A ROADMAP FOR THE COMPUTATION OF PERSISTENT HOMOLOGY

NINA OTTER[†], MASON A. PORTER[†‡], ULRIKE TILLMANN[†], PETER GRINDROD[†], AND HEATHER A. HARRINGTON[†]

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. Despite recent progress, the computation of PH remains a wide 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 applied mathematicians and 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.

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., [57, 61, 75, 108] 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 discipline of "topological data analysis" (TDA) [21] have provided a wealth of new insights in the study of data in an increasingly diverse set of applications — including coverage in sensor networks [37], protein structure [56,77,126], stability of fullerene molecules [125], robotics [10, 105, 119], signals in images [29, 65], periodicity in time series [102], breast-cancer classification [41, 99], viral evolution [25], natural images [23], contagion spread on networks [115], structure of amorphous materials [67], force networks in granular matter [78,79], equities-market networks [84], diverse applications in neuroscience [33,60], collective behavior in biology [117], and time-series output of dynamical systems [87]. 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 attempts to do PH in an object-oriented way [123].

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. Data sets that can be studied with PH are usually finite metric spaces, which are also called "point cloud" data sets in the TDA literature. (For other types of data that can be studied with PH, see Section 5.1.) 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 analyses 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.

[†]Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK
[‡]CABDyN Complexity Centre, University of Oxford, Oxford OX1 1HP, UK

As an illustration, consider the set of points in $\mathbb{R}^2$ that we show in Fig. 1.1. Let ϵ, which we interpret as a "distance parameter," be a nonnegative real number (so in the following $\epsilon = 0$ 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 Fig. 1.1(a).

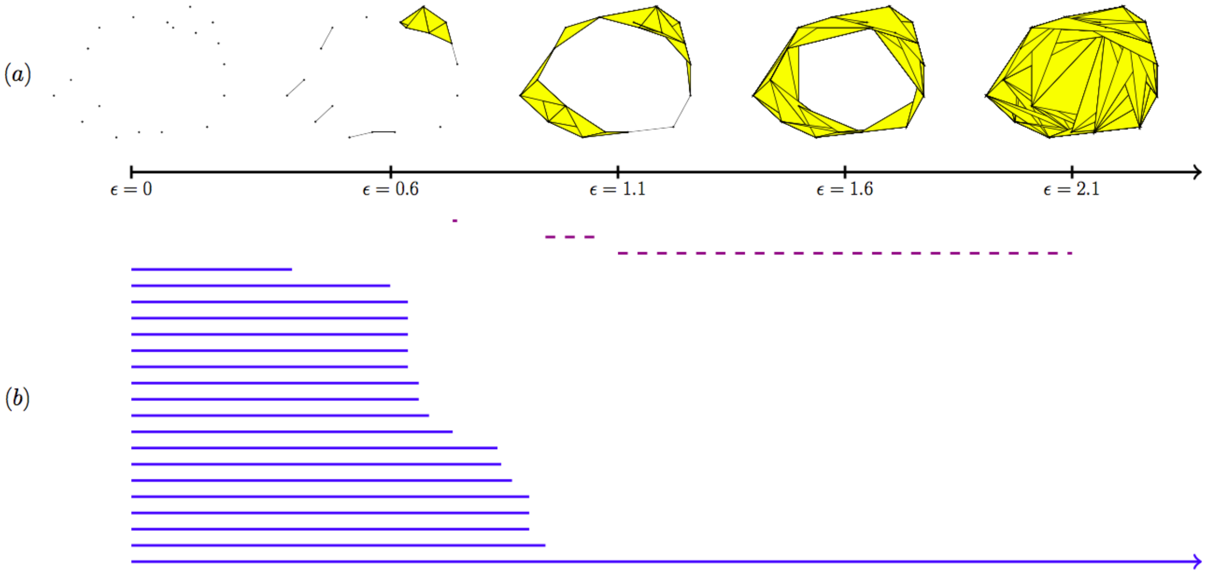


Fig. 1.1: (a) A finite set of points in $\mathbb{R}^2$ (for $\epsilon = 0$) and a nested sequence of spaces obtained from it (from $\epsilon = 0$ to $\epsilon = 2.1$). (b) Barcode for the nested sequence of spaces illustrated in (a). Solid lines represent the lifetime of components, and dashed lines represent the lifetime of holes.

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 collection 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 Fig. 1.1(b), we reproduce such intervals for the number of components (blue solid lines) and the number of holes (violet dashed lines).

In Fig. 1.1(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, one can potentially construe the shorter dashed lines as noise*. 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 Fig. 1.1(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 [57]. We note that one of the most difficult parts of using PH is the statistical interpretation of results. We discuss such interpretation further in Section 5.4.

Our construction of nested spaces gives an example of a "filtered Vietoris–Rips complex," which we define and discuss in Section 5.2. We point to several introductory papers, books, and two videos at the end of Section 4.

In the present article, we focus on persistent homology, but there are other methods in TDA — including the Mapper algorithm [110], Euler calculus (see [59] for an introduction with an eye towards applications),

*However, note that shorter lines cannot always be construed as noise (see, e.g. [18] and [113]).

cellular sheaves [32, 59], and many more. We refer readers who wish to learn more about the foundations of TDA to the excellent article [21], which discusses why topology and functoriality are essential for data analysis.

The first algorithm for the computation of PH was introduced for computation over the field $\mathbb{F}_2$ in [50] and over general fields in [131]. Since then, several algorithms and optimization techniques have been presented, and there are now various powerful implementations of PH [5, 8, 89, 93, 96, 114]. 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 rapidly, 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 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 Perseus, Dionysus, DIPHA, javaPlex, and Gudhi. We include a tutorial (see 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, and cubical complexes for image data. 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. [2, 18, 54]).

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, Gudhi, DIPHA, Dionysus, and Perseus, and we obtain some surprising results (see Section 7.2). There are several benchmarkings in the PH literature; we are aware of the following ones: the benchmarking in [38] compares the implementations of standard and dual algorithms in Dionysus; the one in [97] compares the Morse-theoretic reduction algorithm with the standard algorithm; the one in [8] compares all of the data structures and algorithms implemented in PHAT; the benchmarking in [7] compares PHAT and its spin-off DIPHA; and the benchmarking in C. Maria's doctoral thesis [88] 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 [78, 79, 111, 115]) does not appear in Maria's benchmarking.

3. Homology. Often data lies in a metric space, such as subsets 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, and other deformations. In technical terms, they are *homotopy invariant.* Conversely, under favorable conditions[†], 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.

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 Fig. 3.1.

Definition 3.1. *A* simplicial complex[‡] *is a collection K of non-empty subsets of a set K_0 such that $\tau \subset \sigma$ and $\sigma \in K$ guarantees that $\tau \in K$ and $\{v\} \in K$ for all $v \in K_0$. 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 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. 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 simplicial complex in Fig. 3.1(a). Definition 3.1 is a rather abstract definition, 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 Fig. 3.1(a) has a geometric realization given by the subset of $\mathbb{R}^2$ in Fig. 3.1(b).

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

$$d_p \colon C_p(K) \to C_{p-1}(K) \colon \sigma \mapsto \sum_{\tau \subset \sigma, \tau \in K_{p-1}} \tau \,.$$

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

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

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.

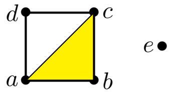
[†] See [66, Corollary 4.33].

