Chapter 4

Persistence

In the previous chapter, we have studied the homology of fixed simplicial complexes. In this chapter, we will look at simplicial complexes that grow over time. Let us start with a small example. Consider the following process of building up a triangle abc. At time t_1, we add the vertices a and b together with the edge ab. This gives birth to a single connected component. At time t_2 we add the vertex c, giving birth to a second connected component. At time t_3 we add the edge ac, connecting the two components. We can interpret this as the younger of the components being absorbed by the older component. In more crude language, we say that the younger component dies. At time t_4 we add the final edge bc, which gives birth to a hole, that is, an element of the homology group H_1. Finally, at time t_5 we add the interior of the triangle, killing the hole born at t_4. We can summarize this process as follows: we have a connected component that was born at t_1 and survived the entire process, and a connected component that was born at t_2 that died again at t_3. Finally, we have a hole born at t_4 dying at t_5. Capturing this information of holes with their birth and death is the goal of persistent homology.

Persistent homology can be applied to data analysis by defining (in a way that we will see soon) a process to build up a simplicial complex from point cloud data and computing the birth and death times of holes. Subtracting the birth time from the death time we get the lifespan of a hole; the underlying idea is that holes with a short lifetime are a byproduct of the process (noise), whereas holes with a long lifespan convey information about the shape of the underlying data.

4.1 Filtrations

We start by a mathematical formulation of the process of growing a simplicial complex or, more general, a topological space. A *filtration* is a nested sequence of subspaces

$$\mathcal{F} : X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots \subseteq X_n = X.$$

For each $i \leqslant j$, we have the inclusion map $\iota_{i,j} : X_i \hookrightarrow X_j$. Given these functions ι, we get induced maps in homology: $h_p^{i,j} = \iota_* : H_p(X_i) \to H_p(X_j)$. Filtrations are a very

general object that appear naturally in many settings. Let us look at some important examples of filtrations.

- Given a function $f : X \to \mathbb{R}$, we can define the (uncountably infinite) *sublevel set filtration* $X_a = f^{-1}(-\infty, a]$.
- A *simplicial filtration* is a nested sequence of subcomplexes

  $$\mathcal{F} : K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K.$$

  We call a simplicial filtration *simplex-wise*, if $K_i \setminus K_{i-1}$ is a single simplex (or empty).
- We call a function $f : K \to \mathbb{R}$ *simplex-wise monotone* if for every $\sigma \subseteq \tau$ we have $f(\sigma) \leqslant f(\tau)$. A simplex-wise monotone function guarantees us that the sublevel set filtration by f gives a proper simplicial filtration. Note that it does not necessarily guarantee us that the sublevel set filtration is simplex-wise (e.g., consider a function f that is not injective).
- We can also define a simplicial filtration by ordering our vertices $v_0, v_1, \ldots, v_n$. Then, let K_i be the simplicial complex induced by the vertices $v_0, \ldots, v_i$. We call the simplices $K_i \setminus K_{i-1}$ added when adding v_i the *lower star* of v_i. Thus, this type of filtration is also called the *lower star filtration*.
- Given some data points in $\mathbb{R}^d$, we can define a filtration based on our intuition of *growing balls*: We consider the nerve of all balls $B(p, r)$; with growing r we get more and more faces in this nerve. We will later formalize this into the so-called *Čech complex*.

4.2 Persistent Homology

As we have seen, from a filtration $X_0 \subseteq X_1 \subseteq \ldots \subseteq X_n$ we get a sequence of homology groups with homomorphisms between them:

$$H_p(\mathcal{F}) : H_p(X_0) \to H_p(X_1) \to H_p(X_2) \to \ldots \to H_p(X_n).$$

Such an object is called a *persistence module*. Given a persistence module, we can now define groups that capture all the holes that are alive during a certain period.

Definition 4.1. *The* p-th persistent homology group $H_p^{i,j}$ *is defined by*

$$H_p^{i,j} := \operatorname{im} h_p^{i,j} = Z_p(K_i)/(B_p(K_j) \cap Z_p(K_i)).$$

This definition characterizes the cycles that that are present already in K_i and that are not boundaries even in K_j.

