Topological Data Analysis

Lecture Notes

Master M2 — 2025–2026

Yaë Ulrich Gaba

"Topology is precisely the mathematical discipline that allows the passage from local to global."

— *René Thom*

March 25, 2026

Contents

Preface

Topological Data Analysis (TDA) is a relatively young field, born in the early 2000s at the confluence of algebraic topology, computational geometry, and data science. Its fundamental goal is to extract *qualitative* and *structural* information from complex datasets by exploiting topological invariants that are robust to noise and continuous deformations.

While classical statistical methods rely on parametric models or distributional assumptions, TDA adopts a radically different perspective: it seeks to characterize the *shape* of data. This approach has proven particularly fruitful in domains as diverse as molecular biology, neuroscience, materials science, quantitative finance, and deep learning.

Course Objectives

This course is designed for Master's and PhD students with a background in:

- **Algebra and topology**: groups, rings, modules; topological spaces, compactness, connectedness.
- **Analysis and probability**: metric spaces, measure theory, convergence.
- **Algorithms and programming**: complexity, data structures, Python.

Upon completion, the student will be able to:

1. Construct and manipulate the main simplicial complexes (Vietoris–Rips, Čech, Alpha) associated with point clouds.
2. Compute simplicial homology and persistent homology, both theoretically and algorithmically.
3. Interpret persistence diagrams and barcodes, and assess their stability.
4. Apply the Mapper algorithm for exploratory visualization.
5. Integrate topological descriptors into machine learning and deep learning pipelines.
6. Conduct a research project applying TDA to real-world data.

Course Organization

The course is structured into eleven chapters, grouped into four parts:

Part I — Foundations (Chapters 1–3) Motivations, simplicial complexes, homology, and introduction to persistent homology.

Part II — Persistence Theory (Chapters 4–6) Persistence diagrams, stability theorems, and construction of filtered complexes.

Part III — Methods and Algorithms (Chapters 7–8) Mapper algorithm, distances between persistence diagrams (Wasserstein, bottleneck).

Part IV — TDA and Learning (Chapters 9–11) Integration with machine learning, concrete applications, and connections to deep learning.

Each chapter contains rigorous definitions, theorems with proofs (or proof sketches), detailed examples, Python implementations using `gudhi`, `ripser`, and `giotto-tda`, as well as exercises of increasing difficulty.

Notation and Conventions

Symbol	Meaning
$\mathbb{R}^d$	Euclidean space of dimension d
$\mathbb{N}, \mathbb{Z}, \mathbb{Q}$	Natural numbers, integers, rationals
$\mathbb{F}$	Finite field (often $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$)
K	Abstract simplicial complex
$\sigma = [v_0, \ldots, v_p]$	p-simplex
$C_p(K)$	Group of p-chains of K
∂_p	Boundary operator in dimension p
$H_p(K)$	p-th homology group of K
β_p	p-th Betti number ($= \dim H_p$)
$\mathrm{VR}(X, r)$	Vietoris–Rips complex at parameter r
$\check{C}(X, r)$	Čech complex at parameter r
Dgm_p	Persistence diagram in dimension p
d_B, W_p	Bottleneck distance, p-Wasserstein distance
$\|\cdot\|$	Norm (Euclidean by default)
$\|S\|$	Cardinality of set S
$\langle \cdot, \cdot \rangle$	Inner product

Software Environment

The implementations in this course use **Python 3.10+** with the following libraries:

- **GUDHI** (`https://gudhi.inria.fr`): C++/Python library developed at INRIA, offering a wide range of simplicial structures, persistence algorithms, and statistical tools.
- **Ripser** (`https://ripser.scikit-tda.org`): ultra-fast implementation of persistent homology for Vietoris–Rips complexes.

- **giotto-tda** (https://giotto-ai.github.io/gtda-docs): machine-learning oriented library, scikit-learn compatible.
- **scikit-learn**, **NumPy**, **SciPy**, **Matplotlib**: preprocessing, numerical computation, and visualization.
- **PyTorch**: for deep learning models integrating topological layers.