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

(a) A simplicial complex:

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

(b) A geometric realization of the simplicial complex in (a) is the following subset of $\mathbb{R}^2$:



(c) We compute the simplicial homology for the simplicial complex in (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.

Fig. 3.1: (a) Example of a simplicial complex, (b) a geometric realisation of this simplicial complex, and (c) computation of its simplicial homology.

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:

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

We give an example of such a sequence in Fig. 3.1(c), for which we also report the Betti numbers.

When working with simplicial complexes, one can define "simple homotopies" by removing or adding a pair of simplices (σ, τ), where τ is a codimension-1 face of σ 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 Ex. 3.1, 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*. 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 7.1 (in Section 7), most of the implementations for the computation of PH work with $\mathbb{F}_2$. See [66] for an introduction to homology, and see [73] for an overview of computational homology.

3.2. Building simplicial complexes. As we discussed in Section 3.1, computing the homology of 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 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 3.3. *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 [12, 48]. 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 $\cup_{x \in S} B(x, \epsilon)$, 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 Subsection 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 [48, 53] for more details on the Čech complex, and see [12, 48] for a precise statement of the Nerve Theorem.

4. Persistent homology. Experimental data often takes 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 [100] 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 Fig. 4.1.

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 subcomplexes is called a *filtered simplicial complex.* We can apply homology to each of the subcomplexes and define the *total pth persistent homology of* K as the graded vector space $\oplus_{i=1}^{l} H_p(K_i)$.

For all homology degrees p, the inclusion maps $K_i \to K_j$ induce $\mathbb{F}_2$-linear maps $f_{i,j} \colon H_p(K_i) \to H_p(K_j)$ for all $i, j \in \{1, \ldots, l\}$ with $i \leq j$. We say that $x \in H_p(K_i)$ is *born* in $H_p(K_i)$ if the preimage $f_{k,i}^{-1}(x)$ is the empty set for all $k < i$. Similarly, we say that x *dies* in $H_p(K_j)$ if $j > i$ is the smallest index for which

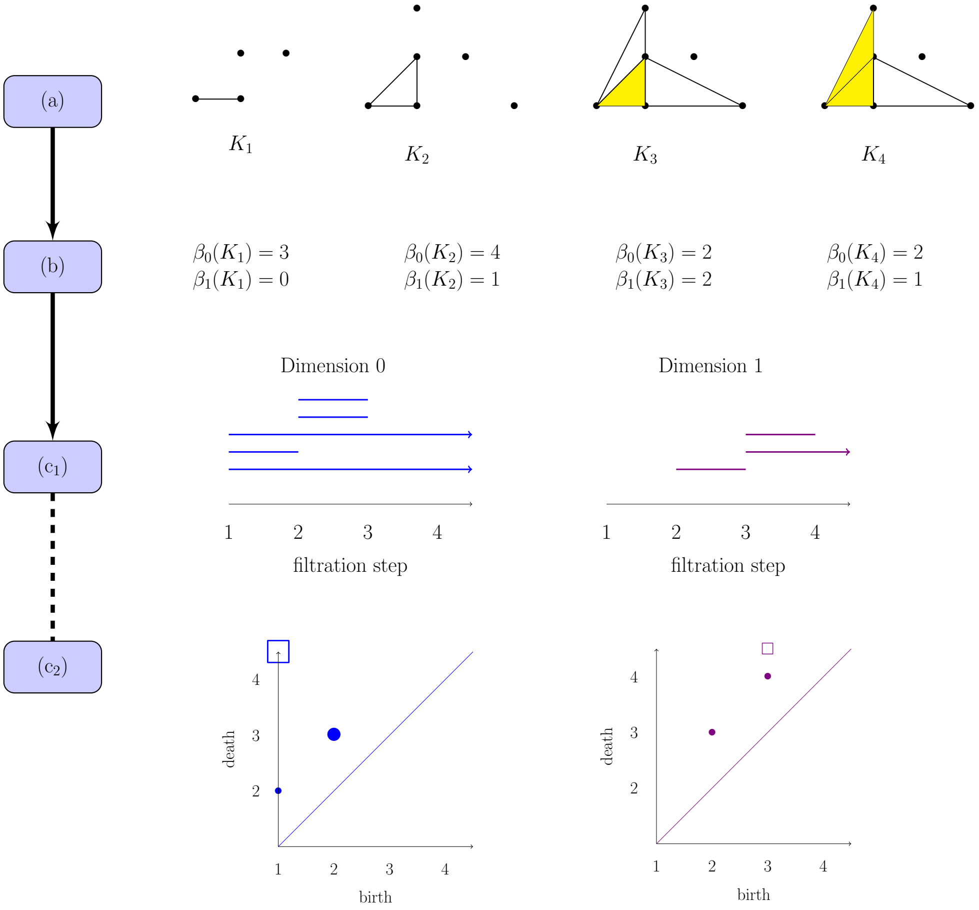


Fig. 4.1: Example of persistent homology for a finite filtered simplicial complex. (a) We start with a finite filtered simplicial complex. That is, we start with a sequence of finite simplicial complexes $K_1, \ldots, K_l$ such that $K_i \subset K_j$ whenever $i \leq j$ for $i, j \in \{1, \ldots, l\}$. Given $i \in \{1, \ldots, l\}$, the subcomplex K_i is called the *ith filtration step*. (b) We compute the simplicial homology of each complex K_i. (c$_1$) By the Fundamental Theorem of Persistent Homology, we obtain a well-defined collection of intervals called the "barcode." We interpret each bar in dimension p as representing the lifetime of a p-homology class across the filtration. Its left endpoint represents the filtration step i at which the class is born, and its right endpoint represents the filtration step $j > i$ at which the class dies. 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. (c$_2$) An alternative way to represent barcodes graphically (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.)

$f_{i,j}(x) = 0$. We can then represent the lifetime of x by the half-open interval $[i, j)$. If $f_{i,j}(x) \neq 0$ for all $i < j \leq l$, we say that x *lives forever*, and we represent its lifetime by the interval $[i, \infty)$.

A more refined and ultimately more useful definition also considers the possibility that two different non-zero classes $x, y \in H_p(K_i)$ map to the same class in $H_p(K_j)$. If y is born earlier than x, we can apply the so-called *Elder Rule* and consider x as subordinate. If both x and y are born at the same time, there remains an ambiguity. One can consider visualizing this situation by merging the half-open lifetime intervals associated to x and y at the point at which their images first become identified. If one draws all lifetime half-lines, one ends up with a rather complicated and unreadable clutter. Fortunately, by the Fundamental Theorem of PH, there exists a collection of elements of the total pth persistent homology such that they and their unique images under the maps $f_{i,j}$ form a basis for the total pth persistent homology. In particular, the half-open lifetime intervals associated to these elements form a collection of disjoint half-open intervals, and the number and lengths of these disjoint intervals do not depend on the choice of these elements. Consequently, there is a well-defined collection of intervals, collectively called the *barcode*, associated to the total pth persistent homology. (See Fig. 4.1 for an example of a barcode.) The Fundamental Theorem of PH, and hence the existence of the barcode, relies on the fact that we are using homology with field coefficients.

There are numerous excellent introductions to PH, such as the books [48, 59, 100, 127] and the papers [21, 46, 47, 58]. For a brief and friendly introduction to PH and some of 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 Fig. 5.1. 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.