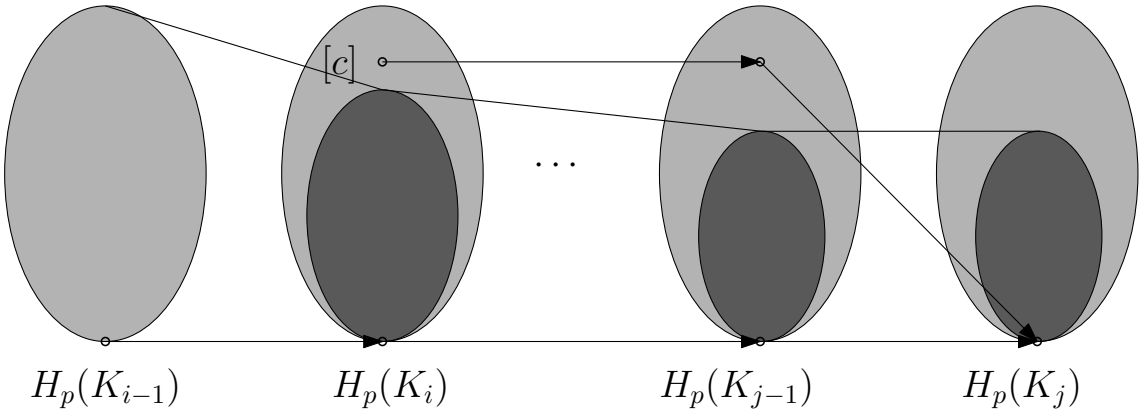

Figure 4.1: *An illustration of a class* $[c]$ *being born at* K_i *and dying entering* K_j.

Definition 4.2. *The* p-th persistent Betti numbers $\beta_p^{i,j}$ *are the dimensions of the* p*-th persistent homology groups:* $\beta_p^{i,j} = \dim H_p^{i,j}$.

Exercise 4.3. *Let* $p \geqslant 1$. *For every* $n \geqslant 1$, *construct a filtration* $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_n$ *such that*

- $H_p(X_k) \neq 0$ *for all* $k \in \{1, \ldots, n\}$ *and*
- $H_p^{i,j} = 0$ *for all* $i < j$.

We say that a p-homology class $[c]$ (a p-hole) is *born at* K_i if $[c] \in H_p(K_i)$ but $[c] \notin H_p^{i-1,i}$. Similarly, $[c]$ *dies entering* K_j, if $[c] \neq 0$ in $H_p(K_{j-1})$ but $h_p^{j-1,j}([c]) = 0$.

It is not always obvious which homology class dies. Consider the following filtration: X_1 consists of two points a and b, and in X_2 the two points are connected by an edge. Let us look at H_0, that is, the connected components. We have that $H_0(X_1) \simeq \mathbb{Z}_2^2$, with the natural basis $\{[a], [b]\}$. On the other hand, in X_2 there is only a single connected component, and $[a] = [b]$. So a homology class is dying, but both our basis elements $[a]$ and $[b]$ survive. What is happening?

It turns out that we were not careful with our choice of basis: $H_0(X_1)$ can also be viewed as being generated by $[a]$ and $[a + b]$, and the class $[a + b]$ indeed dies going into X_2. In general, if two homology classes merge, they both do not die, but their sum does. There is a consistent choice of basis which allows us to only look at persistent homology in terms of basis elements, but we do not go into this at this point.

If we have a simplex-wise filtration, we can circumvent the above issue by sorting homology classes by the time where they were born (recall the solution to Exercise 3.33 to see why this gives a total order). When two classes merge, we just say the "younger one" dies. This can be seen as adapting the considered basis along the way.

