimuli (each represented as a high-dimensional point cloud) in terms of their geometric and topological structure. The approach of this project was to first embed the data into a lower-dimensional manifold using diffusion map and then apply the method of persistent homology to extract the topological features, specifically the persistence diagrams and the persistence barcodes. In the end, we computed the pairwise Wasserstein distance between the persistence diagrams, which measures the pairwise similarity we set out to find.

The following flowchart shows a schematic summary of our approach.



5.2 Data Collection

The neural spiking data were collected from lab experiments conducted on mice. The experimental setup is shown in the figure below. Visual stimuli of moving artificial gratings of six different types were flashed in front of the mouse. Each visual stimuli were moving in eight directions. Neuron output of the mouse retina was recorded with electrodes and encoded in peristimulus (PSTH) diagrams. Each PSTH diagram shows the firing rate of one neuron over time for each of the eight directions respectively. Brighter pixels indicate higher firing rates.

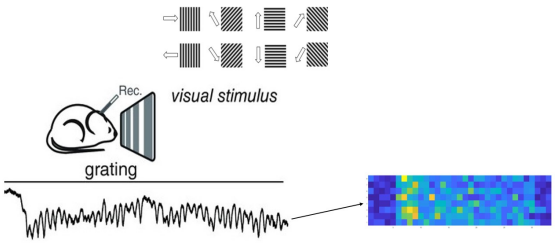


Figure 5.1: Visualising neural data from lab experiments.

Definition 55 (Neural population response)

Suppose $\mathcal{S}$ is a set of S visual stimuli (e.g., images) $\mathcal{S} = \{s_1, s_2, \ldots, s_S\}$, each having T number of shifts (which is equivalent to the number of pixels in the PSTH diagrams in this application).

The neural population response of a set of N neurons to a stimulus m_i over all transformations is $\mathcal{N} = \{\vec{n}_1, \vec{n}_2, \ldots, \vec{n}_N\}$, where $\vec{n}_i \in \mathbb{R}^T$.

Definition 56 (Neural tensors)

Each neural tensor encodes the neural population response of a set of neurons to a set of stimulus over all transformations. It is thus a 3-way tensor of dimension N-by-S-by-T.

The neural spiking data set used in this project is represented by a 3-way tensor of dimension 698-by-6-by-264. The three dimensions represent the following respectively: there are in total 698 neurons 6, 6 types of visual stimuli, and 264 number of pixels in the PSTH diagram.

With this neural spiking data set, we can create six point clouds, each corresponds to the neural population response towards one type of stimuli, which we denote as $X_1, X_2, \ldots, X_6$. Each of the point cloud X_i is thus represented by a matrix of dimension 698-by-264, giving us a point cloud of 698 points in $\mathbb{R}^{264}$.

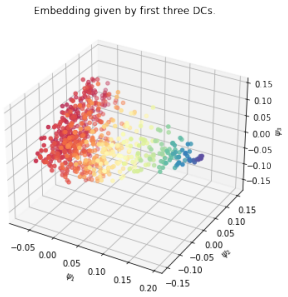
5.3 Step 1: Dimensionality Reduction

The main motivation behind dimensionality reduction is that the intrinsic dimension of the data is usually much lower than the extrinsic dimension and the high-dimensionality is usually just an artifact of the representation. The representation of data has a potentially large degree of freedom, that is, the number of variables that are really necessary to describe the data is much smaller.

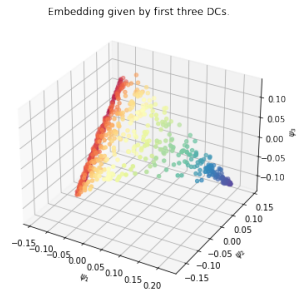
The Manifold Hypothesis states that real-world high-dimensional data lie on low-dimensional manifolds embedded within the high-dimensional space. (DeepAI, 2019)

In the context of neuroscience, neural spiking data is high-dimensional, but the neural connections constrain the possible patterns of population activity (Okun et al., 2015, Sadtler et al., 2014, Tsodyks, 1999) and that the possible patterns are confined to a low-dimensional manifold (Stopfer, Jayaraman, and Laurent, 2003, Yu et al., 2009) spanned by a few independent patterns that are called “neural modes." (Gallego et al., 2017)

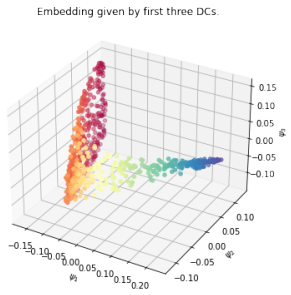
In terms of implementation, we used the pydiffmap package (Eastman et al., 2017). In our implementation, we reduced the dimensionaltity of each point cloud from the space of $\mathbb{R}^{264}$ to $\mathbb{R}^3$ using diffusion map. The resulting three-dimensional embedding of the original point clouds colored by the first three diffusion coordinates are shown below:



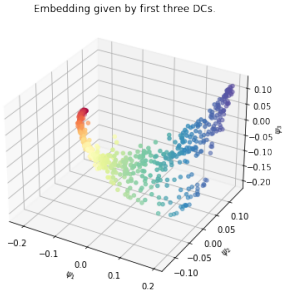
(a) Three-dimensional embedding of the original point cloud X_1.



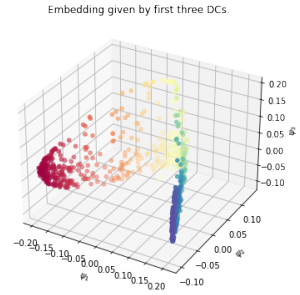
(b) Three-dimensional embedding of the original point cloud X_2.



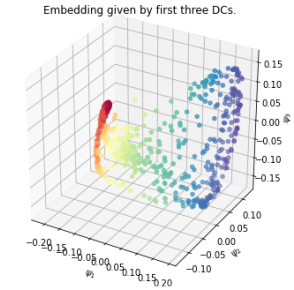
(c) Three-dimensional embedding of the original point cloud X_3.



(d) Three-dimensional embedding of the original point cloud X_4.



(e) Three-dimensional embedding of the original point cloud X_5.

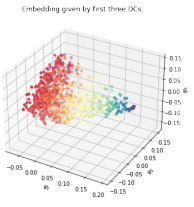


(f) Three-dimensional embedding of the original point cloud X_6.

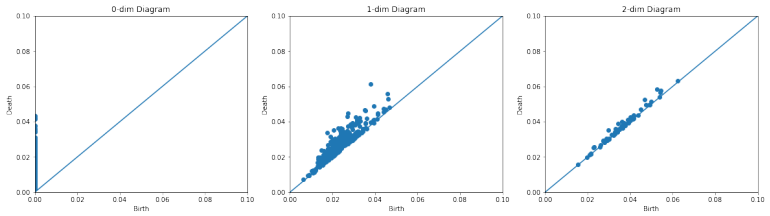
5.4 Step 2: Persistent Homology

Having obtained the three-dimensional embeddings of the point clouds corresponding to six stimuli types, we applied persistent homology to extract the topological features from the six embeddings respectively. In this step, these topological features are represented by persistence barcodes and persistence diagrams.

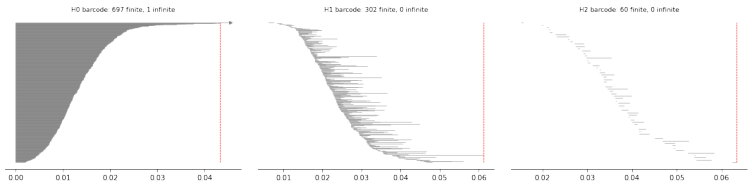
Our implementation used the package ripser (Tralie, Saul, and Bar-On, 2018) to obtain the respective persistence diagrams from the embeddings. Using the (birth, death)-intervals from each persistence diagram, we drew the equivalent persistence barcode representation. The complete code can be found in Appendix A. The following figures show the results from applying persistent homology.



(a) Three-dimensional embedding of the original point cloud X_1.

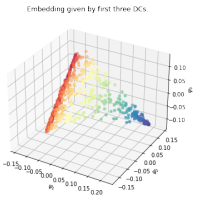


(b) Persistence diagrams.

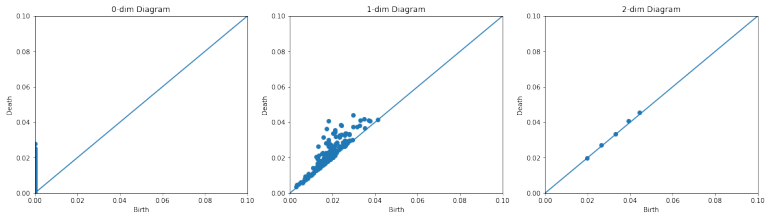


(c) (d) Persistence barcodes. (e)

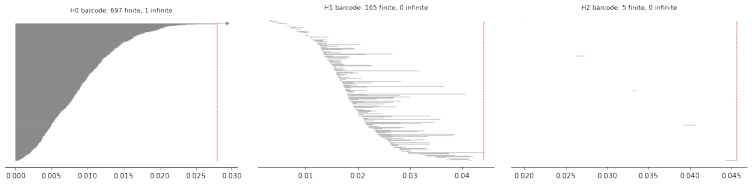
Figure 5.3: Results for applying persistent homology on the three-dimensional embedding of X_1.



(a) Three-dimensional embedding of the original point cloud X_2.

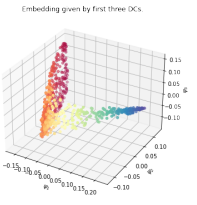


(b) Persistence diagrams.

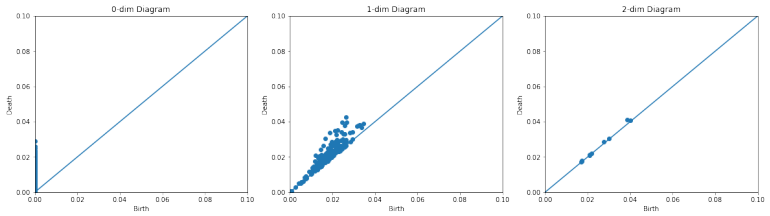


(c) (d) Persistence barcodes. (e)

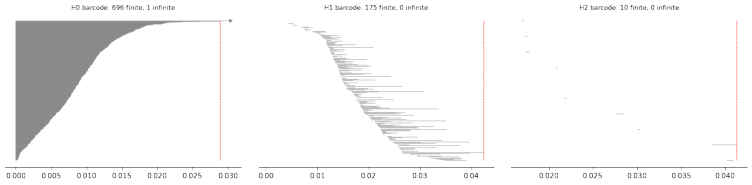
Figure 5.4: Results for applying persistent homology on the three-dimensional embedding of X_2.



(a) Three-dimensional embedding of the original point cloud X_3.

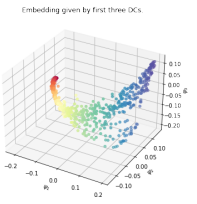


(b) Persistence diagrams.

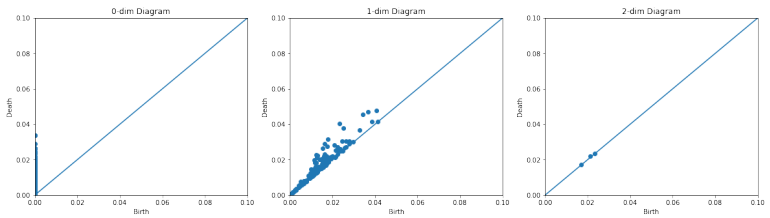


(c) (d) Persistence barcodes. (e)

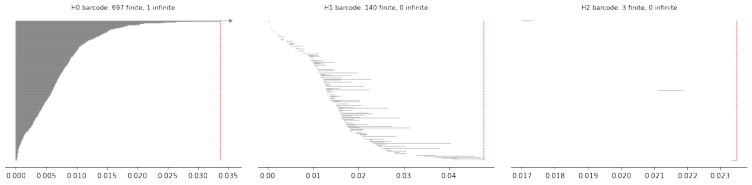
Figure 5.5: Results for applying persistent homology on the three-dimensional embedding of X_3.



(a) Three-dimensional embedding of the original point cloud X_4.

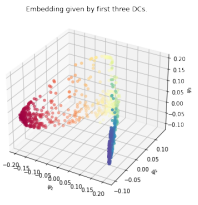


(b) Persistence diagrams.

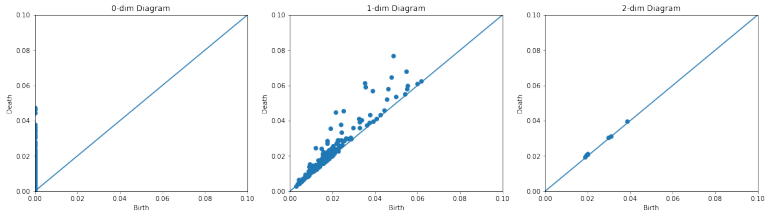


(c) (d) Persistence barcodes. (e)

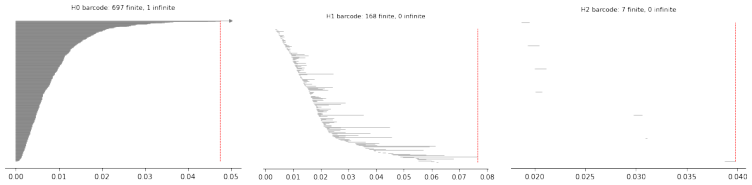
Figure 5.6: Results for applying persistent homology on the three-dimensional embedding of X_4.



(a) Three-dimensional embedding of the original point cloud X_5.

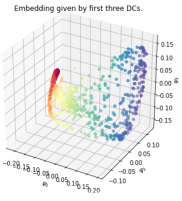


(b) Persistence diagrams.

