Otter et al. *EPJ Data Science* (2017) 6:17
DOI 10.1140/epjds/s13688-017-0109-5

EPJ.org

EPJ Data Science
a SpringerOpen Journal

REGULAR ARTICLE Open Access

A roadmap for the computation of persistent homology

Nina Otter[1,3], Mason A Porter[4,1,2*], Ulrike Tillmann[1,3], Peter Grindrod[1] and Heather A Harrington[1]

*Correspondence:
mason@math.ucla.edu
[4]Department of Mathematics, UCLA, Los Angeles, CA 90095, USA
Full list of author information is available at the end of the article

Abstract

Persistent homology (PH) is a method used in topological data analysis (TDA) to study qualitative features of data that persist across multiple scales. It is robust to perturbations of input data, independent of dimensions and coordinates, and provides a compact representation of the qualitative features of the input. The computation of PH is an open area with numerous important and fascinating challenges. The field of PH computation is evolving rapidly, and new algorithms and software implementations are being updated and released at a rapid pace. The purposes of our article are to (1) introduce theory and computational methods for PH to a broad range of computational scientists and (2) provide benchmarks of state-of-the-art implementations for the computation of PH. We give a friendly introduction to PH, navigate the pipeline for the computation of PH with an eye towards applications, and use a range of synthetic and real-world data sets to evaluate currently available open-source implementations for the computation of PH. Based on our benchmarking, we indicate which algorithms and implementations are best suited to different types of data sets. In an accompanying tutorial, we provide guidelines for the computation of PH. We make publicly available all scripts that we wrote for the tutorial, and we make available the processed version of the data sets used in the benchmarking.

Keywords: persistent homology; topological data analysis; point-cloud data; networks

1 Introduction

The amount of available data has increased dramatically in recent years, and this situation — which will only become more extreme — necessitates the development of innovative and efficient data-processing methods. Making sense of the vast amount of data is difficult: on one hand, the sheer size of the data poses challenges; on the other hand, the complexity of the data, which includes situations in which data is noisy, high-dimensional, and/or incomplete, is perhaps an even more significant challenge. The use of clustering techniques and other ideas from areas such as computer science, machine learning, and uncertainty quantification — along with mathematical and statistical models — are often very useful for data analysis (see, e.g., [1–4] and many other references). However, recent mathematical developments are shedding new light on such 'traditional' ideas, forging new approaches of their own, and helping people to better decipher increasingly complicated structure in data.

Techniques from the relatively new subject of 'topological data analysis' (TDA) have provided a wealth of new insights in the study of data in an increasingly diverse set of applications — including sensor-network coverage [5], proteins [6–9], 3-dimensional structure of DNA [10], development of cells [11], stability of fullerene molecules [12], robotics [13–15], signals in images [16, 17], periodicity in time series [18], cancer [19–22], phylogenetics [23–25], natural images [26], the spread of contagions [27, 28], self-similarity in geometry [29], materials science [30–33], financial networks [34, 35], diverse applications in neuroscience [36–43], classification of weighted networks [44], collaboration networks [45, 46], analysis of mobile phone data [47], collective behavior in biology [48], time-series output of dynamical systems [49], natural-language analysis [50], and more. There are numerous others, and new applications of TDA appear in journals and preprint servers increasingly frequently. There are also interesting computational efforts, such as [51].

TDA is a field that lies at the intersection of data analysis, algebraic topology, computational geometry, computer science, statistics, and other related areas. The main goal of TDA is to use ideas and results from geometry and topology to develop tools for studying qualitative features of data. To achieve this goal, one needs precise definitions of qualitative features, tools to compute them in practice, and some guarantee about the robustness of those features. One way to address all three points is a method in TDA called *persistent homology* (PH). This method is appealing for applications because it is based on algebraic topology, which gives a well-understood theoretical framework to study qualitative features of data with complex structure, is computable via linear algebra, and is robust with respect to small perturbations in input data.

Types of data sets that can be studied with PH include finite metric spaces, digital images, level sets of real-valued functions, and networks (see Section 5.1). In the next two paragraphs, we give some motivation for the main ideas of persistent homology by discussing two examples of such data sets.

Finite metric spaces are also called point-cloud data sets in the TDA literature. From a topological point of view, finite metric spaces do not contain any interesting information. One thus considers a thickening of a point cloud at different scales of resolution and then analyzes the evolution of the resulting shape across the different resolution scales. The qualitative features are given by topological invariants, and one can represent the variation of such invariants across the different resolution scales in a compact way to summarize the 'shape' of the data.

As an illustration, consider the set of points in $\mathbb{R}^2$ that we show in Figure 1. Let ϵ, which we interpret as a distance parameter, be a nonnegative real number (so $\epsilon = 0$ gives the set of points). For different values of ϵ, we construct a space S_ϵ composed of vertices, edges, triangles, and higher-dimensional polytopes according to the following rule: We include an edge between two points i and j if and only if the Euclidean distance between them is no larger than ϵ; we include a triangle if and only if all of its edges are in S_ϵ; we include a tetrahedron if and only if all of its face triangles are in S_ϵ; and so on. For $\epsilon \leq \epsilon'$, it then follows that the space S_ϵ is contained in the space $S_{\epsilon'}$. This yields a nested sequence of spaces, as we illustrate in Figure 1(a). Our construction of nested spaces gives an example of a 'filtered Vietoris–Rips complex,' which we define and discuss in Section 5.2.

By using homology, a tool in algebraic topology, one can measure several features of the spaces S_ϵ — including the numbers of components, holes, and voids (higher-dimensional versions of holes). One can then represent the lifetime of such features using a finite collec-

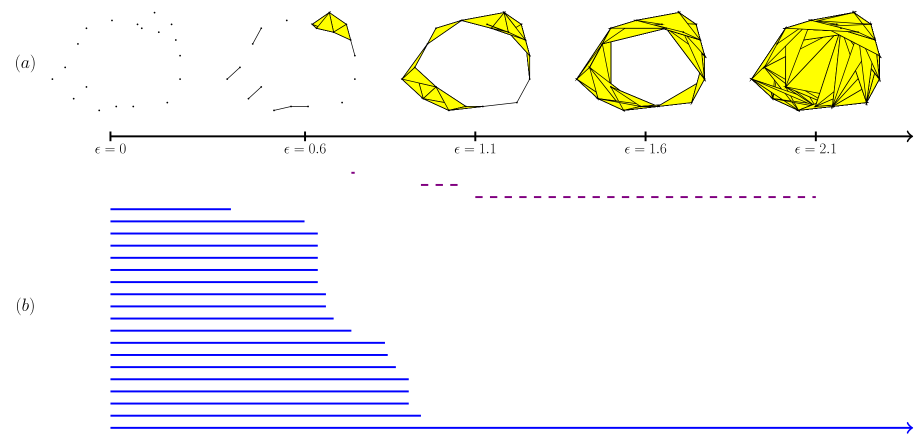


Figure 1 Example of persistent homology for a point cloud. (a) A finite set of points in $\mathbb{R}^2$ (for $\epsilon = 0$) and a nested sequence of spaces obtained from it (from $\epsilon = 0$ to $\epsilon = 2.1$). **(b)** Barcode for the nested sequence of spaces illustrated in (a). Solid lines represent the lifetime of components, and dashed lines represent the lifetime of holes.

tion of intervals known as a 'barcode.' Roughly, the left endpoint of an interval represents the birth of a feature, and its right endpoint represents the death of the same feature. In Figure 1(b), we reproduce such intervals for the number of components (blue solid lines) and the number of holes (violet dashed lines). In Figure 1(b), we observe a dashed line that is significantly longer than the other dashed lines. This indicates that the data set has a long-lived hole. By contrast, in this example one can potentially construe the shorter dashed lines as noise. (However, note that while widespread, such an intepretation is not correct in general; for applications in which one considers some short and medium-sized intervals as features rather than noise, see [52, 53].) When a feature is still 'alive' at the largest value of ϵ that we consider, the lifetime interval is an infinite interval, which we indicate by putting an arrowhead at the right endpoint of the interval. In Figure 1(b), we see that there is exactly one solid line that lives up to $\epsilon = 2.1$. One can use information about shorter solid lines to extract information about how data is clustered in a similar way as with linkage-clustering methods [3].

One of the most challenging parts of using PH is statistical interpretation of results. From a statistical point of view, a barcode like the one in Figure 1(b) is an unknown quantity that one is trying to estimate; one therefore needs methods for quantitatively assessing the quality of the barcodes that one obtains with computations. The challenge is twofold. On one hand, there is a cultural obstacle: practitioners of TDA often have backgrounds in pure topology and are not well-versed in statistical approaches to data analysis [54]. On the other hand, the space of barcodes lacks geometric properties that would make it easy to define basic concepts such as mean, median, and so on. Current research is focused both on studying geometric properties of this space and on studying methods that map this space to spaces that have better geometric properties for statistics. In Section 5.4, we give a brief overview of the challenges and current approaches for statistical interpretation of barcodes. This is an active area of research and an important endeavor, as few statistical tools are currently available for interpreting results in applications of PH.

We now discuss a second example related to digital images. (For an illustration, see Figure 2(a).) Digital images have a cubical structure, given by the pixels (for 2-dimensional

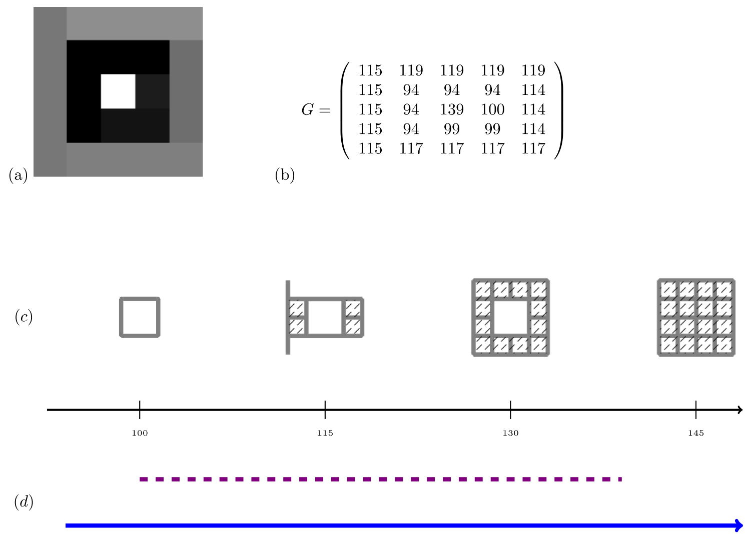


Figure 2 Example of persistent homology for a gray-scale digital image. (a) A gray-scale image, **(b)** the matrix of gray values, **(c)** the filtered cubical complex associated to the digital image, and **(d)** the barcode for the nested sequence of spaces in panel (c). A solid line represents the lifetime of a component, and a dashed line represents the lifetime of a hole.

digital images) or voxels (for 3-dimensional images). Therefore, one approach to study digital images uses combinatorial structures called 'cubical complexes.' (For a different approach to the study of digital images, see Section 5.1.) Roughly, cubical complexes are topological spaces built from a union of vertices, edges, squares, cubes, and higher-dimensional hypercubes. An efficient way [55] to build a cubical complex from a 2-dimensional digital image consists of assigning a vertex to every pixel, then joining vertices corresponding to adjacent pixels by an edge, and filling in the resulting squares. One proceeds in a similar way for 3-dimensional images. One then labels every vertex with an integer that corresponds to the gray value of the pixel, and one labels edges (respectively, squares) with the maximum of the values of the adjacent vertices (respectively, edges). One can then construct a nested sequence of cubical complexes $C_0 \subset C_1 \subset \cdots \subset C_{256}$, where for each $i \in \{0, 1, \dots, 256\}$, the cubical complex C_i contains all vertices, edges, squares, and cubes that are labeled by a number less than or equal to i. (See Figure 2(c) for an example.) Such a sequence of cubical complexes is also called a 'filtered cubical complex.' Similar to the previous example, one can use homology to measure several features of the spaces C_i (see Figure 2(d)).

In the present article, we focus on persistent homology, but there are also other methods in TDA — including the Mapper algorithm [56], Euler calculus (see [57] for an introduction with an eye towards applications), cellular sheaves [57, 58], and many more. We refer readers who wish to learn more about the foundations of TDA to the article [59], which discusses why topology and functoriality are essential for data analysis. We point to several introductory papers, books, and two videos on PH at the end of Section 4.

The first algorithm for the computation of PH was introduced for computation over $\mathbb{F}_2$ (the field with two elements) in [60] and over general fields in [61]. Since then, several algorithms and optimization techniques have been presented, and there are now various powerful implementations of PH [62–68]. Those wishing to try PH for computations may find it difficult to discern which implementations and algorithms are best suited for a given task. The field of PH is evolving continually, and new software implementations and updates are released at a rapid pace. Not all of them are well-documented, and (as is well-known in the TDA community), the computation of PH for large data sets is computationally very expensive.

To our knowledge, there exists neither an overview of the various computational methods for PH nor a comprehensive benchmarking of the state-of-the-art implementations for the computation of persistent homology. In the present article, we close this gap: we introduce computation of PH to a general audience of applied mathematicians and computational scientists, offer guidelines for the computation of PH, and test the existing open-source published libraries for the computation of PH.

The rest of our paper is organized as follows. In Section 2, we discuss related work. We then introduce homology in Section 3 and introduce PH in Section 4. We discuss the various steps of the pipeline for the computation of PH in Section 5, and we briefly examine algorithms for generalized persistence in Section 6. In Section 7, we give an overview of software libraries, discuss our benchmarking of a collection of them, and provide guidelines for which software or algorithm is better suited to which data set. (We provide specific guidelines for the computation of PH with the different libraries in the Tutorial in Additional file 2 of the Supplementary Information (SI).) In Section 8, we discuss future directions for the computation of PH.

2 Related work

In our work, we introduce PH to non-experts with an eye towards applications, and we benchmark state-of-the-art libraries for the computation of PH. In this section, we discuss related work for both of these points.

There are several excellent introductions to the theory of PH (see the references at the end of Section 4.1), but none of them emphasizes the actual computation of PH by providing specific guidelines for people who want to do computations. In the present paper, we navigate the theory of PH with an eye towards applications, and we provide guidelines for the computation of PH using the open-source libraries javaPlex, Perseus, Dionysus, DIPHA, Gudhi, and Ripser. We include a tutorial (see Additional file 2 of the SI) that gives specific instructions for how to use the different functionalities that are implemented in these libraries. Much of this information is scattered throughout numerous different papers, websites, and even source code of implementations, and we believe that it is beneficial to the applied mathematics community (especially people who seek an entry point into PH) to find all of this information in one place. The functionalities that we cover include plots of barcodes and persistence diagrams and the computation of PH with Vietoris–Rips complexes, alpha complexes, Čech complexes, witness complexes, cubical complexes for image data. We also discuss the computation of the bottleneck and Wasserstein distances. We thus believe that our paper closes a gap in introducing PH to people interested in applications, while our tutorial complements existing tutorials (see, e.g. [69–71]).

We believe that there is a need for a thorough benchmarking of the state-of-the-art libraries. In our work, we use twelve different data sets to test and compare the libraries javaPlex, Perseus, Dionysus, DIPHA, Gudhi, and Ripser. There are several benchmarkings in the PH literature; we are aware of the following ones: the benchmarking in [72] compares the implementations of standard and dual algorithms in Dionysus; the one in [73] compares the Morse-theoretic reduction algorithm with the standard algorithm; the one in [62] compares all of the data structures and algorithms implemented in PHAT; the benchmarking in [74] compares PHAT and its spin-off DIPHA; and the benchmarking in C. Maria's doctoral thesis [75] is to our knowledge the only existing benchmarking that compares packages from different authors. However, Maria compares only up to three different implementations at one time, and he used the package JPLEX (which is no longer maintained) instead of the JAVAPLEX library (its successor). Additionally, the widely used library PERSEUS (e.g., it was used in [22, 27, 30, 31]) does not appear in Maria's benchmarking.

3 Homology

Assume that one is given data that lies in a metric space, such as a subset of Euclidean space with an inherited distance function. In many situations, one is not interested in the precise geometry of these spaces, but instead seeks to understand some basic characteristics, such as the number of components or the existence of holes and voids. Algebraic topology captures these basic characteristics either by counting them or by associating vector spaces or more sophisticated algebraic structures to them. Here we are interested in *homology*, which associates one vector space $H_i(X)$ to a space X for each natural number $i \in \{0, 1, 2, \dots\}$. The dimension of $H_0(X)$ counts the number of path components in X, the dimension of $H_1(X)$ is a count of the number of holes, and the dimension of $H_2(X)$ is a count of the number of voids. An important property of these algebraic structures is that they are robust, as they do not change when the underlying space is transformed by bending, stretching, or other deformations. In technical terms, they are *homotopy invariant*.[a]

It can be very difficult to compute the homology of arbitrary topological spaces. We thus approximate our spaces by combinatorial structures called 'simplicial complexes,' for which homology can be easily computed algorithmically. Indeed, often one is not even given the space X, but instead possesses only a discrete sample set S from which to build a simplicial complex following one of the recipes described in Sections 3.2 and 5.2.

3.1 Simplicial complexes and their homology

We begin by giving the definitions of simplicial complexes and of the maps between them. Roughly, a simplicial complex is a space that is built from a union of points, edges, triangles, tetrahedra, and higher-dimensional polytopes. We illustrate the main definitions given in this section with the example in Figure 3. As we pointed out in Section 1, 'cubical complexes' give another way to associate a combinatorial structure to a topological space. In TDA, cubical complexes have been used primarily to study image data sets. One can compute PH for a nested sequence of cubical complexes in a similar way as for simplicial complexes, but the theory of PH for simplicial complexes is richer, and we therefore examine only simplicial homology and complexes in our discussions. See [76] for a treatment of cubical complexes and their homology.

Definition 1 A *simplicial complex*[b] is a collection K of non-empty subsets of a set K_0 such that $\{v\} \in K$ for all $v \in K_0$, and $\tau \subset \sigma$ and $\sigma \in K$ guarantees that $\tau \in K$. The elements of K_0 are called *vertices* of K, and the elements of K are called *simplices*. Additionally, we say that a simplex has *dimension p* or is a *p-simplex* if it has a cardinality of $p + 1$. We use K_p to denote the collection of p-simplices. The *k-skeleton* of K is the union of the sets K_p for

(a) A simplicial complex:

$$K = \{\{a\},\{b\},\{c\},\{d\},\{e\},\{a,b\},\{a,c\},\{a,d\},\{b,c\},\{c,d\},\{a,b,c\}\}\,.$$

(b) Let K be the simplicial complex in (a), and let K' be the following simplicial complex:

$$K' = \{\{g\},\{h\},\{l\},\{g,h\},\{g,l\},\{h,l\},\{g,h,l\}\}\,.$$

We define a map of simplicial complexes, $f\colon K \to K'$, as follows:

$$f\colon K_0 \to K_0'\colon x \mapsto \begin{cases} h\,, & \text{if } x{=}\text{a} \\ g\,, & \text{otherwise}\,. \end{cases}$$

(c) Geometric realizations of the simplicial complexes in (a) and (b) are the following subsets of $\mathbb{R}^2$:

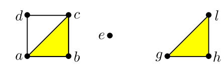


Figure 3 A simple example. (a) A simplicial complex, **(b)** a map of simplicial complexes, and **(c)** a geometric realization of the simplicial complex in (a).

all $p \in \{0,1,\dots,k\}$. If τ and σ are simplices such that $\tau \subset \sigma$, then we call τ a *face* of σ, and we say that τ is a face of σ of *codimension* k' if the dimensions of τ and σ differ by k'. The *dimension* of K is defined as the maximum of the dimensions of its simplices. A *map of simplicial complexes*, $f : K \to L$, is a map $f : K_0 \to L_0$ such that $f(\sigma) \in L$ for all $\sigma \in K$.

We give an example of a simplicial complex in Figure 3(a) and an example of a map of simplicial complexes in Figure 3(b). Definition 1 is rather abstract, but one can always interpret a finite simplicial complex K geometrically as a subset of $\mathbb{R}^N$ for sufficiently large N; such a subset is called a 'geometric realization,' and it is unique up to a canonical piecewise-linear homeomorphism. For example, the simplicial complex in Figure 3(a) has a geometric realization given by the subset of $\mathbb{R}^2$ in Figure 3(c).

We now define homology for simplicial complexes. Let $\mathbb{F}_2$ denote the field with two elements. Given a simplicial complex K, let $C_p(K)$ denote the $\mathbb{F}_2$-vector space with basis given by the p-simplices of K. For any $p \in \{1,2,\dots\}$, we define the linear map (on the basis elements)

$$\begin{aligned} d_p : C_p(K) &\to C_{p-1}(K), \\ \sigma &\mapsto \sum_{\tau \subset \sigma, \tau \in K_{p-1}} \tau\,. \end{aligned}$$

For $p = 0$, we define d_0 to be the zero map. In words, d_p maps each p-simplex to its boundary, the sum of its faces of codimension 1. Because the boundary of a boundary is always empty, the linear maps d_p have the property that composing any two consecutive maps yields the zero map: for all $p \in \{0,1,2,\dots\}$, we have $d_p \circ d_{p+1} = 0$. Consequently, the image of d_{p+1} is contained in the kernel of d_p, so we can take the quotient of kernel(d_p) by image(d_{p+1}). We can thus make the following definition.

Definition 2 For any $p \in \{0,1,2,\dots\}$, the *pth homology* of a simplicial complex K is the quotient vector space

$$H_p(K) := \text{kernel}(d_p)/\,\text{image}(d_{p+1}).$$

(a) We compute the simplicial homology for the simplicial complex in Figure 3(a). We have the following sequence of vector spaces and linear maps:

$$0 \longrightarrow \mathbb{F}_2 \xrightarrow{d_2} \mathbb{F}_2^5 \xrightarrow{d_1} \mathbb{F}_2^5 \xrightarrow{d_0} 0\,.$$

Let abc denote the basis vector that corresponds to the simplex $\{a, b, c\}$. Similarly, we use ab, ac, ad, bc, and cd to denote the basis vectors that correspond to the 1-simplices; and we use a, b, c, d, and e to denote the basis vectors that correspond to the 0-simplices. We order the bases of the vector spaces using lexicographic order. We then have

$$d_2 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \end{pmatrix}^{\mathrm{t}}$$

and

$$d_1 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

One can then compute that $\beta_0(K) = 2$, $\beta_1(K) = 1$, and all higher Betti numbers are 0.