Persistence pairings are another way around this issue. We add some final complex K_{n+1} which has trivial homology (i.e., by adding all simplices that are not yet present). Then, we aim to figure out how many holes get born at K_i and die entering K_j. For this,

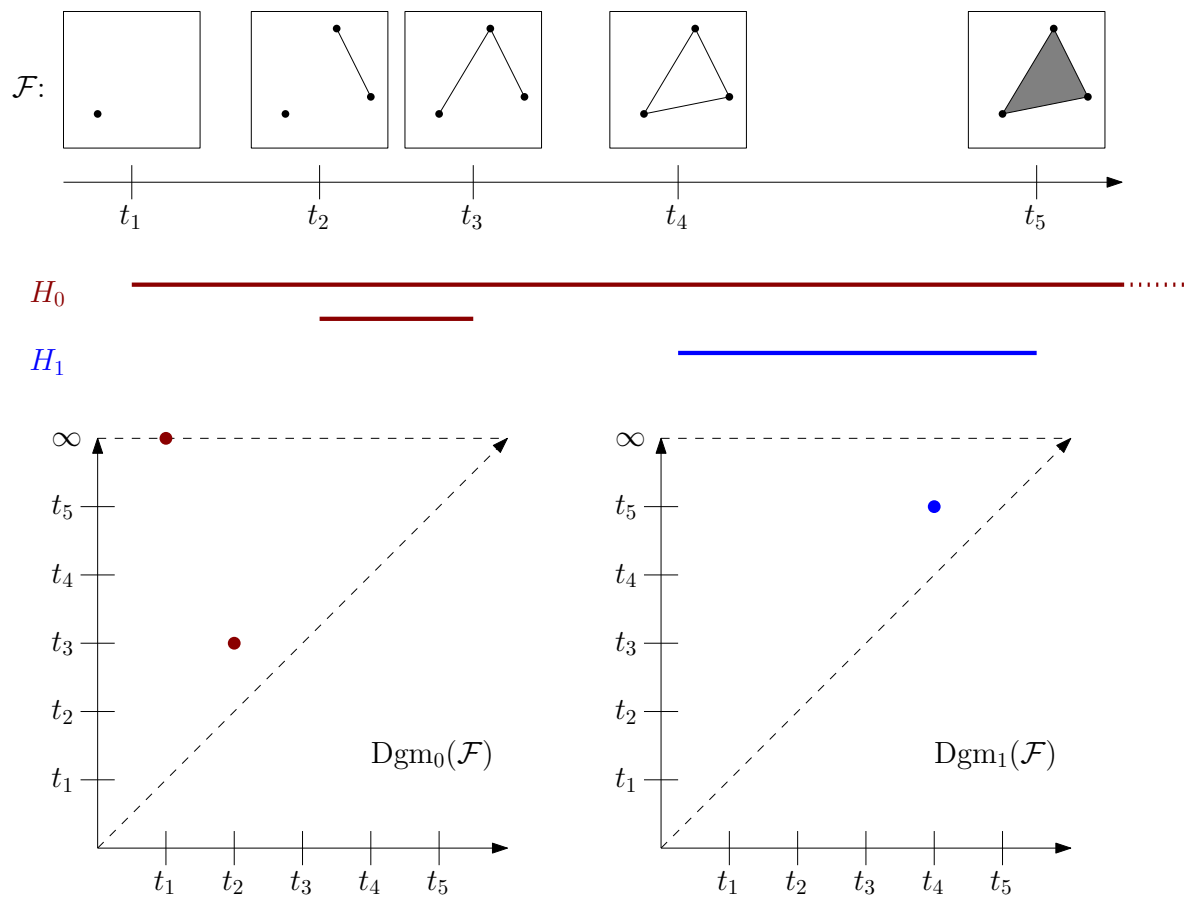

Figure 4.2: *An example of a filtration with the corresponding barcodes and persistence diagrams.*

we define

$$\mu_p^{i,j} := (\beta_p^{i,j-1} - \beta_p^{i,j}) - (\beta_p^{i-1,j-1} - \beta_p^{i-1,j}), \text{ for } i < j \leqslant n+1.$$

Here, the content of the left parenthesis denotes the number of holes born at or before K_i, which die entering K_j. Conversely, the right parenthesis denotes the number of holes born strictly before K_i, and die entering K_j. Thus, subtracting the two, gives the number of holes born exactly at K_i and die entering K_j. Note that this conveys the information that we are interested in, but does not require choosing any basis.