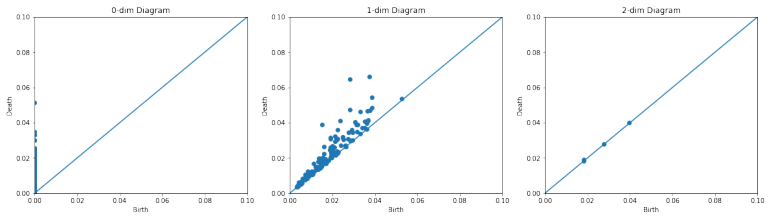


(c) (d) Persistence barcodes. (e)

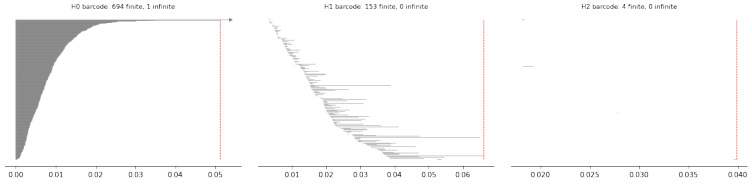
Figure 5.7: Results for applying persistent homology on the three-dimensional embedding of X_5.



(a) Three-dimensional embedding of the original point cloud X_6.



(b) Persistence diagrams.



(c) (d) Persistence barcodes. (e)

Figure 5.8: Results for applying persistent homology on the three-dimensional embedding of X_6.

5.5 Step 3: Pairwise Wasserstein distance

In the last step, we used the gudhi package (The GUDHI Project, 2021) to compute the pairwise Wasserstein distance between the persistence diagrams. The method used is based on (Kerber, Morozov, and Nigmetov,

2016).

We first computed the first pairwise distances between the persistence diagrams for each of the six point clouds.

H_0	X_1	X_2	X_3	X_4	X_5	X_6
X_1	0.0	2.77	3.05	3.95	3.08	3.42
X_2	2.77	0.0	0.43	1.35	1.0	0.84
X_3	3.05	0.43	0.0	1.03	0.94	0.66
X_4	3.95	1.35	1.03	0.0	1.11	0.72
X_5	3.08	1.0	0.94	1.11	0.0	0.51
X_6	3.42	0.84	0.66	0.72	0.51	0.0

Table 5.1: Pairwise Wasserstein distance between persistent diagrams for homology group H_0.

H_1	X_1	X_2	X_3	X_4	X_5	X_6
X_1	0.0	0.46	0.5	0.64	0.64	0.55
X_2	0.46	0.0	0.17	0.29	0.36	0.3
X_3	0.5	0.17	0.0	0.25	0.33	0.3
X_4	0.64	0.29	0.25	0.0	0.27	0.29
X_5	0.64	0.36	0.33	0.27	0.0	0.26
X_6	0.55	0.3	0.3	0.29	0.26	0.0

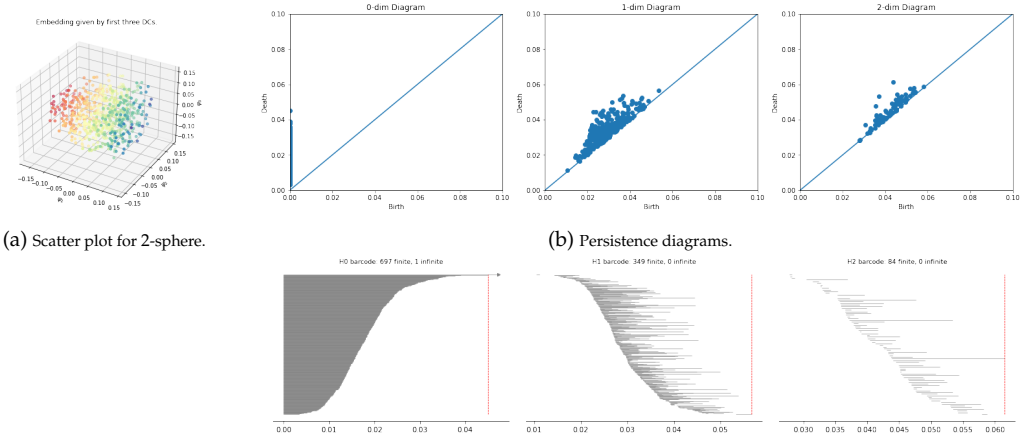
Table 5.2: Pairwise Wasserstein distance between persistent diagrams for homology group H_1.

H_2	X_1	X_2	X_3	X_4	X_5	X_6
X_1	0.0	0.06	0.06	0.06	0.06	0.06
X_2	0.06	0.0	0.01	0.0	0.01	0.0
X_3	0.06	0.01	0.0	0.0	0.01	0.0
X_4	0.06	0.0	0.0	0.0	0.0	0.0
X_5	0.06	0.01	0.01	0.0	0.0	0.0
X_6	0.06	0.0	0.0	0.0	0.0	0.0

Table 5.3: Pairwise Wasserstein distance between persistent diagrams for homology group H_2.

Building upon the same method, we further compared the six point clouds with known three-dimensional shapes (2-sphere and torus) based on the Wasserstein distance.

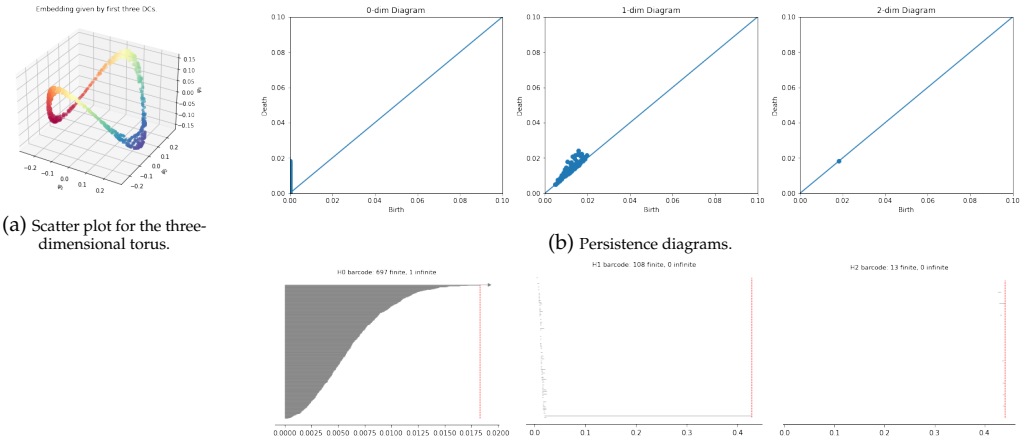
First, we extracted the topological features of the 2-sphere and three-dimensional torus, as we did for the point clouds in Step 2.



(a) Scatter plot for 2-sphere. (b) Persistence diagrams.

(c) (d) Persistence barcodes. (e)

Figure 5.9: Results for applying persistent homology on the 2-sphere.



(a) Scatter plot for the three-dimensional torus.

(b) Persistence diagrams.

(c) (d) Persistence barcodes. (e)

Figure 5.10: Results for applying persistent homology on the three-dimensional torus.

Then, we computed the Wasserstein distance between the six point clouds and the known shapes:

H_0	X_1	X_2	X_3	X_4	X_5	X_6
2-sphere	2.48	4.46	4.64	5.18	4.53	4.83
torus	3.63	1.46	1.31	0.73	1.38	1.06

Table 5.4: Wasserstein distance between persistent diagrams of the point clouds and 2-sphere and torus (homology group H_0).

H_1	X_1	X_2	X_3	X_4	X_5	X_6
2-sphere	0.62	0.77	0.80	0.87	0.85	0.76
torus	0.74	0.49	0.44	0.34	0.45	0.47

Table 5.5: Wasserstein distance between persistent diagrams of the point clouds and 2-sphere and torus (homology group H_1).

H_2	X_1	X_2	X_3	X_4	X_5	X_6
2-sphere	0.11	0.12	0.12	0.12	0.13	0.12
torus	0.057	0.017	0.018	0.016	0.018	0.016

Table 5.6: Wasserstein distance between persistent diagrams of the point clouds and 2-sphere and torus (homology group H_2).

5.6 Analysis of results

Based on the results from the above tables of Wasserstein distances, our observations and inferences are as follows.

1. The topological structure for X_1 is distinct from the rest.

   Based on the pairwise Wasserstein distances in Tables 5.1 and 5.2, X_1 is notably different from the rest of the point clouds in terms

of topological structure. In the context of our application, this implies that the neural population response evoked by stimulus type 1 is significantly different from the other stimulus types. This observation leads us to hypothesize that there is some neuroscientific reason behind this distinction in the topological structure of neural population response. It might be interesting to conduct further lab experiments to investigate what special properties this stimulus has and why it causes such different neural population response.

2. Wasserstein distances between persistence diagrams are nearly negligible for H_2.

   Based on Table 5.3, pairwise Wasserstein distances between persistence diagrams for H_2 are small enough to be nearly negligible. This implies that the intrinsic dimensionality of this neural data might be even lower than three-dimensional since there is no significant differences in homology groups H_2 for the point clouds.

3. Shape comparison with 2-sphere and torus.

   Based on Tables 5.4 and 5.5, the Wasserstein distances between the point clouds and the 2-sphere are smaller than the Wasserstein distances between the point clouds and the torus for all X_i except for X_1. We can thus infer that except for X_1, all other point clouds are more similar to the shape of a torus than the 2-sphere. This implies that the topological structure of the neural population response evoked by stimuli type 1 is more similar to a sphere while the neural population response evoked by the rest of the stimuli types used in the experiments are more similar to a torus.

It would be interesting to compare this result with the hypothesis in (Ben-Yishai, Bar-Or, and Sompolinsky, 1995, Blumenfeld, Rogat, and Krajcik, 2006, Goldberg et al., 2004, Singh et al., 2008): If we are given an oriented stimulus, and if the orientation is a circular variable, then the hypothesis is that the neural population response evoked by such stimulus must have a topological structure equivalent to that of a circle. However, to fully test this hypothesis, further experiments with different types of stimuli need to be conducted.

5.7 Conclusion

In this chapter, we demonstrated our proposed approach to compare the point clouds that represent the neural population response to different stimuli. Our proposed approach is contingent on the topological structures of the point clouds.

One significant limitation in this specific application is that lab data usually involve a lot of sampling errors such as missing data and inconsistent densities, causing noise to the true underlying geometry of the neural population response. Persistent homology works well on synthetic data, but might not work when the sampling errors are too significant.

The advantages of topological approach is that we can provide a succinct and useful summary of the global geometric structure of the neural spiking data, thus obtaining a useful account of how similarly the neurons collectively respond to visual stimuli. This is especially important in applications where the notion of connectedness and clusters are salient.

As an emerging field, TDA will certainly see more applications in solving problems that involve understanding the geometric and topological structure of high-dimensional data.

Bibliography

Aggarwal, Charu C., Alexander Hinneburg, and Daniel A. Keim (2001). "On the Surprising Behavior of Distance Metrics in High Dimensional Space". en. In: *Database Theory — ICDT 2001*. Ed. by Gerhard Goos et al. Vol. 1973. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 420–434. ISBN: 978-3-540-41456-8 978-3-540-44503-6. DOI: 10.1007/3-540-44503-X_27. URL: http://link.springer.com/10.1007/3-540-44503-X_27 (visited on 10/17/2021).

Bardin, Jean-Baptiste, Gard Spreemann, and Kathryn Hess (Jan. 2019). "Topological exploration of artificial neuronal network dynamics". en. In: *Network Neuroscience* 3.3, pp. 725–743. ISSN: 2472-1751. DOI: 10.1162/netn_a_00080. URL: https://direct.mit.edu/netn/article/3/3/725-743/2173 (visited on 09/01/2021).

Ben-Yishai, R, R L Bar-Or, and H Sompolinsky (Apr. 1995). "Theory of orientation tuning in visual cortex." In: *Proceedings of the National Academy of Sciences* 92.9, p. 3844. DOI: 10.1073/pnas.92.9.3844. URL: http://www.pnas.org/content/92/9/3844.abstract.

Beshkov, Kosio and Paul Tiesinga (2021). "Geodesic-based distance reveals non-linear topological features in neural activity from mouse visual cortex". In: *bioRxiv*. Publisher: Cold Spring Harbor Laboratory _eprint: https://www.biorxiv.org/content/early/2021/05/21/2021.05.21.444993.full.

DOI: 10.1101/2021.05.21.444993. URL: https://www.biorxiv.org/content/early/2021/05/21/2021.05.21.444993.

Blumenfeld, Phyllis, Toni Rogat, and Joseph Krajcik (Jan. 2006). "Motivation and Cognitive Engagement in Learning Environments." In: *The Cambridge Handbook of the Learning Sciences*.

Carlsson, Gunnar (Jan. 2009). "Topology and data". en. In: *Bulletin of the American Mathematical Society* 46.2, pp. 255–308. ISSN: 0273-0979. DOI: 10.1090/S0273-0979-09-01249-X. URL: http://www.ams.org/journal-getitem?pii=S0273-0979-09-01249-X (visited on 11/02/2021).

Chaudhuri, Rishidev et al. (Sept. 2019). "The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep". In: *Nature Neuroscience* 22.9, pp. 1512–1520. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0460-x. URL: https://doi.org/10.1038/s41593-019-0460-x.

Chazal, Frédéric and Bertrand Michel (Feb. 2021). "An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists". en. In: *arXiv:1710.04019 [cs, math, stat]*. arXiv: 1710.04019. URL: http://arxiv.org/abs/1710.04019 (visited on 10/17/2021).

Chazal, Frédéric et al. (2009). "Proximity of persistence modules and their diagrams". en. In: *Proceedings of the 25th annual symposium on Computational geometry - SCG '09*. Aarhus, Denmark: ACM Press, p. 237. ISBN: 978-1-60558-501-7. DOI: 10.1145/1542362.1542407. URL: http://portal.acm.org/citation.cfm?doid=1542362.1542407 (visited on 11/03/2021).

Coifman, R. R. et al. (2005). "Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps". In: *Proceedings of the National Academy of Sciences* 102.21, pp. 7426–7431. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.0500334102. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.0500334102 (visited on 08/05/2021).

Coifman, Ronald R. and Stéphane Lafon (2006). "Diffusion maps". In: *Applied and Computational Harmonic Analysis* 21.1, pp. 5–30. ISSN: 10635203. DOI: 10.1016/j.acha.2006.04.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S1063520306000546 (visited on 08/05/2021).

DeepAI (2019). *Manifold hypothesis*. URL: https://deepai.org/machine-learning-glossary-and-terms/manifold-hypothesis.