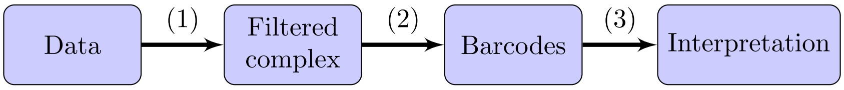


Fig. 5.1: PH pipeline.

5.1. Data. Thus far, we have focused on the context of analyzing data sets that are in the form of finite metric spaces. However, there also exist other types of data for which one can construct filtered complexes. These include networks and images.

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.") We refer the reader to [104] for an application of this method. There are many other methods to associate a simplicial complex to both undirected and directed networks. See the book [69] for an overview of such methods, and see the paper [68] for an overview of PH for networks.

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 [73] 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 [122] for an efficient algorithm and data structure for the computation of PH for cubical data, and [9] for an algorithm that computes PH for cubical data in an approximate way. For an application of PH and cubical complexes to movies, see [97].

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 [23], in which the local structure of natural images was studied by selecting 3×3 patches of pixels of the images.

5.2. From Data Point Clouds to 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 5.1, we summarize the simplicial complexes that we discuss in this subsection.

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 $S_{\epsilon_1}, S_{\epsilon_2}, \dots, S_{\epsilon_l}$, 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

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

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

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

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 $\{x_0, \dots, x_k\}$ is a k-simplex if and only if every pair of vertices in $\{x_0, \dots, x_k\}$ 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 [128] 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 [48, 121]. For a proof of the approximation of the Čech complex by the VR complex, see [48]; see [76] 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

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

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" [§] 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 $d \leq 2$, the best algorithms have complexity $\mathcal{O}(N \log N)$, where N is the cardinality of S. For $d \geq 3$, 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 [13] for a discussion of progress in this direction, and see [62] 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 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 [49], in 3-dimensional Euclidean space in [52], and for Euclidean spaces of arbitrary dimension in [45]. 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 α-complexes [47]. (See [81] for the algorithm, and see [80] 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 [35].)

Definition 5.1. *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 5.2. *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* $\mathrm{Del}^w_{\text{lazy}}(L; X, \epsilon)$. 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 $\mathrm{Del}^w_{\text{lazy}}(L; X, \epsilon)$ whenever its edges are in $\mathrm{Del}^w_{\text{lazy}}(L; X, \epsilon)$. 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

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

[§] 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.

Table 5.1: We summarize several types of complexes used for PH. 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, we indicate by L the set of landmark points, and we indicate by Q the subsample set for the graph induced complex.

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

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 [35], and parametrized witness complexes were introduced in [36]. Witness complexes can be rather useful for applications. Because their complexity depends on the number of landmark points, one can reduce 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 [4,64], but they can fail to recover topology for submanifolds of Euclidean space of three or more dimensions [14]. Consequently, there have been studies of simplicial complexes that are similar to the witness complexes but with better theoretical guarantees. One of them is the "graph-induced complex" [42] (see the next subsection).

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 [42], which is a simplicial complex constructed on a subsample Q, and has better theoretical guarantees than the witness complex (see [71] 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 [109]; and an approximation of the Čech complex [76] whose worst-case size also scales linearly in the data. 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 7.1 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 [107], which was adapted to filtrations of simplicial complexes in [92]. The basic idea of the algorithm developed in [92] 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 total 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 [70], and one thus relies on heuristics to find partial matchings to reduce the size of the complex.

A method for the reduction of the size of a complex for clique complexes, such as the VR complex, was introduced in [129] 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) [130]. The tidy-set method is a heuristic, because it does not give a guarantee on 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, 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;
- 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 $\sigma_1, \dots, \sigma_n$ 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. 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 [50] and for general fields in [131]. For every $j \in \{1, \dots, n\}$, we define $\text{low}(j)$ to be the largest index value i such that $\delta(i,j)$ is different from 0.[¶] If column j only contains 0 entries, then the value of $\text{low}(j)$ is undefined. We say that the boundary matrix is *reduced* if the map low is injective on its domain of definition. In Alg. 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.

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

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

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 $\text{low}(j) = i$, 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 $\text{low}(j)$ 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 $\text{low}(k) = j$, 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.

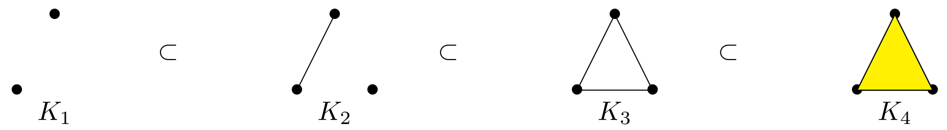
A pair (σ_i, σ_j) gives the half-open interval $[i, j)$ in the barcode, and an unpaired simplex σ_k gives the infinite interval $[k, \infty)$. We give an example of PH computation in Fig. 5.2.

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 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.

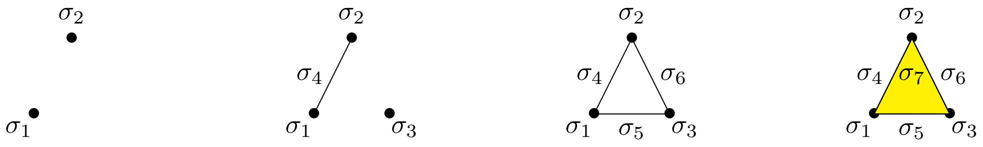
As we mentioned in Section 5.3.1, in the worst case, the standard algorithm has cubic complexity in the number of simplices. This bound is sharp, as Morozov gave an example of a complex with cubic complexity in [94]. Note that in cases such as when matrices are sparse, complexity is less than cubic. Milosavljević, Morozov, and Skraba [91] 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 [31]. 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

[¶] 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.

(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 Alg. 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, since 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.

Fig. 5.2: Example of PH computation with the `standard algorithm` Alg. 1.

complexity in the number of simplices. Sequential algorithms include the `twist algorithm` [27] and the `dual algorithm` [38, 39]. (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 the SI).) Parallel algorithms in a shared setting include the `spectral-sequence algorithm` [48, Section VII.4] and the `chunk algorithm` [6], and parallel algorithms in a distributed setting include the `distributed algorithm` [7]. The `multifield algorithm` is a sequential algorithm that allows the simultaneous computation of PH over several fields [15].

5.4. Statistical interpretation of topological summaries. Once we have obtained the barcodes, we need to interpret the results of our 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.

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 [19]. Roughly, there are two basic types of current approaches to this problem. In the first approach, one studies properties of a metric space whose points are persistence diagrams, while in the second approach one studies "features" of persistence diagrams.

More precisely, in the first 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 Fig. 4.1 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 5.3. *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. We include the diagonal points for technical reasons; roughly, it is desirable to be able to compare persistence diagrams by studying bijections between their points, 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 5.4. *Let $p \in [1, \infty]$. The pth Wasserstein distance between X and Y is defined as*

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

for $1 \leq p < \infty$ 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.

Usually, one takes $\mathrm{d} = L_p$ for $1 \leq p \leq \infty$. 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 at present there are few tools that can be used in applications. We refer the reader to the publications [90, 95, 118] for research in this direction.

The second approach for the development of statistical tools for PH consists in mapping the space of persistence diagrams to spaces (e.g., Banach spaces) that are amenable to statistical analysis. Such methods include persistence landscapes [16] (see also the tutorial [18]), the space of algebraic functions [3], and persistence images [28]. See the papers [77, 113] for applications of persistence landscapes.

The library Dionysus [93] implements the computation of the bottleneck and Wasserstein distance (for $d = L_\infty$), while the Persistence Landscape Toolbox [44] and the TDA Package [55] implement many tools for the statistical interpretation of persistent homology.