(b) The map of simplicial complexes, $f\colon K \to K'$, in Figure 3(b) induces the map $\widetilde{f}_0\colon C_0(K) \to C_0(K')$ on chain complexes that sends the basis element a to h (note that we use the same notation as in part (a) of this figure). Furthermore, it sends the basis elements that correspond to the other 0-simplices of K to g, and it induces the map $\widetilde{f}_1\colon C_1(K) \to C_1(K')$ that sends the basis elements ab, ac, and ad to cd and sends the basis elements cd and bc to 0. This gives a map f_0 on 0th homology that sends both generators of $H_0(K)$ to the generator of $H_0(K')$. It follows that $H_i(K') = 0$ for all $i \geq 1$, so f_i is the zero map for $i \geq 1$.

Figure 4 Examples to illustrate simplicial homology. (a) Computation of simplicial homology for the simplicial complex in Figure 3(a) and **(b)** induced map in 0th homology for the map of simplicial complexes in Figure 3(b).

Its dimension

$$\beta_p(K) := \dim H_p(K) = \dim \operatorname{kernel}(d_p) - \dim \operatorname{image}(d_{p+1})$$

is called the *pth Betti number* of K. Elements in the image of d_{p+1} are called *p-boundaries*, and elements in the kernel of d_p are called *p-cycles*.

Intuitively, the p-cycles that are not boundaries represent p-dimensional holes. Therefore, the pth Betti number 'counts' the number of p-holes. Additionally, if K is a simplicial complex of dimension n, then for all $p > n$, we have that $H_p(K) = 0$, as K_p is empty and hence $C_p(K) = 0$. We therefore obtain the following sequence of vector spaces and linear maps:

$$0 \xrightarrow{d_{n+1}} C_n(K) \xrightarrow{d_n} \cdots \xrightarrow{d_2} C_1(K) \xrightarrow{d_1} C_0(K) \xrightarrow{d_0} 0.$$

We give an example of such a sequence in Figure 4(a), for which we also report the Betti numbers.

One of the most important properties of simplicial homology is 'functoriality.' Any map $f : K \to K'$ of simplicial complexes induces the following $\mathbb{F}_2$-linear map:

$$\widetilde{f}_p : C_p(K) \to C_p\big(K'\big),$$
$$\sum_{\sigma \in K_p} c_\sigma \sigma \mapsto \sum_{\sigma \in K_p \text{ such that } f(\sigma) \in K'_p} c_\sigma f(\sigma) \quad \text{for any } p \in \{0, 1, 2, \dots\},$$

where $c_\sigma \in \mathbb{F}_2$. Additionally, $\widetilde{f_p} \circ d_{p+1} = d'_{p+1} \circ \widetilde{f}_{p+1}$, and the map $\widetilde{f_p}$ therefore induces the following linear map between homology vector spaces:

$$f_p : H_p(K) \to H_p(K'),$$
$$[c] \mapsto [\widetilde{f_p}(c)].$$

(We give an example of such a map in Figure 4(b).) Consequently, to any map $f : K \to K'$ of simplicial complexes, we can assign a map $f_p : H_p(K) \to H_p(K')$ for any $p \in \{0, 1, 2, \dots\}$. This assignment has the important property that given a pair of composable maps of simplicial complexes, $f : K \to K'$ and $g : K' \to K''$, the map $(g \circ f)_p : H_p(K) \to H_p(K'')$ is equal to the composition of the maps induced by f and g. That is, $(g \circ f)_p = g_p \circ f_p$. The fact that a map of simplicial complexes induces a map on homology that is compatible with composition is called *functoriality*, and it is crucial for the definition of persistent homology (see Section 4.1).

When working with simplicial complexes, one can modify a simplicial complex by removing or adding a pair of simplices (σ, τ), where τ is a face of σ of codimension 1 and σ is the only simplex that has τ as a face. The resulting simplicial complex has the same homology as the one with which we started. In Figure 3(a), we can remove the pair $(\{a, b, c\}, \{b, c\})$ and then the pair $(\{a, b\}, \{b\})$ without changing the Betti numbers. Such a move is called an *elementary simplicial collapse* [77]. In Section 5.2.6, we will see an application of this for the computation of PH.

In this section, we have defined simplicial homology over the field $\mathbb{F}_2$ — i.e., 'with coefficients in $\mathbb{F}_2$.' One can be more general and instead define simplicial homology with coefficients in any field (or even in the integers). However, when $1 \neq -1$, one needs to take more care when defining the boundary maps d_p to ensure that $d_p \circ d_{p+1}$ remains the zero map. Consequently, the definition is more involved. For the purposes of the present paper, it suffices to consider homology with coefficients in the field $\mathbb{F}_2$. Indeed, we will see in Section 4 that to obtain topological summaries in the form of barcodes, we need to compute homology with coefficients in a field. Furthermore, as we summarize in Table 2 (in Section 7), most of the implementations for the computation of PH work with $\mathbb{F}_2$.

We conclude this section with a warning: changing the coefficient field can affect the Betti numbers. For example, if one computes the homology of the Klein bottle (see Section 7.1.1) with coefficients in the field $\mathbb{F}_p$ with p elements, where p is a prime, then $\beta_0(K) = 1$ for all primes p. However, $\beta_1(K) = 2$ and $\beta_2(K) = 1$ if $p = 2$, but $\beta_1(K) = 1$ and $\beta_2(K) = 0$ for all other primes p. The fact that $\beta_2(K) = 0$ for $p \neq 2$ arises from the nonorientability of the Klein bottle. The treatment of different coefficient fields is beyond the scope of our article, but interested readers can peruse [78] for an introduction to homology and [76] for an overview of computational homology.

3.2 Building simplicial complexes

As we discussed in Section 3.1, computing the homology of finite simplicial complexes boils down to linear algebra. The same is not true for the homology of an arbitrary space X, and one therefore tries to find simplicial complexes whose homology approximates the homology of the space in an appropriate sense.

An important tool is the Čech (Č) complex. Let $\mathcal{U}$ be a cover of X — i.e., a collection of subsets of X such that the union of the subsets is X. The k-simplices of the Čech complex are the non-empty intersections of $k + 1$ sets in the cover $\mathcal{U}$. More precisely, we define the nerve of a collection of sets as follows.

Definition 3 Let $\mathcal{U} = \{U_i\}_{i\in I}$ be a non-empty collection of sets. The *nerve* of $\mathcal{U}$ is the simplicial complex with set of vertices given by I and k-simplices given by $\{i_0, \ldots, i_k\}$ if and only if $U_{i_0} \cap \cdots \cap U_{i_k} \neq \emptyset$.

If the cover of the sets is sufficiently 'nice,' then the Nerve Theorem implies that the nerve of the cover and the space X have the same homology [79, 80]. For example, suppose that we have a finite set of points S in a metric space X. We then can define, for every $\epsilon > 0$, the space S_ϵ as the union $\bigcup_{x\in S} B(x, \epsilon)$, where $B(x, \epsilon)$ denotes the closed ball with radius ϵ centered at x. It follows that $\{B(x, \epsilon) \mid x \in S\}$ is a cover of S_ϵ, and the nerve of this cover is the *Čech complex on S at scale* ϵ. We denote this complex by $\check{C}_\epsilon(S)$. If the space X is Euclidean space, then the Nerve Theorem guarantees that the simplicial complex $\check{C}_\epsilon(S)$ recovers the homology of S_ϵ.

From a computational point of view, the Čech complex is expensive because one has to check for large numbers of intersections. Additionally, in the worst case, the Čech complex can have dimension $|\mathcal{U}| - 1$, and it therefore can have many simplices in dimensions higher than the dimension of the underlying space. Ideally, it is desirable to construct simplicial complexes that approximate the homology of a space but are easy to compute and have 'few' simplices, especially in high dimensions. This is a subject of ongoing research: In Section 5.2, we give an overview of state-of-the-art methods to associate complexes to point-cloud data in a way that addresses one or both of these desiderata. See [80, 81] for more details on the Čech complex, and see [79, 80] for a precise statement of the Nerve Theorem.

4 Persistent homology

Assume that we are given experimental data in the form of a finite metric space S; there are points or vectors that represent measurements along with some distance function (e.g., given by a correlation or a measure of dissimilarity) on the set of points or vectors. Whether or not the set S is a sample from some underlying topological space, it is useful to think of it in those terms. Our goal is to recover the properties of such an underlying space in a way that is robust to small perturbations in the data S. In a broad sense, this is the subject of *topological inference.* (See [82] for an overview.) If S is a subset of Euclidean space, one can consider a 'thickening' S_ϵ of S given by the union of balls of a certain fixed radius ϵ around its points and then compute the Čech complex. One can thus try to compute qualitative features of the data set S by constructing the Čech complex for a chosen value ϵ and then computing its simplicial homology. The problem with this approach is that there is a priori no clear choice for the value of the parameter ϵ. The key insight of PH is the following: To extract qualitative information from data, one considers several (or even all) possible values of the parameter ϵ. As the value of ϵ increases, simplices are added to the complexes. Persistent homology then captures how the homology of the complexes changes as the parameter value increases, and it detects which features 'persist' across changes in the parameter value. We give an example of persistent homology in Figure 5.

4.1 Filtered complexes and homology

Let K be a finite simplicial complex, and let $K_1 \subset K_2 \subset \cdots \subset K_l = K$ be a finite sequence of nested subcomplexes of K. The simplicial complex K with such a sequence of sub-

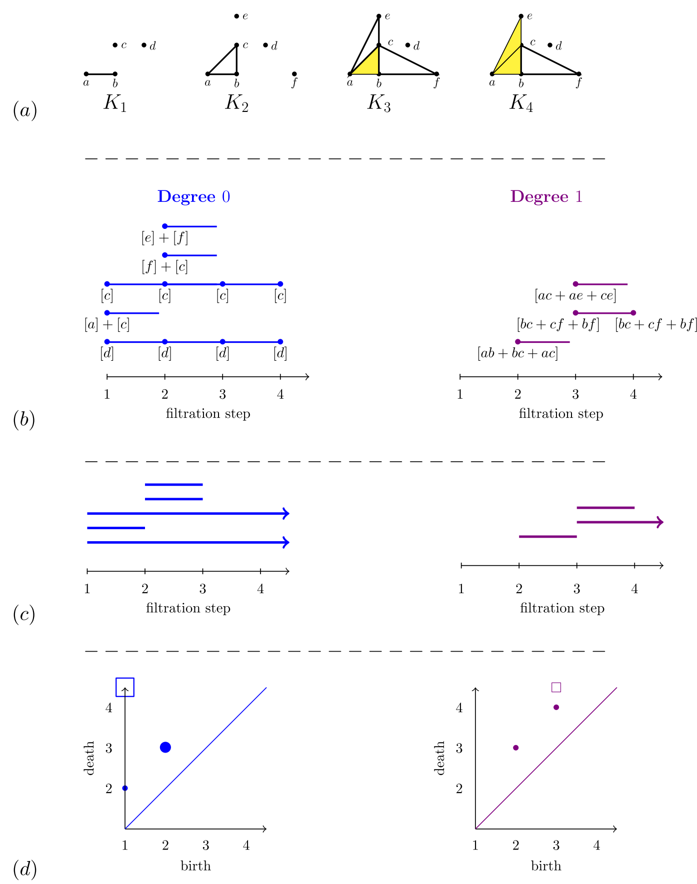


Figure 5 Example of persistent homology for a finite filtered simplicial complex. (a) We start with a finite filtered simplicial complex. **(b)** At each filtration step i, we draw as many vertices as the dimension of (left column) $H_0(K_i)$ and (right column) $H_1(K_i)$. We label the vertices by basis elements, the existence of which is guaranteed by the Fundamental Theorem of Persistent Homology, and we draw an edge between two vertices to represent the maps $f_{i,j}$, as explained in the main text. We thus obtain a well-defined collection of disjoint half-open intervals called a 'barcode.' We interpret each interval in degree p as representing the lifetime of a p-homology class across the filtration. **(c)** We rewrite the diagram in (b) in the conventional way. We represent classes that are born but do not die at the final filtration step using arrows that start at the birth of that feature and point to the right. **(d)** An alternative graphical way to represent barcodes (which gives exactly the same information) is to use *persistence diagrams*, in which an interval $[i,j)$ is represented by the point (i,j) in the extended plane $\overline{\mathbb{R}}^2$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$. Therefore, a persistence diagram is a finite multiset of points in $\overline{\mathbb{R}}^2$. We use squares to signify the classes that do not die at the final step of a filtration, and the size of dots or squares is directly proportional to the number of points being represented. For technical reasons, which we discuss briefly in Section 5.4, one also adds points on the diagonal to the persistence diagrams. (Each of the points on the diagonal has infinite multiplicity.)

complexes is called a *filtered simplicial complex*. See Figure 5(a) for an example of filtered simplicial complex. We can apply homology to each of the subcomplexes. For all p, the inclusion maps $K_i \to K_j$ induce $\mathbb{F}_2$-linear maps $f_{i,j} : H_p(K_i) \to H_p(K_j)$ for all $i, j \in \{1, \dots, l\}$ with $i \leq j$. By functoriality (see Section 3.1), it follows that

$$f_{k,j} \circ f_{i,k} = f_{i,j} \quad \text{for all } i \leq k \leq j. \tag{1}$$

We therefore give the following definition.[c]

Definition 4 Let $K_1 \subset K_2 \subset \cdots \subset K_l = K$ be a filtered simplicial complex. The *pth persistent homology of K* is the pair

$$\left(\left\{H_p(K_i)\right\}_{1\leq i\leq l}, \{f_{i,j}\}_{1\leq i\leq j\leq l}\right),$$

where for all $i,j \in \{1,\ldots,l\}$ with $i \leq j$, the linear maps $f_{i,j} : H_p(K_i) \to H_p(K_j)$ are the maps induced by the inclusion maps $K_i \to K_j$.

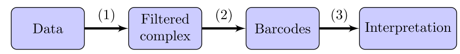
The pth persistent homology of a filtered simplicial complex gives more refined information than just the homology of the single subcomplexes. We can visualize the information given by the vector spaces $H_p(K_i)$ together with the linear maps $f_{i,j}$ by drawing the following diagram: at filtration step i, we draw as many bullets as the dimension of the vector space $H_p(K_i)$. We then connect the bullets as follows: we draw an interval between bullet u at filtration step i and bullet v at filtration step $i + 1$ if the generator of $H_p(K_i)$ that corresponds to u is sent to the generator of $H_p(K_{i+1})$ that corresponds to v. If the generator corresponding to a bullet u at filtration step i is sent to 0 by $f_{i,i+1}$, we draw an interval starting at u and ending at $i + 1$. (See Figure 5(b) for an example.) Such a diagram clearly depends on a choice of basis for the vector spaces $H_p(K_i)$, and a poor choice can lead to complicated and unreadable clutter. Fortunately, by the Fundamental Theorem of Persistent Homology [61], there is a choice of basis vectors of $H_p(K_i)$ for each $i \in \{1,\ldots,l\}$ such that one can construct the diagram as a well-defined and unique collection of disjoint half-open intervals, collectively called a *barcode*.[d] We give an example of a barcode in Figure 5(c). Note that the Fundamental Theorem of PH, and hence the existence of a barcode, relies on the fact that we are using homology with field coefficients. (See [61] for more details.)

There is a useful interpretation of barcodes in terms of births and deaths of generators. Considering the maps $f_{i,j}$ written in the basis given by the Fundamental Theorem of Persistent Homology, we say that $x \in H_p(K_i)$ (with $x \neq 0$) is *born* in $H_p(K_i)$ if it is not in the image of $f_{i-1,i}$ (i.e., $f_{i-1,i}^{-1}(x) = \emptyset$). For $x \in H_p(K_i)$ (with $x \neq 0$), we say that x *dies* in $H_p(K_j)$ if $j > i$ is the smallest index for which $f_{i,j}(x) = 0$. The lifetime of x is represented by the half-open interval $[i, j)$. If $f_{i,j}(x) \neq 0$ for all j such that $i < j \leq l$, we say that x *lives forever*, and its lifetime is represented by the interval $[i,\infty)$.

Remark 5 Note that some references (e.g., [80]) introduce persistent homology by defining the birth and death of generators without using the existence of a choice of compatible bases, as given by the Fundamental Theorem of Persistent Homology. The definition of birth coincides with the definition that we have given, but the definition of death is different. One says that $x \in H_p(K_i)$ (with $x \neq 0$) *dies* in $H_p(K_j)$ if $j > i$ is the smallest index for which either $f_{i,j}(x) = 0$ or there exists $y \in H_p(K_{i'})$ with $i' < i$ such that $f_{i',j}(y) = f_{i,j}(x)$. In words, this means that x and y merge at filtration step j, and the class that was born earlier is the one that survives. In the literature, this is called the *elder rule*. We do not adopt this definition, because the elder rule is not well-defined when two classes are born at the same time, as there is no way to choose which class will survive. For example, in Figure 5, there are two classes in H_0 that are born at the same stage in K_1. These two classes merge in K_2, but neither dies. The class that dies is $[a] + [c]$.

There are numerous excellent introductions to PH, such as the books [57, 80, 82, 83] and the papers [59, 84–88]. For a brief and friendly introduction to PH and some of

Figure 6 PH pipeline.



its applications, see the video https://www.youtube.com/watch?v=h0bnG1Wavag. For a brief introduction to some of the ideas in TDA, see the video https://www.youtube.com/watch?v=XfWibrh6stw.

5 Computation of PH for data

We summarize the pipeline for the computation of PH from data in Figure 6. In the following subsections, we describe each step of this pipeline and state-of-the-art algorithms for the computation of PH. The two features that make PH appealing for applications are that it is computable via linear algebra and that it is stable with respect to perturbations in the measurement of data. In Section 5.5, we give a brief overview of stability results.

5.1 Data

As we mentioned in Section 1, types of data sets that one can study with PH include finite metric spaces, digital images, and networks. We now give a brief overview of how one can study these types of data sets using PH.

5.1.1 Networks

One can construe an undirected network as a 1-dimensional simplicial complex. If the network is weighted, then filtering by increasing or decreasing weight yields a filtered 1-dimensional simplicial complex. To obtain more refined information about the network, it is desirable to construct higher-dimensional simplices. There are various methods to do this. The simplest method, called a *weight rank clique filtration* (WRCF), consists of building a clique complex on each subnetwork. (See Section 5.2.1 for the definition of 'clique complex.') See [89] for an application of this method. Another method to study networks with PH consists of mapping the nodes of the network to points of a finite metric space. There are several ways to compute distances between nodes of a network; the method that we use in our benchmarking in Section 7 consists of computing a shortest path between nodes. For such a distance to be well-defined, note that one needs the network to be connected (although conventionally one takes the distance between nodes in different components to be infinity). There are many methods to associate an unfiltered simplicial complex to both undirected and directed networks. See the book [90] for an overview of such methods, and see the paper [91] for an overview of PH for networks.

5.1.2 Digital images

As we mentioned in Section 1, digital images have a natural cubical structure: 2-dimensional digital images are made of pixels, and 3-dimensional images are made of voxels. Therefore, to study digital images, cubical complexes are more appropriate than simplicial complexes. Roughly, cubical complexes are spaces built from a union of vertices, edges, squares, cubes, and so on. One can compute PH for cubical complexes in a similar way as for simplicial complexes, and we will therefore not discuss this further in this paper. See [76] for a treatment of computational homology with cubical complexes rather than simplicial complexes and for a discussion of the relationship between simplicial and cubical homology. See [55] for an efficient algorithm and data structure for the computation of PH for cubical data, and [92] for an algorithm that computes PH for cubical data in an approximate way. For an application of PH and cubical complexes to movies, see [73].

Other approaches for studying digital images are also useful. In general, given a digital image that consists of N pixels or voxels, one can consider this image as a point in a $c \times N$-dimensional space, with each coordinate storing a vector of length c representing the color of a pixel or voxel. Defining an appropriate distance function on such a space allows one to consider a collection of images (each of which has N pixels or voxels) as a finite metric space. A version of this approach was used in [26], in which the local structure of natural images was studied by selecting 3×3 patches of pixels of the images.

5.1.3 Finite metric spaces

As we mentioned in the previous two subsections, both undirected networks and image data can be construed as finite metric spaces. Therefore, methods to study finite metric spaces with PH apply to the study of networks and image data sets.

In some applications, points of a metric space have associated 'weights.' For instance, in the study of molecules, one can represent a molecule as a union of balls in Euclidean space [93, 94]. For such data sets, one would therefore also consider a minimum filtration value (see Section 5.2 for the description of such filtration values) at which the point enters the filtration. In Table 2(g), we indicate which software libraries implement this feature.

5.2 Filtered simplicial complexes

In Section 3.2, we introduced the Čech complex, a classical simplicial complex from algebraic topology. However, there are many other simplicial complexes that are better suited for studying data from applications. We discuss them in this section.

To be a useful tool for the study of data, a simplicial complex has to satisfy some theoretical properties dictated by topological inference; roughly, if we build the simplicial complex on a set of points sampled from a space, then the homology of the simplicial complex has to approximate the homology of the space. For the Čech complex, these properties are guaranteed by the Nerve Theorem. Some of the complexes that we discuss in this subsection are motivated by a 'sparsification paradigm': they approximate the PH of known simplicial complexes but have fewer simplices than them. Others, like the Vietoris–Rips complex, are appealing because they can be computed efficiently. In this subsection, we also review reduction techniques, which are heuristics that reduce the size of complexes without changing the PH. In Table 1, we summarize the simplicial complexes that we discuss in this subsection.

Table 1 We summarize several types of complexes that are used for PH

Complex *K*	**Size of *K***	**Theoretical guarantee**
Čech	$2^{\mathcal{O}(N)}$	Nerve theorem
Vietoris–Rips (VR)	$2^{\mathcal{O}(N)}$	Approximates Čech complex
Alpha	$N^{\mathcal{O}(\lceil d/2 \rceil)}$ (N points in $\mathbb{R}^d$)	Nerve theorem
Witness	$2^{\mathcal{O}(\lvert L \rvert)}$	For curves and surfaces in Euclidean space
Graph-induced complex	$2^{\mathcal{O}(\lvert Q \rvert)}$	Approximates VR complex
Sparsified Čech	$\mathcal{O}(N)$	Approximates Čech complex
Sparsified VR	$\mathcal{O}(N)$	Approximates VR complex

We indicate the theoretical guarantees and the worst-case sizes of the complexes as functions of the cardinality N of the vertex set. For the witness complexes (see Section 5.2.4), L denotes the set of landmark points, while Q denotes the subsample set for the graph-induced complex (see Section 5.2.5).