Eastman, Peter et al. (2017). "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics". In: *PLOS Computational Biology* 13.7. DOI: 10.1371/journal.pcbi.1005659.

Edelsbrunner, Letscher, and Zomorodian (Nov. 2002). "Topological Persistence and Simplification". In: *Discrete & Computational Geometry* 28.4, pp. 511–533. ISSN: 1432-0444. DOI: 10.1007/s00454-002-2885-2. URL: https://doi.org/10.1007/s00454-002-2885-2.

Gallego, Juan A. et al. (2017). "Neural Manifolds for the Control of Movement". In: *Neuron* 94.5, pp. 978–984. ISSN: 08966273. DOI: 10.1016/j.neuron.2017.05.025. URL: https://linkinghub.elsevier.com/retrieve/pii/S0896627317304634 (visited on 10/23/2021).

Giusti, Chad et al. (Nov. 2015). "Clique topology reveals intrinsic geometric structure in neural correlations". en. In: *Proceedings of the National*

Academy of Sciences 112.44, pp. 13455–13460. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1506407112. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1506407112 (visited on 09/01/2021).

Goldberg, Richard M. et al. (Jan. 2004). "A Randomized Controlled Trial of Fluorouracil Plus Leucovorin, Irinotecan, and Oxaliplatin Combinations in Patients With Previously Untreated Metastatic Colorectal Cancer". en. In: *Journal of Clinical Oncology* 22.1, pp. 23–30. ISSN: 0732-183X, 1527-7755. DOI: 10.1200/JCO.2004.09.046. URL: http://ascopubs.org/doi/10.1200/JCO.2004.09.046 (visited on 11/05/2021).

Kerber, Michael, Dmitriy Morozov, and Arnur Nigmetov (June 2016). "Geometry Helps to Compare Persistence Diagrams". en. In: *arXiv:1606.03357 [cs]*. arXiv: 1606.03357. URL: http://arxiv.org/abs/1606.03357 (visited on 10/01/2021).

Mémoli, Facundo and Guillermo Sapiro (2004). "Comparing point clouds". en. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing - SGP '04*. ISSN: 17278384. Nice, France: ACM Press, p. 32. ISBN: 978-3-905673-13-5. DOI: 10.1145/1057432.1057436. URL: http://portal.acm.org/citation.cfm?doid=1057432.1057436 (visited on 10/01/2021).

Okun, Michael et al. (May 2015). "Diverse coupling of neurons to populations in sensory cortex". In: *Nature* 521.7553, pp. 511–515. ISSN: 1476-4687. DOI: 10.1038/nature14273. URL: https://doi.org/10.1038/nature14273.

Phillips, Jeff M. (2013). "Course Notes for CS 6955 Data Mining Spring 2013". en. In:

Sadtler, Patrick T. et al. (Aug. 2014). "Neural constraints on learning". In: *Nature* 512.7515, pp. 423–426. ISSN: 1476-4687. DOI: 10.1038/nature13665. URL: https://doi.org/10.1038/nature13665.

Singh, Gurjeet et al. (June 2008). "Topological analysis of population activity in visual cortex". In: *Journal of Vision* 8.8, pp. 11–11. ISSN: 1534-7362. DOI: 10.1167/8.8.11. eprint: https://arvojournals.org/arvo/content_public/journal/jov/933530/jov-8-8-11.pdf. URL: https://doi.org/10.1167/8.8.11.

Stopfer, Mark, Vivek Jayaraman, and Gilles Laurent (Sept. 2003). "Intensity versus Identity Coding in an Olfactory System". In: *Neuron* 39.6, pp. 991–1004. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2003.08.011. URL: https://www.sciencedirect.com/science/article/pii/S089662730300535X.

The GUDHI Project (2021). *GUDHI User and Reference Manual*. 3.4.1. GUDHI Editorial Board. URL: https://gudhi.inria.fr/python/latest/wasserstein_distance_user.html.

Tralie, Christopher, Nathaniel Saul, and Rann Bar-On (2018). "Ripser.py: A Lean Persistent Homology Library for Python". In: *The Journal of Open Source Software* 3.29, p. 925. DOI: 10.21105/joss.00925. URL: https://doi.org/10.21105/joss.00925.

Tsodyks, Misha (Jan. 1999). "Attractor neural network models of spatial maps in hippocampus". In: *Hippocampus* 9.4. Publisher: John Wiley & Sons, Ltd, pp. 481–489. ISSN: 1050-9631. DOI: 10.1002/(SICI)1098-1063(1999)9:4<481::AID-HIPO14>3.0.CO;2-S. URL: https://doi.org/10.1002/(SICI)1098-1063(1999)9:4<481::AID-HIPO14>3.0.CO;2-S (visited on 11/02/2021).

Yu, Byron M. et al. (July 2009). "Gaussian-Process Factor Analysis for Low-Dimensional Single-Trial Analysis of Neural Population Activity". In: *Journal of Neurophysiology* 102.1. Publisher: American Physiological Society, pp. 614–635. ISSN: 0022-3077. DOI: 10.1152/jn.90941.2008. URL: https://doi.org/10.1152/jn.90941.2008 (visited on 11/02/2021).

Zomorodian, Afra (2005a). "Computing Persistent Homology". en. In: p. 15.

Zomorodian, Afra J. (2005b). *Topology for Computing*. Cambridge University Press.

Appendix A

Implementation Code

Persistent_Homology_Features_retina_DiffMap+TDA_report

November 5, 2021

0.1 Apply TDA on retina data

The following code includes:

1. dimensionality reduction using diffusion maps

- diffusion map: pydiffmap

https://pydiffmap.readthedocs.io/en/master/usage.html

2. generate both persistence barcodes and persistence diagrams from the resulting 3D point clouds for six stimuli types

- persistence diagram: ripser (https://ripser.scikit-tda.org/en/latest/index.html)
- persistence barcode: persim
  - https://persim.scikit-tda.org/en/latest/index.html
  - https://github.com/scikit-tda/persim/pull/24
  - https://github.com/scikit-tda/persim/blob/4702251c22d4fffbb1c29409f466745c6b6c26c5/persim/visual

3. Compute the pairwise Wasserstein distance between the persistence diagrams:

- package used: gudhi hera
- useful reference: https://github.com/giotto-ai/giotto-tda/issues/603

[1]:
```
import persim
```

[2]:
```
# Basic imports
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import networkx as nx
# from IPython.display import Video

# scikit-tda imports..... Install all with -> pip install scikit-tda
#--- this is the main persistence computation workhorse
import ripser
# from persim import plot_diagrams
import persim
# import persim.plot
```

```
# teaspoon imports..... Install with -> pip install teaspoon
#---these are for generating data and some drawing tools
import teaspoon.MakeData.PointCloud as makePtCloud
import teaspoon.TDA.Draw as Draw

#---these are for generating time series network examples
from teaspoon.SP.network import ordinal_partition_graph
from teaspoon.TDA.PHN import PH_network
from teaspoon.SP.network_tools import make_network
from teaspoon.parameter_selection.MsPE import MsPE_tau
import teaspoon.MakeData.DynSysLib.DynSysLib as DSL
```

[3]:
```
import kmapper as km
from sklearn.cluster import AgglomerativeClustering
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.manifold import Isomap
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import numpy as np
from kmapper import jupyter
import pydiffmap
from pydiffmap import diffusion_map as dm
from pydiffmap.visualization import embedding_plot, data_plot
```

[4]:
```
# import dionysus as ds
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

import pylab as pl
from matplotlib import collections  as mc
from sklearn.datasets import load_digits
from skimage.morphology import skeletonize
import math
import sys
```

[5]:
```
from scipy.io import loadmat
retina_original_data = loadmat('retina-201205_bgonlyb50RelNormconsecDel142.
 ↪mat')['X']
retina_original_data.shape
```

[5]: (698, 6, 264)

0.2 Draw barcode diagrmas:

[6]:
```
class Barcode:
    __doc__ = """
        Barcode visualisation made easy!
        Note that this convenience class requires instantiation as the number
        of subplots produced depends on the dimension of the data.
        """

    def __init__(self, diagrams, verbose=False):
        """
        Parameters
        ==========
        diagrams: list-like
            typically the output of ripser(nodes)['dgms']
        verbose: bool
            Execute print statemens for extra information; currently only echoes
            number of bars in each dimension (Default=False).
        Examples
        ==========
        >>> n = 300
        >>> t = np.linspace(0, 2 * np.pi, n)
        >>> noise = np.random.normal(0, 0.1, size=n)
        >>> data = np.vstack([((3+d) * np.cos(t[i]+d), (3+d) * np.sin(t[i]+d))␣
↪for i, d in enumerate(noise)])
        >>> diagrams = ripser(data)
        >>> bc = Barcode(diagrams['dgms'])
        >>> bc.plot_barcode()
        """
        if not isinstance(diagrams, list):
            diagrams = [diagrams]

        self.diagrams = diagrams
        self._verbose = verbose
        self._dim = len(diagrams)

    def plot_barcode(self, figsize=None, show=True, export_png=False, dpi=100,␣
↪**kwargs):
        """Wrapper method to produce barcode plot
        Parameters
        ==========
        figsize: tuple
            figure size, default=(6,6) if H0+H1 only, (6,4) otherwise
        show: boolean
            show the figure via plt.show()
        export_png: boolean
            write image to png data, returned as io.BytesIO() instance,
```

```
            default=False
        **kwargs: artist paramters for the barcodes, defaults:
            c='grey'
            linestyle='-'
            linewidth=0.5
            dpi=100 (for png export)
        Returns
        ===========
        out: list or None
            list of png exports if export_png=True, otherwise None
        """
        if self._dim == 2:
            if figsize is None:
                figsize = (6, 6)

            return self._plot_H0_H1(
                figsize=figsize,
                show=show,
                export_png=export_png,
                dpi=dpi,
                **kwargs
            )

        else:
            if figsize is None:
                figsize = (6, 4)

            return self._plot_Hn(
                figsize=figsize,
                show=show,
                export_png=export_png,
                dpi=dpi,
                **kwargs
            )

    def _plot_H0_H1(self, *, figsize, show, export_png, dpi, **kwargs):
        out = []

        fig, ax = plt.subplots(2, 1, figsize=figsize)

        for dim, diagram in enumerate(self.diagrams):
            self._plot_many_bars(dim, diagram, dim, ax, **kwargs)

        if export_png:
            fp = io.BytesIO()
            plt.savefig(fp, dpi=dpi)
            fp.seek(0)
```

```
            out += [fp]

        if show:
            plt.show()
        else:
            plt.close()

        if any(out):
            return out

    def _plot_Hn(self, *, figsize, show, export_png, dpi, **kwargs):
        out = []

        for dim, diagram in enumerate(self.diagrams):
            fig, ax = plt.subplots(1, 1, figsize=figsize)

            self._plot_many_bars(dim, diagram, 0, [ax], **kwargs)

            if export_png:
                fp = io.BytesIO()
                plt.savefig(fp, dpi=dpi)
                fp.seek(0)

                out += [fp]

            if show:
                plt.show()
            else:
                plt.close()

        if any(out):
            return out

    def _plot_many_bars(self, dim, diagram, idx, ax, **kwargs):
        number_of_bars = len(diagram)
        if self._verbose:
            print("Number of bars in dimension %d: %d" % (dim, number_of_bars))

        if number_of_bars > 0:
            births = np.vstack([(elem[0], i) for i, elem in enumerate(diagram)])
            deaths = np.vstack([(elem[1], i) for i, elem in enumerate(diagram)])

            inf_bars = np.where(np.isinf(deaths))[0]
            max_death = deaths[np.isfinite(deaths[:, 0]), 0].max()

            number_of_bars_fin = births.shape[0] - inf_bars.shape[0]
```

```
            number_of_bars_inf = inf_bars.shape[0]

            _ = [self._plot_a_bar(ax[idx], birth, deaths[i], max_death,␣
↪**kwargs) for i, birth in enumerate(births)]

        # the line below is to plot a vertical red line showing the maximal␣
↪finite bar length
        ax[idx].plot(
            [max_death, max_death],
            [0, number_of_bars - 1],
            c='r',
            linestyle='--',
            linewidth=0.7
        )

        title = "H%d barcode: %d finite, %d infinite" % (dim,␣
↪number_of_bars_fin, number_of_bars_inf)
        ax[idx].set_title(title, fontsize=9)
        ax[idx].set_yticks([])

        for loc in ('right', 'left', 'top'):
            ax[idx].spines[loc].set_visible(False)

    @staticmethod
    def _plot_a_bar(ax, birth, death, max_death, c='gray', linestyle='-',␣
↪linewidth=0.5):
        if np.isinf(death[0]):
            death[0] = 1.05 * max_death
            ax.plot(
                death[0],
                death[1],
                c=c,
                markersize=4,
                marker='>',
            )

        ax.plot(
            [birth[0], death[0]],
            [birth[1], death[1]],
            c=c,
            linestyle=linestyle,
            linewidth=linewidth,
        )
```

0.3 Draw persistent diagrams:

[7]:
```
## My code from here:

def drawTDAtutorial(P,diagrams, R):
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize = (20,5))

    # Draw diagrams
    plt.sca(axes[0])
    plt.title('0-dim Diagram')
    Draw.drawDgm(diagrams[0])
#     R = max(diagrams[0][1])
    plt.axis([0,R,0,R])

    plt.sca(axes[1])
    plt.title('1-dim Diagram')
    Draw.drawDgm(diagrams[1])
#     R = max(diagrams[1][1])
    plt.axis([0,R,0,R])

    plt.sca(axes[2])
    plt.title('2-dim Diagram')
    Draw.drawDgm(diagrams[2])
#     R = max(diagrams[2][1])
    plt.axis([0,R,0,R])
```