Recommended Installation

We recommend using a virtual environment:

```
python -m venv tda-env
source tda-env/bin/activate
pip install gudhi ripser giotto-tda scikit-learn torch matplotlib
```

Historical Milestones

Topological Data Analysis draws from several foundational contributions:

- **1990s**: Edelsbrunner, Letscher, and Zomorodian introduce the first notions of persistent homology.
- **2002**: Zomorodian and Carlsson formalize persistent homology and its computation via graded module decomposition.
- **2005**: Cohen-Steiner, Edelsbrunner, and Harer prove the fundamental stability theorem for persistence diagrams.
- **2007**: Singh, Mémoli, and Carlsson introduce the Mapper algorithm.
- **2009**: Chazal, Cohen-Steiner, Glisse, Guibas, and Oudot extend stability results to Wasserstein distances.
- **2015–present**: Growing integration of TDA with machine learning and deep learning.

Acknowledgments

This course owes much to the pioneering work of Gunnar Carlsson, Herbert Edelsbrunner, Frédéric Chazal, Steve Oudot, and the entire TDA community. We also thank the developers of GUDHI, Ripser, and giotto-tda for the quality of their tools.

Intuition

TDA rests on a simple but profound idea: *the shape of data carries information.* The tools of algebraic topology allow us to quantify this shape in a rigorous, stable, and computable manner. This course will give you the keys to exploit this idea in your own research.

The authors, March 2026

Chapter 1

Motivations and Overview

Imagine a point cloud in a high-dimensional space. Classical statistics will give you its mean, its variance, its principal components. But they will not tell you whether there is a "hole" in the middle, a loop, or two clusters separated by a void. These *topological* features — the global shape of the data — are often the most informative. This is the founding intuition of *topological data analysis* (TDA), a field born in the early 2000s from the work of Gunnar Carlsson, Herbert Edelsbrunner, and others, at the frontier of algebraic topology, computational geometry, and data science.

Intuition

Why do we need topology to analyze data? Because the *shape* of data — its holes, connected components, and cavities — carries information that classical statistics cannot capture.

1.1 The Fundamental Problem

Consider a point cloud $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$. This cloud is often a noisy sample from an underlying topological space $\mathcal{M}$ (a manifold, curve, surface, etc.). The central problem of TDA is:

Fundamental Question

How can we recover the topological properties of $\mathcal{M}$ from the finite sample X?

Example 1.1. Let $\mathcal{M} = S^1$ be the unit circle in $\mathbb{R}^2$. A sample of 100 noisy points on this circle forms a ring. Topologically, S^1 has one connected component ($\beta_0 = 1$) and one hole ($\beta_1 = 1$). TDA allows us to recover these invariants from the point cloud.

1.2 Limitations of Classical Approaches

Classical statistical and learning methods have several limitations when dealing with data of complex shape:

1. **Coordinate dependence**: descriptive statistics (mean, variance) depend on the chosen coordinate system. Topology is invariant under homeomorphism.

2. **Loss of structural information**: clustering identifies connected components but ignores holes and cavities.

3. **Curse of dimensionality**: in high dimensions, Euclidean distances become uninformative, but topological invariants remain meaningful.

4. **Lack of multi-scale analysis**: fixing a distance threshold is arbitrary; persistence explores *all* scales simultaneously.

1.3 The Philosophy of TDA

TDA rests on three guiding principles:

Definition 1.2 (Principles of TDA)**.**

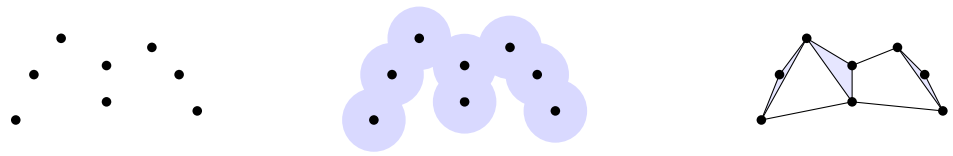
1. **Deformation invariance**: topological properties are preserved under continuous deformation (homeomorphism, homotopy equivalence).

2. **Multi-scale approach**: rather than fixing a resolution parameter, we study the evolution of topology across all resolution scales.

3. **Stability**: small perturbations of the data produce small perturbations of the topological descriptors.

1.4 Visual Illustration: From Point Cloud to Complex

The fundamental construction of TDA consists of progressively thickening a point cloud into a geometric object whose topology can be computed.



(a) Point cloud X (b) Balls $B(x_i, r)$ (c) Simplicial complex

1.5 Betti Numbers: A First Intuition

Betti numbers are the simplest topological invariants. Intuitively:

Betti Numbers

$$\beta_0 = \text{number of connected components} \tag{1.1}$$
$$\beta_1 = \text{number of holes (loops)} \tag{1.2}$$
$$\beta_2 = \text{number of cavities (enclosed voids)} \tag{1.3}$$
$$\beta_p = \text{number of “holes” of dimension } p \tag{1.4}$$

Example 1.3.