For the rest of this subsection (X, d) denotes a metric space, and S is a subset of X, which becomes a metric space with the induced metric. In applications, S is the collection of measurements together with a notion of distance, and we assume that S lies in the (unknown) metric space X. Our goal is then to compute persistent homology for a sequence of nested spaces $S_{\epsilon_1}, S_{\epsilon_2}, \ldots, S_{\epsilon_l}$, where each space gives a 'thickening' of S in X.

5.2.1 Vietoris–Rips complex

We have seen that one of the disadvantages of the Čech complex is that one has to check for a large number of intersections. To circumvent this issue, one can instead consider the Vietoris–Rips (VR) complex, which approximates the Čech complex. For a non-negative real number ϵ, the *Vietoris–Rips complex* $\mathrm{VR}_\epsilon(S)$ *at scale* ϵ is defined as

$$\mathrm{VR}_\epsilon(S) = \big\{\sigma \subseteq S \mid \mathrm{d}(x,y) \leq 2\epsilon \text{ for all } x, y \in \sigma\big\}.$$

The sense in which the VR complex approximates the Čech complex is that, when S is a subset of Euclidean space, we have $\check{C}_\epsilon(S) \subseteq \mathrm{VR}_\epsilon(S) \subseteq \check{C}_{\sqrt{2}\epsilon}(S)$. Deciding whether a subset $\sigma \subseteq S$ is in $\mathrm{VR}_\epsilon(S)$ is equivalent to deciding if the maximal pairwise distance between any two vertices in σ is at most 2ϵ. Therefore, one can construct the VR complex in two steps. One first computes the *ϵ-neighborhood graph of S*. This is the graph whose vertices are all points in S and whose edges are

$$\big\{(i,j) \in S \times S \mid i \neq j \text{ and } \mathrm{d}(i,j) \leq 2\epsilon\big\}.$$

Second, one obtains the VR complex by computing the *clique complex* of the ϵ-neighborhood graph. The clique complex of a graph is a simplicial complex that is defined as follows: The subset $\{x_0, \ldots, x_k\}$ is a k-simplex if and only if every pair of vertices in $\{x_0, \ldots, x_k\}$ is connected by an edge. Such a collection of vertices is called a *clique*. This construction makes it very easy to compute the VR complex, because to construct the clique complex one has only to check for pairwise distances — for this reason, clique complexes are also called 'lazy' in the literature. Unfortunately, the VR complex has the same worst-case complexity as the Čech complex. In the worst case, it can have up to $2^{|S|} - 1$ simplices and dimension $|S| - 1$.

In applications, one therefore usually only computes the VR complex up to some dimension $k \ll |S| - 1$. In our benchmarking, we often choose $k = 2$ and $k = 3$.

The paper [95] overviews different algorithms to perform both of the steps for the construction of the VR complex, and it introduces fast algorithms to construct the clique complex. For more details on the VR complex, see [80, 96]. For a proof of the approximation of the Čech complex by the VR complex, see [80]; see [97] for a generalization of this result.

5.2.2 The Delaunay complex

To avoid the computational problems of the Čech and VR complexes, we need a way to limit the number of simplices in high dimensions. The Delaunay complex gives a geometric tool to accomplish this task, and most of the new simplicial complexes that have been introduced for the study of data are based on variations of the Delaunay complex. The Delaunay complex and its dual, the Voronoi diagram, are central objects of study in computational geometry because they have many useful properties.

For the Delaunay complex, one usually considers $X = \mathbb{R}^d$, so we also make this assumption. We subdivide the space $\mathbb{R}^d$ into regions of points that are closest to any of the points in S. More precisely, for any $s \in S$, we define

$$V_s = \{x \in \mathbb{R}^d \mid \mathrm{d}(x,s) \le \mathrm{d}(x,s') \text{ for all } s' \in S\}.$$

The collection of sets V_s is a cover for $\mathbb{R}^d$ that is called the *Voronoi decomposition of* $\mathbb{R}^d$ *with respect to* S, and the nerve of this cover is called the *Delaunay complex* of S and is denoted by $\mathrm{Del}(S;\mathbb{R}^d)$. In general, the Delaunay complex does not have a geometric realization in $\mathbb{R}^d$. However, if the points S are 'in general position'[e] then the Delaunay complex has a geometric realization in $\mathbb{R}^d$ that gives a triangulation of the convex hull of S. In this case, the Delaunay complex is also called the *Delaunay triangulation*.

The complexity of the Delaunay complex depends on the dimension d of the space. For $d \le 2$, the best algorithms have complexity $\mathcal{O}(N \log N)$, where N is the cardinality of S. For $d \ge 3$, they have complexity $\mathcal{O}(N^{\lceil d/2 \rceil})$. The construction of the Delaunay complex is therefore costly in high dimensions, although there are efficient algorithms for the computation of the Delaunay complex for $d = 2$ and $d = 3$. Developing efficient algorithms for the construction of the Delaunay complex in higher dimensions is a subject of ongoing research. See [98] for a discussion of progress in this direction, and see [99] for more details on the Delaunay complex and the Voronoi diagram.

5.2.3 Alpha complex

We continue to assume that S is a finite set of points in $\mathbb{R}^d$. Using the Voronoi decomposition, one can define a simplicial complex that is similar to the Čech complex, but which has the desired property that (if the points S are in general position) its dimension is at most that of the space. Let $\epsilon > 0$, and let S_ϵ denote the union $\bigcup_{s \in S} B(s,\epsilon)$. For every $s \in S$, consider the intersection $V_s \cap B(s,\epsilon)$. The collection of these sets forms a cover of S_ϵ, and the nerve complex of this cover is called the *alpha (α) complex of S at scale* ϵ and is denoted by $A_\epsilon(S)$. The Nerve Theorem applies, and it therefore follows that $A_\epsilon(S)$ has the same homology as S_ϵ.

Furthermore, $A_\infty(S)$ is the Delaunay complex; and for $\epsilon < \infty$, the alpha complex is a subcomplex of the Delaunay complex. The alpha complex was introduced for points in the plane in [100], in 3-dimensional Euclidean space in [101], and for Euclidean spaces of arbitrary dimension in [102]. For points in the plane, there is a well-known speed-up for the alpha complex that uses a duality between 0-dimensional and 1-dimensional persistence for alpha complexes [86]. (See [103] for the algorithm, and see [104] for an implementation.)

5.2.4 Witness complexes

Witness complexes are very useful for analyzing large data sets, because they make it possible to construct a simplicial complex on a significantly smaller subset $L \subseteq S$ of points that are called 'landmark' points. Meanwhile, because one uses information about all points in S to construct the simplicial complex, the points in S are called 'witnesses.' Witness complexes can be construed as a 'weak version' of Delaunay complexes. (See the characterization of the Delaunay complex in [105].)

Definition 6 Let (S, d) be a metric space, and let $L \subseteq S$ be a finite subset. Suppose that σ is a non-empty subset of L. We then say that $s \in S$ is a *weak witness for σ with respect to L* if and only if $\mathrm{d}(s,a) \leq \mathrm{d}(s,b)$ for all $a \in \sigma$ and for all $b \in L \setminus \sigma$. The *weak Delaunay complex* $\mathrm{Del}^w(L;S)$ *of S with respect to L* has vertex set given by the points in L, and a subset σ of L is in $\mathrm{Del}^w(L;S)$ if and only if it has a weak witness in S.

To obtain nested complexes, one can extend the definition of witnesses to ϵ-witnesses.

Definition 7 A point $s \in S$ is a *weak ϵ-witness* for σ with respect to L if and only if $\mathrm{d}(s,a) \leq \mathrm{d}(s,b) + \epsilon$ for all $a \in \sigma$ and for all $b \in L \setminus \sigma$.

Now we can define the *weak Delaunay complex* $\mathrm{Del}^w(L;S,\epsilon)$ *at scale* ϵ to be the simplicial complex with vertex set L, and such that a subset $\sigma \subseteq L$ is in $\mathrm{Del}^w(L;S,\epsilon)$ if and only if it has a weak ϵ-witness in S. By considering different values for the parameter ϵ, we thereby obtain nested simplicial complexes. The weak Delaunay complex is also called the 'weak witness complex' or just the 'witness complex' in the literature.

There is a modification of the witness complex called the *lazy witness complex* $\mathrm{Del}^w_{\mathrm{lazy}}(L;X,\epsilon)$. It is a clique complex, and it can therefore be computed more efficiently than the witness complex. The lazy witness complex has the same 1-skeleton as $\mathrm{Del}^w(L;X,\epsilon)$, and one adds a simplex σ to $\mathrm{Del}^w_{\mathrm{lazy}}(L;X,\epsilon)$ whenever its edges are in $\mathrm{Del}^w_{\mathrm{lazy}}(L;X,\epsilon)$. Another type of modification of the witness complex yields *parametrized witness complexes*. Let $\nu = 1,2,\ldots$ and for all $s \in S$ define $m_\nu(s)$ to be the distance to the νth closest landmark point. Furthermore, define $m_0(s) = 0$ for all $s \in S$. Let $\mathrm{W}_\nu(L;S,\epsilon)$ be the simplicial complex whose vertex set is L and such that a 1-simplex $\sigma = \{x_0, x_1\}$ is in $\mathrm{W}_\nu(L;X,\epsilon)$ if and only if there exists s in S for which

$$\max\big\{\mathrm{d}(x_0,s), \mathrm{d}(x_1,s)\big\} \leq m_\nu(s) + \epsilon.$$

A simplex σ is in $\mathrm{W}_\nu(L;X,\epsilon)$ if and only if all of its edges belong to $\mathrm{W}_\nu(L;X,\epsilon)$. For $\nu = 2$, note that $\mathrm{W}_2(L;X,\epsilon) = \mathrm{Del}^w_{\mathrm{lazy}}(L;X,\epsilon)$. For $\nu = 0$, we have that $\mathrm{W}_0(L;X,\epsilon)$ approximates the VR complex $\mathrm{VR}(L;\epsilon)$. That is,

$$\mathrm{W}_0(L;X,\epsilon) \subseteq \mathrm{VR}(L;2\epsilon) \subseteq \mathrm{W}_0(L;X,2\epsilon).$$

Note that parametrized witness complexes are often called 'lazy witness complexes' in the literature, because they are clique complexes.

The weak Delaunay complex was introduced in [105], and parametrized witness complexes were introduced in [106]. Witness complexes can be rather useful for applications. Because their complexity depends on the number of landmark points, one can reduce the complexity by computing simplicial complexes using a smaller number of vertices. However, there are theoretical guarantees for the witness complex only when S is the metric space associated to a low-dimensional Euclidean submanifold. It has been shown that witness complexes can be used to recover the topology of curves and surfaces in Euclidean space [107, 108], but they can fail to recover topology for submanifolds of Euclidean space of three or more dimensions [109]. Consequently, there have been studies of simplicial complexes that are similar to the witness complexes but with better theoretical guarantees (see Section 5.2.5).

5.2.5 Additional complexes

Many more complexes have been introduced for the fast computation of PH for large data sets. These include the graph-induced complex [110], which is a simplicial complex constructed on a subsample Q, and has better theoretical guarantees than the witness complex (see [111] for the companion software); an approximation of the VR complex that has a worst-case size that is linear in the number of data points [112]; an approximation of the Čech complex [97] whose worst-case size also scales linearly in the data; and an approximation of the VR complex via simplicial collapses [113]. We do not discuss such complexes in detail, because thus far (at the time of writing) none of them have been implemented in publicly-available libraries for the computation of PH. (See Table 2 in Section 7 for information about which complexes have been implemented.)

5.2.6 Reduction techniques

Thus far, we have discussed techniques to build simplicial complexes with possibly 'few' simplices. One can also take an alternative approach to speed up the computation of PH. For example, one can use a heuristic (i.e., a method without theoretical guarantees on the speed-up) to reduce the size of a filtered complex while leaving the PH unchanged.

For simplicial complexes, one such method is based on discrete Morse theory [114], which was adapted to filtrations of simplicial complexes in [115]. The basic idea of the algorithm developed in [115] is that one can compute a partial matching of the simplices in a filtered simplicial complex so that (i) pairs occur only between simplices that enter the filtration at the same step, (ii) unpaired simplices determine the homology, and (iii) one can remove paired simplices from the filtered complex without altering the PH. Such deletions are examples of the elementary simplicial collapses that we mentioned in Section 3.1. Unfortunately, the problem of finding an optimal partial matching was shown to be NP complete [116], and one thus relies on heuristics to find partial matchings to reduce the size of the complex.

One particular family of elementary collapses, called *strong collapses*, was introduced in [117]. Strong collapses preserve cycles of shortest length in the representative class of a generator of a hole [118]; this feature makes strong collapses useful for finding holes in networks [118]. A distributed version of the algorithm proposed in [118] was presented in [119] and adapted for the computation of PH in [120].

A method for the reduction of the size of a complex for clique complexes, such as the VR complex, was proposed in [121] and is called the *tidy-set method*. Using maximal cliques, this method extracts a minimal representation of the graph that determines the clique complex. Although the tidy-set method cannot be extended to filtered complexes, it can be used for the computation of zigzag PH (see Section 6) [122]. The tidy-set method is a heuristic, because it does not give a guarantee to minimize the size of the output complex.

5.3 From a filtered simplicial complex to barcodes

To compute the PH of a filtered simplicial complex K and obtain a barcode like the one illustrated in Figure 5(c), we need to associate to it a matrix — the so-called *boundary matrix* — that stores information about the faces of every simplex. To do this, we place a total ordering on the simplices of the complex that is compatible with the filtration in the following sense:

- a face of a simplex precedes the simplex;

Algorithm 1 The standard algorithm for the reduction of the boundary matrix to barcodes

---

for $j = 1$ to n **do**
  while there exists $i < j$ with $\mathrm{low}(i) = \mathrm{low}(j)$ **do**
    add column i to column j
  end while
end for

- a simplex in the ith complex K_i precedes simplices in K_j for $j > i$, which are not in K_i.

Let n denote the total number of simplices in the complex, and let $\sigma_1, \ldots, \sigma_n$ denote the simplices with respect to this ordering. We construct a square matrix δ of dimension $n \times n$ by storing a 1 in $\delta(i,j)$ if the simplex σ_i is a face of simplex σ_j of codimension 1; otherwise, we store a 0 in $\delta(i,j)$.

Once one has constructed the boundary matrix, one has to reduce it using Gaussian elimination.[f] In the following subsections, we discuss several algorithms for reducing the boundary matrix.

5.3.1 Standard algorithm

The so-called `standard algorithm` for the computation of PH was introduced for the field $\mathbb{F}_2$ in [60] and for general fields in [61]. For every $j \in \{1, \ldots, n\}$, we define $\mathrm{low}(j)$ to be the largest index value i such that $\delta(i,j)$ is different from 0.[g] If column j only contains 0 entries, then the value of $\mathrm{low}(j)$ is undefined. We say that the boundary matrix is *reduced* if the map low is injective on its domain of definition. In Algorithm 1, we illustrate the standard algorithm for reducing the boundary matrix. Because this algorithm operates on columns of the matrix from left to right, it is also sometimes called the 'column algorithm.' In the worst case, the complexity of the standard algorithm is cubic in the number of simplices.

5.3.2 Reading off the intervals

Once the boundary matrix is reduced, one can read off the intervals of the barcode by pairing the simplices in the following way:

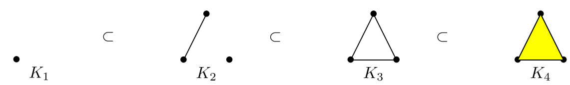
- If $\mathrm{low}(j) = i$, then the simplex σ_j is paired with σ_i, and the entrance of σ_i in the filtration causes the birth of a feature that dies with the entrance of σ_j.
- If $\mathrm{low}(j)$ is undefined, then the entrance of the simplex σ_j in the filtration causes the birth of a feature. It there exists k such that $\mathrm{low}(k) = j$, then σ_j is paired with the simplex σ_k, whose entrance in the filtration causes the death of the feature. If no such k exists, then σ_j is unpaired.

A pair (σ_i, σ_j) gives the half-open interval $[\mathrm{dg}(\sigma_i), \mathrm{dg}(\sigma_j))$ in the barcode, where for a simplex $\sigma \in K$ we define $\mathrm{dg}(\sigma)$ to be the smallest number l such that $\sigma \in K_l$. An unpaired simplex σ_k gives the infinite interval $[\mathrm{dg}(\sigma_k), \infty)$. We give an example of PH computation in Figure 7.

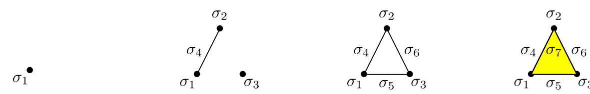
5.3.3 Other algorithms

After the introduction of the standard algorithm, several new algorithms were developed. Each of these algorithms gives the same output for the computation of PH, so we only give a brief overview and references to these algorithms, as one does not need to know them

(a) A filtered simplicial complex:



(b) We put a total order on the simplices that is compatible with the filtration:



where σ_i denotes the ith simplex in this order.

(c) (Left) The boundary matrix B for the filtered simplicial complex in (a) with respect to order on simplices in (b), and (right) its reduction $\overline{B}$ given by applying Algorithm 1 (one first adds column 5 to column 6, and then column 4 to column 6):

$$B = \begin{pmatrix} 0&0&0&1&1&0&0\\ 0&0&0&1&0&1&0\\ 0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0 \end{pmatrix} \qquad \overline{B} = \begin{pmatrix} 0&0&0&1&1&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0 \end{pmatrix}.$$

(d) We read off the following intervals from the matrix $\overline{B}$ in (c):

- σ_1 is positive, unpaired; this gives the interval $[1, \infty)$ in H_0.
- σ_2 is positive, paired with σ_4; this gives no interval, because σ_2 and σ_4 enter at the same time in the filtration.
- σ_3 is positive, paired with σ_5: this gives the interval $[2, 3)$ in H_0.
- σ_6 is positive, paired with σ_7: this gives the interval $[3, 4)$ in H_1.

Figure 7 Example of PH computation with the `standard algorithm` (see Algorithm 1).

to compute PH with one of the publicly-available software packages. In Section 7.2, we indicate which implementation of these libraries is best suited to which data set.

As we mentioned in Section 5.3.1, in the worst case, the standard algorithm has cubic complexity in the number of simplices. This bound is sharp, as Morozov gave an example of a complex with cubic complexity in [123]. Note that in cases such as when matrices are sparse, complexity is less than cubic. Milosavljević, Morozov, and Skraba [124] introduced an algorithm for the reduction of the boundary matrix in $\mathcal{O}(n^\omega)$, where ω is the matrix-multiplication coefficient (i.e., $\mathcal{O}(n^\omega)$ is the complexity of the multiplication of two square matrices of size n). At present, the best bound for ω is 2.376 [125]. Many other algorithms have been proposed for the reduction of the boundary matrix. These algorithms give a heuristic speed-up for many data sets and complexes (see the benchmarkings in the forthcoming references), but they still have cubic complexity in the number of simplices. Sequential algorithms include the `twist algorithm` [126] and the `dual algorithm` [72, 127]. (Note that the dual algorithm is known to give a speed-up when one computes PH with the VR complex, but not necessarily for other types of complexes (see also the results of our benchmarking for the **vertebra** data set in Additional file 1 of the SI).) Parallel algorithms in a shared setting include the `spectral-sequence algorithm` (see Section VII.4 of [80]) and the `chunk algorithm` [128]; parallel algorithms in a distributed setting include the `distributed algorithm` [74]. The `multifield algorithm` is a sequential algorithm that allows the simultaneous computation of PH over several fields [129].

5.4 Statistical interpretation of topological summaries

Once one has obtained barcodes, one needs to interpret the results of computations. In applications, one often wants to compare the output of a computation for a certain data set with the output for a null model. Alternatively, one may be studying data sets from the output of a generative model (e.g., many realizations from a model of random networks), and it is then necessary to average results over multiple realizations. In the first instance, one needs both a way to compare the two different outputs and a way to evaluate the significance of the result for the original data set. In the second case, one needs a way to calculate appropriate averages (e.g., summary statistics) of the result of the computations.

From a statistical perspective, one can interpret a barcode as an unknown quantity that one tries to estimate by computing PH. If one wants to use PH in applications, one thus needs a reliable way to apply statistical methods to the output of the computation of PH. To our knowledge, statistical methods for PH were addressed for the first time in the paper [130]. Roughly speaking, there are three current approaches to the problem of statistical analysis of barcodes. In the first approach, researchers study topological properties of random simplicial complexes (see, e.g., [131, 132]) and the review papers [133, 134]. One can view random simplicial complexes as null models to compare with empirical data when studying PH. In the second approach, one studies properties of a metric space whose points are persistence diagrams. In the third approach, one studies 'features' of persistence diagrams. We will provide a bit more detail about the second and third approaches.

In the second approach, one considers an appropriately defined 'space of persistence diagrams,' defines a distance function on it, studies geometric properties of this space, and does standard statistical calculations (means, medians, statistical tests, and so on). Recall that a persistence diagram (see Figure 5 for an example) is a multiset of points in $\overline{\mathbb{R}}^2$ and that it gives the same information as a barcode. We now give the following precise definition of a persistence diagram.

Definition 8 A *persistence diagram* is a multiset that is the union of a finite multiset of points in $\overline{\mathbb{R}}^2$ with the multiset of points on the diagonal $\Delta = \{(x,y) \in \mathbb{R}^2 \mid x = y\}$, where each point on the diagonal has infinite multiplicity.

In this definition, we include all of the points on the diagonal in $\mathbb{R}^2$ with infinite multiplicity for technical reasons. Roughly, it is desirable to be able to compare persistence diagrams by studying bijections between their elements, and persistence diagrams must thus be sets with the same cardinality.

Given two persistence diagrams X and Y, we consider the following general definition of distance between X and Y.

Definition 9 Let $p \in [1,\infty]$. The *pth Wasserstein distance* between X and Y is defined as

$$W_p[\mathrm{d}](X,Y) := \inf_{\phi:X\to Y} \left[\sum_{x\in X} \mathrm{d}\big[x,\phi(x)\big]^p \right]^{1/p}$$

for $p \in [1,\infty)$ and as

$$W_\infty[\mathrm{d}](X,Y) := \inf_{\phi:X\to Y} \sup_{x\in X} \mathrm{d}\big[x,\phi(x)\big]$$

for $p = \infty$, where d is a metric on $\mathbb{R}^2$ and ϕ ranges over all bijections from X to Y.

Usually, one takes $\mathrm{d} = L_q$ for $q \in [1, \infty]$. One of the most commonly employed distance functions is the *bottleneck distance* $W_\infty[L_\infty]$.