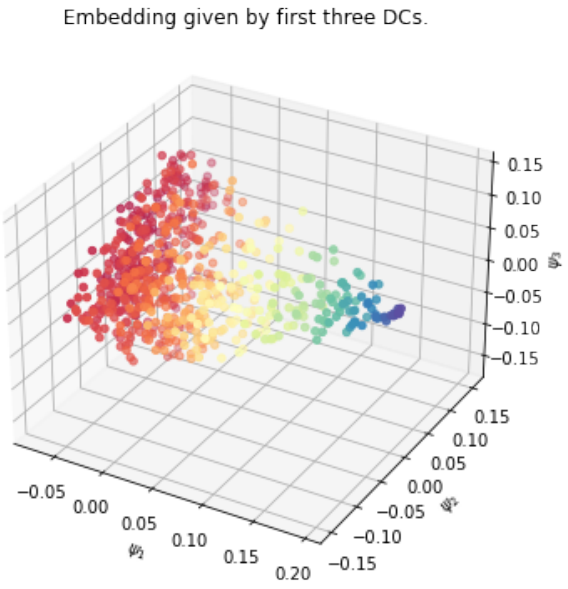
0.4 Dimensionality reduction and TDA:

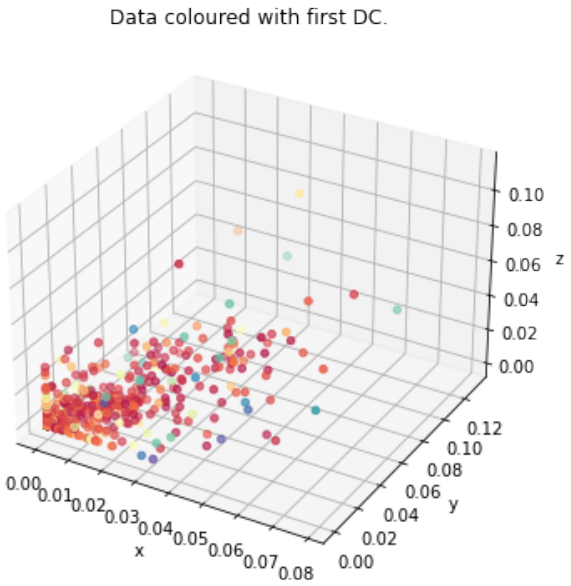
pydiffmap: n_evecs is the number of eigenvectors that are computed, epsilon is a scale parameter used to rescale distances between data points, alpha is a normalization parameter (typically between 0.0 and 1.0) that influences the effect of the sampling density, and k is the number of nearest neighbors considered when the kernel is computed. A larger k means increased accuracy but larger computation time

[8]:
```
def diffusion_tda(X):
  ## diffusion map with automatic epsilon detection:
    mydmap = dm.DiffusionMap.from_sklearn(n_evecs = 3, alpha = 1, epsilon =  1.
↪0 , k=100)

  # Fit to and transform the data
    X_dmap = mydmap.fit_transform(X)

    embedding_plot(mydmap, dim=3, scatter_kwargs = {'c': X_dmap[:,0], 'cmap':␣
↪'Spectral'})
    data_plot(mydmap, dim=3, scatter_kwargs = {'cmap': 'Spectral'})
    plt.show()
    print("SHAPE",X_dmap.shape)
```

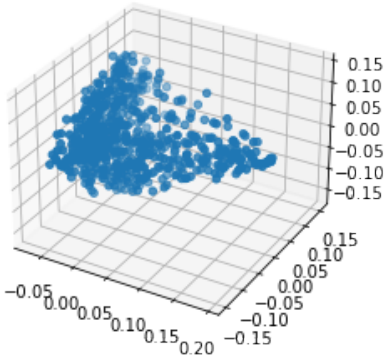
```
    ax = plt.axes(projection ="3d")
    ax.scatter3D(X_dmap[:,0], X_dmap[:,1], X_dmap[:,2])
    plt.show()

#     plt.scatter(X_dmap[:,0], X_dmap[:,1])
#     plt.show()

    X_diagrams = ripser.ripser(X_dmap, maxdim = 2)['dgms']

  ## draw persistence diagrams
    drawTDAtutorial(X_dmap,X_diagrams,R=0.1)

  ## draw persistence barcodes
    Barcode(X_diagrams).plot_barcode()

    return X_diagrams, Barcode(X_diagrams)
```

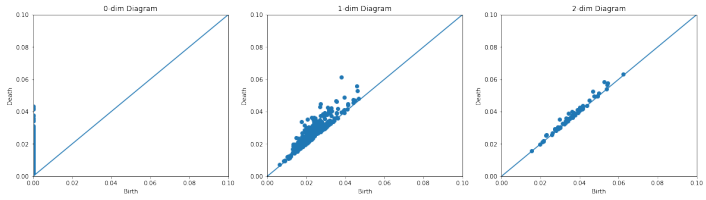
[9]:
```
X1 = retina_original_data[:,0,:]
X1.shape
X1_diagrams, X1_barcode = diffusion_tda(X1)

## isomap what info is lost and the specific distance measure
## possible loss of info: analyze and say something
## show the barcode


## compare with the traditional method: directly compute the distance between
## metric space: 698 points
# dsitance between two sequence of images
# two distance between images
# compare some of the distances
# goromov hausdorff distance between barcodes - carlsson review
# observe the diff compared to traditional distance
```

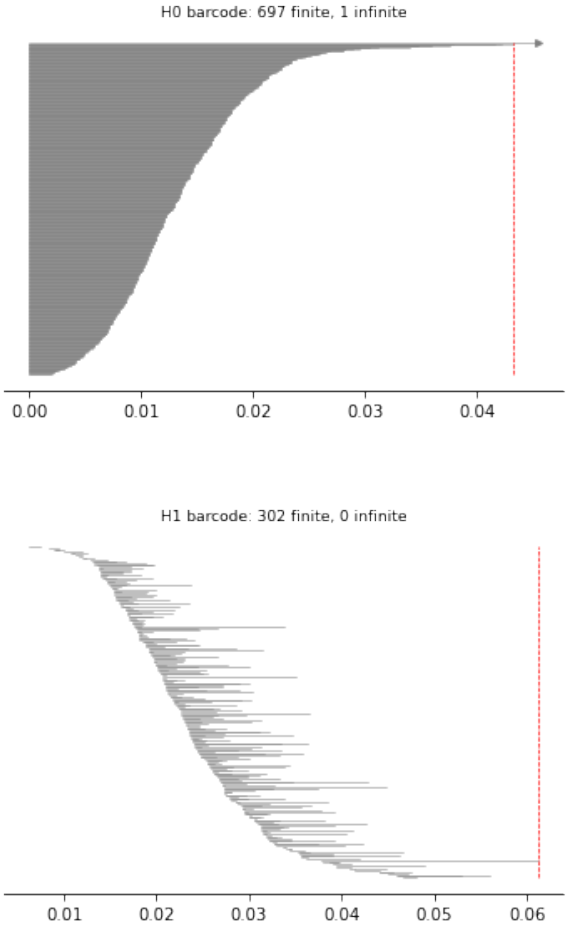

Embedding given by first three DCs.
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ3
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ2
−0.05
0.00
0.05
0.10
0.15
0.20
ψ1

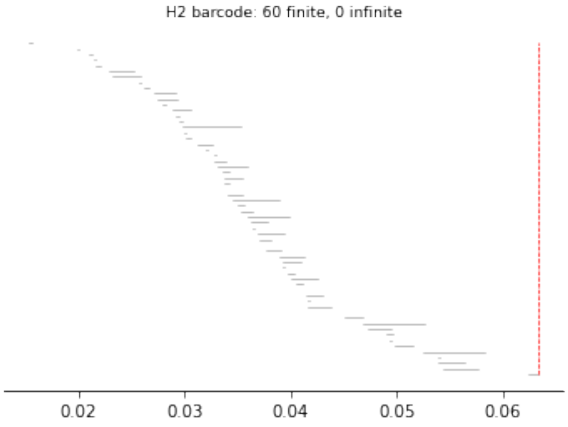

Data coloured with first DC.
0.10
0.08
0.06
z
0.04
0.02
0.00
0.12
0.10
0.08
0.06
y
0.04
0.02
0.00
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
x

```
SHAPE (698, 3)
```

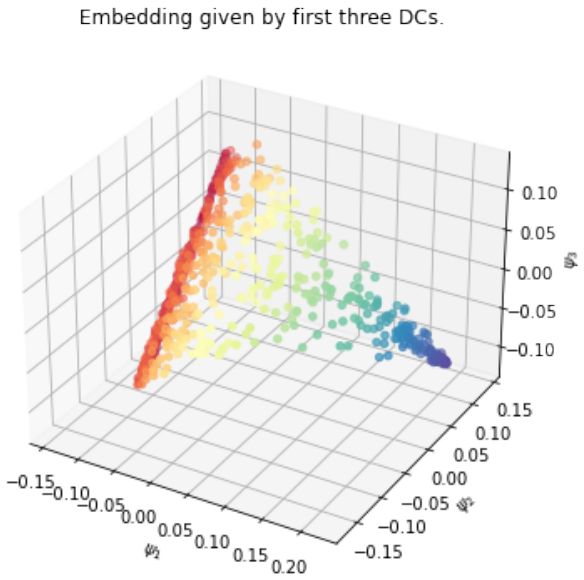

0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.05
0.00
0.05
0.10
0.15
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15

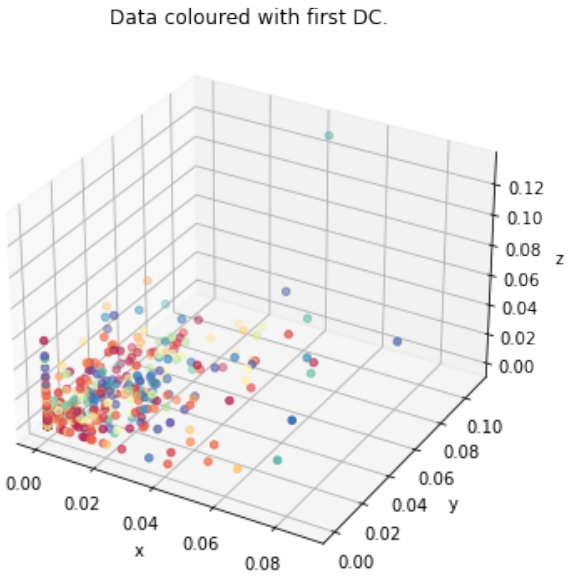

0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth


H0 barcode: 697 finite, 1 infinite
0.00
0.01
0.02
0.03
0.04
H1 barcode: 302 finite, 0 infinite
0.01
0.02
0.03
0.04
0.05
0.06

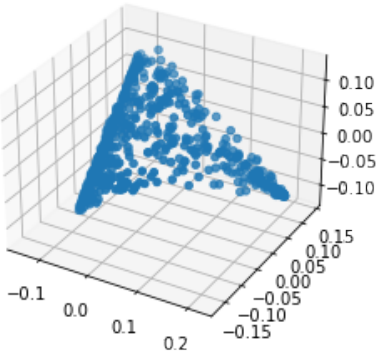


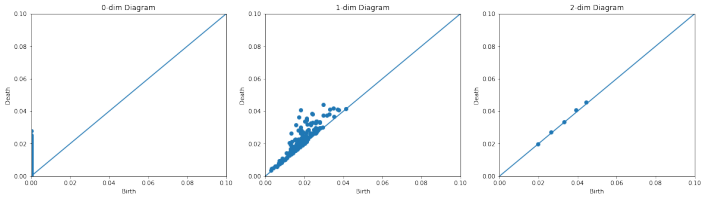
[10]:
```
X2 = retina_original_data[:,1,:]
X2_diagrams, X2_barcode = diffusion_tda(X2)
```

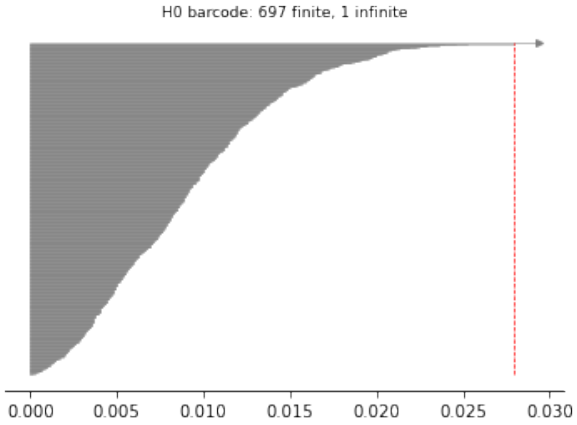

Embedding given by first three DCs.
0.10
0.05
0.00
−0.05
−0.10
ψ_3
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ_2
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
ψ_1

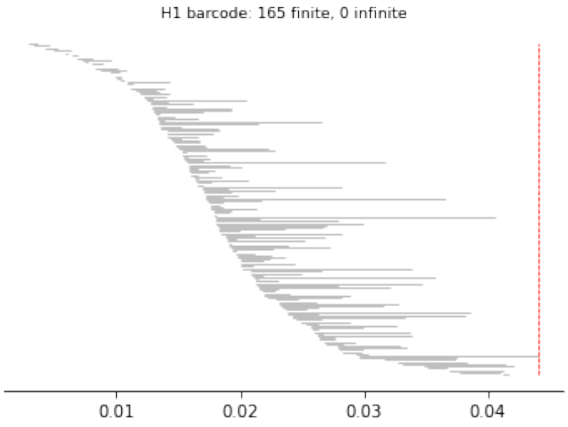

Data coloured with first DC.
0.12
0.10
0.08
z
0.06
0.04
0.02
0.00
0.10
0.08
0.06
y
0.04
0.02
0.00
0.00
0.02
0.04
x
0.06
0.08

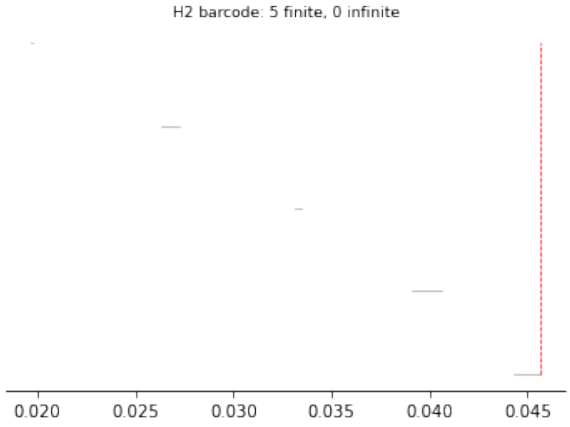
```
SHAPE (698, 3)
```


0.10
0.05
0.00
−0.05
−0.10
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.1
0.0
0.1
0.2