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 [30], asserts that, under favorable conditions, step (2) in the pipeline in Fig. 5.1 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 [17, 20, 26]. Stability for PH is an active area of research; for an overview of stability results, their history and recent developments, see [100, 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:

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

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

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

was introduced in [22]. In the more general setting in which maps are not inclusions, one can still compute PH using the `simplicial map algorithm` [43].

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:

$$\begin{tikzcd}
& \vdots & \vdots & \vdots & \\
\dots \arrow[r] & K_{j+1,i-1} \arrow[u] \arrow[r] & K_{j+1,i} \arrow[u] \arrow[r] & K_{j+1,i+1} \arrow[u] \arrow[r] & \dots \\
\dots \arrow[r] & K_{j,i-1} \arrow[u] \arrow[r] & K_{j,i} \arrow[u] \arrow[r] & K_{j,i+1} \arrow[u] \arrow[r] & \dots \\
\dots \arrow[r] & K_{j-1,i-1} \arrow[u] \arrow[r] & K_{j-1,i} \arrow[u] \arrow[r] & K_{j-1,i+1} \arrow[u] \arrow[r] & \dots \\
& \vdots \arrow[u] & \vdots \arrow[u] & \vdots \arrow[u] &
\end{tikzcd}$$

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 [24, 100] for the theory of multi-parameter persistent homology, and see [86] (and [85] for its companion paper) for upcoming software for the visualization of 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 7.1.

The software package javaPlex [114], which was developed by the computational topology group at Stanford University, is based on the Plex library [103], which to our knowledge is the first piece of software to implement the computation of PH. Perseus [96] was developed to implement Morse-theoretic reductions [92] (see Section 5.2.6). jHoles [11] is a Java library for computing the weight rank clique filtration for weighted undirected networks [104]. Dionysus [93] is the first software package to implement the `dual algorithm` [38,39]. PHAT [8] 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 [5], a spin-off of PHAT, implements a distributed computation of the matrix-reduction algorithm. Gudhi [89], the most recently developed software of the set that we examine, 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 [15]. This library is currently under intense development. SimpPers [72] implements the `simplicial map algorithm`. Libraries that implement techniques for the statistical interpretation of barcodes include the TDA Package [55] and the Persistence Landscape Toolbox [44]. RIVET, a package for visualizing 2-parameter persistent homology, will be released soon [86]. We summarize the properties of all publicly-available libraries with accompanying peer-reviewed publication in Table 7.1, and we report on performance for a selection of them in Section 7.1.3. 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 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

[‖] In the SI, we report benchmarking of some features implemented by only some of the five libraries that we test.

Table 7.1: Overview of existing software (which have an accompanying peer-reviewed publication) for the computation of PH. The symbol "–" signifies that the associated feature is not implemented, and "✓" signifies that it is implemented. For each software package, we indicate the following. (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, W stands for the weak witness complex, W_ν stands for parametrized witness complexes, WRCF stands for the weight rank clique filtration, and α stands for the alpha complex. Perseus, DIPHA, and Gudhi implement the computation of the lower-star filtration [51] 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 also Tutorial in the SI for more details). Note that DIPHA uses the efficient representation of cubical complexes presented in [122], and thus the size of the cubical complex in DIPHA is smaller than the size of the resulting complex with the other libraries. (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 [66]; in contrast with simplicial complexes, currently there do not 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 [114] for details), and Dionysus implements the computation of vineyards [22] and circle-valued functions [39]. Both javaPlex and Dionysus support the output of representatives of homology classes for the intervals in a barcode. (h) Whether executable files are provided. (i) Whether visualization of the output is provided.

Software	javaPlex	Perseus	jHoles	Dionysus	PHAT	DIPHA	Gudhi	SimpPers
(a) Language	Java	C++	Java	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 seq.	twist, dual, distributed	dual, multifield	simplicial map
(c) Coeff. field	$\mathbb{Q}$ $\mathbb{F}_p$	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_2$ (standard, zigzag) $\mathbb{F}_p$ (dual)	$\mathbb{F}_2$	$\mathbb{F}_2$	$\mathbb{F}_p$	$\mathbb{F}_2$
(d) Homology	simplicial, cellular	simplicial, cubical	simplicial	simplicial	simplicial, cubical	simplicial, cubical	simplicial, cubical	simplicial
(e) Filtrations computed	VR W W_ν	VR, lower star of cubical complex	WRCF	VR α Čech	–	VR, lower star of cubical complex	VR α W lower star of cubical complex	–
(f) Filtrations as input	simplicial complex, zigzag, CW	simplicial complex, cubical complex	–	simplicial complex, zigzag	boundary matrix of simpl. complex	boundary matrix of simpl. complex	–	map of simpl. complexes
(g) Additional features	Computes some hom. alg. constructions, homology generators	–	–	vineyards, circle-valued functions, homology generators	–	–	–	–
(h) Precompiled	✓	✓	✓	–	–	–	–	✓
(i) Visualization	barcodes	persistence diagram	–	–	–	persistence diagram	–	–

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.

We study the software packages JAVAPLEX, PERSEUS, DIONYSUS, DIPHA, and GUDHI 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: these arise from topology (1), stochastic topology (2), dynamical systems (3), and from an area at the intersection of network theory and fractal geometry (4) and which was first used to study connection patterns of the cerebral cortex (see below for details). Data sets (5)–(12) are from empirical experiments and measurements: they arise from phylogenetics (5)–(6), genomics (9), neuroscience (8), image analysis (7), 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 grey-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 on edges as distances between nodes (except for the US Congress networks; 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 Fig. 7.1). 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 they do have the same singular homology** with $\mathbb{F}_2$ coefficients. 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 $\mathbb{F}_2$ coefficients.

(2) Random VR complexes (uniform distribution) [74]. 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 $G(N, \epsilon)$ [101].) 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 [74].

(3) Vicsek biological aggregation model. This model was first introduced in [120] and was studied using PH in [117]. We implement the model in the form in which it appears in [117]. 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 fixed absolute value for the velocity v_0, the number of particles N, 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. We run three simulations of the model using the parameter values used in [117]. For each simulation, we

** *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. We refer the reader to [66] for an account of singular homology.

choose two point clouds that correspond to two different time frames. See [117] for further details. We denote this data set by **Vicsek**.

(4) Fractal networks. These are self-similar networks introduced in [112] 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 on 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 connnection 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 three ways of defining distances between weights using the abbreviations "r" for random, "l" for linear, and "o" for the network with distance 1 between any two adjacent nodes.

(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 1088 different genomic sequences and compute distances between them by using the Hamming distance. We use the aligned sequences studied using PH in [25]. (The authors of that paper retrieved the sequences from [82].) We denote this data set by **HIV**.

(6) Genomic sequences of H3N2. These are 1000 different genomic sequences of H3N2 influenza. We compute the Hamming distance between sequences. We use the aligned sequences studied using PH in [25]. We denote this data set by **H3N2**.

(7) Stanford Dragon graphic. We sample points uniformly at random from 3-dimensional scans of the dragon [83], whose reconstruction we show in Fig. 7.1. The sample sizes contain 1000 and 2000 points. We denote these data sets by **drag 1** and **drag 2**, respectively.

(8) *C. elegans* neuronal network [104]. A weighted, undirected network in which each node is a neuron and edges represent synapses or gaps junctions. We denote this data set by **eleg**.