The development of statistical analysis on the space of persistence diagrams is an area of ongoing research, and presently there are few tools that can be used in applications. See [135–137] for research in this direction. Until recently, the library Dionysus [64] was the only library to implement computation of the bottleneck and Wasserstein distances (for $\mathrm{d} = L_\infty$); the library hera [138] implements a new algorithm [139] for the computation of the bottleneck and Wasserstein distances that significantly outperforms the implementation in Dionysus. The library TDA Package [140] (see [69] for the accompanying tutorial) implements the computation of confidence sets for persistence diagrams that was developed in [141], distance functions that are robust to noise and outliers [142], and many more tools for interpreting barcodes.

The third approach for the development of statistical tools for PH consists of mapping the space of persistence diagrams to spaces (e.g., Banach spaces) that are amenable to statistical analysis and machine-learning techniques. Such methods include persistence landscapes [143], using the space of algebraic functions [144], persistence images [145], and kernelization techniques [146–149]. See the papers [6, 52] for applications of persistence landscapes. The package Persistence Landscape Toolbox [150] (see [70] for the accompanying tutorial) implements the computation of persistence landscapes, as well as many statistical tools that one can apply to persistence landscapes, such as mean, ANOVA, hypothesis tests, and many more.

5.5 Stability

As we mentioned in Section 1, PH is useful for applications because it is stable with respect to small perturbations in the input data.

The first stability theorem for PH, proven in [151], asserts that, under favorable conditions, step (2) in the pipeline in Figure 6 is 1-Lipschitz with respect to suitable distance functions on filtered complexes and the bottleneck distance for barcodes (see Section 5.4). This result was generalized in the papers [152–154]. Stability for PH is an active area of research; for an overview of stability results, their history and recent developments, see [82], Chapter 3.

6 Excursus: generalized persistence

One can use the algorithms that we described in Section 5 to compute PH when one has a sequence of complexes with inclusion maps that are all going in the same direction, as in the following diagram:

$$\cdots \to K_{i-1} \to K_i \to K_{i+1} \to \cdots .$$

An algorithm, called the `zigzag algorithm`, for the computation of PH for inclusion maps that do not all go in the same direction, as, e.g., in the diagram

$$\cdots \to K_{i-1} \to K_i \leftarrow K_{i+1} \to \cdots$$

was introduced in [155]. In the more general setting in which maps are not inclusions, one can still compute PH using the `simplicial map algorithm` [156].

One may also wish to vary two or more parameters instead of one. This yields multi-filtered simplicial complexes, as, e.g., in the following diagram:

$$\begin{array}{ccccccccc}
 & & \vdots & & \vdots & & \vdots & & \\
 & & \downarrow & & \downarrow & & \downarrow & & \\
\cdots & \rightarrow & K_{j+1,i-1} & \rightarrow & K_{j+1,i} & \rightarrow & K_{j+1,i+1} & \rightarrow & \cdots \\
 & & \downarrow & & \downarrow & & \downarrow & & \\
\cdots & \rightarrow & K_{j,i-1} & \rightarrow & K_{j,i} & \rightarrow & K_{j,i+1} & \rightarrow & \cdots \\
 & & \downarrow & & \downarrow & & \downarrow & & \\
\cdots & \rightarrow & K_{j-1,i-1} & \rightarrow & K_{j-1,i} & \rightarrow & K_{j-1,i+1} & \rightarrow & \cdots \\
 & & \downarrow & & \downarrow & & \downarrow & & \\
 & & \vdots & & \vdots & & \vdots & &
\end{array}$$

In this case, one speaks of *multi-parameter persistent homology*. Unfortunately, the Fundamental Theorem of Persistent Homology is no longer valid if one filters with more than one parameter, and there is no such thing as a 'generalized interval.' The topic of multi-parameter persistence is under active research, and several approaches are being studied to extract topological information from multi-filtered simplicial complexes. See [82, 157] for the theory of multi-parameter persistent homology, and see [158] (and [159] for its companion paper) for upcoming software for the visualization of invariants for 2-parameter persistent homology.

7 Software

There are several publicly-available implementations for the computation of PH. We give an overview of the libraries with accompanying peer-reviewed publication and summarize their properties in Table 2.

The software package javaPlex [66], which was developed by the computational topology group at Stanford University, is based on the Plex library [161], which to our knowledge is the first piece of software to implement the computation of PH. Perseus [65] was developed to implement Morse-theoretic reductions [115] (see Section 5.2.6). jHoles [162] is a Java library for computing the weight rank clique filtration for weighted undirected networks [89]. Dionysus [64] is the first software package to implement the `dual algorithm` [72, 127]. PHAT [62] is a library that implements several algorithms and data structures for the fast computation of barcodes, takes a boundary matrix as input, and is the first software to implement a matrix-reduction algorithm that can be executed in parallel. DIPHA [63], a spin-off of PHAT, implements a distributed computation of the matrix-reduction algorithm. Gudhi [67] implements new data structures for simplicial complexes and the boundary matrix. It also implements the `multi-field algorithm`, which allows simultaneous computation of PH over several fields [129]. This library is currently under intense development, and a Python interface was just released in the most recent version of the library (namely, Version 2.0.0, whereas the version that we study in our tests is Version 1.3.1). The library ripser [68], the most recently developed software of the set that we examine, uses several optimizations and shortcuts to speed up the computation of PH with the VR complex. This library does not have an accompanying peer-reviewed publication. However, because it is currently the best-performing

Table 2 Overview of existing software for the computation of PH that have an accompanying peer-reviewed publication (and also RIPSER [68], because of its performance)

Software	JAVAPLEX	PERSEUS	JHOLES	DIONYSUS	PHAT	DIPHA	GUDHI	SIMPPERS	RIPSER
(a) Language	Java	C++	Java	C++	C++	C++	C++	C++	C++
(b) Algorithms for PH	standard, dual, zigzag	Morse reductions, standard	standard (uses JAVAPLEX)	standard, dual, zigzag	standard, dual, twist, chunk, spectral sequence	twist, dual, distributed	dual, multifield	simplicial map	twist, dual
(c) Coefficient field	$\mathbb{Q}, \mathbb{F}_p$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$ (standard, zigzag), $\mathbb{F}_p$ (dual)	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_p$	$\mathbb{F}_2$	$\mathbb{F}_p$
(d) Homology	simplicial, cellular	simplicial, cubical	simplicial	simplicial	simplicial, cubical	simplicial, cubical	simplicial, cubical	simplicial	simplicial
(e) Filtrations computed	VR, $\mathbb{W}$, $\mathbb{W}_\nu$	VR, lower star of cubical complex	WRCF	VR, α, Č	-	VR, lower star of cubical complex	VR, α, $\mathbb{W}$, lower star of cubical complex	-	VR
(f) Filtrations as input	simplicial complex, zigzag, CW	simplicial complex, cubical complex	-	simplicial complex, zigzag	boundary matrix of simplicial complex	boundary matrix of simplicial complex	-	map of simplicial complexes	-
(g) Additional features	Computes some homological algebra constructions, homology generators	weighted points for VR	-	vineyards, circle-valued functions, homology generators	-	-	-	-	-
(h) Visualization	barcodes	persistence diagram	-	-	-	persistence diagram	-	-	-

The symbol '-' signifies that the associated feature is not implemented. For each software package, we indicate the following items. (a) The language in which it is implemented. (b) The implemented algorithms for the computation of barcodes from the boundary matrix. (c) The coefficient fields for which PH is computed, where the letter p denotes any prime number in the coefficient field $\mathbb{F}_p$. (d) The type of homology computed. (e) The filtered complexes that are computed, where VR stands for Vietoris–Rips complex, $\mathbb{W}$ stands for the weak witness complex, $\mathbb{W}_\nu$ stands for parametrized witness complexes, WRCF stands for the weight rank clique filtration, α stands for the alpha complex, and Č for the Čech complex. PERSEUS, DIPHA, and GUDHI implement the computation of the lower-star filtration [160] of a weighted cubical complex; one inputs data in the form of a d-dimensional array; the data is then interpreted as a d-dimensional cubical complex, and its lower-star filtration is computed. (See the Tutorial in Additional file 2 of the SI, for more details.) Note that DIPHA and GUDHI use the efficient representation of cubical complexes presented in [55], so the size of the cubical complex that is computed by these libraries is smaller than the size of the resulting complex with PERSEUS. (f) The filtered complexes that one can give as input. JAVAPLEX supports the input of a filtered CW complex for the computation of cellular homology [78]; in contrast with simplicial complexes, there do not currently exist algorithms to assign a cell complex to point-cloud data. (g) Additional features implemented by the library. JAVAPLEX supports the computation of some constructions from homological algebra (see [66] for details), and PERSEUS implements the computation of PH with the VR for points with different 'birth times' (see Section 5.1.3). The library DIONYSUS implements the computation of vineyards [155] and circle-valued functions [127]. Both JAVAPLEX and DIONYSUS support the output of representatives of homology classes for the intervals in a barcode. (h) Whether visualization of the output is provided (both in terms of memory usage and in terms of wall-time seconds[h]) library for the computation of PH with the VR complex, we include it in our study. The library SimpPers [163] implements the `simplicial map algorithm`. Libraries that implement techniques for the statistical interpretation of barcodes include the TDA Package [140] and the Persistence Landscape Toolbox [150]. (See Section 5.4 for additional libraries for the interpretation of barcodes.) RIVET, a package for visualizing 2-parameter persistent homology, is slated to be released soon [158]. We summarize the properties of the libraries for the computation of PH that we mentioned in this paragraph in Table 2, and we discuss the performance for a selection of them in Section 7.1.3 and in Additional file 1 of the SI. For a list of programs, see https://github.com/n-otter/PH-roadmap.

7.1 Benchmarking

We benchmark a subset of the currently available open-source libraries with peer-reviewed publication for the computation of PH. To our knowledge, the published open-source libraries are jHoles, javaPlex, Perseus, Dionysus, PHAT, DIPHA, SimpPers, and Gudhi. To these, we add the library ripser, which is currently the best-performing library to compute PH with the VR complex. To study the performance of the packages, we restrict our attention to the algorithms that are implemented by the largest number of libraries. These are the VR complex and the standard and dual algorithms for the reduction of the boundary matrix. PHAT only takes a boundary matrix as input, so it is not possible to conduct a direct comparison of it with the other implementations. However, the fast data structures and algorithms implemented in PHAT are also implemented in its spin-off software DIPHA, which we include in the benchmarking. The software jHoles computes PH using the WRCF for weighted undirected networks, and SimpPers takes a map of simplicial complexes as input, so these two libraries cannot be compared directly to the other libraries. In Additional file 1 of the SI, we report benchmarking of some additional features that are implemented by some of the six libraries (i.e., javaPlex, Perseus, Dionysus, DIPHA, Gudhi, and Ripser) that we test. Specifically, we report results for the computation of PH with cubical complexes for image data sets and the computation of PH with witness, alpha, and Čech complexes.

We study the software packages javaPlex, Perseus, Dionysus, DIPHA, Gudhi, and Ripser using both synthetic and real-world data from three different perspectives:

1. Performance measured in CPU seconds and wall-time (i.e., elapsed time) seconds.
2. Memory required by the process.
3. Maximum size of simplicial complex allowed by the software.

7.1.1 Data sets

In this subsection, we describe the data sets that we use for our benchmarking. We use data sets from a variety of different mathematical and scientific areas and applications. In each case, when possible, we use data sets that have already been studied using PH. Our list of data sets is far from complete; we view this list as an initial step towards building a comprehensive collection of benchmarking data sets for PH.

Data sets (1)–(4) are synthetic; they arise from topology (1), stochastic topology (2), dynamical systems (3), and from an area at the intersection of network theory and fractal geometry (4). (As we discuss below, data set (4) was used originally to study connection patterns in the cerebral cortex.) Data sets (5)–(12) are from empirical experiments and

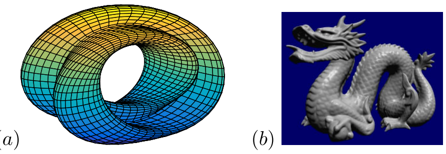


Figure 8 Two well-known examples. (a) Plot of the image of the figure-8 immersion of the Klein bottle and **(b)** the reconstruction of the Stanford Dragon (retrieved from [164]).

measurements: they arise from phylogenetics (5)–(6), image analysis (7), genomics (9), neuroscience (8), medical imaging (10), political science (11), and scientometrics (12).

In each case, these data sets are of one of the following three types: point clouds, weighted undirected networks, and gray-scale digital images. To obtain a point cloud from a real-world weighted undirected network, we compute shortest paths using the inverse of the nonzero weights of edges as distances between nodes (except for the US Congress networks and the human genome network; see below). For the synthetic networks, the values assigned to edges are interpreted as distances between nodes, and we therefore use these values to compute shortest paths. We make all processed versions of the data sets that we use in the benchmarking available at https://github.com/n-otter/PH-roadmap/tree/master/data_sets. We provide the scripts that we used to produce the synthetic data sets at https://github.com/n-otter/PH-roadmap/tree/master/matlab/synthetic_data_sets_scripts.

We now describe all data sets in detail:

(1) Klein bottle. The Klein bottle is a one-sided nonorientable surface (see Figure 8). We linearly sample points from the Klein bottle using its 'figure-8' immersion in $\mathbb{R}^3$ and size sample of 400 points. We denote this data set by **Klein**. Note that the image of the immersion of the Klein bottle does not have the same homotopy type as the original Klein bottle, but it does have the same singular homology[i] with coefficients in $\mathbb{F}_2$. We have $H_0(B) = \mathbb{F}_2$, $H_1(B) = \mathbb{F}_2 \oplus \mathbb{F}_2$, and $H_2(B) = \mathbb{F}_2$, where B denotes the Klein bottle and $H_i(B)$ is the ith singular homology group with coefficients in $\mathbb{F}_2$.

(2) Random VR complexes (uniform distribution) [165]. The parameters for this model are positive integers N and d; the random VR complex for parameters N and d is the VR complex $\mathrm{VR}_\epsilon(X)$, where X is a set of N points sampled from $\mathbb{R}^d$. (Equivalently, the random VR complex is the clique complex on the random geometric graph $G(N, \epsilon)$ [166].) We sample N points uniformly at random from $[0,1]^d$. We choose $(N,d) = (50,16)$ and we denote this data set by **random**. The homology of random VR complexes was studied in [165].

(3) Vicsek biological aggregation model. This model was first introduced in [167] and was studied using PH in [48]. We implement the model in the form in which it appears in [48]. The model describes the motion of a collection of particles that interact in a square with periodic boundary conditions. The parameters for the model are the length l of the side of the square, the initial angle θ_0, the (constant) absolute value v_0 for the velocity, the number N of particles, a noise parameter η, and the number T of time steps. The output of the model is a point cloud in 3-dimensional Euclidean space in which each point is specified by its position in the 2-dimensional box and its velocity angle ('heading'). We run three simulations of the model using the parameter values from [48]. For each simulation, we choose

two point clouds that correspond to two different time frames. See [48] for further details. We denote this data set by **Vicsek**.

(4) Fractal networks. These are self-similar networks introduced in [168] to investigate whether connection patterns of the cerebral cortex are arranged in self-similar patterns. The parameters for this model are natural numbers b, k, and n. To generate a fractal network, one starts with a fully-connected network with 2^b nodes. Two copies of this network are connected to each other so that the 'connection density' between them is k^{-1}, where the connection density is the number of edges between the two copies divided by the number of total possible edges between them. Two copies of the resulting network are connected with connection density k^{-2}. One repeats this type of connection process until the network has size 2^n, but with a decrease in the connection density by a factor of $1/k$ at each step.

   We define distances between nodes in two different ways: (1) uniformly at random, and (2) with linear weight-degree correlations. In the latter, the distance between nodes i and j is distributed as $k_i k_j X$, where k_i is the degree of node i and X is a random variable uniformly distributed on the unit interval. We use the parameters $b = 5$, $n = 9$, and $k = 2$; and we compute PH for the weighted network and for the network in which all adjacent nodes have distance 1. We denote this data set by **fract** and distinguish between the two ways of defining distances between weights using the abbreviations 'r' for random, and 'l' for linear.

(5) Genomic sequences of the HIV virus. We construct a finite metric space using the independent and concatenated sequences of the three largest genes — `gag`, `pol`, and `env` — of the HIV genome. We take 1,088 different genomic sequences and compute distances between them by using the Hamming distance. We use the aligned sequences studied using PH in [23]. (The authors of that paper retrieved the sequences from [169].) We denote this data set by **HIV**.

(6) Genomic sequences of H3N2. This data set consists of 1,000 different genomic sequences of H3N2 influenza. We compute the Hamming distance between sequences. We use the aligned sequences studied using PH in [23]. We denote this data set by **H3N2**.

(7) Stanford Dragon graphic. We sample points uniformly at random from 3-dimensional scans of the dragon [164], whose reconstruction we show in Figure 8. The sample sizes include 1,000 and 2,000 points. We denote these data sets by **drag 1** and **drag 2**, respectively.

(8) *C. elegans* neuronal network. This is a weighted, undirected network in which each node is a neuron and edges represent synapses or gap junctions. We use the network studied using PH in [89]. (The authors of the paper used the data set studied in [170], which first appeared in [171].) Recall that for this example, and also for the other real-world weighted networks (except for the human genome network and the US Congress networks), we convert each nonzero edge weight to a distance by taking its inverse. We denote this data set by **eleg**.

(9) Human genome. This is a weighted, undirected network representing a sample of the human genome. We use the network studied using PH in [89]. (The authors of that paper created the sample using data retrieved from [172].) Each node represents a gene, and two nodes are adjacent if there is a nonzero correlation

between the expression levels of the corresponding genes. We define the weight of an edge as the inverse of the correlation.[j] We denote this data set by **genome**.

(10) Gray-scale image: 3-dimensional rotational angiography scan of a head with an aneurysm. This data set was used in the benchmarking in [74]. This data set is given by a 3-dimensional array of size $512 \times 512 \times 512$, where each entry stores an integer that represents the grey value for the corresponding voxel. We retrieved the data set from the repository [173]. We denote this data set by **vertebra**.

(11) US Congress roll-call voting networks. These two networks (the Senate and House of Representatives from the 104th United States Congress) are constructed using the procedure in [174] from data compiled by [175]. In each network, a node is a legislator (Senators in one data set and Representatives in the other), and there is a weighted edge between legislators i and j, where the weight $w_{i,j}$ is a number in $[0, 1]$ (it is equal to 0 if and only if legislators i and j never voted the same way on any bill) given by the number of times the two legislators voted in the same way divided by the total number of bills on which they both voted. See [174] for further details. We denote the networks from the Senate and House by **senate** and **house**, respectively. The network **senate** has 103 nodes, and the network **house** has 445 nodes. To compute shortest paths, we define the distance between two nodes i and j to be $1 - w_{i,j}$. In the 104th Congress, no two politicians voted in the same way on every bill, so we do not have distinct nodes with 0 distance between them. (This is important, for example, if one wants to apply multidimensional scaling.)

(12) Network of network scientists. This is a weighted, undirected network representing the largest connected component of a collaboration network of network scientists [176]. Nodes represent authors and edges represent collaborations, and weights indicate the number of joint papers. The largest connected component consists of 379 nodes. We denote this data set by **netw-sc**.

7.1.2 Machines and compilers

We tested the libraries on both a cluster and a shared-memory system. The cluster is a Dell Sandybridge cluster, it has 1,728 (i.e., 108×16) cores of 2.0 GHz Xeon SandyBridge, RAM of 64 GiB in 80 nodes and RAM of 128 GiB in 4 nodes, and a scratch disk of 20 TB. It runs the operating system (OS) Red Hat Enterprise Linux 6. The shared-memory system is an IBM System x3550 M4 server with 16 (i.e., 2×8) cores of 3.3 GHz, RAM of 768 GB, and storage of 3 TB. It runs the OS Ubuntu 14.04.01.[k] The major difference in running shared algorithms on the shared-memory system versus the distributed-memory system is that each node in the former has much more available RAM than in the latter. (See also the difference in performance between computations on cluster and shared memory system in Tables 3 and 4.) To compile Gudhi, DIPHA, Perseus, and Dionysus, we used the compiler gcc 4.8.2 on the cluster and gcc 4.8.4 on the shared-memory system; for both machines, we used the (default) optimization -O3. Additionally, we used openmpi 1.8.3 for DIPHA.

7.1.3 Tests and results

We now report the details and results of the tests that we performed. We have made the data sets, header file to measure memory, and other information related to the tests available at https://github.com/n-otter/PH-roadmap. Of the six software packages that

Table 3 Performance of the software packages measured in wall-time (i.e., elapsed time), and CPU seconds (for the computations running on the cluster)

Data set	(a) Computations on cluster: wall-time seconds					
	eleg	**Klein**	**HIV**	**drag 2**	**random**	**genome**
Size of complex	4.4×10^6	1.1×10^7	2.1×10^8	1.3×10^9	3.1×10^9	4.5×10^8
Max. dim.	2	2	2	2	8	2
JAVAPLEX (st)	84	747	-	-	-	-
DIONYSUS (st)	474	1,830	-	-	-	-
DIPHA (st)	6	90	1,631	142,559	-	9,110
PERSEUS	543	1,978	-	-	-	-
DIONYSUS (d)	513	145	-	-	-	-
DIPHA (d)	4	6	81	2,358	5,096	232
GUDHI	36	89	1,798	14,368	-	4,753
RIPSER	1	1	2	6	349	3

Data set	(b) Computations on cluster: CPU seconds					
	eleg	**Klein**	**HIV**	**drag 2**	**random**	**genome**
Size of complex	4.4×10^6	1.1×10^7	2.1×10^8	1.3×10^9	3.1×10^9	4.5×10^8
Max. dim.	2	2	2	2	8	2
JAVAPLEX (st)	284	1,031	-	-	-	-
DIONYSUS (st)	473	1,824	-	-	-	-
DIPHA (st)	68	1,360	25,950	1,489,615	-	130,972
PERSEUS	542	1,974	-	-	-	-
DIONYSUS (d)	513	145	-	-	-	-
DIPHA (d)	39	73	1,276	37,572	79,691	3,622
GUDHI	36	88	1,794	14,351	-	4,764
RIPSER	1	1	2	5	348	2

Data set	(c) Computations on shared-memory system: wall-time seconds					
	eleg	**Klein**	**HIV**	**drag 2**	**genome**	**fract r**
Size of complex	3.2×10^8	1.1×10^7	2.1×10^8	1.3×10^9	4.5×10^8	2.8×10^9
Max. dim.	3	2	2	2	2	3
JAVAPLEX (st)	13,607	1,358	43,861	-	28,064	-
PERSEUS	-	1,271	-	-	-	-
DIONYSUS (d)	-	100	142,055	35,366	-	572,764
DIPHA (d)	926	13	773	4,482	1,775	3,923
GUDHI	381	6	177	1,518	442	4,590
RIPSER	2	1	2	5	3	1,517

For each data set, we indicate the size of the simplicial complex and the maximum dimension up to which we construct the VR complex. For all data sets, we construct the filtered VR complex up to the maximum distance between any two points. We indicate the implementation of the standard algorithm using the abbreviation 'st' following the name of the package, and we indicate the implementation of the dual algorithm using the abbreviation 'd.' The symbol '-' signifies that we were unable to finish computations for this data set, because the machine ran out of memory. PERSEUS implements only the standard algorithm, and GUDHI and RIPSER implement only the dual algorithm. (a), (b) We run DIPHA on one node and 16 cores for the data sets **eleg**, **Klein**, and **genome**; on 2 nodes of 16 cores for the **HIV** data set; on 2 and 3 nodes of 16 cores for the dual and standard implementations, respectively, for **drag 2**; and on 8 nodes of 16 cores for **random**. (The maximum number of processes that we could use at any one time was 128.) (c) We run DIPHA on a single core.

we study, four implement the computation of the dual algorithm, and four implement the standard algorithm. It is reported in [66] that JAVAPLEX implements the dual algorithm, but the implementation of the algorithm has a bug and gives a wrong output. To our knowledge, this bug has not yet been fixed (at the time of writing), and we therefore test only the standard algorithm.