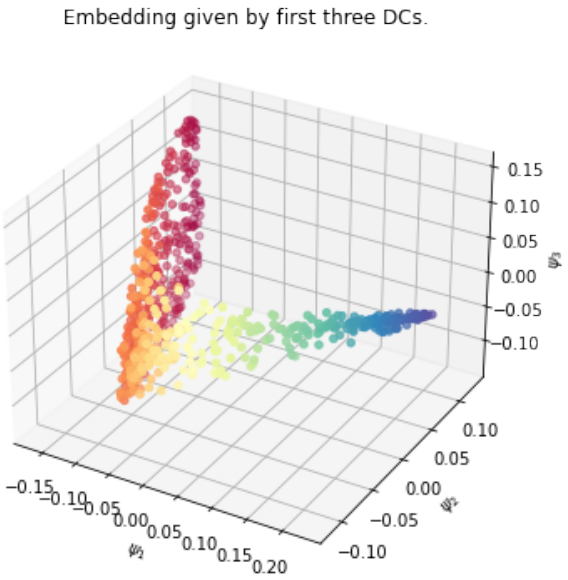

0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth

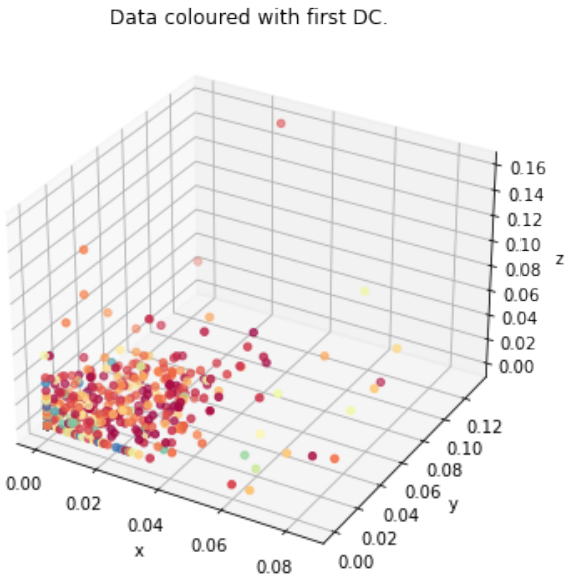

H0 barcode: 697 finite, 1 infinite
0.000
0.005
0.010
0.015
0.020
0.025
0.030


H1 barcode: 165 finite, 0 infinite
0.01
0.02
0.03
0.04

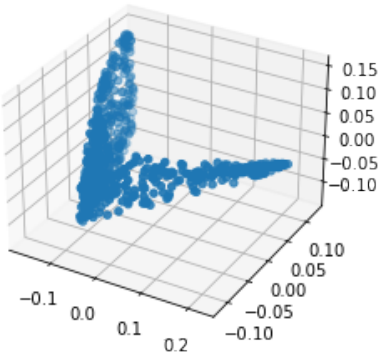


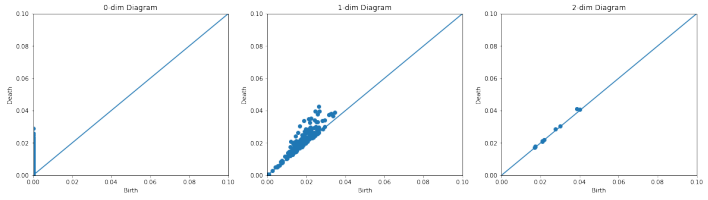
[11]:
```
X3 = retina_original_data[:,2,:]
X3_diagrams, X3_barcode = diffusion_tda(X3)
```

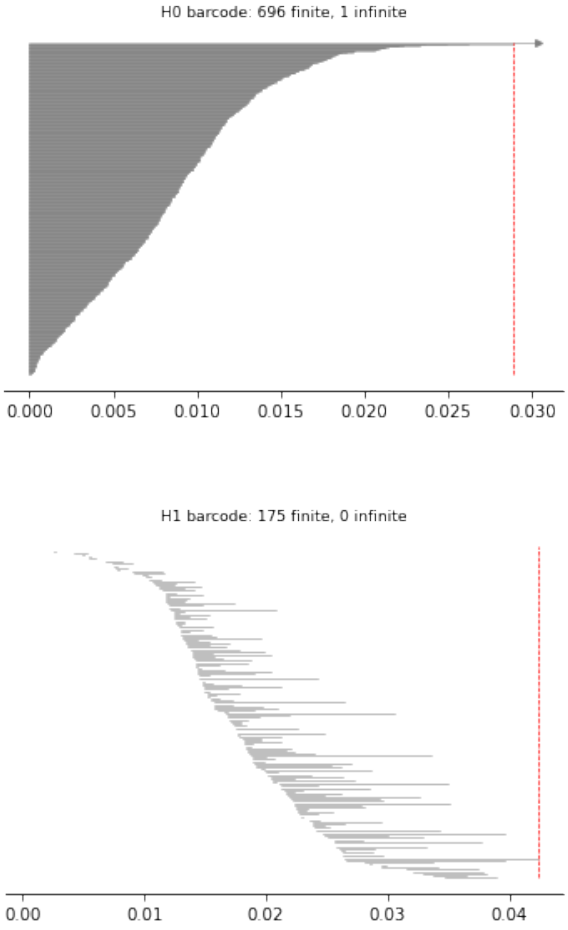

Embedding given by first three DCs.
0.15
0.10
0.05
0.00
−0.05
−0.10
ψ3
0.10
0.05
0.00
−0.05
−0.10
ψ2
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
ψ1

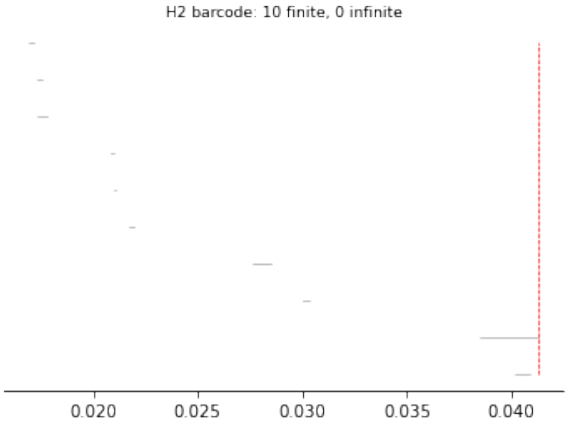

Data coloured with first DC.
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
z
0.12
0.10
0.08
0.06
0.04
0.02
0.00
y
0.00
0.02
0.04
0.06
0.08
x

```
SHAPE (698, 3)
```

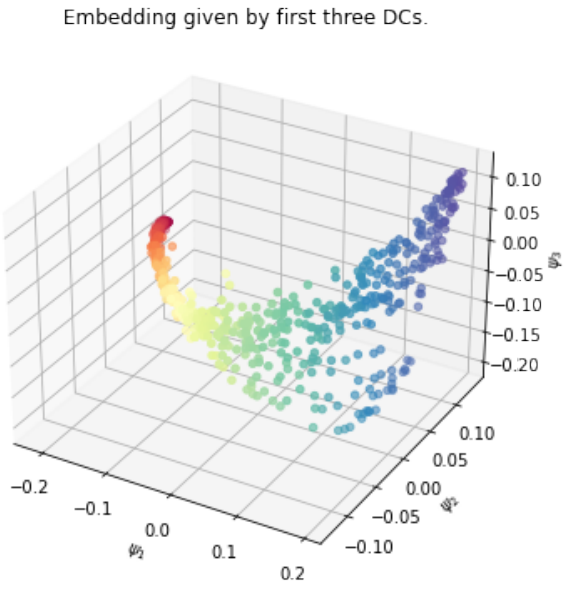

0.15
0.10
0.05
0.00
−0.05
−0.10
0.10
0.05
0.00
−0.05
−0.10
−0.1
0.0
0.1
0.2

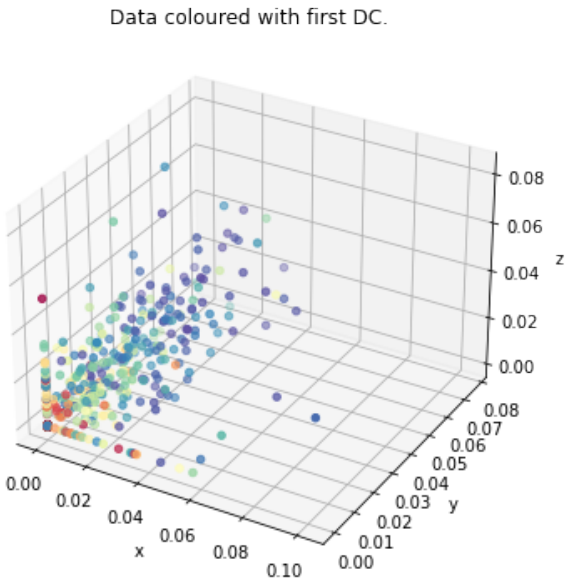

0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth


H0 barcode: 696 finite, 1 infinite
0.000
0.005
0.010
0.015
0.020
0.025
0.030
H1 barcode: 175 finite, 0 infinite
0.00
0.01
0.02
0.03
0.04

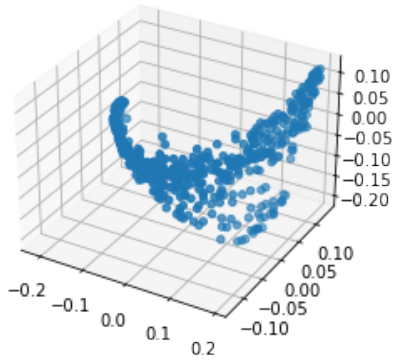


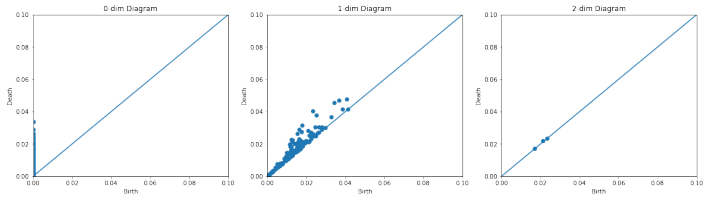
[12]:
```
X4 = retina_original_data[:,3,:]

X4_diagrams, X4_barcode = diffusion_tda(X4)
```

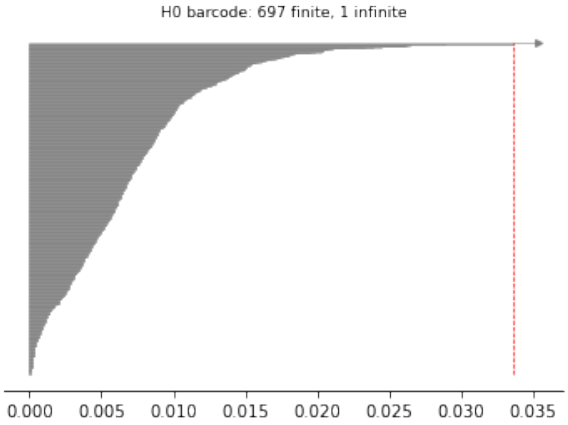

Embedding given by first three DCs.
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
ψ_3
0.10
0.05
0.00
−0.05
−0.10
ψ_2
−0.2
−0.1
0.0
0.1
0.2
ψ_1

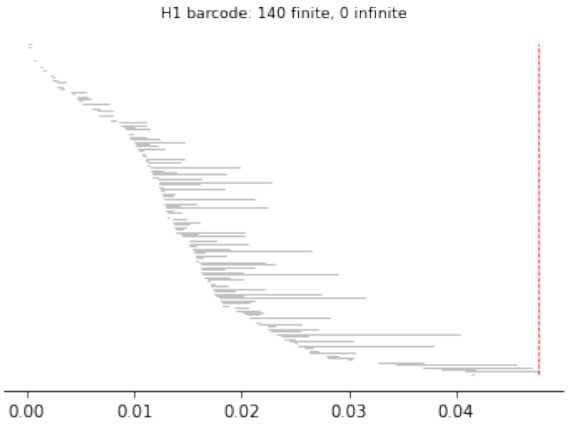

Data coloured with first DC.
0.08
0.06
0.04
0.02
0.00
z
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
y
0.00
0.02
0.04
0.06
0.08
0.10
x

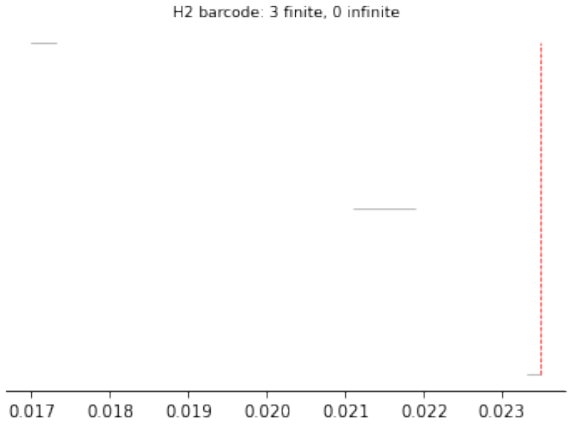
```
SHAPE (698, 3)
```


0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.2
−0.1
0.0
0.1
0.2
0.10
0.05
0.00
−0.05
−0.10