- A filled disk: $\beta_0 = 1$, $\beta_1 = 0$.
- A circle S^1: $\beta_0 = 1$, $\beta_1 = 1$.
- A torus T^2: $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = 1$.
- A sphere S^2: $\beta_0 = 1$, $\beta_1 = 0$, $\beta_2 = 1$.

1.6 A Typical TDA Pipeline

The general topological analysis process follows these steps:

Standard TDA Pipeline

1. **Data**: point cloud $X \subset \mathbb{R}^d$ or distance matrix.
2. **Filtration**: build a nested family of simplicial complexes $K_{r_1} \subseteq K_{r_2} \subseteq \cdots$.
3. **Persistent homology**: compute the birth and death of each topological feature.
4. **Representation**: encode the result as a persistence diagram or barcode.
5. **Vectorization**: transform into a feature vector for learning.
6. **Analysis/Prediction**: apply statistical or ML methods.

First Example with Ripser

```python
import numpy as np
from ripser import ripser
import matplotlib.pyplot as plt
from persim import plot_diagrams

# Generate a noisy circle
theta = np.linspace(0, 2*np.pi, 100, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])
X += 0.1 * np.random.randn(*X.shape)

# Compute persistent homology
result = ripser(X, maxdim=1)
diagrams = result['dgms']

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].scatter(X[:, 0], X[:, 1], s=10)
axes[0].set_title("Point Cloud")
axes[0].set_aspect('equal')
plot_diagrams(diagrams, ax=axes[1])
axes[1].set_title("Persistence Diagram")
plt.tight_layout()
```

```
plt.savefig("first_tda_example.pdf")
```

1.7 Overview of Applications

TDA has demonstrated its usefulness across many domains:

1.7.1 Biology and Medicine

- **Protein shape analysis**: persistent Betti numbers characterize cavities and tunnels in molecular structures.
- **Genomics**: Mapper reveals subgroups in breast cancer gene expression data.
- **Medical imaging**: tumor detection via persistent homology of images.

1.7.2 Neuroscience

- **Brain connectome**: topological analysis of neural networks.
- **Neural codes**: persistent homology reveals the structure of spatial representations (place cells).

1.7.3 Materials Science

- **Amorphous materials**: characterization of medium-range order.
- **Porous materials**: porosity quantification via Betti numbers.

1.7.4 Time Series and Signals

- **Anomaly detection**: topological changes signal abnormal regimes.
- **Takens reconstruction**: embedding a time series and analyzing the resulting shape.

1.8 Mathematical Formalism: A First Glimpse

We conclude this introductory chapter with a formal overview of the main objects, which will be developed in subsequent chapters.

Definition 1.4 (Filtration)**.** A *filtration* is a family of topological subspaces (or simplicial complexes) $\{K_r\}_{r\in\mathbb{R}}$ such that:

$$r \leq s \implies K_r \subseteq K_s.$$

Definition 1.5 (Persistence Module)**.** A *persistence module* is a functor $\mathbf{V} : (\mathbb{R}, \leq) \to \mathbf{Vec}_{\mathbb{F}}$ that assigns to each $r \in \mathbb{R}$ a vector space V_r and to each pair $r \leq s$ a linear map $v_r^s : V_r \to V_s$.

Theorem 1.6 (Decomposition of Persistence Modules)**.** *Every* pointwise finite-dimensional *(p.f.d.) persistence module decomposes uniquely (up to isomorphism) as a direct sum of interval modules:*

$$\mathbf{V} \cong \bigoplus_{j \in J} \mathbb{I}[b_j, d_j)$$

where $\mathbb{I}[b, d)$ *is the module that equals* $\mathbb{F}$ *on* $[b, d)$ *and* 0 *elsewhere, with identity maps on the interval.*

1.9 Why Topology and Not Just Geometry?

Remark 1.7. Geometry studies metric properties (distances, curvatures, angles) that depend on the embedding. Topology studies properties invariant under continuous deformation: it is coarser but more robust. TDA exploits this robustness to extract essential information.

For instance, a coffee mug and a donut are topologically identical (both have one hole), even though they are geometrically very different.

1.10 Exercises

Exercise 1.1. Generate a cloud of 200 points sampled uniformly on a torus in $\mathbb{R}^3$ (standard parametrization with $R = 2, r = 1$). Add Gaussian noise with variance $\sigma^2 = 0.05$. Compute persistent homology with `ripser` and verify that you recover $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = 1$.