For the computations on the cluster, we compare the libraries running both the dual algorithm and the standard algorithm. The package DIPHA is the only one to implement a distributed computation. As a default, we run the software on one node and 16 cores; we only increase the number of nodes and cores employed when the machine runs out of memory. However, augmenting the number of nodes can make the computations faster

Table 4 Memory usage in GB for the computations summarized in Table 3

Data set	(a) Computations on cluster					
	eleg	**Klein**	**HIV**	**drag 2**	**random**	**genome**
Size of complex	4.4×10^6	1.1×10^7	2.1×10^8	1.3×10^9	3.1×10^9	4.5×10^8
Max. dim.	2	2	2	2	8	2
JAVAPLEX (st)	<5	<15	>64	>64	>64	>64
DIONYSUS (st)	1.3	11.6	-	-	-	-
DIPHA (st)	0.1	0.2	2.7	4.9	-	4.8
PERSEUS	5.1	12.7	-	-	-	-
DIONYSUS (d)	0.5	1.1	-	-	-	-
DIPHA (d)	0.1	0.2	1.8	13.8	9.6	6.3
GUDHI	0.2	0.5	8.5	62.8	-	21.5
RIPSER	0.007	0.02	0.06	0.2	24.7	0.07

Data set	(b) Computations on shared-memory system					
	eleg	**Klein**	**HIV**	**drag 2**	**genome**	**fract r**
Size of complex	3.2×10^8	1.1×10^7	2.1×10^8	1.3×10^9	4.5×10^8	2.8×10^9
Max. dim.	3	2	2	2	2	3
JAVAPLEX (st)	<600	<15	<700	>700	<700	>700
PERSEUS	-	11.7	-	-	-	-
DIONYSUS (d)	-	1.1	16.8	134.2	-	268.5
DIPHA (d)	31.2	0.9	17.7	109.5	37.3	276.1
GUDHI	15.4	0.5	10.2	62.8	21.4	134.8
RIPSER	0.2	0.03	0.07	0.2	0.07	155

For JAVAPLEX, we indicate the value of the maximum heap size that was sufficient to perform the computation. The value that we give is an upper bound to the memory usage. For DIPHA, we indicate the maximum memory used by a single core (considering all cores). See Table 3 for details on the number of cores used.

(in terms of CPU seconds) for complexes of all sizes.[1] We see this in our experiments, and it is also discussed in [74]. For the other packages, we run the computations on a single node with one core.

For computations on the shared-memory system, we compare the libraries using only the dual algorithm if they implement it, and we otherwise use the standard algorithm. For the shared-memory system, we run all packages (including DIPHA) on a single core.

In our benchmarking, we report mean computation times and memory measurements. In Table 3, we give the computation times for the different software packages. We measure elapsed and CPU time by using the `time` function in Linux. We report computation times with a precision of one second; if a computation took only fractions of a second, we report 'one second' as the computation time. For space reasons, we report results for a subset of the computations. (In Additional file 1 of the SI, we tabulate the rest of our computations.) In Table 4, we report the memory used by the processes in terms of maximum resident set size (RSS); in other words, we give the maximum amount of real RAM a program has used during its execution. We measure the maximum RSS using the `getrusage` function in Linux. The header file that we use to measure memory is available at https://github.com/n-otter/PH-roadmap. In DIPHA, the measurement of memory is already implemented by the authors of the software. They also use the `getrusage` function in Linux. The package JAVAPLEX is written in Java, and we thus cannot measure its memory as we do for the other packages. However, one can infer memory requirements for this software package using the value of the maximal heap size necessary to perform the computations; we report this value in Table 4. In Table 5, we give the maximum size of the simplicial complex for which we were able to compute PH with each software package in our benchmarkings.

Table 5 For each software package in (a), we indicate in (b) the maximal size of the simplicial complex supported by it thus far in our tests

(a)	javaPlex	Dionysus		DIPHA		Perseus	Gudhi	Ripser
	st	st	d	st	d	st	d	d
(b)	$4.5 \cdot 10^8$	$1.1 \cdot 10^7$	2.8×10^9	$1.3 \cdot 10^9$	$3.4 \cdot 10^9$	$1 \cdot 10^7$	$3.4 \cdot 10^9$	$3.4 \cdot 10^9$

7.2 Conclusions from our benchmarking

Our tests suggest that Ripser is the best-performing library currently available for the computation of PH with the Vietoris–Rips complex, and in order of decreasing performance, that Gudhi and DIPHA are the next-best implementations. For the computation of PH with cubical complexes, Gudhi outperforms DIPHA by a factor of 3 to 4 in terms of memory usage, and DIPHA outperforms Gudhi in terms of wall-time seconds by a factor of 1 to 2 (when running on one core on a shared-memory system). Both DIPHA and Gudhi significantly outperform the implementation in Perseus. For the computation of PH with the alpha complex, we did not observe any significant differences in performance between the libraries Gudhi and Dionysus. Because the alpha complex has fewer simplices than the other complexes that we used in our tests, further tests with larger data sets may be appropriate in future benchmarkings.

There is a huge disparity between implementations of the dual and standard algorithms when computing PH with the VR complex. In our benchmarking, the dual implementations outperformed standard ones both in terms of computation time (with respect to both CPU and wall-time seconds) and in terms of the amount of memory used. This significant difference in performance and memory usage was also revealed for the software package Dionysus in [72]. However, note that when computing PH for other types of complexes, the standard algorithm may be better suited than the dual algorithm. (See, e.g., the result of our test for the **vertebra** data set in Additional file 1 of the SI.)

To conclude, in our benchmarking, the fastest software packages were Ripser, Gudhi, and DIPHA. For small complexes, the software packages Perseus and javaPlex are good choices, because they are the easiest ones to use. (They are the only libraries that need only to be downloaded and are then 'plug-and-play,' and they have user-friendly interfaces.) Because the library javaPlex implements the computation of a variety of complexes and algorithms, we feel that it is the best software for an initial foray into PH computation.

We now give guidelines for the computation of PH based on our benchmarking. We list several types of data sets in Table 6 and indicate which software or algorithm that we feel is best-suited to each one. These guidelines are based on the findings of our benchmarking. Note that one can transform networks into distance matrices, and distance matrices can yield points in Euclidean space using a method such as multi-dimensional scaling. Naturally, given a finite set of points in Euclidean space, we can compute their distance matrix. As we discussed in Section 5.1, image data can also be considered as a finite metric space, so the indications for distance matrices and points in Euclidean space also apply to image data.

8 Future directions

We conclude by discussing some future directions for the computation of PH. As we saw in Section 5, much work has been done on step 2 (i.e., going from filtered complexes to barcodes) of the PH pipeline of Figure 6, and there exist implementations of many fast

Table 6 Guidelines for which implementation is best-suited for which data set, based on our benchmarking

Data type	Complex	Suggested software
networks	WRCF	jHoles
image data	cubical	Gudhi or DIPHA (st)
distance matrix	VR	Ripser
distance matrix	W	javaPlex
points in Euclidean space	VR	Gudhi
points in Euclidean space	Č	Dionysus
points in Euclidean space	α (only in dim 2 and 3)	Dionysus ((st) in dim 2, (d) in dim 3) or Gudhi

Recall that we indicate the implementation of the dual algorithm using the abbreviation 'd' following the name of a package, and similarly we indicate the implementation of the standard algorithm by 'st'. Note that for smaller data sets one can also use javaPlex to compute PH with VR complexes from points in Euclidean space, and Perseus to compute PH with cubical complexes for image data, and with VR complexes for distance matrices. The library jHoles can only handle networks with density much less than 1.

algorithms for the reduction of the boundary matrix. Step 1 (i.e., going from data to a filtered complex) of the PH pipeline is an active area of research, but many sparsification techniques (see, e.g., [97, 112]) for complexes have yet to be implemented, and more research needs to be done on steps 1 and 3 (i.e., interpreting barcodes; see, e.g., [130, 136, 143]) of the PH pipeline. In particular, it is important to develop approaches for statistical analysis of persistent homology.

We believe that there needs to be a community-wide effort to build a library that implements the algorithms and data structures for the computation of PH, and that it should be done in a way that new algorithms and methods can be implemented easily in this framework. This would parallel similar community-wide efforts in fields such as computational algebra and computational geometry, and libraries such as Macaulay2 [177], Sage [178], and CGAL [179].

We also believe that there is a need to create guidelines and benchmark data sets for the test of new algorithms and data structures. The methods and collection of data sets that we used in our benchmarking provide an initial step towards establishing such guidelines and a list of test problems.

9 List of abbreviations

1. α: alpha complex
2. d (following the name of a library): implementation of the dual algorithm
3. Č: Čech complex
4. PH: persistent homology
5. SI: Supplementary Information
6. st (following the name of a library): implementation of the standard algorithm
7. TDA: topological data analysis
8. VR: Vietoris–Rips complex
9. W: weak witness complex
10. W_ν: parametrized witness complexes
11. WRCF: weight rank clique filtration

10 Availability of data and materials

The processed version of the data sets used in the benchmarking and the scripts written for the tutorial are available at https://github.com/n-otter/PH-roadmap. The open-source libraries for the computation of PH studied in this paper are available at the references indicated in the associated citations.

Additional material

Additional file 1: Additional computations. (pdf)
Additional file 2: Tutorial for 'A roadmap for the computation of persistent homology'. (pdf)

Acknowledgements
We thank the Rabadan Lab at Columbia University for providing the HIV and H3N2 sequences used in [23] and Giovanni Petri for sharing the data sets used in [89]. We thank Krishna Bhogaonker, Adrian Clough, Patrizio Frosini, Florian Klimm, Yacoub Kureh, Vitaliy Kurlin, Robert MacKay, James Meiss, Dane Taylor, Leo Speidel, Parker Edwards, and Bernadette Stolz for helpful comments on a draft of this paper. We also thank the anonymous referees for their many helpful comments. The first author thanks Ulrich Bauer, Michael Lesnick, Hubert Wagner, and Matthew Wright for helpful discussions, and thanks Florian Klimm, Vidit Nanda, and Bernadette Stolz for precious advice. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility (http://dx.doi.org/10.5281/zenodo.22558) in carrying out some of the computations performed in this work. The first author thanks the support team at the ARC for their assistance. NO and PG are grateful for support from the EPSRC grant EP/G065802/1 (The Digital Economy HORIZON Hub). HAH gratefully acknowledges EPSRC Fellowship EP/K041096/1. NO and UT were supported by The Alan Turing Institute through EPSRC grant EP/N510129/1. NO and HAH were supported by the EPSRC institutional grant D4D01270 BKA1.01.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
NO did the benchmarking and wrote the original version of the manuscript. NO, MAP, UT, PG, and HAH designed the study and revised the manuscript. NO, MAP, and HAH obtained the example data sets to be analyzed.

Author details
[1]Mathematical Institute, University of Oxford, Oxford, OX2 6GG, UK. [2]CABDyN Complexity Centre, University of Oxford, Oxford, OX1 1HP, UK. [3]The Alan Turing Institute, 96 Euston Road, London, NW1 2DB, UK. [4]Department of Mathematics, UCLA, Los Angeles, CA 90095, USA.

Endnotes

[a] Conversely, under favorable conditions (see [78], Corollary 4.33), these algebraic invariants determine the topology of a space up to homotopy — an equivalence relation that is much coarser (and easier to work with) than the more familiar notion of homeomorphy.

[b] Note that this is usually called an 'abstract simplicial complex' in the literature.

[c] A pair $(\{M_i\}_{i\in I}, \{\phi_{i,j} : M_i \to M_j\}_{i\leq j})$, where $(I, \leq)$ is a totally ordered set, such that for each i, we have that M_i is a vector space and the maps $\phi_{i,j}$ are linear maps satisfying the functoriality property (1), is usually called a *persistence module*. With this terminology, the homology of a filtered simplicial complex is an example of persistence module.

[d] Although the collection of intervals is unique, note that one has to choose a vertical order when drawing the intervals in the diagram, and there is therefore an ambiguity in the representation of the intervals as a barcode. However, there is no ambiguity when representing the intervals as points in a persistence diagram (see Figure 5(d)).

[e] A set S of points in $\mathbb{R}^d$ is *in general position* if no $d+2$ points of S lie on a d-dimensional sphere, and for any $d' < d$, no $d'+2$ points of S lie on a d'-dimensional subspace that is isometric to $\mathbb{R}^{d'}$. In particular, a set of points S in $\mathbb{R}^2$ is in general position if no four points lie on a 2-dimensional sphere and no three points lie on a line.

[f] As we mentioned in Section 4, for the reduction of the boundary matrix and thus the computation of PH, it is crucial that one uses simplicial homology with coefficients in a field; see [61] for details.

[g] This map is called 'low' in the literature, because one can think of it as indicating the index of the 'lowest' row — the one that is nearest to the bottom of the page on which one writes the boundary matrix — that contains a 1 in column j.

[h] 'Wall time' is the amount of elapsed time perceived by a human.

[i] *Singular homology* is a method that assigns to every topological space homology groups encoding invariants of the space, in an analogous way as simplicial homology assigns homology groups to simplicial complexes. See [78] for an account of singular homology.

[j] We note that the weight should be the correlation; this issue came to our attention when the paper was in press.

[k] Note that we performed the computations for Gudhi and Ripser at a different point in time, during which the shared-memory system was running the OS Ubuntu 16.04.01.

[l] Based on the results of our tests, we think of small, medium, and large complexes, respectively, as complexes with a size of order of magnitude of up to 10 million simplices, between 10 million and 100 million simplices, and between 100 million and a billion simplices.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 30 January 2017 Accepted: 7 June 2017 Published online: 09 August 2017

References

1. Kaufman L, Rousseeuw PJ (1990) Finding groups in data: an introduction to cluster analysis. Wiley, New York
2. Goldenberg A, Zheng AX, Fienberg SE, Airoldi EM (2010) A survey of statistical network models. Found Trends Mach Learn 2:129-233
3. Gan G, Ma C, Wu J (2007) Data clustering: theory, algorithms, and applications. SIAM, Philadelphia
4. Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1:27-64
5. de Silva V, Ghrist R (2007) Coverage in sensor networks via persistent homology. Algebraic Geom Topol 7:339-358
6. Kovacev-Nikolic V, Bubenik P, Nikolić D, Heo G (2014) Using persistent homology and dynamical distances to analyze protein binding. arXiv:1412.1394
7. Gameiro M, Hiraoka Y, Izumi S, Kramár M, Mischaikow K, Nanda V (2015) A topological measurement of protein compressibility. Jpn J Ind Appl Math 32:1-17
8. Xia K, Wei G-W (2014) Persistent homology analysis of protein structure, flexibility, and folding. Int J Numer Methods Biomed Eng 30:814-844
9. Xia K, Li Z, Mu L (2016) Multiscale persistent functions for biomolecular structure characterization. arXiv:1612.08311
10. Emmett K, Schweinhart B, Rabadán R (2016) Multiscale topology of chromatin folding. In: Proceedings of the 9th EAI international conference on bio-inspired information and communications technologies (formerly BIONETICS), BICT'15. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, pp 177-180
11. Rizvi A, Camara P, Kandror E, Roberts T, Schieren I, Maniatis T, Rabadan R (2017) Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. Nat Biotechnol 35:551-560. doi:10.1038/nbt.3854
12. Xia K, Feng X, Tong Y, Wei GW (2015) Persistent homology for the quantitative prediction of fullerene stability. J Comput Chem 36:408-422
13. Bhattacharya S, Ghrist R, Kumar V (2015) Persistent homology for path planning in uncertain environments. IEEE Trans Robot 31:578-590
14. Pokorny FT, Hawasly M, Ramamoorthy S (2016) Topological trajectory classification with filtrations of simplicial complexes and persistent homology. Int J Robot Res 35:204-223
15. Vasudevan R, Ames A, Bajcsy R (2013) Persistent homology for automatic determination of human-data based cost of bipedal walking. Nonlinear Anal Hybrid Syst 7:101-115
16. Chung MK, Bubenik P, Kim PT (2009) Persistence diagrams of cortical surface data. In: Prince JL, Pham DL, Myers KJ (eds) Information processing in medical imaging. Lecture notes in computer science, vol 5636. Springer, Berlin, pp 386-397
17. Guillemard M, Boche H, Kutyniok G, Philipp F (2013) Signal analysis with frame theory and persistent homology. In: 10th international conference on sampling theory and applications, pp 309-312
18. Perea JA, Deckard A, Haase SB, Harer J (2015) Sw1pers: sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data. BMC Bioinform 16:Article ID 257
19. Nicolau M, Levine AJ, Carlsson G (2011) Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. Proc Natl Acad Sci USA 108:7265-7270
20. DeWoskin D, Climent J, Cruz-White I, Vazquez M, Park C, Arsuaga J (2010) Applications of computational homology to the analysis of treatment response in breast cancer patients. Topol Appl 157:157-164
21. Crawford L, Monod A, Chen AX, Mukherjee S, Rabadán R (2016) Topological summaries of tumor images improve prediction of disease free survival in glioblastoma multiforme. arXiv:1611.06818
22. Singh N, Couture HD, Marron JS, Perou C, Niethammer M (2014) Topological descriptors of histology images. In: Wu G, Zhang D, Zhou L (eds) Machine learning in medical imaging. Lecture notes in computer science, vol 8679. Springer, Cham, pp 231-239
23. Chan JM, Carlsson G, Rabadan R (2013) Topology of viral evolution. Proc Natl Acad Sci USA 110:18566-18571
24. Cámara PG, Levine AJ, Rabadán R (2016) Inference of ancestral recombination graphs through topological data analysis. PLoS Comput Biol 12:Article ID e1005071
25. Emmett K, Rosenbloom D, Camara P, Rabadan R (2014) Parametric inference using persistence diagrams: a case study in population genetics. arXiv:1406.4582
26. Carlsson G, Ishkhanov T, de Silva V, Zomorodian A (2008) On the local behavior of spaces of natural images. Int J Comput Vis 76:1-12
27. Taylor D, Klimm F, Harrington HA, Kramár M, Mischaikow K, Porter MA, Mucha PJ (2015) Topological data analysis of contagion maps for examining spreading processes on networks. Nat Commun 6:Article ID 7723
28. Lo D, Park B (2016) Modeling the spread of the Zika virus using topological data analysis. arXiv:1612.03554
29. MacPherson R, Schweinhart B (2012) Measuring shape with topology. J Math Phys 53:Article ID 073516
30. Kramár M, Goullet A, Kondic L, Mischaikow K (2013) Persistence of force networks in compressed granular media. Phys Rev E 87:Article ID 042207
31. Kramár M, Goullet A, Kondic L, Mischaikow K (2014) Quantifying force networks in particulate systems. Physica D 283:37-55
32. Hiraoka Y, Nakamura T, Hirata A, Escolar E, Matsue K, Nishiura Y (2016) Hierarchical structures of amorphous solids characterized by persistent homology. Proc Natl Acad Sci USA 113:7035-7040
33. Lee Y, Barthel SD, Dłotko P, Mohamad Moosavi S, Hess K, Smit B (2017) Pore-geometry recognition: on the importance of quantifying similarity in nanoporous materials. arXiv:1701.06953
34. Leibon G, Pauls S, Rockmore D, Savell R (2008) Topological structures in the equities market network. Proc Natl Acad Sci USA 105:20589-20594
35. Gidea M (2017) Topology data analysis of critical transitions in financial networks. arXiv:1701.06081