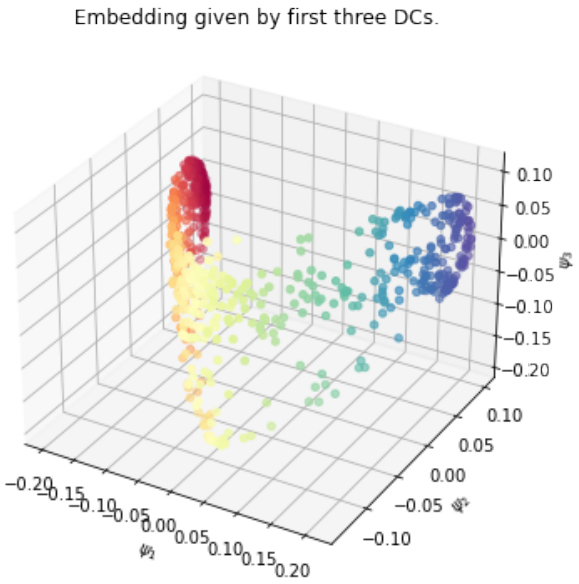

0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth

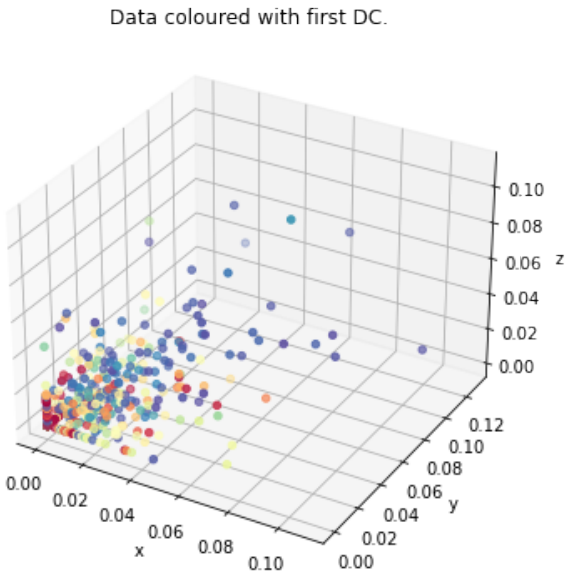

H0 barcode: 697 finite, 1 infinite
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035


H1 barcode: 140 finite, 0 infinite
0.00
0.01
0.02
0.03
0.04

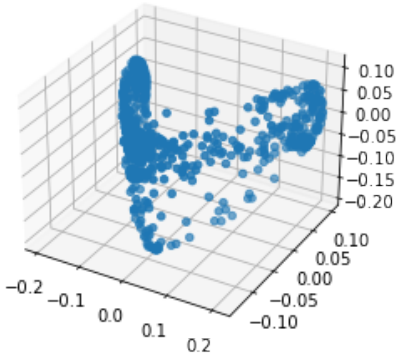


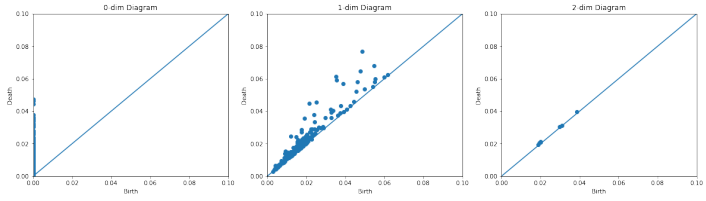
[13]:
```
X5 = retina_original_data[:,4,:]


X5_diagrams, X5_barcode = diffusion_tda(X5)
```

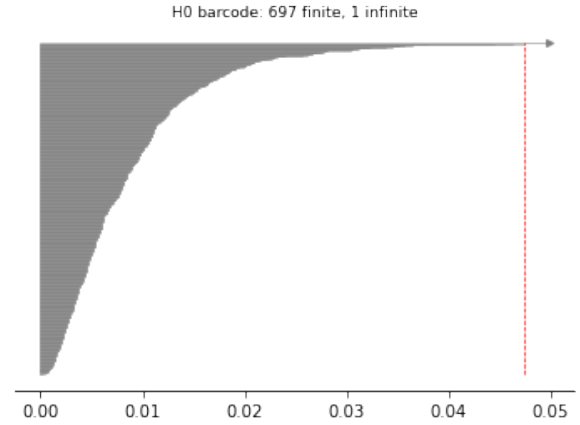

Embedding given by first three DCs.
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
ψ_3
0.10
0.05
0.00
−0.05
ψ_2
−0.10
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
ψ_1

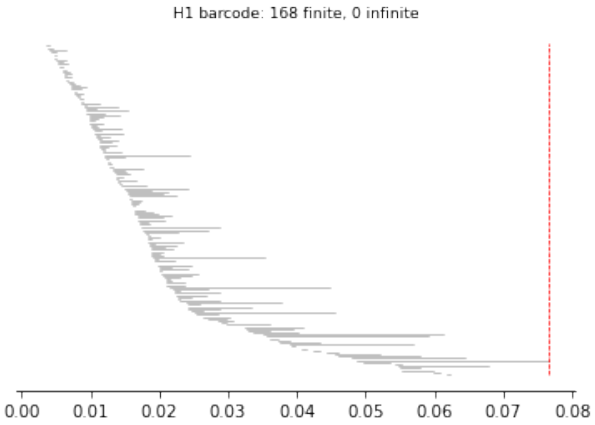

Data coloured with first DC.
0.10
0.08
0.06
z
0.04
0.02
0.00
0.12
0.10
0.08
0.06
y
0.04
0.02
0.00
0.00
0.02
0.04
0.06
x
0.08
0.10

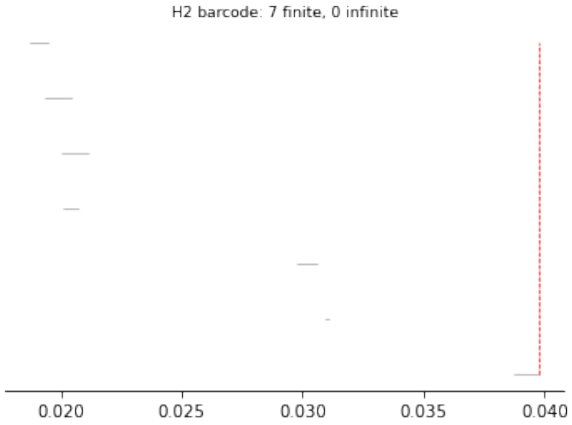
```
SHAPE (698, 3)
```


0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.2
−0.1
0.0
0.1
0.2
0.10
0.05
0.00
−0.05
−0.10

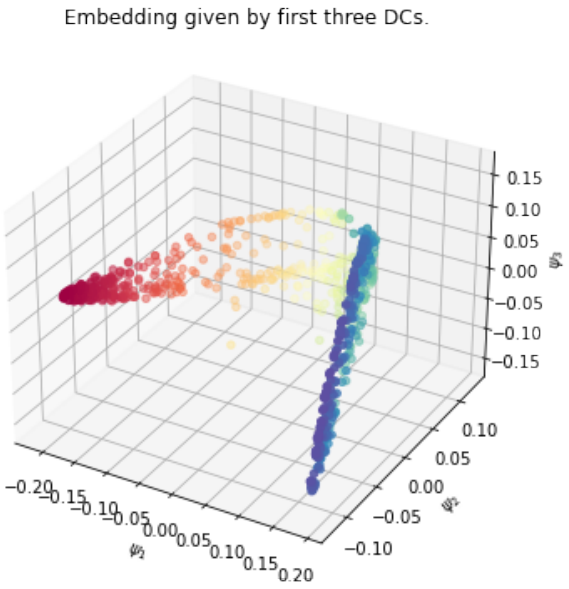

0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth

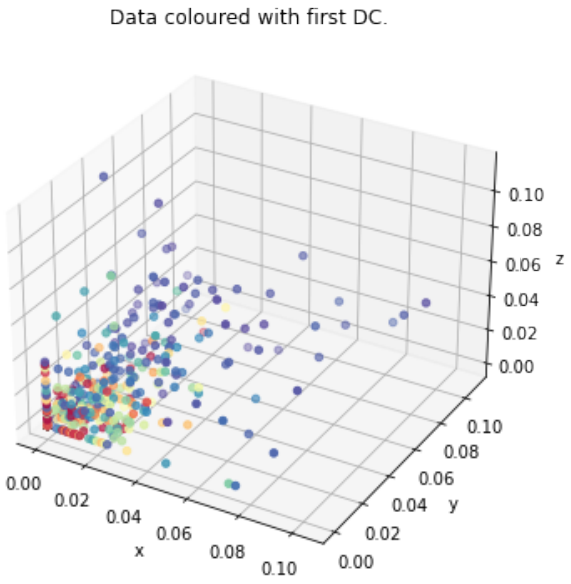

H0 barcode: 697 finite, 1 infinite
0.00
0.01
0.02
0.03
0.04
0.05


H1 barcode: 168 finite, 0 infinite
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08

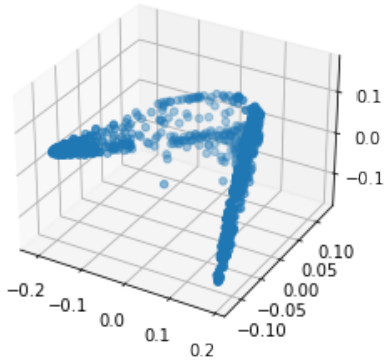


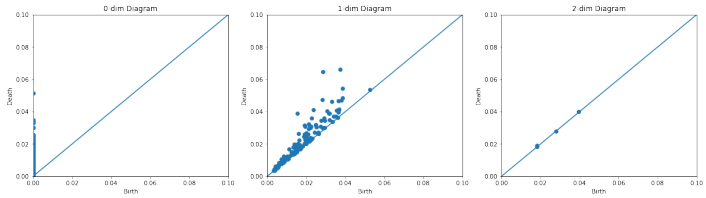
[14]:
```
X6 = retina_original_data[:,5,:]

X6_diagrams, X6_barcode = diffusion_tda(X6)
```

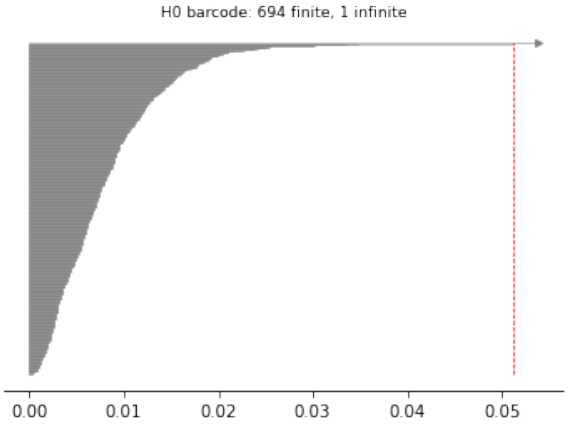

Embedding given by first three DCs.
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ3
0.10
0.05
0.00
−0.05
−0.10
ψ2
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
ψ1

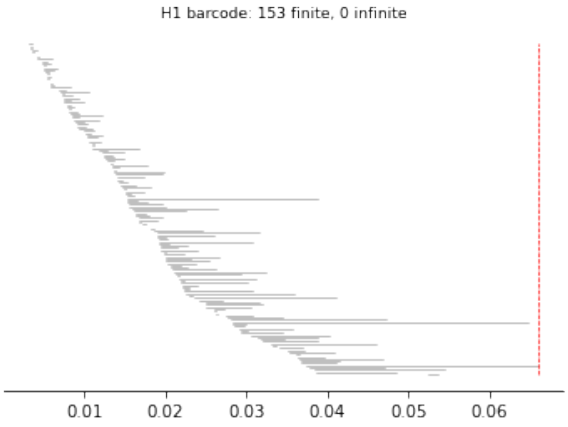

Data coloured with first DC.
0.10
0.08
0.06
z
0.04
0.02
0.00
0.10
0.08
0.06
0.04
y
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10
x

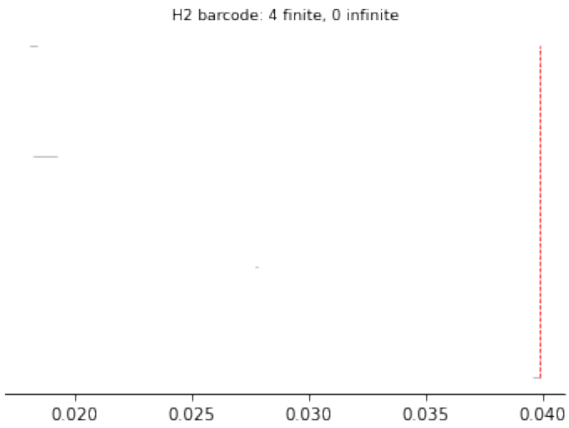
```
SHAPE (698, 3)
```


0.1
0.0
−0.1
0.10
0.05
0.00
−0.05
−0.10
−0.2
−0.1
0.0
0.1
0.2


0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth


H0 barcode: 694 finite, 1 infinite
0.00
0.01
0.02
0.03
0.04
0.05


H1 barcode: 153 finite, 0 infinite
0.01
0.02
0.03
0.04
0.05
0.06



0.5 Compute the pairwise Wasserstein distance:

useful resource: https://github.com/giotto-ai/giotto-tda/issues/603

0.5.1 first check that Wasserstein distance computed by two packages are the same:

[15]:
```
from gtda.diagrams import PairwiseDistance
pd = PairwiseDistance(['wasserstein'])
```

[16]:
```
pd_X1_X2_persim = persim.wasserstein(X1_diagrams[0], X2_diagrams[0])
```

```
C:\Users\ifisa\anaconda3\lib\site-packages\persim\wasserstein.py:51:
UserWarning: dgm1 has points with non-finite death times;ignoring those points
  warnings.warn(
C:\Users\ifisa\anaconda3\lib\site-packages\persim\wasserstein.py:61:
UserWarning: dgm2 has points with non-finite death times;ignoring those points
  warnings.warn(
```

[17]:
```
pd_X1_X2_persim
```

[17]:
```
2.7718896795122
```

[18]:
```
## check that the result is consistent with the persim package:
from gudhi.wasserstein import wasserstein_distance
from gudhi.hera import wasserstein_distance as hera
```

```
pd_X1_X2_gudhi = hera(X1_diagrams[0], X2_diagrams[0],internal_p=2)
pd_X1_X2_gudhi
```

```
POT (Python Optimal Transport) package is not installed. Try to run $ conda
install -c conda-forge pot ; or $ pip install POT
```