The *persistence diagram* $\mathrm{Dgm}_p(\mathcal{F})$ is a birth-death diagram which contains a point for every pair i, j for which $\mu_p^{i,j} > 0$. If we give each K_i a timestamp a_i, the point is drawn at the coordinates (a_i, a_j). We give each point multiplicity $\mu_p^{i,j}$. On the diagonal we add points with infinite multiplicity, for some technical reasons that will become apparent later. We can also represent the same information by *barcodes*: For every i, j, we draw $\mu_p^{i,j}$ intervals $[a_i, a_j]$. This is then called the p-th persistence barcode.

Exercise 4.4. *Consider the simplex-wise filtration induced by the order* $\sigma_1, \ldots, \sigma_N$ *on*

the simplices of a complex K*. When does the order*

$$\sigma_1, \ldots, \sigma_{k-1}, \sigma_{k+1}, \sigma_k, \sigma_{k+2}, \ldots, \sigma_N$$

induce a simplex-wise filtration too? When it does, describe the relation between the corresponding persistence diagrams.

Exercise 4.5. *Give two filtrations* $X_1 \subseteq \ldots \subseteq X_n$ *and* $Y_1 \subseteq \ldots \subseteq Y_n$ *that have the same persistence diagrams but for which for any* $i \in \{1, \ldots, n\}$*,* X_i *is not homotopy-equivalent to* Y_i*.*

4.3 Algorithms for Persistent Homology

So far we have considered homology and persistent homology only on a mathematical level. However, for practical applications we are interested in actually computing homological information. In this section we discuss how we can compute persistence pairings given simplicial filtrations. This will of course also allow us to compute persistence diagrams and persistence barcodes.

4.3.1 Persistence Pairing Algorithm

The first algorithm we consider is the so-called persistence pairing algorithm. It only works on simplex-wise filtrations, we thus restrict our attention to such filtrations. In any time step j, we add a single simplex $\sigma_j := K_j \setminus K_{j-1}$. Let p be its dimension. There are only two things that can happen to the homology when adding σ_j: Either, a new non-boundary p-cycle c (i.e., a hole) is born, or a $(p-1)$-cycle becomes a boundary (i.e., a hole dies). In the first case we say that σ_j is a *creator*. Otherwise, we say that σ_j is a *destructor*. The fact that in every step exactly one of the two events happens is a consequence of the Euler characteristic, as discussed in Exercise 3.33.

When a new simplex σ_j destroys a hole, this corresponds to an interval of the persistence barcode ending. The beginning of that interval is at the time step when this hole was born, which corresponds to a unique simplex (recall, we are considering simplex-wise filtrations only). This unique simplex must be a creator, since when it was inserted a hole was born. The idea of the persistence pairing algorithm is to form pairings between destructors and creators. To do this, the algorithm assumes the newly added simplex σ_j to be a destructor, and tries to find the corresponding unpaired creator using a simple heuristic. If no such creator can be found by the procedure, we know that σ_j must actually be a creator itself.

The heuristic is quite simple to describe. We have to look for an unpaired creator only within a cycle c that becomes a boundary due to the insertion of σ_j. Among this cycle c, we wish to pair σ_j with the youngest unpaired creator. Any such cycle c must be homologous to $\delta\sigma_j$, which is the simplest candidate for such a cycle c. This is thus where the search begins. We first try to pair σ_j with the youngest $(p-1)$-simplex ρ of

its boundary. If ρ is unpaired, we pair it to σ_j and we are done. Otherwise, ρ is already paired with some (p-simplex) τ. In this case we replace c by $c + \delta\tau$. This is now a new candidate cycle, in which we can try pairing σ_j to the youngest simplex. We repeat this process until we found an unpaired creator we can pair σ_j to, or until we cannot continue because $c = 0$. In this case we label σ_j as a new creator. At the end of the algorithm (after processing all steps of the filtration), all remaining unpaired creators correspond to holes present at the last step of the filtration, and we pair them with the element ∞.