36. Giusti C, Ghrist R, Bassett D (2016) Two's company and three (or more) is a simplex. J Comput Neurosci 41:1-14
37. Curto C (2017) What can topology tell us about the neural code? Bull, New Ser, Am Math Soc 54:63-78
38. Dłotko P, Hess K, Levi R, Nolte M, Reimann M, Scolamiero M, Turner K, Muller E, Markram H (2016) Topological analysis of the connectome of digital reconstructions of neural microcircuits. arXiv:1601.01580
39. Kanari L, Dłotko P, Scolamiero M, Levi R, Shillcock J, Hess K, Markram H (2016) Quantifying topological invariants of neuronal morphologies. arXiv:1603.08432
40. Lord L-D, Expert P, Fernandes HM, Petri G, Van Hartevelt TJ, Vaccarino F, Deco G, Turkheimer F, Kringelbach M (2016) Insights into brain architectures from the homological scaffolds of functional connectivity networks. Front Syst Neurosci 10:Article ID 85
41. Bendich P, Marron JS, Miller E, Pieloch A, Skwerer S (2016) Persistent homology analysis of brain artery trees. Ann Appl Stat 10:198-218
42. Yoo J, Kim EY, Ahn YM, Ye JC (2016) Topological persistence vineyard for dynamic functional brain connectivity during resting and gaming stages. J Neurosci Methods 267:1-13
43. Dabaghian Y, Brandt VL, Frank LM (2014) Reconceiving the hippocampal map as a topological template. eLife 3:Article ID e03476
44. Sizemore A, Giusti C, Bassett D (2017) Classification of weighted networks through mesoscale homological features. J Complex Netw 5:245-273
45. Pal S, Moore TJ, Ramanathan R, Swami A (2017) Comparative topological signatures of growing collaboration networks. In: Complex networks VIII. Springer, Cham, pp 201-209
46. Carstens CJ, Horadam KJ (2013) Persistent homology of collaboration networks. Math Probl Eng 2013:Article ID 815035
47. Bajardi P, Delfino M, Panisson A, Petri G, Tizzoni M (2015) Unveiling patterns of international communities in a global city using mobile phone data. EPJ Data Sci 4:Article ID 3
48. Topaz CM, Ziegelmeier L, Halverson T (2015) Topological data analysis of biological aggregation models. PLoS ONE 10:Article ID e0126383
49. Maletic S, Zhao Y, Rajkovic M (2015) Persistent topological features of dynamical systems. arXiv:1510.06933
50. Zhu X (2013) Persistent homology: an introduction and a new text representation for natural language processing. In: Proceedings of the twenty-third international joint conference on artificial intelligence, IJCAI '13, Beijing, China AAAI Press, Menlo Park, pp 1953-1959
51. Wang B, Wei G-W (2016) Object-oriented persistent homology. J Comput Phys 305:276-299
52. Stolz BJ, Harrington HA, Porter MA (2017) Persistent homology of time-dependent functional networks constructed from coupled time series. Chaos 27:Article ID 047410
53. Bendich P, Marron JS, Miller E, Pieloch A, Skwerer S (2016) Persistent homology analysis of brain artery trees. Ann Appl Stat 10:198-218
54. Adler R (2014) TOPOS, and why you should care about it. IMS Bull 43:4-5
55. Wagner H, Chen C, Vuçini E (2012) Efficient computation of persistent homology for cubical data. In: Peikert R, Hauser H, Carr H, Fuchs R (eds) Topological methods in data analysis and visualization II. Mathematics and visualization. Springer, Berlin, pp 91-106
56. Singh G, Mémoli F, Carlsson G (2007) Topological methods for the analysis of high dimensional data sets and 3D object recognition. In: Eurographics symposium on point-based graphics, pp 91-100
57. Ghrist R (2014) Elementary applied topology, 1.0 edn
58. Curry J (2013) Sheaves, cosheaves and applications. arXiv:1303.3255
59. Carlsson G (2009) Topology and data. Bull Am Math Soc 46:255-308
60. Edelsbrunner H, Letscher D, Zomorodian A (2002) Topological persistence and simplification. Discrete Comput Geom 28:511-533
61. Zomorodian A, Carlsson G (2005) Computing persistent homology. Discrete Comput Geom 33:249-274
62. Bauer U, Kerber M, Reininghaus J, Wagner H (2014) PHAT: persistent homology algorithms toolbox. In: Hong H, Yap C (eds) Mathematical software - ICMS 2014. Lecture notes in computer science, vol 8592. Springer, Berlin, pp 137-143. Software available at https://code.google.com/p/phat/
63. Bauer U, Kerber M, Reininghaus J (2014) DIPHA (a distributed persistent homology algorithm). https://code.google.com/p/dipha/
64. Morozov D Dionysus. http://www.mrzv.org/software/dionysus/
65. Nanda V Perseus, the persistent homology software. http://www.sas.upenn.edu/~vnanda/perseus
66. Tausz A, Vejdemo-Johansson M, Adams H (2014) JavaPlex: a research software package for persistent (co)homology. In: Hong H, Yap C (eds) Mathematical software - ICMS 2014. Lecture notes in computer science, vol 8592, pp 129-136. Software available at http://appliedtopology.github.io/javaplex/
67. Maria C, Boissonnat J-D, Glisse M, Yvinec M (2014) The Gudhi library: simplicial complexes and persistent homology. In: Hong H, Yap C (eds) Mathematical software - ICMS 2014. Lecture notes in computer science, vol 8592. Springer, Berlin, pp 167-174. Software available at https://project.inria.fr/gudhi/software/
68. Bauer U (2016) Ripser. https://github.com/Ripser/ripser
69. Fasy BT, Kim J, Lecci F, Maria C (2014) Introduction to the R package TDA. arXiv:1411.1830
70. Bubenik P, Dłotko P (2017) A persistence landscapes toolbox for topological statistics. J Symb Comput 78:91-114
71. Adams H, Tausz A JavaPlex tutorial. https://github.com/appliedtopology/javaplex
72. de Silva V, Morozov D, Vejdemo-Johansson M (2011) Dualities in persistent (co)homology. Inverse Probl 27:Article ID 124003
73. Nanda V (2012) Discrete Morse theory for filtrations. PhD thesis, Rutgers, The State University of New Jersey
74. Bauer U, Kerber M, Reininghaus J (2014) Distributed computation of persistent homology. In: 2014 proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX). SIAM, Philadelphia, pp 31-38
75. Maria C (2014) Algorithms and data structures in computational topology. PhD thesis, Université de Nice-Sophia Antipolis. http://www-sop.inria.fr/members/Clement.Maria/docs/ClementMaria_PhDdissertation.pdf
76. Kaczynski T, Mischaikow K, Mrozek M (2004) Computational homology. Applied mathematical sciences, vol 157. Springer, New York

77. Cohen MM (1970) A course in simple homotopy theory. Graduate texts in mathematics. Springer, New York
78. Hatcher A (2002) Algebraic topology. Cambridge University Press, Cambridge
79. Björner A (1995) Topological methods. In: Graham R, Grötschel M, Lovász L (eds) Handbook of combinatorics. Elsevier, Amsterdam, pp 1819-1872
80. Edelsbrunner H, Harer J (2010) Computational topology: an introduction. Applied mathematics. Am. Math. Soc., Providence
81. Eilenberg S, Steenrod NE (1952) Foundations of algebraic topology. Princeton mathematical series. Princeton University Press, Princeton
82. Oudot SY (2015) Persistence theory: from quiver representations to data analysis. AMS mathematical surveys and monographs, vol 209. Am. Math. Soc., Providence
83. Zomorodian A (2009) Topology for computing. Cambridge monographs on applied and computational mathematics. Cambridge University Press, Cambridge
84. Weinberger S (2011) What is…persistent homology? Not Am Math Soc 58:36-39
85. Ghrist R (2008) Barcodes: the persistent topology of data. Bull Am Math Soc 45:61-75
86. Edelsbrunner H, Harer J (2008) Persistent homology — a survey. In: Goodman JE, Pach J, Pollack R (eds) Surveys on discrete and computational geometry: twenty years later. Contemporary mathematics, vol 453. Am. Math. Soc., Providence, pp 257-282
87. Edelsbrunner H, Morozov D (2012) Persistent homology: theory and practice. In: Proceedings of the European congress of mathematics, pp 31-50
88. Patania A, Vaccarino F, Petri G (2017) Topological analysis of data. EPJ Data Sci 6(1):7
89. Petri G, Scolamiero M, Donato I, Vaccarino F (2013) Topological strata of weighted complex networks. PLoS ONE 8:Article ID e66506
90. Jonsson J (2007) Simplicial complexes of graphs. Lecture notes in mathematics. Springer, Berlin
91. Horak D, Maletić S, Rajković M (2009) Persistent homology of complex networks. J Stat Mech Theory Exp 2009:Article ID P03034
92. Bendich P, Edelsbrunner H, Kerber M (2010) Computing robustness and persistence for images. IEEE Trans Vis Comput Graph 16:1251-1260
93. Zhou W, Yan H (2014) Alpha shape and Delaunay triangulation in studies of protein-related interactions. Brief Bioinform 15:54-64
94. Xia K, Wei G-W (2016) A review of geometric, topological and graph theory apparatuses for the modeling and analysis of biomolecular data. arXiv:1612.01735
95. Zomorodian A (2010) Technical section: fast construction of the Vietoris–Rips complex. Comput Graph 34:263-271
96. Vietoris L (1927) Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen. Math Ann 97:454-472
97. Kerber M, Sharathkumar R (2013) Approximate Čech complex in low and high dimensions. In: Cai L, Cheng S-W, Lam T-W (eds) 24th international symposium on algorithms and computation (ISAAC 2013). Lecture notes in computer science, vol 8283, pp 666-676
98. Boissonnat J-D, Devillers O, Hornus S (2009) Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In: Proceedings of the twenty-fifth annual symposium on computational geometry, SoCG '09. ACM, New York, pp 208-216
99. Goodman JE, O'Rourke J (eds) (2004) Handbook of discrete and computational geometry, 2nd edn. CRC Press, Boca Raton
100. Edelsbrunner H, Kirkpatrick D, Seidel R (1983) On the shape of a set of points in the plane. IEEE Trans Inf Theory 29:551-559
101. Edelsbrunner H, Mücke EP (1994) Three-dimensional alpha shapes. ACM Trans Graph 13:43-72
102. Edelsbrunner H (1995) The union of balls and its dual shape. Discrete Comput Geom 13:415-440
103. Kurlin V (2015) A one-dimensional homologically persistent skeleton of an unstructured point cloud in any metric space. Comput Graph Forum 34:253-262
104. Kurlin V (2015) http://kurlin.org/projects/persistent-skeletons.cpp
105. de Silva V (2008) A weak characterisation of the Delaunay triangulation. Geom Dedic 135:39-64
106. de Silva V, Carlsson G (2004) Topological estimation using witness complexes. In: Proceedings of the first Eurographics conference on point-based graphics, pp 157-166
107. Guibas LJ, Oudot SY (2008) Reconstruction using witness complexes. Discrete Comput Geom 40:325-356
108. Attali D, Edelsbrunner H, Mileyko Y (2007) Weak witnesses for Delaunay triangulations of submanifolds. In: Proceedings of the 2007 ACM symposium on solid and physical modeling, SPM '07. ACM, New York, pp 143-150
109. Boissonnat J-D, Guibas LJ, Oudot SY (2009) Manifold reconstruction in arbitrary dimensions using witness complexes. Discrete Comput Geom 42:37-70
110. Dey TK, Fan F, Wang Y (2013) Graph induced complex on point data. In: Proceedings of the twenty-ninth annual symposium on computational geometry, SoCG '13. ACM, New York, pp 107-116
111. Jyamiti research group (2013) GIComplex. http://web.cse.ohio-state.edu/~tamaldey/GIC/GICsoftware/
112. Sheehy DR (2013) Linear-size approximations to the Vietoris–Rips filtration. Discrete Comput Geom 49:778-796
113. Dey TK, Shi D, Wang Y (2016) SimBa: an efficient tool for approximating Rips-filtration persistence via simplicial batch-collapse. In: 24th annual European symposium on algorithms (ESA 2016). LIPIcs - Leibniz international proceedings in informatics, vol 57. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Saarbrücken, pp 35:1-35:16
114. Robin F (1998) Morse theory for cell complexes. Adv Math 134:90-145
115. Mischaikow K, Nanda V (2013) Morse theory for filtrations and efficient computation of persistent homology. Discrete Comput Geom 50:330-353
116. Joswig M, Pfetsch ME (2006) Computing optimal Morse matchings. SIAM J Discrete Math 20:11-25
117. Barmak JA, Minian EG (2012) Strong homotopy types, nerves and collapses. Discrete Comput Geom 47:301-328
118. Wilkerson AC, Moore TJ, Swami A, Krim H (2013) Simplifying the homology of networks via strong collapses. In: 2013 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 5258-5262
119. Wilkerson AC, Chintakunta H, Krim H, Moore TJ, Swami A (2013) A distributed collapse of a network's dimensionality. In: 2013 IEEE global conference on signal and information processing, pp 595-598

120. Wilkerson AC, Chintakunta H, Krim H (2014) Computing persistent features in big data: a distributed dimension reduction approach. In: 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 11-15
121. Zomorodian A (2010) The tidy set: a minimal simplicial set for computing homology of clique complexes. In: Proceedings of the twenty-sixth annual symposium on computational geometry, SoCG '10. ACM, New York, pp 257-266
122. Zomorodian A (2012) Topological data analysis. In: Zomorodian A (ed) Advances in applied and computational topology. Proceedings of symposia in applied mathematics, vol 70. Am. Math. Soc., Providence, pp 1-39
123. Morozov D (2005) Persistence algorithm takes cubic time in worst case. BioGeometry News (Feb 2005), Department of Computer Science, Duke University
124. Milosavljević N, Morozov D, Skraba P (2011) Zigzag persistent homology in matrix multiplication time. In: Proceedings of the twenty-seventh annual symposium on computational geometry, SoCG '11. ACM, New York, pp 216-225
125. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9:251-280
126. Chen C, Kerber M (2011) Persistent homology computation with a twist. In: Proceedings of the 27th European workshop on computational geometry, pp 197-200
127. de Silva V, Morozov D, Vejdemo-Johansson M (2011) Persistent cohomology and circular coordinates. Discrete Comput Geom 45:737-759
128. Bauer U, Kerber M, Reininghaus J (2014) Clear and compress: computing persistent homology in chunks. In: Bremer P-T, Hotz I, Pascucci V, Peikert R (eds) Topological methods in data analysis and visualization III. Mathematics and visualization. Springer, Cham, pp 103-117
129. Boissonnat J-D, Maria C (2014) Computing persistent homology with various coefficient fields in a single pass. In: Schulz AS, Wagner D (eds) Algorithms - ESA 2014. Lecture notes in computer science, vol 8737. Springer, Berlin, pp 185-196
130. Bubenik P, Kim PT (2007) A statistical approach to persistent homology. Homol Homotopy Appl 9:337-362
131. Adler R, Bobrowski O, Weinberger S (2014) Crackle: the homology of noise. Discrete Comput Geom 52:680-704
132. Young J-G, Petri G, Vaccarino F, Patania A (2017) Construction of and efficient sampling from the simplicial configuration model. arXiv:1705.10298
133. Adler RJ, Bobrowski O, Borman MS, Subag E, Weinberger S (2010) Persistent homology for random fields and complexes. In: Borrowing strength: theory powering applications - a festschrift for Lawrence D. Brown. IMS collections, vol 6. Institute of Mathematical Statistics, Beachwood, pp 124-143
134. Kahle M (2014) Topology of random simplicial complexes: a survey. In: Applied algebraic topology: new directions and applications. Contemporary mathematics, vol 620. Am. Math. Soc., Providence, pp 221-241
135. Mileyko Y, Mukherjee S, Harer J (2011) Probability measures on the space of persistence diagrams. Inverse Probl 27:Article ID 124007
136. Turner K, Mileyko Y, Mukherjee S, Harer J (2014) Fréchet means for distributions of persistence diagrams. Discrete Comput Geom 52:44-70
137. Munch E, Turner K, Bendich P, Mukherjee S, Mattingly J, Harer J (2015) Probabilistic Fréchet means for time varying persistence diagrams. Electron J Stat 9:1173-1204
138. Kerber M, Morozov D, Nigmetov A (2016). https://bitbucket.org/grey_narn/hera
139. Kerber M, Morozov D, Nigmetov A (2016) Geometry helps to compare persistence diagrams. arXiv:1606.03357
140. Fasy BT, Kim J, Lecci F, Maria C, Rouvreau V TDA: statistical tools for topological data analysis. https://cran.r-project.org/web/packages/TDA/index.html
141. Fasy B, Lecci F, Rinaldo A, Wasserman L, Balakrishnan S, Singh A (2014) Confidence sets for persistence diagrams. Ann Stat 42:2301-2339
142. Chazal F, Fasy BT, Lecci F, Michel B, Rinaldo A, Wasserman L (2014) Robust topological inference: distance to a measure and kernel distance. arXiv:1412.7197
143. Bubenik P (2015) Statistical topological data analysis using persistence landscapes. J Mach Learn Res 16:77-102
144. Adcock A, Carlsson E, Carlsson G (2013) The ring of algebraic functions on persistence bar codes. arXiv:1304.0530
145. Chepushtanova S, Emerson T, Hanson E, Kirby M, Motta F, Neville R, Peterson C, Shipman P, Ziegelmeier L (2015) Persistence images: an alternative persistent homology representation. arXiv:1507.06217
146. Kwitt R, Huber S, Niethammer M, Lin W, Bauer U (2015) Statistical topological data analysis - a kernel perspective. In: Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R (eds) Advances in neural information processing systems, vol 28. Curran Associates, Red Hook, pp 3052-3060
147. Reininghaus J, Huber S, Bauer U, Kwitt R (2015) A stable multi-scale kernel for topological machine learning. In: 2015 IEEE conference on computer vision and pattern recognition (CVPR), pp 4741-4748
148. Bobrowski O, Mukherjee S, Taylor J (2017) Topological consistency via kernel estimation. Bernoulli 23:288-328
149. Zhu X, Vartanian A, Bansal M, Nguyen D, Brandl L (2016) Stochastic multiresolution persistent homology kernel. In: Proceedings of the twenty-fifth international joint conference on artificial intelligence, IJCAI'16. AAAI Press, Palo Alto, pp 2449-2455
150. Dłotko P Persistence landscape toolbox. https://www.math.upenn.edu/~dlotko/persistenceLandscape.html
151. Cohen-Steiner D, Edelsbrunner H, Harer J (2007) Stability of persistence diagrams. Discrete Comput Geom 37:103-120
152. Chazal F, Cohen-Steiner D, Glisse M, Guibas LJ, Oudot SY (2009) Proximity of persistence modules and their diagrams. In: Proceedings of the twenty-fifth annual symposium on computational geometry, SoCG '09. ACM, New York, pp 237-246
153. Bubenik P, Scott JA (2014) Categorification of persistent homology. Discrete Comput Geom 51:600-627
154. Bubenik P, de Silva V, Scott J (2014) Metrics for generalized persistence modules. Found Comput Math 15:1501-1531
155. Carlsson G, de Silva V, Morozov D (2009) Zigzag persistent homology and real-valued functions. In: Proceedings of the twenty-fifth annual symposium on computational geometry, SoCG '09. ACM, New York, pp 247-256
156. Dey TK, Fan F, Wang Y (2014) Computing topological persistence for simplicial maps. In: Proceedings of the thirtieth annual symposium on computational geometry, SoCG '14. ACM, New York, pp 345-354

157. Carlsson G, Zomorodian A (2009) The theory of multidimensional persistence. Discrete Comput Geom 42:71-93
158. Lesnick M, Wright M (2016) RIVET: the rank invariant visualization and exploration tool. http://rivet.online/
159. Lesnick M, Wright M (2015) Interactive visualization of 2-D persistence modules. arXiv:1512.00180
160. Edelsbrunner H, Morozov D, Pascucci V (2006) Persistence-sensitive simplification functions on 2-manifolds. In: Proceedings of the twenty-second annual symposium on computational geometry, SoCG '06. ACM, New York, pp 127-134
161. Perry P, de Silva V (2000– 2006) Plex. http://mii.stanford.edu/research/comptop/programs/
162. Binchi J, Merelli E, Rucco M, Petri G, Vaccarino F (2014) jHoles: a tool for understanding biological complex networks via clique weight rank persistent homology. Electron Notes Theor Comput Sci 306:5-18
163. Jyamiti research group (2014) SimpPers. http://web.cse.ohio-state.edu/~tamaldey/SimpPers/SimpPers-software/
164. Stanford University Computer Graphics Laboratory, The Stanford 3D scanning repository. https://graphics.stanford.edu/data/3Dscanrep
165. Kahle M (2011) Random geometric complexes. Discrete Comput Geom 45:553-573
166. Penrose M (2003) Random geometric graphs. Oxford University Press, Oxford
167. Vicsek T, Czirók A, Ben-Jacob E, Cohen I, Shochet O (1995) Novel type of phase transition in a system of self-driven particles. Phys Rev Lett 75:1226-1229
168. Sporns O (2006) Small-world connectivity, motif composition, and complexity of fractal neuronal connections. Biosystems 85:55-64
169. Los Alamos National Laboratory, HIV database. http://www.hiv.lanl.gov/content/index
170. Watts DJ, Strogatz SH (1998) Collective dynamics of 'small world' networks. Nature 393(6684):440-442
171. White JG, Southgate E, Thomson JN, Brenner S (1986) The structure of the nervous system of the nematode Caenorhabditis elegans. Philos Trans R Soc Lond B, Biol Sci 314(1165):1-340
172. Davis TA, Hu Y (2011) The University of Florida sparse matrix collection. ACM Trans Math Softw 38:1-25. http://www.cise.ufl.edu/research/sparse/matrices
173. Volvis repository. http://volvis.org
174. Waugh AS, Pei L, Fowler JH, Mucha PJ, Porter MA (2012) Party polarization in congress: a network science approach. arXiv:0907.3509. Data available at http://figshare.com/articles/Roll_Call_Votes_United_States_House_and_Senate/1590036
175. Poole KT (2016) Voteview. http://voteview.com
176. Newman MEJ (2006) Finding community structure in networks using the eigenvectors of matrices. Phys Rev E 74:Article ID 036104
177. Grayson DR, Stillman ME Macaulay2, a software system for research in algebraic geometry. http://www.math.uiuc.edu/Macaulay2/
178. T. S. Developers, Sage mathematics software. http://www.sagemath.org
179. The CGAL Project (2015) CGAL user and reference manual, 4.7 edn. CGAL Editorial Board

ADDITIONAL COMPUTATIONS

1. Introduction. We report additional computations that we perform with the libraries that we study. See the main paper for a description of the data sets and details on how we perform the computations.

2. Vietoris–Rips complex. In this section, we report additional results for the computation of PH with the VR complex, as explained in Section 7 of the main manuscript. In Tables 1–4, we give results for the computations on a cluster; in Table 5, we give results for the computations on a shared-memory system. For the data set **Vicsek**, we computed PH for six different point clouds, as we also explain in the main paper. We report results only for one of these points clouds, because the results of the computations — in terms of wall-time seconds and memory used — were all similar. The point cloud for which we report the results has the following parameters (see the main manuscript for a discussion of these parameters): $N = 300$, $l = 5$, $\theta_0 = 1$, $v_0 = 0.03$, $\eta = 0.1$, and $T = 600$. Of the 600 different points clouds (one for each time step t), we choose the one that corresponds to time step $t = 300$.