[18]:
```
2.7719287214918436
```

0.5.2 Compute all pairwise distance between the six point clouds:

[95]:
```
from gudhi.wasserstein import wasserstein_distance
from gudhi.hera import wasserstein_distance as hera
X_diagrams = []
X_diagrams.append(X1_diagrams)
X_diagrams.append(X2_diagrams)
X_diagrams.append(X3_diagrams)
X_diagrams.append(X4_diagrams)
X_diagrams.append(X5_diagrams)
X_diagrams.append(X6_diagrams)

pd_gudhi = np.zeros((6,6,3))
for dim in range(3):
    print(dim)
    print("--------------------")
    for i in range(6):
        print("$X" + "_" + str(i+1) + '$ &')
        for j in range(6):
#             if i != j:
            pd_gudhi[i,j,dim] = hera(X_diagrams[i][dim],␣
 ↪X_diagrams[j][dim],internal_p=2)
#                 print('Wasserstein distance between persistence diagrams for'␣
 ↪+ str(i+1) +' and '+ str(j+1) + ' (H' + str(dim) + ') is:')
            print(str(float("{0:.2f}".format(pd_gudhi[i,j,dim])))+'&')
        print("\\" + "\\")
        print("\hline")
```

```
0
--------------------
$X_1$ &
0.0&
2.77&
3.05&
3.95&
3.08&
3.42&
\\
\hline
```

```
$X_2$ &
2.77&
0.0&
0.43&
1.35&
1.0&
0.84&
\\
\hline
$X_3$ &
3.05&
0.43&
0.0&
1.03&
0.94&
0.66&
\\
\hline
$X_4$ &
3.95&
1.35&
1.03&
0.0&
1.11&
0.72&
\\
\hline
$X_5$ &
3.08&
1.0&
0.94&
1.11&
0.0&
0.51&
\\
\hline
$X_6$ &
3.42&
0.84&
0.66&
0.72&
0.51&
0.0&
\\
\hline
1
--------------------
$X_1$ &
```

```
0.0&
0.46&
0.5&
0.64&
0.64&
0.55&
\\
\hline
$X_2$ &
0.46&
0.0&
0.17&
0.29&
0.36&
0.3&
\\
\hline
$X_3$ &
0.5&
0.17&
0.0&
0.25&
0.33&
0.3&
\\
\hline
$X_4$ &
0.64&
0.29&
0.25&
0.0&
0.27&
0.29&
\\
\hline
$X_5$ &
0.64&
0.36&
0.33&
0.27&
0.0&
0.26&
\\
\hline
$X_6$ &
0.55&
0.3&
0.3&
```

```
0.29&
0.26&
0.0&
\\
\hline
2
-------------------
$X_1$ &
0.0&
0.06&
0.06&
0.06&
0.06&
0.06&
\\
\hline
$X_2$ &
0.06&
0.0&
0.01&
0.0&
0.01&
0.0&
\\
\hline
$X_3$ &
0.06&
0.01&
0.0&
0.0&
0.01&
0.0&
\\
\hline
$X_4$ &
0.06&
0.0&
0.0&
0.0&
0.0&
0.0&
\\
\hline
$X_5$ &
0.06&
0.01&
0.01&
0.0&
```

```
0.0&
0.0&
\\
\hline
$X_6$ &
0.06&
0.0&
0.0&
0.0&
0.0&
0.0&
\\
\hline
```

Pairwise Wasserstein distance between persistence diagrams for homology group H_0:

[20]:
```
print(pd_gudhi[:,:,0])
```

```
[[0.         2.77192872 3.04745634 3.95276261 3.07892065 3.41508927]
 [2.77192524 0.         0.42930521 1.34856909 0.99806364 0.84377773]
 [3.04747962 0.42930766 0.         1.03143331 0.94296253 0.65615236]
 [3.95283538 1.34850632 1.0315476  0.         1.10848366 0.72082808]
 [3.07892922 0.99806358 0.94295643 1.10848369 0.         0.51151537]
 [3.41507195 0.84376872 0.65615035 0.72081878 0.51151656 0.        ]]
```

Pairwise Wasserstein distance between persistence diagrams for homology group H_1:

[21]:
```
print(pd_gudhi[:,:,1])
```

```
[[0.         0.45973786 0.50093082 0.64066787 0.63832201 0.54722478]
 [0.45974146 0.         0.16690206 0.29471953 0.36368453 0.30304126]
 [0.50092305 0.1668978  0.         0.24557967 0.33290296 0.29892867]
 [0.64068023 0.29472335 0.24558021 0.         0.26870174 0.28557107]
 [0.63833715 0.36368295 0.33290654 0.26869569 0.         0.26458892]
 [0.54724428 0.30303734 0.29893539 0.28556935 0.26458771 0.        ]]
```

Pairwise Wasserstein distance between persistence diagrams for homology group H_2:

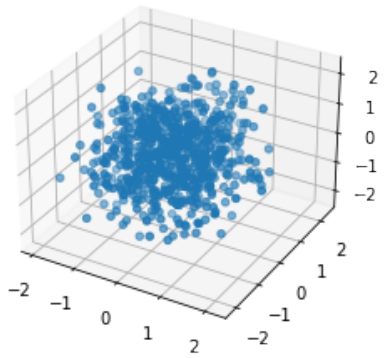
[22]:
```
print(pd_gudhi[:,:,2])
```

```
[[0.         0.05800773 0.05728896 0.0593703  0.06085189 0.0596126 ]
 [0.05800773 0.         0.00500477 0.00380341 0.00596403 0.0036489 ]
 [0.05728896 0.00500477 0.         0.00485482 0.00647532 0.00494322]
 [0.0593703  0.00380341 0.00485482 0.         0.00479106 0.00202047]
 [0.06085189 0.00596403 0.00647532 0.00479106 0.         0.00432514]
 [0.0596126  0.0036489  0.00494322 0.00202047 0.00432514 0.        ]]
```

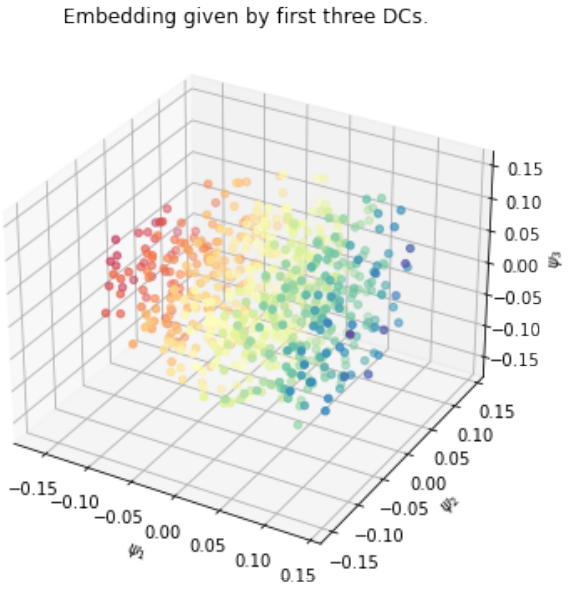
0.5.3 compare with known shape, eg a circle:

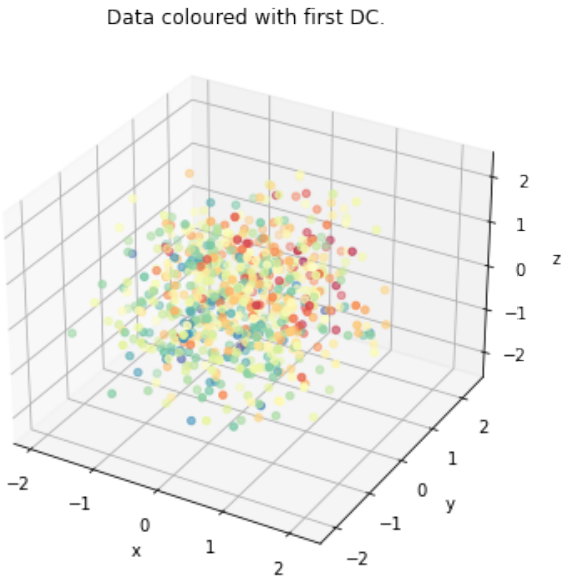
[23]:
```
import tadasets
```

[24]:
```
dsphere = tadasets.dsphere(n=698, d=12, r=3.14, ambient=14, noise=0.14)
ax = plt.axes(projection ="3d")
ax.scatter3D(dsphere[:,0], dsphere[:,1], dsphere[:,2])
plt.show()
```

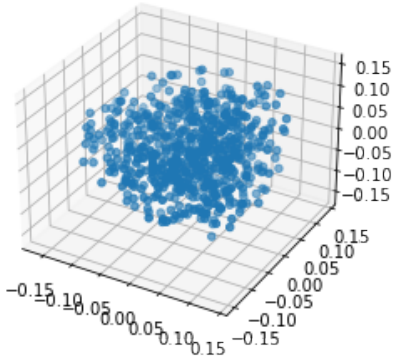


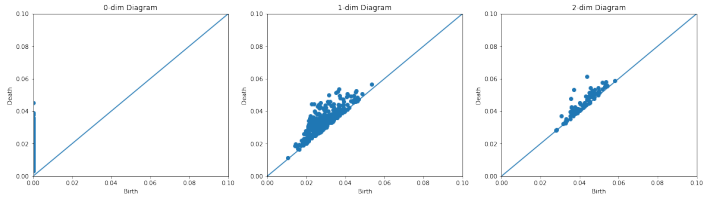
[25]:
```
dsphere_diagrams, dsphere_barcode = diffusion_tda(dsphere)
```

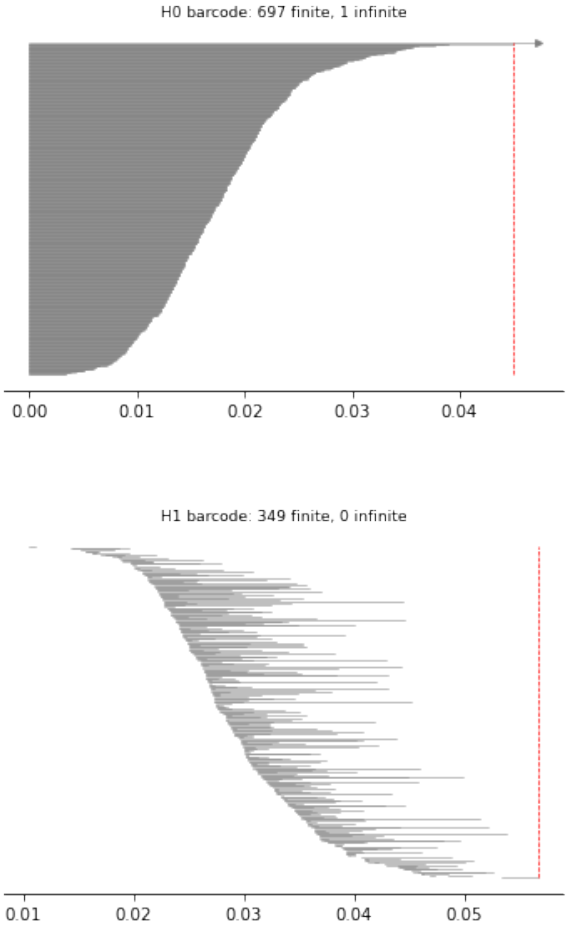

Embedding given by first three DCs.
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ3
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ2
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
ψ1

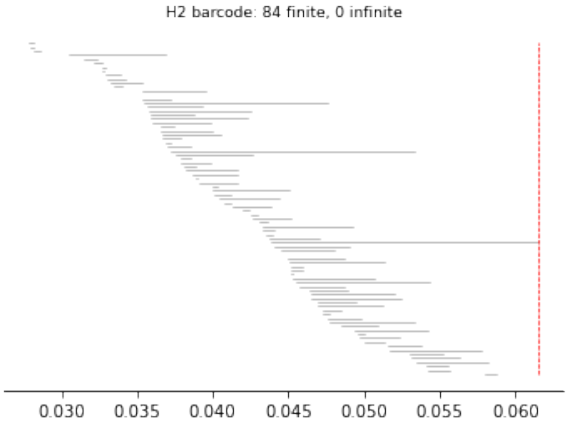



```
SHAPE (698, 3)
```

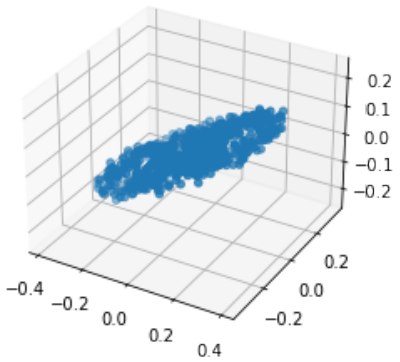

0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15


0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth


H0 barcode: 697 finite, 1 infinite
0.00
0.01
0.02
0.03
0.04
H1 barcode: 349 finite, 0 infinite
0.01
0.02
0.03
0.04
0.05



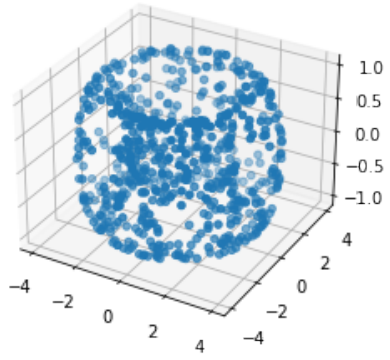
[26]:
```
torus = tadasets.torus(n=698, c=2, a=1, ambient=200, noise=0.2)
ax = plt.axes(projection ="3d")
ax.scatter3D(torus[:,0], torus[:,1], torus[:,2])
plt.show()
```



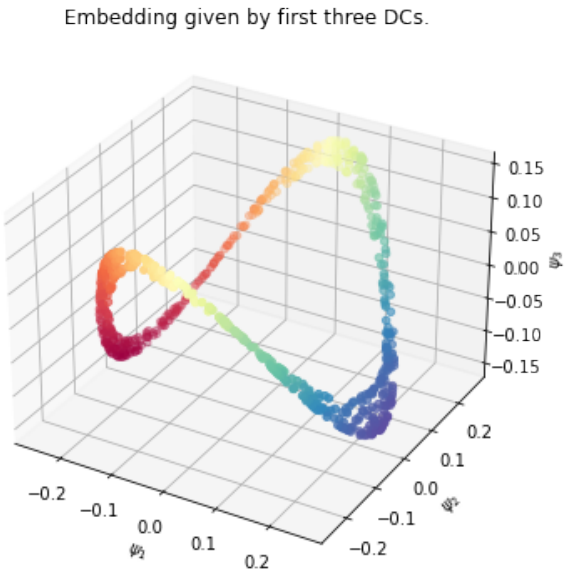
[42]:
```
class TorusSampler():

    def __init__(self, r=1, R=3, x0=0, y0=0, z0=0):
        self._r = r
        self._R = R
        self._x0 = x0
        self._y0 = y0
        self._z0 = z0

    '''
    This sampling may not be uniform
    '''
    def Sample(self, n):
        u = 2 * np.pi * np.random.rand(n)
        v = 2 * np.pi * np.random.rand(n)

        cosu = np.cos(u)
        sinu = np.sin(u)

        cosv = np.cos(v)
        sinv = np.sin(v)

        x = self._x0 + (self._R + self._r * cosu) * cosv
        y = self._y0 + (self._R + self._r * cosu) * sinv
        z = self._z0 + self._r * sinu

        return np.array([x, y, z]).T
```