(9) Human genome. A weighted, undirected network representing a sample of the human genome. We use the network studied using PH in [104]. (The authors of that paper created the sample using data retrieved from [34].) Each node represents a gene, and weighted edges between nodes represent the correlation of the expression level of two genes. We denote this data set by **genome**.

(10) Grey-scale image: 3-dimensional rotational angiography scan of a head with an aneurysm. This data set was used in the benchmarking in [7]. This data set is given by a 3-dimensional array of size 512 × 512 × 512, with each entry storing an integer that represents the grey value for the corresponding voxel. We retrieved the data set from the repository [1]. 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 [124] from data compiled by [106]. 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 [124] 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 [98]. Nodes represent authors and edges represent collaborations, where weights indicate the number of joint papers. The largest connected components consists of 379 nodes. We denote this data set by **netw-sc**.

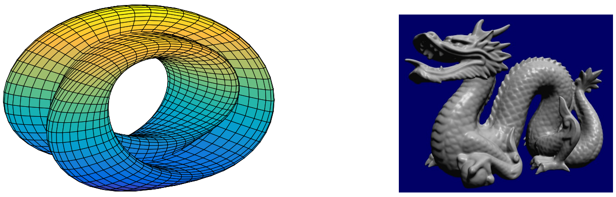


Fig. 7.1: Plot of the image of the figure-8 immersion of the Klein bottle and the reconstruction of the Stanford Dragon (retrieved from [83]).

7.1.2. Machines and compilers. We tested all of the libraries on both a cluster and a shared-memory system. The cluster is a Dell Sandybridge cluster, it has 1728 (i.e., 108×16) cores of 2.0GHz 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 3.3GHz, RAM of 768 GB, and storage of 3 TB. It runs the OS Ubuntu 14.04.01. 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 7.2 and 7.3.) 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 five software packages that we study, three implement the computation of the dual algorithm, and four implement the standard algorithm. It is reported in [114] 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 (as of 26 May 2016), 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 (in terms of CPU seconds) for complexes of all sizes.[††] We see this in our experiments, and it is also discussed in [7]. 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 7.2, we give the computation times for the different software packages. We measure elapsed and CPU time by using the `time` function in Linux. For space reasons, we report results for a subset of the computations, and we refer the reader to the SI for a tabulation of the rest of our computations. In Table 7.3, 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

[††] 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. At present, there are no software packages that can handle complexes with 10 billion simplices or more.

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 7.3. In Table 7.4, 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 7.2: Performance of the software packages measured in wall-time (i.e., elapsed time), and CPU seconds (for the computations running on the cluster). 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 implements 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**; on eight nodes of 16 cores for **random**. (Note that 128 is the maximum amount of processes that we could use at any one time.) (c) We run DIPHA on a single core.

(a) Computations on cluster: wall-time seconds

Data set	**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	1830	-	-	-	-
DIPHA (st)	6	90	1631	142559	-	9110
Perseus	543	1978	-	-	-	-
Dionysus (d)	513	145	-	-	-	-
DIPHA (d)	4	6	81	2358	5096	232
Gudhi	6	11	249	15419	-	754

(b) Computations on cluster: CPU seconds

Data set	**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	1031	-	-	-	-
Dionysus (st)	473	1824	-	-	-	-
DIPHA (st)	68	1360	25950	1489615	-	130972
Perseus	542	1974	-	-	-	-
Dionysus (d)	513	145	-	-	-	-
DIPHA (d)	39	73	1276	37572	79691	3622
Gudhi	4	10	248	3151	-	739

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

Data set	**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)	13607	1358	43861	-	28064	-
Perseus	-	1271	-	-	-	-
Dionysus (d)	-	100	142055	35366	-	572764
DIPHA (d)	926	13	773	4482	1775	3923
Gudhi	464	9	245	2458	595	-

7.2. Conclusions from our benchmarking. Our tests suggest that DIPHA is the most powerful library currently available, as it is fast and can handle complexes with up to 3 billion simplices. The library Gudhi, even though it is faster than DIPHA for many computations, failed to compute PH for complexes with more than 2 billion simplices in our benchmarking. However, the implementation of the

Table 7.3: Memory usage in GB for the computations summarized in Table 7.2. 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 memory usage. For DIPHA, we indicate the maximum memory used by a single core (considering all cores). See Table 7.2 for details on the number of cores used.

(a) Computations on cluster

Data set	**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	0.6	9.9	64.5	-	25

(b) Computations on shared-memory system

Data set	**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	17.9	0.6	11.8	73.2	25	-

Table 7.4: Maximal size of simplicial complex supported by the software thus far in our tests.

Software	javaPlex	Dionysus		DIPHA		Perseus	Gudhi
	st	st	d	st	d	st	d
Max. size	$4.5 \cdot 10^8$	$1.1 \cdot 10^7$	2.8×10^9	$1.3 \cdot 10^9$	$3.4 \cdot 10^9$	$1 \cdot 10^7$	$1.3 \cdot 10^9$

dual algorithm in Dionysus was able compute PH for the data set **fract**, for which the simplicial complex has approximately 2.8 billion simplices, whereas the library Gudhi ran out of memory for this data set. This result is surprising, because Gudhi and DIPHA are widely considered to be the best libraries currently available.

There is a huge disparity between implementations of the dual and standard algorithms. 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 [38].

To conclude, in our benchmarking, the fastest software packages were Gudhi and DIPHA, and the packages that were able to handle the largest number of simplices (about three billion) were DIPHA and Dionysus. 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.

In the following we give guidelines for the computation of PH based on our benchmarking. We list several types of data sets in Table 7.5 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.

Table 7.5: Guidelines for which implementation is best-suited for which data set, based on our benchmarking. Recall that we indicate the implementation of the dual algorithm using the abbreviation "d" following the name of a package. 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 smaller than 1. Note that the newest version (as of 26 May 2016) of the library Gudhi implements the computation of PH with cubical complexes, witness complexes and alpha complexes (as we report in Table 7.1). We could not test these implementations because the new version of the Gudhi depends on a version of CGAL that is not supported yet on our machines.

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

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 Fig. 5.1, and there exist implementations of many fast 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., [76, 109]) 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., [16, 19, 118]) of the PH pipeline.

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 [63], Sage [40], and CGAL [116].

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

Acknowledgements. We thank the Rabadan Lab at Columbia University for providing the HIV and H3N2 sequences used in [25] and Giovanni Petri for sharing the data sets used in [104]. We thank Adrian Clough, Patrizio Frosini, Florian Klimm, Vitaliy Kurlin, Robert MacKay, 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.

REFERENCES

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

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

[3] A. Adcock, E. Carlsson, and G. Carlsson, *The ring of algebraic functions on persistence bar codes*, ArXiv e-prints, (2013). 1304.0530.

[4] D. Attali, H. Edelsbrunner, and Y. Mileyko, *Weak witnesses for Delaunay triangulations of submanifolds*, in Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling, SPM '07, New York, NY, USA, 2007, ACM, pp. 143–150.

[5] U. Bauer, M. Kerber, and J. Reininghaus, *DIPHA (A distributed persistent homology algorithm)*. Software available at https://code.google.com/p/dipha/.

[6] ———, *Clear and compress: Computing persistent homology in chunks*, in Topological Methods in Data Analysis and Visualization III, P.-T. Bremer, I. Hotz, V. Pascucci, and R. Peikert, eds., Mathematics and Visualization, Springer International Publishing, 2014, pp. 103–117.