TABLE 1
*Computations of PH with VR complex for the data set **drag 1**. The size of the complex is* 1.7×10^8*, and the dimension is* 2*. We run* DIPHA *on one node and* 16 *cores.*

	Wall-time seconds	CPU seconds	Memory in GB
javaPlex (st)	-	-	> 64
Dionysus (st)	-	-	-
DIPHA (st)	7356	117417	2.4
Perseus	-	-	-
Dionysus (d)	4360	4362	16.8
DIPHA (d)	75	1176	1.8
GUDHI	1524	1517	7.9
Ripser	2	1	0.04

TABLE 2
*Computations of PH with VR complex for the data set **house**. The size of the complex is* 1.6×10^9*, and the dimension is* 3*. We run* DIPHA *(d) on three nodes of* 16 *cores, and* DIPHA *(st) on four nodes of* 16 *cores.*

	Wall-time seconds	CPU seconds	Memory in GB
javaPlex (st)	-	-	> 64
DIPHA (st)	53686	243587	5.3
Perseus	-	-	-
Dionysus (d)	-	-	-
DIPHA (d)	1450	22981	7.1
GUDHI	46377	25925	64.6
Ripser	11	11	0.8

3. Image data set. In this section, we report timings and memory usage for the computation of PH with cubical complexes with the libraries DIPHA, Perseus, and GUDHI for the data set **vertebra**. For this data set, the standard implementation in DIPHA performs better in terms of wall time, CPU seconds, and memory usage than the dual implementation. This does not necessarily contradict the fact that the dual implementation gives a heuristic speed-up with respect to the standard imple-

TABLE 3

*Computations of PH with VR complex for the data set **senate**. The size of the complex is* 4.6×10^6*, and the dimension is* 3*. We run* DIPHA *on two nodes and* 16 *cores.*

	Wall-time seconds	CPU seconds	Memory in GB
JAVAPLEX (st)	222	571	< 5
DIPHA (st)	5	58	0.1
PERSEUS	457	457	4.9
DIONYSUS (d)	183	183	0.5
DIPHA (d)	5	34	0.1
GUDHI	51	51	0.2
RIPSER	1	1	0.01

TABLE 4

*Computations of PH with VR complex for the data set **netw-sc**. The size of the complex is* 8.5×10^8*, and the dimension is* 3*. We run* DIPHA *on two nodes of* 16 *cores.*

	Wall-time seconds	CPU seconds	Memory in GB
JAVAPLEX (st)	-	-	> 64
DIPHA (st)	16004	202755	5.6
PERSEUS	-	-	-
DIONYSUS (d)	-	-	-
DIPHA (d)	618	9842	5.8
GUDHI	13483	13465	40.6
RIPSER	9	9	0.5

TABLE 5

Computations on the shared-memory system.

Data set	**H3N2**	**Vicsek**	**drag 1**	**fract l**
Size of complex	3.4×10^9	3.3×10^8	1.7×10^8	2.8×10^9
Max. dim.	2	3	2	3
JAVAPLEX (st)	-	26257	19054	-
PERSEUS	-	-	-	-
DIONYSUS (d)	-	4637	2889	593528
DIPHA (d)	14362	369	1775	3920
GUDHI	3377	433	144	4154
RIPSER	24	3	2	14

(a) Computations on the shared-memory system: wall-time seconds

Data set	**H3N2**	**Vicsek**	**drag 1**	**fract l**
Size of complex	3.4×10^9	3.3×10^8	1.7×10^8	2.8×10^9
Max. dim.	2	3	2	3
JAVAPLEX (st)	> 700	< 600	< 600	> 700
PERSEUS	-	-	-	-
DIONYSUS (d)	-	33.5	16.8	269
DIPHA (d)	276.1	32.5	13.8	276.1
GUDHI	158.2	15.9	7.9	134.5
RIPSER	0.2	0.3	0.04	1.2

(b) Computations on the shared-memory system: memory in GB

mentation, because it is well-known that the dual implementation gives a speed-up when one computes PH with the VR complex but not in general.

TABLE 6

Computations of PH with cubical complexes for the data set ***vertebra*** *on the shared-memory system. The complex, as constructed by* DIPHA *and* GUDHI*, has* 1.1×10^9 *cells, and its dimension is* 3*. Note that* PERSEUS *implements a less efficient way for building filtered cubical complexes from grey-scale images, and the resulting complex has more cells than the one constructed by* DIPHA *and* GUDHI.

	DIPHA (d)	DIPHA (st)	PERSEUS	GUDHI (d)
Wall-time seconds	2115	1793	19604	3668
Memory in GB	82.1	80.9	628.8	59.8

4. Additional functionalities. In this section, we report timings and memory usage for the computation of PH with the alpha (α), Čech (Č), weak witness complex (W), and parametrized witness complex for $\nu = 2$ (W_2) (see the main text for a description of these complexes). For the witness complexes we choose the landmark points uniformly at random. We use the two data sets consisting of point clouds in $\mathbb{R}^3$ (i.e., the data sets **Klein** and **Vicsek**). Additionally, note that the newest version of GUDHI (Version 1.3.1) implements the computation of witness complexes, but it does not implement the computation of filtered witness complexes.

TABLE 7

Computations of alpha and Čech complexes on the shared-memory system.

	DIONYSUS α	DIONYSUS Č	GUDHI α
Size of complex	9.2×10^3	-	9.2×10^3
Wall-time seconds	1	-	2
Memory in GB	0.008	-	0.006

(a) Time and memory usage for the data set **Klein**.

	DIONYSUS α	DIONYSUS Č	GUDHI α
Size of complex	7.7×10^3	-	7.7×10^3
Wall-time seconds	1	-	2
Memory in GB	0.007	-	0.006

(b) Time and memory usage for the data set **Vicsek**. For the witness complexes we choose 100 landmark points uniformly at random.

Table 8

Computations of PH with the witness complex on the shared-memory system.

	javaPlex W	javaPlex W_2
Size of complex	6.8×10^5	2.8×10^6
Wall-time seconds	32	328
Memory in GB	< 2	< 5

(a) Time and memory usage for the data set **Klein**. We choose 100 landmark points uniformly at random.

	javaPlex W	javaPlex W_2
Size of complex	1×10^6	4.1×10^6
Wall-time seconds	50	84
Memory in GB	< 3	< 5

(b) Time and memory usage for the data set **Vicsek**. For the witness complexes, we choose 100 landmark points uniformly at random.

Tutorial for “A roadmap for the computation of persistent homology”

1 Introduction

In this tutorial, we give detailed guidelines for the computation of persistent homology and for several of the functionalities that are implemented by the libraries in Table 2 in the main manuscript. We first give some advice on how to install the various libraries, and we then give guidelines for how to compute PH for every step of the pipeline in Fig. 3 of the main paper. We explain how to compute PH for networks with the weight rank clique filtration (WRCF); for point clouds with the VR, alpha, Čech, and witness complexes; and for image data sets with cubical complexes. We then give guidelines for visualizing the outputs of the computations and for computing the bottleneck and Wasserstein distances with Dionysus and Hera. In addition to the bottleneck and Wasserstein distances, there are also other tools (such as persistence landscapes and confidence sets) that are useful for statistical assessment of barcodes, but we do not discuss them here, as there are already comprehensive tutorials [4,5] for the packages that implement these methods. All MATLAB scripts written for this tutorial are available at `https://github.com/n-otter/PH-roadmap/tree/master/matlab`. In Fig. 1, we give instructions for how to navigate this tutorial.

Many tears and much sweat and blood were spent learning about the different libraries, writing this tutorial, and the scripts. If you find this tutorial helpful, please acknowledge it.

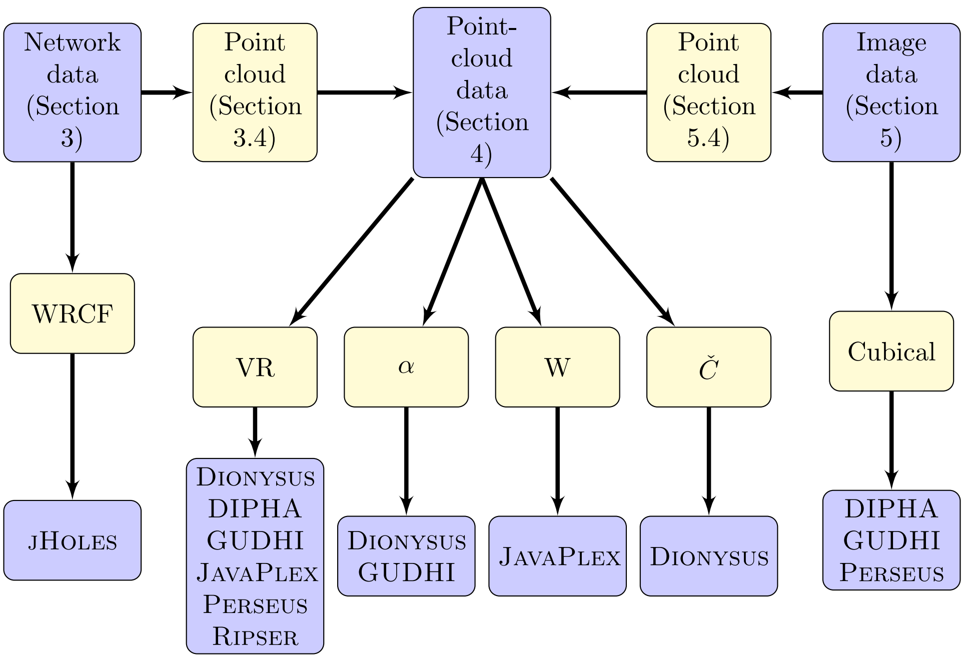


Figure 1: Schematic for how to navigate the tutorial.

Contents

2 Installation

In this section, we give guidelines on how to get and/or install the software packages.

2.1 Dionysus

The code for DIONYSUS is available at `http://www.mrzv.org/software/dionysus/get-build-install.html`, where one can also find information on dependencies and how to build the library.

The library is written in C++, but it also supports python bindings (i.e., there is a python interface to some of the functionalities that are implemented in the library). Depending on the machine on which one is building DIONYSUS, the python bindings can create some issues. Additionally, from the perspective of performance, it is better to directly use the C++ code. If one wishes to build the library without the python bindings, one needs to delete the `bindings` directory and also to delete the (last) line `add_subdirectory (bindings)` in the file `CMakeLists.txt`.

If one seeks to compute the bottleneck or Wasserstein distances with the library (see Section 7), then before building the library, one needs to amend a mistake in the bottleneck-distance.cpp script as follows: in the subdirectory `examples`, one finds the file `bottleneck-distance.cpp`. One needs to uncomment the following line (which occurs towards the end of the file):

```
std::cout << "Distance: " << bottleneck_distance(dgm1, dgm2) << std::endl;
```

Now one can build the library as follows (from the directory in which the `CMakeLists.txt` file is):

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

2.2 DIPHA

The DIPHA library is available at `https://github.com/DIPHA/dipha`. One can build the library as follows (from the directory in which the `CMakeLists.txt` file is):

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

2.3 GUDHI

The GUDHI library is available at `https://gforge.inria.fr/frs/?group_id=3865`. Information about dependencies and how to build the library is available at `http://gudhi.gforge.inria.fr/doc/latest/installation.html`. One can build the library in a similar way as explained for DIPHA.

We note that a python interface was released with the most recent version (at the time of this writing) of the library GUDHI; in this tutorial, we give instructions on how to use the C++ implementation, and we point readers who are familiar with python to the documentation available at `http://gudhi.gforge.inria.fr/python/latest/`.

2.4 JavaPlex

The JAVAPLEX library does not require installation or to be built, and all implementations that we listed in Table 2 in the main text can be found in the directory `matlab-examples_x.y.z` (where `x.y.z` stands for the version number). This directory, and the accompanying tutorial, can be downloaded at `https://github.com/appliedtopology/javaplex/releases/`. All of the scripts in `matlab-examples_x.y.z` are written in MATLAB.

2.5 Hera

The HERA library is available at `https://bitbucket.org/grey_narn/hera`. The root folder includes two subfolders: `geom_bottleneck` contains the source code for computing bottleneck distance, and `geom_matching` contains the source code for computing Wasserstein distance.

One can build the library by running the following commands in each of the subfolders `geom_bottleneck/` and `geom_matching/wasserstein`:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

2.6 jHoles

Ideally, the JHOLES library should be available for download at `http://cuda.unicam.it/jHoles`. However, this website is often down, so the best way to obtain the library is to contact Matteo Rucco, who is the corresponding author of the companion paper [3]. The library does not require installation or to be built.

2.7 Perseus

A compiled version of the PERSEUS library is available at `http://people.maths.ox.ac.uk/nanda/perseus/`. Those wishing to build the library from source code can find the source code at the same website.

2.8 Ripser

The RIPSER library is available at `https://github.com/Ripser/ripser`. One can build the library by running `make` in the folder that includes the `Makefile`. Note that RIPSER supports several options that can be passed to `make`. See `https://github.com/Ripser/ripser` for more information.

3 Computation of PH for networks

In this section, we explain how to compute PH for undirected weighted networks. We represent the nodes of a network with N nodes using the natural numbers $1, \ldots, N$.

3.1 Sample network data

To create weighted networks, one can use the script `fractal_weighted.m` available at `https://github.com/n-otter/PH-roadmap/tree/master/matlab/synthetic_data_sets_scripts`. We recall that a fractal network is determined by three non-negative integers n, b and k (see the main paper for details). The script `fractal_weighted.m` takes four parameters as input: (1) a natural number n (the total number of nodes of the graph is 2^n); (2) a natural number b (the number of nodes of the initial network); (3) a natural number k (this is the connection density parameter); and (4) a string which indicates how weights are associated to edges: this is either `'random'`or `'linear'` (see the main paper for details).

Example: The command

```
>> fractal_weighted(4,2,1,'random')
```

saves the files `fractal_4_2_1_random.txt` and `fractal_4_2_1_random_edge_list.txt`, where the first file is a text file storing the weighted adjacency matrix of a fractal network with 16 nodes and random weight on every edge, while the second file is a text file storing the weighted edge list of the same network.

3.2 Adjacency matrix versus edge-list file

We assume that a network is given either as an adjacency matrix in a MAT-file or as a text file with a list of weighted edges. A typical entry on one line of such a file is a triple "$i\ j\ w_{ij}$", where i and j are the nodes incident to the edge and w_{ij} is the weight of the edge. We call such a file an "edge-list file". We provide the script `adj_matrix_to_edge_list.m` to obtain edge-list files from adjacency matrices. (Note that we also provide the script `edgelist_to_point_cloud_dist_mat.m` to obtain distance matrices from edge-list files, where the distances between nodes are computed using shortest paths.)

3.3 PH with the WRCF (with jHoles)

Using edge-list files, we compute PH by constructing the weight rank clique filtration (WRCF) with the library JHOLES. Here we give instructions on how to compute PH with Version 3 of JHOLES. One needs to run the following command in the terminal:

```
$ java -Xmx<value> -jar jHoles.jar input-file output-file1 output-file2
```

where `-Xmx<value>` is optional and can be used to set the maximum heap size of the garbage collector (we recommend doing this for networks with a large number of nodes or high density). For example, `-Xmx4g` sets the maximum heap size to 4 gigabytes. The file `input-file` is the edge-list file of the network, and `output-file1` is a file in which information (e.g., number of edges, average degree, etc.) about the network is saved, and `output-file2` is the file in which the intervals are saved.

Example: With the command

```
$ java -Xmx4g -jar jHoles.jar fractal_4_2_1_random_edge_list.txt fractal_info.txt \
 fractal_intervals.txt
```

one computes PH with the WRCF for the fractal network with parameters $(n, b, k) = (4, 2, 1)$ and random weight on every edge. The persistence diagram is saved in the file `fractal_intervals.txt`, where for every interval also representative cycles are given, while in file `fractal_info.txt` information such as average degree, density, or average cluster is given. Note that the backslash in the above command indicates that the command continues on the next line, and should therefore be omitted if one writes the whole command on the same line.

3.4 Networks as point clouds

One can construe a connected weighted network as a finite metric space and then compute PH by using one of the methods from Section 4. We now explain how to compute a distance matrix from an undirected weighted network using information about shortest paths between nodes. If two nodes i and j are connected by an edge with weight w, we set the distance between i and j to be $1/w$. Otherwise, we define the distance between i and j to be the minimum of the lengths of all paths between them, where we define the length of a path to be the the sum of the inverse of the weights of the edges in this shortest path. One can compute this distance matrix with the script `shortest_paths.m`. As input, it takes an edge-list file. (See Section 3.2 for how to obtain an edge-list file from a MAT-file that stores an adjacency matrix.) The output of the script is a text file in which each line gives the entries of a row of the distance matrix. (Note that if a network is not connected, then one sets the distance between nodes in two distinct components of the network to be infinite, and one thereby obtains an extended metric space.)

Example:

```
>> shortest_paths('fractal_4_2_1_random_edge_list.txt')
```

saves the text file `'fractal_4_2_1_random_edge_list_SP_distmat.txt'`.

Additionally, using tools like multidimensional scaling, one can convert a distance matrix into a finite set of points in Euclidean space. This is handy if one wants to use a library that does not support distance matrices as an input type. We provide the script `distmat_to_pointcloud.m` to obtain a point cloud from a distance matrix using multidimensional scaling.

Example:

```
>> distmat_to_pointcloud('fractal_4_2_1_random_edge_list_SP_distmat.txt')
```

4 Computation of PH for point clouds

In this section, we explain how to compute PH for finite metric spaces.

4.1 Sample point-cloud data

We provide scripts to create point-cloud data. These are available at `https://github.com/n-otter/PH-roadmap/tree/master/matlab/synthetic_data_sets_scripts`. To create points clouds in $\mathbb{R}^3$ and $\mathbb{R}^4$, one can use the scripts `klein_bottle_imm.m` and `klein_bottle_emb.m`, respectively.

Example:

```
>> klein_bottle_imm(5)
```

samples 25 points uniformly at random from the image of the immersion of the Klein bottle in $\mathbb{R}^3$ and saves the point cloud in the text file `klein_bottle_pointcloud_25.txt`, with each line storing the coordinates of one point, as well as in the MAT-file `klein_bottle_25.mat`.

4.2 Distance matrices versus points clouds

Given a finite set of points in Euclidean space, one can compute an associated distance matrix. To get a distance matrix from a point cloud, we provide the script `pointcloud_to_distmat.m`.

Example:

```
pointcloud_to_distmat('klein_bottle_pointcloud_25.txt')
```

computes the distance matrix for the 25 points sampled from the Klein bottle and saves it in the text file `'klein_bottle_pointcloud_25_distmat.txt'`.

Conversely, a distance matrix can yield a finite set of points in Euclidean space by using a method such as multidimensional scaling. We implement such a conversion in the script `distmat_to_pointcloud.m`.

Example:

```
>> distmat_to_pointcloud('klein_bottle_pointcloud_25_distmat.txt')
```

4.3 VR complex

4.3.1 Input data

The standard input for the construction of the VR complex is a distance matrix. The software packages that take a distance matrix as input are Perseus and DIPHA, but the other packages do not. Instead, they take a set of points in Euclidean space as input. Note that Perseus can also take a set of points in Euclidean space as input, but the implementation does not allow one to set a bound on the upper dimension in the computation of the VR complex, so this implementation is impractical to use for most data sets. We next compute the VR complex with each library. With most of the libraries, one has to indicate the maximum filtration value for which one wants to compute the filtered simplicial complex. We set this value to the maximum distance between any two points. Note, however, that a smaller value often suffices. To compute the maximum distance given a text file `input-file` with the coordinates of a point on each line, one can type the following in MATLAB:

```
>> A=load(input-file);
>> D=pdist(A);
>> D=squareform(D);
>> M=max(max(D));
```

Example:

```
>> A = load('klein_bottle_pointcloud_25.txt');
>> D=pdist(A);
>> D=squareform(D);
>> M=max(max(D));
>> M

M =

    7.0513
```

The maximum distance is given by `M`. Similarly, if one is given a text file `input-file` that stores a distance matrix, then one can compute the maximum distance by typing

```
>> D=load(input-file);
>> M=max(max(D));
```

4.3.2 Dionysus

The library DIONYSUS takes a point cloud as input. We can use either the standard or dual algorithm. For the former, we use the command

```
$ ./rips-pairwise -s max-dimension -m max-distance \
-d output-file input-file
```

where the file `./rips-pairwise` is in the `dionysus/build/examples/rips` directory, further `max-dimension` is the maximum dimension for which we want to compute the simplicial complex, `max-distance` is the maximum parameter value for which we want to compute the complex, `output-file` is the file to which the intervals are written, and `input-file` is a text file with the coordinates of a point on each line. To compute the intervals with the dual algorithm, we use the command

```
$ ./rips-pairwise-cohomology -s max-dimension -m max-distance \
-p prime -d output-file input-file
```

where the file `./rips-pairwise-cohomology` is in the `dionysus/build/examples/cohomology` directory, further `prime` is a prime number and indicates the coefficient field for the computation of cohomology. (For the standard algorithm, there is no choice: one must use $p = 2$.)

Example:

```
$  ./rips-pairwise-cohomology -s 3 -m 7.0513 -p 2 -d klein_bottle_25_output.txt \
klein_bottle_pointcloud_25.txt
```

4.3.3 DIPHA

The DIPHA library reads and writes to binary files. One can convert the text file that stores the distance matrix (see Section 4.2 for how to obtain a distance matrix from a point cloud) into a binary file of the right input type for DIPHA by using the file `save_distance_matrix.m` provided by the developers of DIPHA, which can be found in `dipha-master/matlab`.

Example:

```
>> D=load('klein_bottle_pointcloud_25_distmat.txt');
>> save_distance_matrix(D,'kleinbottle_25.bin');
```

To run DIPHA using a single process, one types the command

```
$ ./dipha [options] --upper_dim d input-file output-file
```

where options include `--benchmark` to display profiling information and `--dual` to run the dual algorithm. (The default is the standard algorithm.) To run DIPHA on more than one process, one uses the command

```
$ mpiexec -n N dipha options --upper-dim d input-file output_file
```

where `N` is the number of process and the above options are again available.

Example:

```
$ ./dipha --benchmark --dual --upper_dim 3 klein_bottle_25.bin klein_bottle_out.bin
```

4.3.4 GUDHI

The library GUDHI takes a point cloud as input. We run the command

```
$ ./rips_persistence -r max-distance -d max-dimension \
-p prime -o output-file input-file
```

where `max-distance`, `max-dimension`, and `prime` are as above, `output-file` is a file to which the intervals are written, and `input-file` is a text file with the coordinates of a point on each line.

Example:

```
$ ./rips_persistence -r 7.0513 -d 3 -p 2 -o klein_bottle_25_output.txt \
 klein_bottle_pointcloud_25.txt
```

4.3.5 JavaPlex

The library JavaPlex takes a point cloud as input. Before starting using this library one has to run the script `load_javaplex.m` which is located in the JavaPlex directory `matlab_examples`. We wrote the script `vietoris_rips_javaplex.m` to compute PH with the VR complex in JavaPlex. This script takes four parameters as input: (1) the name of the text file with the point cloud; (2) the maximum dimension for which we want to compute the simplicial complex; (3) the maximum filtration step for which we want to compute the VR complex; and (4) the number of filtration steps for which we compute the VR complex. The script saves text files containing the barcode intervals, one file for each homological dimension. These are the files ending with `i_right_format.txt` where i indicates the homological dimension.