Exercise 1.2. Explain why Betti numbers alone are not sufficient to distinguish all topological spaces. Give an example of two spaces with the same Betti numbers that are not homeomorphic.

Exercise 1.3. Implement the naive computation of the pairwise distance matrix for a cloud of n points in $\mathbb{R}^d$. What is the complexity? How is this matrix used in the construction of a Vietoris–Rips complex?

Exercise 1.4. Find a recent paper (post-2020) using TDA in an application domain of your choice. Summarize the methodology in one page: what data, which complex, which topological descriptors, what main result?

Chapter 2

Simplicial Complexes and Homology

How do you count the holes in a shape? The question sounds almost childish, yet it took some of the greatest minds in mathematics—Euler, Riemann, Poincaré, Emmy Noether—to arrive at a satisfying answer. The key idea, developed by Henri Poincaré in his landmark *Analysis Situs* (1895), is to decompose a space into simple building blocks—points, edges, triangles, tetrahedra—and then use algebra to detect which combinations of these blocks form "cycles" that do not bound anything. Each such cycle witnesses a hole.

This is the programme of *simplicial homology*: build a combinatorial skeleton of your space (a simplicial complex), define boundary operators that encode how pieces fit together, and compute the quotient "cycles modulo boundaries." The resulting homology groups are topological invariants—they do not depend on the particular triangulation chosen. For topological data analysis, this machinery is indispensable: it is the engine that detects connected components, loops, cavities, and higher-dimensional voids in point cloud data.

Intuition

Simplicial complexes are the building blocks of TDA: they generalize graphs to higher dimensions. Homology is the algebraic tool that allows us to count the holes of each dimension in these complexes.

2.1 Simplices

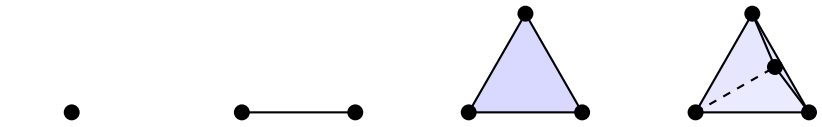
Definition 2.1 (Geometric Simplex)**.** A *geometric p-simplex* is the convex hull of $p+1$ *affinely independent* points $v_0, \ldots, v_p \in \mathbb{R}^d$:

$$\sigma = [v_0, \ldots, v_p] = \left\{ \sum_{i=0}^{p} \lambda_i v_i \;\middle|\; \lambda_i \geq 0,\ \sum_{i=0}^{p} \lambda_i = 1 \right\}.$$

The integer p is the *dimension* of σ.

Example 2.2.

- 0-simplex: a point (vertex).
- 1-simplex: a segment (edge).
- 2-simplex: a triangle (filled).
- 3-simplex: a tetrahedron (filled).



Definition 2.3 (Face)**.** A *face* of a p-simplex $\sigma = [v_0, \ldots, v_p]$ is any simplex obtained by removing one or more vertices. The *i-th opposite face* is:

$$d_i\sigma = [v_0, \ldots, \hat{v}_i, \ldots, v_p]$$

where $\hat{v}_i$ indicates omission of vertex v_i.

2.2 Abstract Simplicial Complexes

Definition 2.4 (Abstract Simplicial Complex)**.** An *abstract simplicial complex* is a pair $K = (V, \Sigma)$ where V is a finite set of vertices and $\Sigma \subseteq 2^V$ is a collection of non-empty subsets of V such that:

1. For every $v \in V$, $\{v\} \in \Sigma$.
2. If $\sigma \in \Sigma$ and $\tau \subseteq \sigma$ with $\tau \neq \emptyset$, then $\tau \in \Sigma$.

Elements of Σ are called *simplices* and condition (2) is the *closure under faces* property.

Definition 2.5 (Dimension)**.** The *dimension* of a simplex σ is $\dim \sigma = |\sigma| - 1$. The *dimension* of the complex is $\dim K = \max_{\sigma \in \Sigma} \dim \sigma$.

Example 2.6. Consider $V = \{a, b, c, d\}$ and:

$$\Sigma = \big\{\{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{b, c\}, \{a, c\}, \{c, d\}, \{a, b, c\}\big\}.$$

This is a simplicial complex of dimension 2. It contains one 2-simplex $\{a, b, c\}$ (filled triangle), four 1-simplices (edges), and four 0-simplices (vertices).

2.3 Chains, Boundaries, and Cycles

Fix a field $\mathbb{F}$ (typically $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$).