[7] ———, *Distributed computation of persistent homology*, in 2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX), Society for industrial and applied mathematics, 2014, ch. 3, pp. 31–38.

[8] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner, *PHAT: Persistent homology algorithms toolbox*, in Mathematical Software, ICMS 2014, Hoon Hong and Chee Yap, eds., vol. 8592 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 137–143. Software available at https://code.google.com/p/phat/.

[9] P. Bendich, H. Edelsbrunner, and M. Kerber, *Computing robustness and persistence for images*, IEEE Transactions on Visualization and Computer Graphics, 16 (2010), pp. 1251–1260.

[10] S. Bhattacharya, R. Ghrist, and V. Kumar, *Persistent homology for path planning in uncertain environments*, IEEE Transactions on Robotics, 31 (2015), pp. 578–590.

[11] 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 (2014), pp. 5–18. Proceedings of the 5th International Workshop on Interactions between Computer Science and Biology (CS2Bio14).

[12] A. Björner, *Topological methods*, in Handbook of combinatorics, R. Graham, M. Grötschel, and L. Lovász, eds., Elsevier Science B.V., 1995, ch. 34, pp. 1819–1872.

[13] J.-D. Boissonnat, O. Devillers, and S. Hornus, *Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension*, in Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry, SCG '09, New York, NY, USA, 2009, ACM, pp. 208–216.

[14] J.-D. Boissonnat, L. J. Guibas, and S. Y. Oudot, *Manifold reconstruction in arbitrary dimensions using witness complexes*, Discrete & Computational Geometry, 42 (2009), pp. 37–70.

[15] J.-D. Boissonnat and C. Maria, *Computing persistent homology with various coefficient fields in a single pass*, in Algorithms - ESA 2014, A. S. Schulz and D. Wagner, eds., vol. 8737 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 185–196.

[16] P. Bubenik, *Statistical topological data analysis using persistence landscapes*, Journal of Machine Learning Research, 16 (2015), pp. 77–102.

[17] P. Bubenik, V. de Silva, and J. Scott, *Metrics for generalized persistence modules*, Foundations of Computational Mathematics, 15 (2014), pp. 1501–1531.

[18] P. Bubenik and P. Dłotko, *A persistence landscapes toolbox for topological statistics*, ArXiv e-prints, (2015). 1501.00179.

[19] Peter Bubenik and Peter T. Kim, *A statistical approach to persistent homology*, Homology, Homotopy and Applications, 9 (2007), pp. 337–362.

[20] P. Bubenik and J. A. Scott, *Categorification of persistent homology*, Discrete & Computational Geometry, 51 (2014), pp. 600–627.

[21] G. Carlsson, *Topology and data*, Bulletin of the American Mathematical Society, 46 (2009), pp. 255–308.

[22] G. Carlsson, V. de Silva, and D. Morozov, *Zigzag persistent homology and real-valued functions*, in Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry, SCG '09, New York, NY, USA, 2009, ACM, pp. 247–256.

[23] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian, *On the local behavior of spaces of natural images*, International Journal of Computer Vision, 76 (2008), pp. 1–12.

[24] G. Carlsson and A. Zomorodian, *The theory of multidimensional persistence*, Discrete & Computational Geometry, 42 (2009), pp. 71–93.

[25] J. Minhow Chan, G. Carlsson, and R. Rabadan, *Topology of viral evolution*, Proceedings of the National Academy of Sciences, 110 (2013), pp. 18566–18571.

[26] F. Chazal, D. Cohen-Steiner, M. Glisse, L. J. Guibas, and S. Y. Oudot, *Proximity of persistence modules and their diagrams*, in Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry, SCG '09, New York, NY, USA, 2009, ACM, pp. 237–246.

[27] C. Chen and M. Kerber, *Persistent homology computation with a twist*, in Proceedings of the 27th European Workshop on Computational Geometry, 2011, pp. 197–200.

[28] S. Chepushtanova, T. Emerson, E. Hanson, M. Kirby, F. Motta, R. Neville, C. Peterson, P. Shipman, and L. Ziegelmeier, *Persistence images: An alternative persistent homology representation*, ArXiv e-prints, (2015). 1507.06217.

[29] M. K. Chung, P. Bubenik, and P. T. Kim, *Persistence diagrams of cortical surface data*, in Information Processing in Medical Imaging, J. L. Prince, D. L. Pham, and K. J. Myers, eds., vol. 5636 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 386–397.

[30] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer, *Stability of persistence diagrams*, Discrete & Computational Geometry, 37 (2007), pp. 103–120.

[31] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation, 9 (1990), pp. 251–280. Computational algebraic complexity editorial.

[32] J. Curry, *Sheaves, cosheaves and applications*, ArXiv e-prints, (2013). 1303.3255.

[33] C. Curto, *What can topology tell us about the neural code?* in American Mathematical Society Current Events Bulletin (Joint Mathematics Meetings), available at http://www.ams.org/meetings/currentevents2016final.pdf, 2016.

[34] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–25. Available at http://www.cise.ufl.edu/research/sparse/matrices.

[35] Vin de Silva, *A weak characterisation of the Delaunay triangulation*, Geometriae Dedicata, 135 (2008), pp. 39–64.

[36] V. de Silva and G. Carlsson, *Topological estimation using witness complexes*, in Proceedings of the First Eurographics conference on Point-Based Graphics, Eurographics Association, 2004, pp. 157–166.

[37] V. de Silva and R. Ghrist, *Coverage in sensor networks via persistent homology*, Algebraic & Geometric Topology, (2007), pp. 339–358.

[38] V. de Silva, D. Morozov, and M. Vejdemo-Johansson, *Dualities in persistent (co)homology*, Inverse Problems, 27 (2011), p. 124003.

[39] ———, *Persistent cohomology and circular coordinates*, Discrete & Computational Geometry, 45 (2011), pp. 737–759.

[40] The Sage Developers, *Sage Mathematics Software*. http://www.sagemath.org.

[41] D. DeWoskin, J. Climent, I. Cruz-White, M. Vazquez, C. Park, and J. Arsuaga, *Applications of computational homology to the analysis of treatment response in breast cancer patients*, Topology and its Applications, 157 (2010), pp. 157–164. Proceedings of the International Conference on Topology and its Applications 2007 at Kyoto; Jointly with 4th Japan Mexico Topology Conference.

[42] T. K. Dey, F. Fan, and Y. Wang, *Graph induced complex on point data*, in Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry, SoCG '13, New York, NY, USA, 2013, ACM, pp. 107–116.

[43] ———, *Computing topological persistence for simplicial maps*, in Proceedings of the Thirtieth Annual Symposium on Computational Geometry, SOCG'14, New York, NY, USA, 2014, ACM, pp. 345–354.

[44] P. Dłotko, *Persistence landscape toolbox*. Available at https://www.math.upenn.edu/~dlotko/persistenceLandscape.html.

[45] H. Edelsbrunner, *The union of balls and its dual shape*, Discrete & Computational Geometry, 13 (1995), pp. 415–440.

[46] H. Edelsbrunner and Morozov D., *Persistent homology: Theory and practice*, in Proceedings of the European Congress of Mathematics, 2012, pp. 31–50.

[47] H. Edelsbrunner and J. Harer, *Persistent Homology — A Survey*, in Surveys on Discrete and Computational Geometry. Twenty Years Later, J. E. Goodman, J. Pach, and R. Pollack, eds., vol. 453 of Contemporary Mathematics, 2008, pp. 257–282.