Example:

```
>> vietoris_rips_javaplex('klein_bottle_pointcloud_25.txt',7.0513,3,20);
```

4.3.6 Perseus

The library PERSEUS takes a distance matrix as input (see Section 4.2 for how to obtain a distance matrix from a point cloud). One has to prepare the input file by adding two lines at the beginning of the file that stores the distance matrix. We do this as follows:

```
N
first-step step-increment steps max-dimension
d11 d12 ...
d21 d22 ...
...
```

where `N` is the number of points (and hence the number of rows (or columns) in the distance matrix), `first-step` is the value for the first filtration step, `step-increment` is the step size between any two filtration steps, `steps` is the number of total steps, and `max-dimension` is the maximum dimension for which we compute the complex. We now can compute PH with the command

```
$ ./perseus distmat input-file output-file
```

in the terminal, where `input-file` is the name of the input file and `output-file` is the name of the file in which the intervals will be saved. PERSEUS creates a series of files named `output-file_i.txt` for $i \in \{0, 1, \dots\}$, where `output-file_i.txt` contains the intervals for the homological degree i.

Example:

```
$ ./perseus distmat klein_bottle_25.txt klein_bottle_25_output.txt
```

where these are the first two lines of the file `klein_bottle_25.txt`:

```
25
0 0.1 71 3
```

4.3.7 Ripser

The library RIPSER takes both a point cloud and a distance matrix as input, and it supports four different format types for the distance matrix. (See `https://github.com/Ripser/ripser#description` for more details.) One of the supported input types for the distance matrix is the format accepted by DIPHA (see Section 4.3.3).

We run the command

```
$ ./ripser --format input-type --dim max-dimension [options] input-file
```

where `input-type` is a string that indicates the type of the input, `max-dimension` is the maximum dimension of persistent homology that is computed (note the difference with respect to the other libraries, for which one indicates the maximum dimension of the complex), and options includes `-- modulus p` (with which one can choose the coefficient field $\mathbb{F}_p$). Note that one has to enable this option at compilation (see Section 2.8). The output of the computation is written to the standard output.

Example:

```
$ ./ripser --format dipha --dim 2 klein_bottle_25.bin > klein_bottle_25_out.log
```

where `klein_bott_25.bin` is the input file from Section 4.3.3 and the standard output is saved to the file `klein_bottle_25_out.log`.

4.4 Alpha

In this section, we explain how to compute PH with the alpha complex with Dionysus and GUDHI.

4.4.1 Dionysus

One can compute PH with the alpha complex for finite subsets of points in $\mathbb{R}^2$ or $\mathbb{R}^3$. For points clouds in $\mathbb{R}^2$, one runs the command

```
$ ./alphashapes2d < input-file > output-file
```

where `input-file` a text file with coordinates of a point in $\mathbb{R}^2$ on each line and `output-file` is the file to which the intervals in the persistence diagram are written. For points clouds in $\mathbb{R}^3$, one runs the command

```
$ ./alphashapes3d-cohomology input-file output-file
```

where `input-file` and `output-file` are as above.[1]

Example:

```
$ ./alphashapes3d-cohomology klein_bottle_pointcloud_25.txt klein_bottle_25_output.txt
```

4.4.2 GUDHI

The program GUDHI supports both points clouds in $\mathbb{R}^2$ and $\mathbb{R}^3$. To compute PH with the alpha complex one can use the script `./alpha_complex_persistence` which is in the folder `example/Persistent_cohomology`. The script takes as input an OFF file, as decribed here `http://www.geomview.org/docs/html/OFF.html`. Namely, the first lines of the input file are as follows:

```
OFF
embedding-dimension V 0 0
x11 x12 ... x1d
x21 x22 ... x2d
...
```

where `embedding-dimension` is the dimension d of the Euclidean space, `V` is the number of points, and all other lines store coordinates `xi1`,..., `xid` of the points.

One then computes PH by running the following command in the terminal

```
$ ./alpha_complex_persistence -p prime -o output-file input-file
```

where `p` is a prime number and indicates that one does computations over the coefficient field $\mathbb{F}_p$.

Example:

```
$ ./alpha_complex_persistence  -p 2 -o klein_bottle_25_output.txt klein_bottle_25_input.txt
```

where the first two lines of the file `klein_bottle_25_input.txt` are as follows:

```
OFF
3 25 0 0
```

[1] There is also a script `./alphashapes3d` , but this script has a bug and does not compute.

4.5 Čech

One can compute PH with the Čech complex for a point cloud in Euclidean space using the implementation in Dionysus. One runs the command

```
$ ./cech-complex < input-file > output-file
```

where `input-file` is a text file of the following form:

```
embedding-dimension max-dimension
x11 x12 ... x1d
x21 x22 ... x2d
...
```

where `embedding-dimension` is the dimension d of the Euclidean space, `max-dimension` is the dimension up to which we compute the complex, and all other lines store coordinates `xi1`,..., `xid` of the points.

Example:

```
$ ./cech-complex < klein_bottle_25_input.txt > klein_bottle_output.txt
```

where the first line of the file `klein_bottle_25_input.txt` is as follows:

```
3 3
```

4.6 Witness

One can compute the witness complex using JavaPlex. Recall that before starting using this library one has to run the script `load_javaplex.m` which is located in the JavaPlex directory `matlab_examples`. Given a point cloud S, the witness complex is a simplicial complex constructed on a subset $L \subseteq S$ of so-called "landmark" points. As we explained in the main manuscript, there are several versions of the witness complex. The ones implemented in JavaPlex are the weak Delaunay complex, which is also just called the "witness complex", and parametrized witness complexes, which are also known as "lazy witness complexes". Given a point cloud L, one can compute the witness complex or lazy witness complex using the scripts `witness_javaPlex.m` and `lazy_witness_javaPlex.m`.

The script `witness_javaPlex.m` takes four parameters as input: (1) the name of the text file with the point cloud; (2) the maximum dimension for which we want to compute the simplicial complex; (3) the maximum filtration value for which we want to compute PH; and (4) the number of filtration steps for which we compute the complex.

The script `lazy_witness_javaPlex.m` takes six parameters as input: (1) the name of the text file with the point cloud; (2) the maximum dimension for which we want to compute the simplicial complex; (3) the number of landmark points; (4) how the landmark points are selected (this is either `'random'` or `'maxmin'`); (5) the value for the parameter ν; and (6) the number of filtration steps for which we compute the complex.

See the scripts for further details on input parameters, and see the main manuscript and the JavaPlex tutorial [2] for further detail on witness complexes.

Example:

```
lazy_witness_javaPlex('klein_bottle_pointcloud_25.txt',3,10,'random',2,20)
```

5 Computation of PH for image data

In this section, we discuss how to compute PH for image data using cubical complexes. The packages DIPHA, Perseus, and GUDHI support the construction of filtered cubical complexes from grey-scale image data. As an example of grey-scale image data, we use the data set "Nucleon" from the Volvis repository [1]. This is a 3-dimensional grey-scale image data set; one is given a 3-dimensional lattice of resolution $41 \times 41 \times 41$, where each lattice point is labeled by an integer that represents the grey-scale value for the voxel anchored at that lattice point. The .raw data file from [1] is binary, and it stores 8 bits for each voxel. We read the .raw data file in MATLAB as follows:

```
>> fileID=fopen('nucleon.raw','r');
>> A=fread(fileID,41*41*41,'int8');
>> B=reshape(A,[41 41 41]);
```

so B is a 3-dimensional array of size $41 \times 41 \times 41$ that stores the grey-scale values.

Note for this example that the cubical complex constructed in DIPHA and GUDHI has dimension 3 and size 531441, while the cubical complex constructed with Perseus has dimension 3 and size 571787. This is because DIPHA and GUDHI implement the optimized way to represent a cubical complex that was introduced in [7]. However, all three libraries implement the same algorithm for the computation of PH from cubical complexes. When interpreting the results of the computations with GUDHI and Perseus, one needs to take into account the rescaling of the grey values (see Section 5.2).

5.1 DIPHA

To save the array in a file that can be given as input to DIPHA, one can use the MATLAB script `save_image_data.m` provided by the developers of DIPHA, which can be found in `dipha-master/matlab`. One gives the array B as input and a name for the input file. One then proceeds in a similar way as for the computation of the VR complex (see Section 4.3).

Example:

```
>> save_image_data(B,'nucleon.bin');
$ ./dipha --benchmark nucleon.bin nucleon_out.bin
```

5.2 Perseus

To compute PH with cubical complexes with Perseus, one needs to rescale the grey values so that all grey values are positive, because Perseus does not allow cells to have negative birth times. We wrote the script `save_image_data_perseus.m` to save the array in a file that can be given as input to Perseus. This script takes the array B as input and a name for the input file for Perseus.

Example:

```
>> save_image_data_perseus(B,'nucleon.txt');
```

To compute PH with Perseus one then runs the following command:

```
$ ./perseus cubtop input-file output-file
```

where `input-file` is the text file prepared with the script `save_image_data_perseus.m`, and `output-file` is the name of the text file to which the barcode intervals are written.

Example:

```
$ ./perseus cubtop nucleon.txt nucleon_output.txt
```

5.3 GUDHI

To compute PH with GUDHI, one can use the same input file as for Perseus. We run the command

```
$ ./Bitmap_cubical_complex input-file
```

and the output is then saved to a file with name `input-file_persistence`.

5.4 Images as point clouds

As we discussed in Section 5.1 of the main manuscript, one can construe a collection of images as a metric space, and one can then apply the methods for computing the PH for point clouds that we discussed in Section 4.

6 Barcodes and persistence diagrams

Once we have computed the intervals, we plot the barcodes and persistence diagrams. The format of the output files varies widely across the different packages. To address this issue and to interpret the results of the computations, we first need to change the format of the output files to a common format. In the unified format, each homological dimension has an accompanying text file in which we store the intervals. In this file, the entry in line i has the form

$$x_i \; y_i \,,$$

where x_i is the left endpoint of the interval and y_i is the right endpoint. If the interval is infinite, we set $y_i = -1$.

- DIONYSUS: We provide the script `dionysus_reformat_output.m` to obtain the right format. This script takes two parameters as input, namely the name of the text file to which the ouput of the computations with DIONYSUS were stored, and a string of five letters indicating the type of file: "dcech" for the output of PH computation with the Čech complex; "alpha" for the output of PH computation with the alpha complex; "VR-st" for the output of PH computation with a Vietoris–Rips complex and the standard algorithm; and "VR-co' for the output of PH computation with a Vietoris–Rips complex and the dual algorithm.
- DIPHA: We provide the script `dipha_reformat_output.m` to obtain the right format. The script takes as input the name of the binary file to which the ouput of the computations with DIPHA were stored.
- GUDHI: We provide the script `gudhi_reformat_output.m` to obtain the right format. The script takes as input the name of the text file to which the output of the computations with GUDHI are stored.
- JAVAPLEX: We wrote the three scripts `vietoris_rips_javaplex.m`, `lazy_witness_javaPlex.m`, and `witness_javaPlex.m`, which have been written to give this type of output.
- JHOLES: We provide the script `jholes_reformat_output.m` to obtain the right format. The script takes as input the name of the text file to which the barcode intervals obtained with JHOLES were stored.
- PERSEUS: The output is already in the right format.
- RIPSER: The script `ripser_reformat_output.m` gives the right format.

Example:

```
>> dionysus_reformat_output('klein_bottle_25_output.txt','alpha')
```

creates the three files `klein_bottle_25_output_0.txt`, `klein_bottle_25_output_1.txt`, and further `klein_bottle_25_output_2.txt`, each storing intervals for PH in dimension 0, 1, and 2, respectively.

We can then plot barcodes using the script `plot_barcodes.m` and plot persistence diagrams using the script `plot_pdg.m`. Both scripts take as inputs (1) the name of a text file storing the intervals for PH in a certain dimension and (2) the title for the plot.

Example:

```
>>  plot_pdg('klein_bottle_25_output_1.txt','Klein bottle alpha dim 1')
```

produces the plot in Fig. 2(a) and saves it as a .pdf file `klein_bottle_25_output_1.pdf`.

Example:

```
>>  plot_barcodes('klein_bottle_25_output_1.txt','Klein bottle alpha dim 1')
```

produces the plot in Fig. 2(a) and saves it as a .pdf file `klein_bottle_25_output_1_barcodes.pdf`.

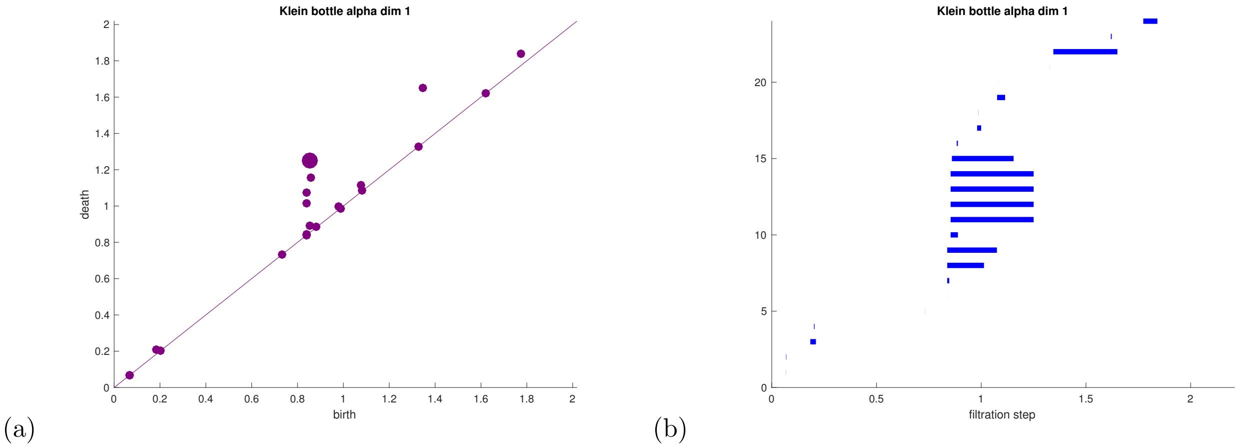


Figure 2: Visualisation of output for the computation of PH in dimension 1 with the alpha complex (with DIONYSUS) for 25 points sampled uniformly at random form the Klein bottle: (a) persistence diagram and (b) barcode.

In the plots in Fig. 2, there are no infinite intervals, so we give an additional example to illustrate how infinite intervals are plotted with our scripts.

Example:

```
>> gudhi_reformat_output('klein_bottle_25_output.txt')
>> plot_barcodes('klein_bottle_25_output_0.txt','Klein bottle VR dim 0')
>> plot_pdg('klein_bottle_25_output_0.txt','Klein bottle VR dim 0')
```

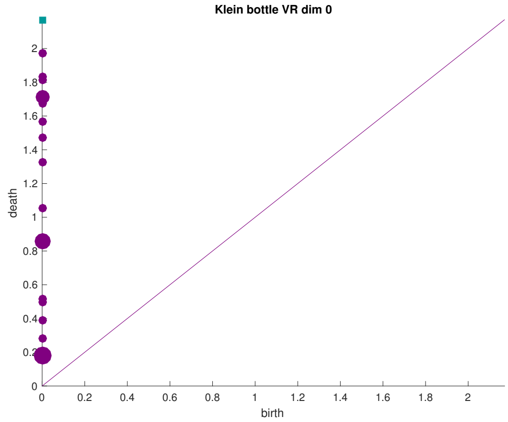
produces the two plots in Fig. 3.

7 Statistical interpretation of barcodes

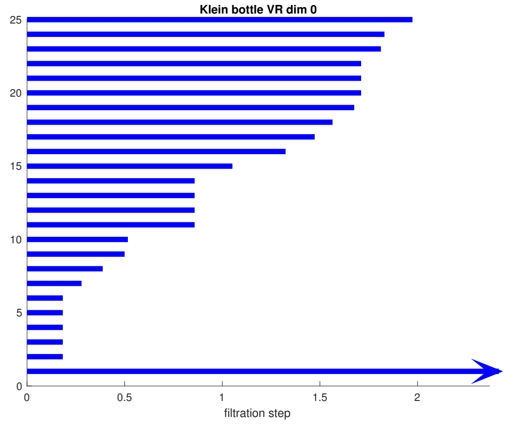
Once one has computed barcodes, one can interpret the results using available implementations of tools (such as bottleneck distance, Wasserstein distance, and persistence landscapes) that are useful for statistical assessment of barcodes. In this section, we give instructions for how to compute the bottleneck and Wasserstein distances with DIONYSUS and HERA. See the tutorial for the Persistence landscape toolbox [4] and the TDA package [5] for instructions on how to use these packages. For ease of reference, we recall the definition of Wasserstein distance from the main manuscript:

Definition 1 *Let $p \in [1, \infty]$. The pth Wasserstein distance between X and Y is defined as*

$$W_p[\mathrm{d}](X, Y) := \inf_{\phi:\, X \to Y} \left[\sum_{x \in X} \mathrm{d}[x, \phi(x)]^p \right]^{1/p}$$



(a)



(b)

Figure 3: Visualisation of output for the computation of PH in dimension 0 with the VR complex (with GUDHI) for 25 points sampled uniformly at random form the Klein bottle: (a) persistence diagram and (b) barcode. Infinite intervals are represented by a square in the persistence diagram, by an arrow in the barcode plot.

for $p \in [1, \infty)$ *and as*

$$W_\infty[\mathrm{d}](X, Y) := \inf_{\phi\colon X \to Y} \sup_{x \in X} \mathrm{d}[x, \phi(x)]$$

for $p = \infty$*, where* d *is a metric on* $\mathbb{R}^2$ *and* ϕ *ranges over all bijections from* X *to* Y*.*

7.1 Bottleneck distance

The bottleneck distance is the Wasserstein distance for $p = \infty$ and $\mathrm{d} = L_\infty$ (see Definition 1).

7.1.1 Dionysus

To compute the bottleneck distance between two barcodes, one can use the script `bottleneck-distance.cpp` (appropriately modified as explained in Section 2.1) in the DIONYSUS subdirectory `examples`. This script requires right endpoints of infinite intervals to be denoted by `inf` ; additionally, if there are intervals of length 0, the script will compute the wrong distance. To make sure that no intervals of length 0 are in the input files and that the intervals of infinite length are in the right format, one can use the script `bottleneck_dionysus.m`. This script takes as input two text files corresponding to two persistence diagrams in the unified format (see Section 6), with one interval per line, as follows:
`>> bottleneck_dionysus('pdg1','pdg2')`
and saves the persistence diagrams to two files called `diagram1.txt` and `diagram2.txt` to the current directory. Now one can compute the bottleneck distance as follows:

```
$ ./bottleneck-distance diagram1.txt diagram2.txt
```

7.1.2 Hera

To compute the bottleneck distance with HERA, one can use the script `bottleneck_dist` in the subdirectory `geom_bottleneck/build/example`. This script requires right endpoints of infinite intervals to be denoted by `-1` ; this corresponds to the convention in the unified format (see Section 6). One can compute the bottleneck distance as follows:

```
$ ./bottleneck_dist diagram1.txt diagram2.txt error
```

where `diagram1.txt` and `diagram2.txt` are two text files corresponding to two persistence diagrams in the unified format, and `error` is a nonnegative real number that is an optional input argument. If `error` is nonzero, instead of the exact distance, an approximation to the bottleneck distance with relative error `error` will be computed. (See the explanation in [6].) This option can be useful when dealing with persistence diagrams that include many off-diagonal points, as it can speed up computations.

7.2 Wasserstein distance

With Dionysus, one can compute the Wasserstein distance for $\mathrm{d} = L_\infty$ and $p = 2$, and one can compute this distance for other values of p with a straightforward modification of the source code. With Hera, one can compute the Wasserstein distance for any choice of metric $\mathrm{d} = L_q$, with $q \in [1, \ldots, \infty]$, for any $p \in [1, \infty)$.

7.2.1 Dionysus

The script `bottleneck-distance.cpp` computes both the bottleneck distance and the Wasserstein distance for $\mathrm{d} = L_\infty$ and $p = 2$, so one can follow the instructions in 7.1.1 to compute the Wasserstein distance for these choices. If one wishes to compute the Wasserstein distance for other values of p, one has to modify the script `bottleneck-distance.cpp` as follows. Towards the end of the file, in the line

```
std::cout << "L2-Distance: " << wasserstein_distance(dgm1, dgm2, 2) << std::endl;
```

one can substitute the third input of the script `wasserstein_distance` with any number $p \in [1, \infty)$.

7.2.2 Hera

With Hera, one can compute the approximate Wasserstein distance discussed in [6]. One can use the script `wasserstein_dist` in the subdirectory `geom_matching/wasserstein/build`. This script requires right endpoints of infinite intervals to be denoted by `-1` ; this corresponds to the convention in the unified format (see Section 6). One can compute the approximate Wasserstein distance as follows:

```
$ ./wasserstein_dist power error distance diagram1.txt diagram2.txt
```

where `power` is the value for p, `error` is the relative error, `distance` is the value for q (where $\mathrm{d} = L_q$ is the employed distance), and `diagram1.txt` and `diagram2.txt` are two text files corresponding to two persistence diagrams in the unified format.

References

[1] Volvis repository. http://volvis.org.

[2] H. Adams and A. Tausz. JavaPlex tutorial. available at `https://github.com/appliedtopology/javaplex`.

[3] J. Binchi, E. Merelli, M. Rucco, G. Petri, and F. Vaccarino. jHoles: A tool for understanding biological complex networks via clique weight rank persistent homology. Electronic Notes in Theoretical Computer Science, 306(0):5–18, 2014. Proceedings of the 5th International Workshop on Interactions between Computer Science and Biology (CS2Bio14).

[4] P. Bubenik and P. Dłotko. A persistence landscapes toolbox for topological statistics. J. Symb. Comput., 78(C):91–114, January 2017.

[5] B. T. Fasy, J. Kim, F. Lecci, and C. Maria. Introduction to the R package TDA. ArXiv:1411.1830, November 2014.

[6] M. Kerber, D. Morozov, and A. Nigmetov. Geometry Helps to Compare Persistence Diagrams. ArXiv e-prints, June 2016. 1606.03357.

[7] H. Wagner, C. Chen, and E. Vuçini. Efficient computation of persistent homology for cubical data. In Ronald Peikert, Helwig Hauser, Hamish Carr, and Raphael Fuchs, editors, Topological Methods in Data Analysis and Visualization II, Mathematics and Visualization, pages 91–106. Springer Berlin Heidelberg, 2012.