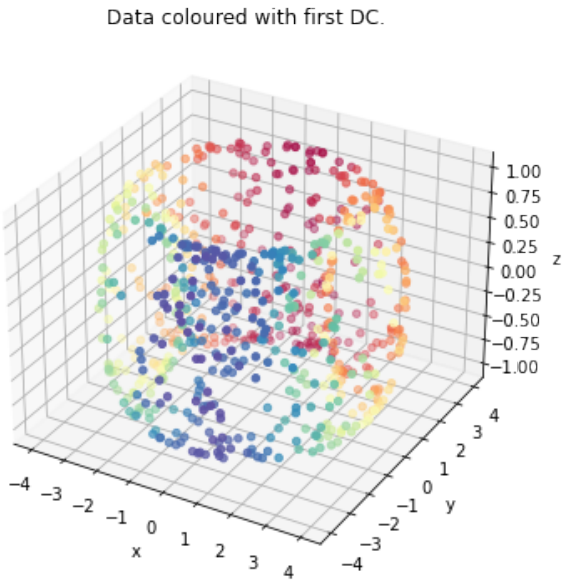
[77]:
```
torus = TorusSampler(1, 3).Sample(698)
```

[78]:
```
ax = plt.axes(projection ="3d")
ax.scatter3D(torus[:,0], torus[:,1], torus[:,2])
plt.show()
```

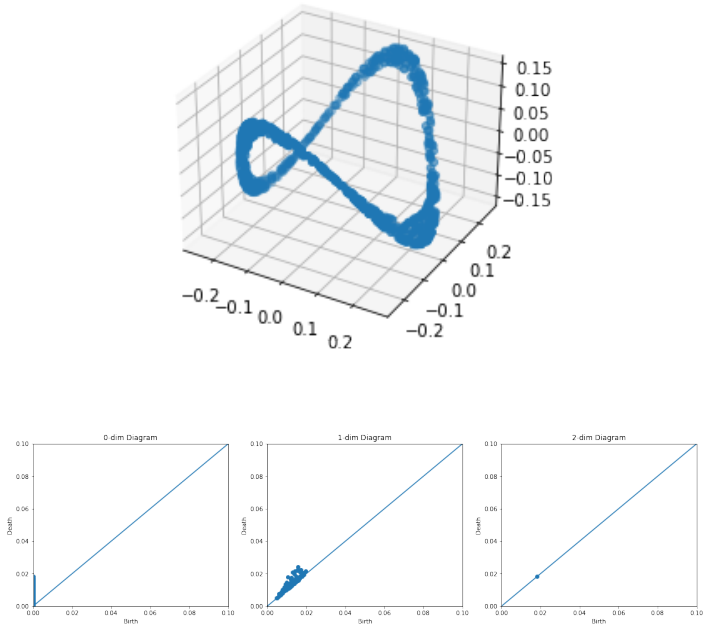


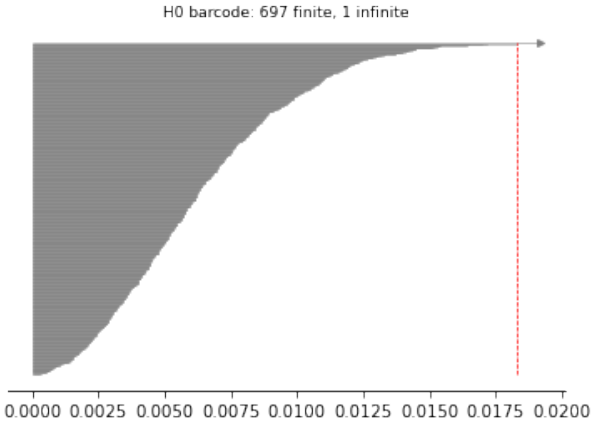
[79]:
```
torus_diagrams, torus_barcode = diffusion_tda(torus)
```

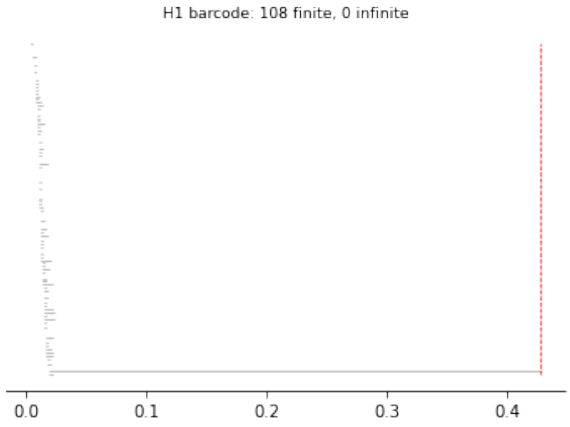

Embedding given by first three DCs.
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
ψ_3
0.2
0.1
0.0
−0.1
−0.2
ψ_2
−0.2
−0.1
0.0
0.1
0.2
ψ_1

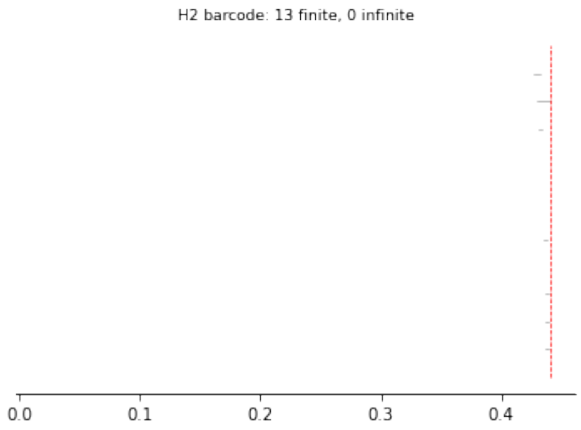

Data coloured with first DC.
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
z
−4 −3 −2 −1 0 1 2 3 4
x
−4 −3 −2 −1 0 1 2 3 4
y

```
SHAPE (698, 3)
```


0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0.2
0.1
0.0
−0.1
−0.2
−0.2
−0.1
0.0
0.1
0.2
0 dim Diagram
1 dim Diagram
2 dim Diagram
Death
Birth


H0 barcode: 697 finite, 1 infinite
0.0000 0.0025 0.0050 0.0075 0.0100 0.0125 0.0150 0.0175 0.0200


H1 barcode: 108 finite, 0 infinite
0.0 0.1 0.2 0.3 0.4



[76]:
```
from gudhi.wasserstein import wasserstein_distance
from gudhi.hera import wasserstein_distance as hera
X_diagrams = []
X_diagrams.append(X1_diagrams)
X_diagrams.append(X2_diagrams)
X_diagrams.append(X3_diagrams)
X_diagrams.append(X4_diagrams)
X_diagrams.append(X5_diagrams)
X_diagrams.append(X6_diagrams)

pd_dsphere_gudhi = np.zeros((6,3))
for dim in range(3):
    for i in range(6):
#         print(X_diagrams[i][dim])
#         print(dsphere_diagrams[dim])
        pd_dsphere_gudhi[i,dim] = hera(X_diagrams[i][dim],␣
↪dsphere_diagrams[dim])
        print('Wasserstein distance between persistence diagrams for circle and␣
↪point cloud ' + str(i+1) + ' (H' + str(dim) + ') is:')
        print(pd_dsphere_gudhi[i,dim])
```

```
Wasserstein distance between persistence diagrams for circle and point cloud 1
(H0) is:
2.4809560721041635
Wasserstein distance between persistence diagrams for circle and point cloud 2
```

```
(H0) is:
4.463884397916445
Wasserstein distance between persistence diagrams for circle and point cloud 3
(H0) is:
4.641338810155503
Wasserstein distance between persistence diagrams for circle and point cloud 4
(H0) is:
5.17980453916789
Wasserstein distance between persistence diagrams for circle and point cloud 5
(H0) is:
4.531922676127579
Wasserstein distance between persistence diagrams for circle and point cloud 6
(H0) is:
4.827321461983956
Wasserstein distance between persistence diagrams for circle and point cloud 1
(H1) is:
0.6241310045588762
Wasserstein distance between persistence diagrams for circle and point cloud 2
(H1) is:
0.7671945061301813
Wasserstein distance between persistence diagrams for circle and point cloud 3
(H1) is:
0.7996520071465056
Wasserstein distance between persistence diagrams for circle and point cloud 4
(H1) is:
0.866591851816338
Wasserstein distance between persistence diagrams for circle and point cloud 5
(H1) is:
0.8496004594489932
Wasserstein distance between persistence diagrams for circle and point cloud 6
(H1) is:
0.7646189627703279
Wasserstein distance between persistence diagrams for circle and point cloud 1
(H2) is:
0.10986399976536632
Wasserstein distance between persistence diagrams for circle and point cloud 2
(H2) is:
0.12334374524652958
Wasserstein distance between persistence diagrams for circle and point cloud 3
(H2) is:
0.12340327631682158
Wasserstein distance between persistence diagrams for circle and point cloud 4
(H2) is:
0.12458728905767202
Wasserstein distance between persistence diagrams for circle and point cloud 5
(H2) is:
0.12582713179290295
Wasserstein distance between persistence diagrams for circle and point cloud 6
```

```
(H2) is:
0.12466537859290838
```

[80]:
```
from gudhi.wasserstein import wasserstein_distance
from gudhi.hera import wasserstein_distance as hera
X_diagrams = []
X_diagrams.append(X1_diagrams)
X_diagrams.append(X2_diagrams)
X_diagrams.append(X3_diagrams)
X_diagrams.append(X4_diagrams)
X_diagrams.append(X5_diagrams)
X_diagrams.append(X6_diagrams)

pd_torus_gudhi = np.zeros((6,3))
for dim in range(3):
    for i in range(6):
#         print(X_diagrams[i][dim])
#         print(dsphere_diagrams[dim])
        pd_torus_gudhi[i,dim] = hera(X_diagrams[i][dim], torus_diagrams[dim])
        print('Wasserstein distance between persistence diagrams for circle and␣
 ↪point cloud ' + str(i+1) + ' (H' + str(dim) + ') is:')
        print(pd_torus_gudhi[i,dim])
```

```
Wasserstein distance between persistence diagrams for circle and point cloud 1
(H0) is:
3.6310234030825086
Wasserstein distance between persistence diagrams for circle and point cloud 2
(H0) is:
1.4639444853646637
Wasserstein distance between persistence diagrams for circle and point cloud 3
(H0) is:
1.3077519815615233
Wasserstein distance between persistence diagrams for circle and point cloud 4
(H0) is:
0.7306229655132483
Wasserstein distance between persistence diagrams for circle and point cloud 5
(H0) is:
1.3777969512157142
Wasserstein distance between persistence diagrams for circle and point cloud 6
(H0) is:
1.0552870598621666
Wasserstein distance between persistence diagrams for circle and point cloud 1
(H1) is:
0.7361079549882561
Wasserstein distance between persistence diagrams for circle and point cloud 2
(H1) is:
0.48583426291588694
Wasserstein distance between persistence diagrams for circle and point cloud 3
```

```
(H1) is:
0.4428885536908638
Wasserstein distance between persistence diagrams for circle and point cloud 4
(H1) is:
0.3395286252562073
Wasserstein distance between persistence diagrams for circle and point cloud 5
(H1) is:
0.45408774679526687
Wasserstein distance between persistence diagrams for circle and point cloud 6
(H1) is:
0.468785529024899
Wasserstein distance between persistence diagrams for circle and point cloud 1
(H2) is:
0.05699422350153327
Wasserstein distance between persistence diagrams for circle and point cloud 2
(H2) is:
0.017472262494266033
Wasserstein distance between persistence diagrams for circle and point cloud 3
(H2) is:
0.018393791280686855
Wasserstein distance between persistence diagrams for circle and point cloud 4
(H2) is:
0.016043948009610176
Wasserstein distance between persistence diagrams for circle and point cloud 5
(H2) is:
0.018177962861955166
Wasserstein distance between persistence diagrams for circle and point cloud 6
(H2) is:
0.01596159115433693
```

[97]:
```
print("Comparing six point-clouds with 3-sphere (H0):")
print(pd_dsphere_gudhi[:,0])
print("Comparing six point-clouds with torus (H0):")
print(pd_torus_gudhi[:,0])
```

```
Comparing six point-clouds with 3-sphere (H0):
[2.48095607 4.4638844  4.64133881 5.17980454 4.53192268 4.82732146]
Comparing six point-clouds with torus (H0):
[3.6310234  1.46394449 1.30775198 0.73062297 1.37779695 1.05528706]
```

[82]:
```
print("Comparing six point-clouds with 3-sphere (H1):")
print(pd_dsphere_gudhi[:,1])
print("Comparing six point-clouds with torus (H1):")
print(pd_torus_gudhi[:,1])
```

```
Comparing six point-clouds with 3-sphere (H1):
[0.624131   0.76719451 0.79965201 0.86659185 0.84960046 0.76461896]
Comparing six point-clouds with torus (H1):
```

```
[0.73610795 0.48583426 0.44288855 0.33952863 0.45408775 0.46878553]
```

[83]:
```
print("Comparing six point-clouds with 3-sphere (H2):")
print(pd_dsphere_gudhi[:,2])
print("Comparing six point-clouds with torus (H2):")
print(pd_torus_gudhi[:,2])
```

```
Comparing six point-clouds with 3-sphere (H2):
[0.109864   0.12334375 0.12340328 0.12458729 0.12582713 0.12466538]
Comparing six point-clouds with torus (H2):
[0.05699422 0.01747226 0.01839379 0.01604395 0.01817796 0.01596159]
```

List of Tables

List of Figures