[48] ———, *Computational Topology: An Introduction*, Applied mathematics, American Mathematical Society, 2010.

[49] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, *On the shape of a set of points in the plane*, IEEE Transactions on Information Theory, 29 (1983), pp. 551–559.

[50] H. Edelsbrunner, D. Letscher, and A. Zomorodian, *Topological persistence and simplification*, Discrete & Computational Geometry, 28 (2002), pp. 511–533.

[51] H. Edelsbrunner, D. Morozov, and V. Pascucci, *Persistence-sensitive simplification functions on 2-manifolds*, in Proceedings of the Twenty-second Annual Symposium on Computational Geometry, SCG '06, New York, NY, USA, 2006, ACM, pp. 127–134.

[52] H. Edelsbrunner and E. P. Mücke, *Three-dimensional alpha shapes*, ACM Transactions on Graphics, 13 (1994), pp. 43–72.

[53] S. Eilenberg and N. E. Steenrod, *Foundations of algebraic topology*, Princeton Mathematical Series, Princeton University Press, 1952.

[54] B. T. Fasy, J. Kim, F. Lecci, and C. Maria, *Introduction to the R package TDA*, ArXiv e-prints, (2014). 1411.1830.

[55] B. T. Fasy, J. Kim, F. Lecci, C. Maria, and V. Rouvreau, *TDA: Statistical Tools for Topological Data Analysis*. Available at https://cran.r-project.org/web/packages/TDA/index.html.

[56] M. Gameiro, Y. Hiraoka, S. Izumi, M. Kramár, K. Mischaikow, and V. Nanda, *A topological measurement of protein compressibility*, Japan Journal of Industrial and Applied Mathematics, 32 (2015), pp. 1–17.

[57] G. Gan, C. Ma, and J. Wu, *Data Clustering: Theory, Algorithms, and Applications*, SIAM, Philadelphia, PA, 2007.

[58] R. Ghrist, *Barcodes: The persistent topology of data*, Bulletin of the American Mathematical Society, 45 (2008), pp. 61–75.

[59] ———, *Elementary Applied Topology*, Createspace, 2014. ed. 1.0.

[60] C. Giusti, R. Ghrist, and D. S. Bassett, *Two's company, three (or more) is a simplex: Algebraic-topological tools for understanding higher-order structure in neural data*, ArXiv e-prints, (2016). 1601.01704.

[61] A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi, *A survey of statistical network models*, Foundations and Trends in Machine Learning, 2 (2010), pp. 129–233.

[62] J. E. Goodman and J. O'Rourke, eds., *Handbook of Discrete and Computational Geometry*, CRC Press, Inc., Boca Raton, FL, USA, 2 ed., 2004.

[63] D. R. Grayson and M. E. Stillman, *Macaulay2, a software system for research in algebraic geometry*. Available at http://www.math.uiuc.edu/Macaulay2/.

[64] L. J. Guibas and S. Y. Oudot, *Reconstruction using witness complexes*, Discrete & Computational Geometry, 40 (2008), pp. 325–356.

[65] M. Guillemard, H. Boche, G. Kutyniok, and F. Philipp, *Signal analysis with frame theory and persistent homology*, in 10th International Conference on Sampling Theory and Applications, 2013, pp. 309–312.

[66] A. Hatcher, *Algebraic Topology*, Cambridge University Press, Cambridge, New York, 2002.

[67] Y. Hiraoka, A. Hirata, E. G. Escolar, and Y. Nishiura, *Persistent homology and many-body atomic structure for medium-range order in the glass*, Nanotechnology, 26 (2015), p. 304001.

[68] D. Horak, S. Maletić, and M. Rajković, *Persistent homology of complex networks*, Journal of Statistical Mechanics: Theory and Experiment, 2009 (2009).

[69] J. Jonsson, *Simplicial Complexes of Graphs*, Lecture Notes in Mathematics, Springer Berlin Heidelberg, 2007.

[70] M. Joswig and M. E. Pfetsch, *Computing optimal Morse matchings*, SIAM Journal on Discrete Mathematics, 20 (2006), pp. 11–25.

[71] Department of Computer Science Jyamiti research group (Prof. Tamal K. Dey) and Ohio State University Engineering, *GIComplex*, 2013. Available at http://web.cse.ohio-state.edu/~tamaldey/GIC/GICsoftware/.

[72] Department of Computer Science Jyamiti research group (Prof. Tamal K. Dey) and Ohio State University Engineering, *SimpPers*, 2014. Available at `http://web.cse.ohio-state.edu/~tamaldey/SimpPers/SimpPers-software/`.

[73] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*, no. Bd. 157 in Applied Mathematical Sciences, Springer, 2004.

[74] M. Kahle, *Random geometric complexes*, Discrete & Computational Geometry, 45 (2011), pp. 553–573.

[75] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley and Sons, Inc., New York, NY, 1990.

[76] M. Kerber and R. Sharathkumar, *Approximate čech complex in low and high dimensions*, in 24th International Symposium on Algorithms and Computation (ISAAC 2013), L. Cai, S.-W. Cheng, and T.-W. Lam, eds., Lecture Notes in Computer Science 8283, 2013, pp. 666–676. Available at `http://arxiv:1307.3272`.

[77] V. Kovacev-Nikolic, P. Bubenik, D. Nikolić, and G. Heo, *Using persistent homology and dynamical distances to analyze protein binding*, ArXiv e-prints, (2014). 1412.1394.

[78] M. Kramár, A. Goullet, L. Kondic, and K. Mischaikow, *Persistence of force networks in compressed granular media*, Physical Review E, 87 (2013), p. 042207.

[79] ———, *Quantifying force networks in particulate systems*, Physica D, 283 (2014), pp. 37–55.

[80] V. Kurlin, 2015. `http://kurlin.org/projects/persistent-skeletons.cpp`.

[81] ———, *A one-dimensional homologically persistent skeleton of an unstructured point cloud in any metric space*, Computer Graphics Forum, (2015).

[82] Los Alamos National Laboratory, *HIV Database*. `http://www.hiv.lanl.gov/content/index`.

[83] Stanford University Computer Graphics Laboratory, *The stanford 3d scanning repository*. `https://graphics.stanford.edu/data/3Dscanrep`.

[84] G. Leibon, S. Pauls, D. Rockmore, and R. Savell, *Topological structures in the equities market network*, Proceedings of the National Academy of Sciences of the United States of America, 105 (2008), pp. 20589–20594.

[85] M. Lesnick and M. Wright, *Interactive Visualization of 2-D Persistence Modules*, ArXiv e-prints, (2015). 1512.00180.

[86] M. Lesnick and M. Wright, *RIVET: the Rank Invariant Visualization and Exploration Tool*, 2016. Software available at `http://rivet.online/`.

[87] S. Maletic, Y. Zhao, and M. Rajkovic, *Persistent topological features of dynamical systems*, ArXiv e-prints, (2015). 1510.06933.

[88] C. Maria, *Algorithms and data structures in computational topology*, (2014). PhD Thesis, Université de Nice-Sophia Antipolis. Available at `http://www-sop.inria.fr/members/Clement.Maria/docs/ClementMaria_PhDdissertation.pdf`.

[89] C. Maria, J.-D. Boissonnat, M. Glisse, and M. Yvinec, *The Gudhi library: Simplicial complexes and persistent homology*, in Mathematical Software, ICMS 2014, H. Hong and C. Yap, eds., vol. 8592 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 167–174. Software available at `https://project.inria.fr/gudhi/software/`.