We refrain from giving a complete proof of this algorithm's correctness. Such a proof can be found in [1], however the algorithm presented there is slightly more complex and more efficient. We would only like to note that when we label a simplex a creator that this is correct to do so: If we reach $c = 0$ we know that the boundary of σ_j is homologous to 0 (we obtained 0 by adding boundaries to $\delta\sigma_j$). Thus, σ_j cannot be a destructor. We can thus safely label σ_j as a new creator.

We summarize this algorithm in the following pseudocode:

Algorithm 1: The persistence pairing algorithm.

```
Input: A simplex-wise filtration of K given by an order of simplices σ_1, ..., σ_N
for j = 1 to n do
    c := δσ_j;
    while c ≠ 0 do
        i := largest integer such that σ_i ∈ c and σ_i is creator;
        ρ := σ_i;
        if ρ is unpaired then
            Label σ_j as destructor and pair ρ and σ_j;
            c := 0
        else
            τ := simplex ρ is paired to;
            c := c + δτ;
        end
    end
    if σ_j has not been labelled a destructor then
        Label σ_j a constructor;
    end
end
Pair all unpaired constructors with ∞;
```

Exercise 4.6. *Let G be a weighted connected graph, where all edge weights are pairwise distinct. Consider a filtration that first inserts all vertices (in some arbitrary order) and then inserts the edges one by one, ordered by increasing weight. What is the set of destructors?*

4.3.2 Matrix Reduction Algorithm

In practice, a different algorithm is used, the *Matrix Reduction Algorithm*. This algorithm implements the same intuition as the persistence pairing algorithm. It has a few advantages: First off, it is more efficient (it avoids the need to add the same boundaries multiple times, similarly to the version of the persistence pairing algorithm provided in [1]). Second, it is phrased in the language of matrices, which allows us to implement it more efficiently using matrix-multiplication techniques. Lastly, the way we describe it in the following it also works with non-simplex-wise filtrations.

In the matrix reduction algorithm, we first find a total order on our simplices. If the input filtration is simplex-wise, this is just the insertion order. Otherwise, we order the simplices primarily by insertion order, and within each set of simultaneously added simplices, we order the simplices by increasing dimension, and then lexicographically. Then, we construct an $N \times N$ matrix, which is the so-called *boundary matrix*. Each row and column is labelled by a simplex, ordered by the order we defined above. We then insert a 1 at row σ and column τ, if σ is part of the boundary of τ.

We now modify this boundary matrix to obtain the *reduced boundary matrix*, from which the persistence pairings can then be read off. We process the columns from left to right. For each column c, we look at the lowest 1 in the column. We call this 1 the *pivot element* of the column. If there is a column $c' < c$ to the left that also has a pivot element in the same row, we add c' to c (in $\mathbb{Z}_2$). This is repeated until no such column $c' < c$ exists.

After processing all the columns, the matrix is in a reduced form: For every row, there is at most one column whose lowest 1 (its pivot element) lies in that row. From this we can now read the persistence pairings: Empty columns correspond to creators (births). To find the death of a creator, look at its corresponding row, and search for a column that has a pivot element in that row. This column is the destructor corresponding to the creator. If there is no such column, this creator never dies, i.e., is unpaired or paired with ∞.

We again summarize this algorithm in the pseudocode below. Let us now analyze at the runtime of this algorithm. For each column ($O(N)$), we might have to add $O(N)$ times a column, and each addition takes $O(N)$. So, by this very rough analysis we have a runtime of $O(N^3)$. But, since the reduction process is very similar to Gaussian elimination, we can actually perform the reduction using techniques that yield a runtime of $O(N^\omega)$, where ω is the matrix-multiplication exponent. However, in practice this is not very useful since efficient matrix-multiplication algorithms are very complex and have large constants, while the naive implementation runs in essentially $O(N)$ time anyways since the involved matrices are so sparse.