[90] Y. Mileyko, S. Mukherjee, and J. Harer, *Probability measures on the space of persistence diagrams*, Inverse Problems, 27 (2011), p. 124007.

[91] N. Milosavljević, D. Morozov, and P. Skraba, *Zigzag persistent homology in matrix multiplication time*, in Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry, SoCG '11, New York, NY, USA, 2011, ACM, pp. 216–225.

[92] K. Mischaikow and V. Nanda, *Morse theory for filtrations and efficient computation of persistent homology*, Discrete & Computational Geometry, 50 (2013), pp. 330–353.

[93] D. Morozov, *Dionysus*. Software available at `http://www.mrzv.org/software/dionysus/`.

[94] ———, *Persistence algorithm takes cubic time in worst case*, BioGeometry News, Dept. Comput. Sci., Duke Univ, (2005).

[95] E. Munch, K. Turner, P. Bendich, S. Mukherjee, J. Mattingly, and J. Harer, *Probabilistic Fréchet means for time varying persistence diagrams*, Electron. J. Statist., 9 (2015), pp. 1173–1204.

[96] V. Nanda, *Perseus, the persistent homology software.* Software available at http://www.sas.upenn.edu/ vnanda/perseus.

[97] ———, *Discrete morse theory for filtrations*, (2012). PhD Thesis. Rutgers, The State University of New Jersey.

[98] M. E. J. Newman, *Finding community structure in networks using the eigenvectors of matrices*, Physical Review E, 74 (2006), p. 036104.

[99] M. Nicolau, A. J. Levine, and G. Carlsson, *Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival*, Proceedings of the National Academy of Sciences of the United States of America, 108 (2011), pp. 7265–7270.

[100] S. Y. Oudot, *Persistence Theory: From Quiver Representations to Data Analysis*, vol. 209 of AMS Mathematical Surveys and Monographs, American Mathematical Society, 2015.

[101] M. Penrose, *Random Geometric Graphs*, Oxford University Press, Oxford, UK, 2003.

[102] J. A. Perea, A. Deckard, S. B. Haase, and J. Harer, *Sw1pers: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data*, BMC Bioinformatics, 16 (2015), pp. 1–12.

[103] P. Perry and V. de Silva, *Plex*, 2000–2006. Available at `http://mii.stanford.edu/research/comptop/programs/`.

[104] G. Petri, M. Scolamiero, I. Donato, and F. Vaccarino, *Topological strata of weighted complex networks*, PLoS ONE, 8 (2013), pp. 1–8.

[105] F. T. Pokorny, M. Hawasly, and S. Ramamoorthy, *Topological trajectory classification with filtrations of simplicial complexes and persistent homology*, The International Journal of Robotics Research, (2015).

[106] K. T. Poole, *Voteview*. `http://voteview.com`, 2016.

[107] F. Robin, *Morse theory for cell complexes*, Advances in Mathematics, 134 (1998), pp. 90–145.

[108] S. E. Schaeffer, *Graph clustering*, Computer Science Review, 1 (2007), pp. 27–64.

[109] D. R. Sheehy, *Linear-size approximations to the Vietoris-Rips filtration*, Discrete & Computational Geometry, 49 (2013), pp. 778–796.

[110] G. Singh, F. Mémoli, and G. Carlsson, *Topological methods for the analysis of high dimensional data sets and 3d object recognition*, in Eurographics Symposium on Point-Based Graphics, 2007, pp. 91–100.

[111] N. Singh, H. D. Couture, J. S. Marron, C. Perou, and M. Niethammer, *Topological descriptors of histology images*, in Machine Learning in Medical Imaging, Guorong Wu, Daoqiang Zhang, and Luping Zhou, eds., vol. 8679 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 231–239.

[112] O. Sporns, *Small-world connectivity, motif composition, and complexity of fractal neuronal connections*, Biosystems, 85 (2006), pp. 55–64.

[113] B. J. Stolz, H. A. Harrington, and M. A. Porter, *Persistent homology of time-dependent functional networks constructed from coupled time series*, ArXiv e-prints, (2016). 1605.00562.

[114] A. Tausz, M. Vejdemo-Johansson, and H. Adams, *JavaPlex: A research software package for persistent (co)homology*, in Proceedings of ICMS 2014, Han Hong and Chee Yap, eds., Lecture Notes in Computer Science 8592, 2014, pp. 129–136. Software available at `http://appliedtopology.github.io/javaplex/`.

[115] D. Taylor, F. Klimm, H. A. Harrington, M. Kramár, K. Mischaikow, M. A. Porter, and P. J. Mucha, *Topological data analysis of contagion maps for examining spreading processes on networks*, Nature Communications, 6 (2015). Article number 7723.

[116] The CGAL Project, *CGAL User and Reference Manual*, CGAL Editorial Board, 4.7 ed., 2015.

[117] C. M. Topaz, L. Ziegelmeier, and T. Halverson, *Topological Data Analysis of Biological Aggregation Models*, PLoS ONE, 10 (2015), pp. 1–26.

[118] K. Turner, Y. Mileyko, S. Mukherjee, and J. Harer, *Fréchet means for distributions of persistence diagrams*, Discrete & Computational Geometry, 52 (2014), pp. 44–70.

[119] R. Vasudevan, A. Ames, and R. Bajcsy, *Persistent homology for automatic determination of human-data based cost of bipedal walking*, Nonlinear Analysis: Hybrid Systems, 7 (2013), pp. 101–115. IFAC World Congress 2011.

[120] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, *Novel type of phase transition in a system of self-driven particles*, Physical Review Letters, 75 (1995), pp. 1226–1229.

[121] L. Vietoris, *Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen*, Mathematische Annalen, 97 (1927), pp. 454–472.

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

[123] B. Wang and G.-W. Wei, *Object-oriented persistent homology*, Journal of Computational Physics, 305 (2016), pp. 276–299.

[124] A. S Waugh, L. Pei, J. H Fowler, P. J Mucha, and M. A Porter, *Party polarization in congress: A network science approach*, arXiv:0907.3509, (2012). data available at `http://figshare.com/articles/Roll_Call_Votes_United_States_House_and_Senate/1590036`.

[125] K. Xia, X. Feng, Y. Tong, and G. W. Wei, *Persistent homology for the quantitative prediction of fullerene stability*, Journal of Computational Chemistry, 36 (2015), pp. 408–422.

[126] K. Xia and G.-W. Wei, *Persistent homology analysis of protein structure, flexibility, and folding*, International journal for numerical methods in biomedical engineering, 30 (2014), pp. 814–844.

[127] A. Zomorodian, *Topology for Computing*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2009.

[128] ———, *Technical section: Fast construction of the Vietoris-Rips complex*, Comput. Graph., 34 (2010), pp. 263–271.

[129] ———, *The tidy set: A minimal simplicial set for computing homology of clique complexes*, 2010, pp. 257–266. SCG10.

[130] ———, *Topological data analysis*, in Advances in Applied and Computational Topology: American Mathematical Society Short Course on Computational Topology, January 4-5, 2011, New Orleans, Louisiana, Afra Zomorodian, ed., vol. 70 of Proceedings of symposia in applied mathematics, American Mathematical Society, 2012, pp. 1–39.

[131] A. Zomorodian and G. Carlsson, *Computing persistent homology*, Discrete Comput. Geom., 33 (2005), pp. 249–274.