Exercise 4.7. *Consider the following simplicial complex, and the simplex-wise filtration which first inserts the vertices in the order* a, b, c, d, e*, and the rest of the simplices as specified by the numbering in Figure 4.3.*

Execute both the persistence pairing algorithm and matrix reduction algorithm on this filtration. What are the similarities and differences in the algorithms? To

Algorithm 2: The matrix reduction algorithm.

Input: A filtration of K.
Find an ordering $\sigma_1, \ldots, \sigma_N$ corresponding to a simplex-wise filtration of K consistent with the given filtration;
$M := 0^{N \times N}$;
for $1 \leqslant i, j \leqslant N$ **do**
  if $\sigma_i \in \delta\sigma_j$ **then**
    $M_{ij} := 1$;
  end
end
for $j = 1$ *to* n **do**
  $\ell := \max(\{-1\} \cup \{i \mid M_{ij} = 1\})$;
  while $\ell \neq -1$ *and* $\exists j' < j$ *such that* $\ell = \max(\{-1\} \cup \{i \mid M_{ij'} = 1\})$ **do**
    $M_{\cdot j} := M_{\cdot j} + M_{\cdot j'}$;
    $\ell := \max(\{-1\} \cup \{i \mid M_{ij} = 1\})$;
  end
end
for $j = 1$ *to* n **do**
  if $M_{\cdot j} = 0^N$ **then**
    Label σ_j a constructor;
    for $j' = 1$ *to* n **do**
      if $j = \max(\{-1\} \cup \{i \mid M_{ij'} = 1\})$ **then**
        Pair σ_j to $\sigma_{j'}$;
        Label $\sigma_{j'}$ a destructor;
      end
    end
  end
end
Pair all unpaired constructors with ∞;

better see what happens, label the columns in the matrix by the sum of columns they currently represent.

Represent the results you obtained by a persistence diagram, and also by the persistence barcodes.

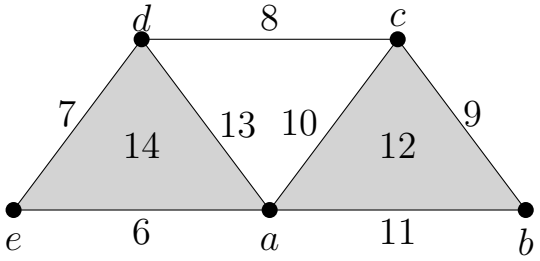

Figure 4.3: *The filtration for Exercise 4.7.*

Exercise 4.8. *A Union-Find data structure is a data structure that maintains disjoint sets dynamically. Given a ground set* X*, such a data structure maintains a family* S *of disjoint subsets of* X*, where each subset is represented by the* smallest *element contained in it. It supports three operations:* `MakeSet`(x) *creates a new set* $\{x\}$*.* `FindSet`(x) *returns the representative (minimum) of the set in* S *which contains* x *(or "no" if* x *is not contained in any set).* `Union`(x, y) *merges the sets containing* x *and* y *into a single one. All of these operations can be implemented in amortized* $\Theta(\alpha(n))$ *time, where* α *is the extremely slowly growing inverse Ackermann function and can be considered a constant for any real world application.*

Consider a simplicial complex K *with its vertices ordered* $v_0, \dots, v_n$*, and consider its lower star filtration. Find an algorithm to compute the* 0*-dimensional persistence diagram (i.e., the persistence pairings) of* K *which makes use of a Union-Find data structure. How many Union-Find operations do you need to perform?*

Questions

13. *What is a filtration?* State the definition and describe different ways how filtrations appear in topology and data analysis.
14. *What is persistent homology?* State the formal definitions and give examples.
15. *How can persistent homology be computed?* Discuss the two algorithms described in Section 4.3.

References

[1] Tamal Krishna Dey and Yusu Wang, *Computational topology for data analysis*, Cambridge University Press, 2022.