Definition 2.7 (Chain Group)**.** The *group of p-chains* of K with coefficients in $\mathbb{F}$ is the vector space:

$$C_p(K; \mathbb{F}) = \bigoplus_{\sigma \in \Sigma,\, \dim \sigma = p} \mathbb{F} \cdot \sigma.$$

An element of C_p is a formal linear combination of p-simplices.

Definition 2.8 (Boundary Operator)**.** The *boundary operator* $\partial_p : C_p(K) \to C_{p-1}(K)$ is defined on generators by:

$$\partial_p[v_0, \ldots, v_p] = \sum_{i=0}^{p} (-1)^i [v_0, \ldots, \hat{v}_i, \ldots, v_p].$$

Fundamental Property of the Boundary

$$\partial_{p-1} \circ \partial_p = 0 \quad \forall\, p.$$

In other words, *the boundary of a boundary is zero.*

Proposition 2.9. For all $p \geq 0$, $\partial_{p-1} \circ \partial_p = 0$.

Proof. Let $\sigma = [v_0, \ldots, v_p]$. Then:

$$\begin{aligned}\partial_{p-1}(\partial_p \sigma) &= \partial_{p-1}\left(\sum_{i=0}^{p} (-1)^i [v_0, \ldots, \hat{v}_i, \ldots, v_p]\right) \\ &= \sum_{i=0}^{p} (-1)^i \sum_{\substack{j=0 \\ j \neq i}}^{p} (-1)^{j'} [v_0, \ldots, \hat{v}_j, \ldots, \hat{v}_i, \ldots, v_p]\end{aligned}$$

where $j' = j$ if $j < i$ and $j' = j - 1$ if $j > i$. Each term appears exactly twice with opposite signs, so the sum vanishes. □

Definition 2.10 (Cycles and Boundaries)**.**

- The group of *p-cycles* is $Z_p(K) = \ker \partial_p$.
- The group of *p-boundaries* is $B_p(K) = \operatorname{im} \partial_{p+1}$.
- Since $\partial_p \circ \partial_{p+1} = 0$, we have $B_p(K) \subseteq Z_p(K)$.

2.4 Simplicial Homology

Definition 2.11 (Homology Group)**.** The *p-th homology group* of K with coefficients in $\mathbb{F}$ is the quotient:

$$H_p(K; \mathbb{F}) = Z_p(K) / B_p(K) = \ker \partial_p \,/\, \operatorname{im} \partial_{p+1}.$$

The *p-th Betti number* is $\beta_p = \dim_{\mathbb{F}} H_p(K; \mathbb{F})$.

Betti Number Formula

$$\beta_p = \dim \ker \partial_p - \dim \operatorname{im} \partial_{p+1} = \operatorname{rank}(Z_p) - \operatorname{rank}(B_p).$$

Theorem 2.12 (Euler–Poincaré Characteristic)**.** *If K is a simplicial complex of dimension d, its* Euler characteristic *satisfies:*

$$\chi(K) = \sum_{p=0}^{d} (-1)^p \, |\Sigma_p| = \sum_{p=0}^{d} (-1)^p \beta_p$$

where Σ_p denotes the set of p-simplices of K.

2.5 Matrix Computation of Homology

In practice, homology computation reduces to linear algebra. Fix an ordering on simplices of each dimension. The operator ∂_p is then represented by a matrix D_p with entries in $\mathbb{F}$.

Computing Betti Numbers

1. Build the boundary matrices D_p for $p = 1, \ldots, d$.
2. Compute $r_p = \operatorname{rank}(D_p)$ (via Smith normal form or Gaussian elimination over $\mathbb{F}$).
3. Betti numbers are: $\beta_p = |\Sigma_p| - r_p - r_{p+1}$.

Homology via Matrix Computation

```python
import numpy as np

def boundary_matrix(simplices_p, simplices_pm1):
    """Boundary matrix over F_2."""
    n_rows = len(simplices_pm1)
    n_cols = len(simplices_p)
    D = np.zeros((n_rows, n_cols), dtype=int)
    idx_map = {tuple(s): i for i, s in enumerate(simplices_pm1)}
    for j, sigma in enumerate(simplices_p):
        for k in range(len(sigma)):
            face = tuple(sigma[:k] + sigma[k+1:])
            if face in idx_map:
                D[idx_map[face], j] = 1  # mod 2
    return D % 2

def betti_numbers_f2(K, dim):
    """Compute beta_0, ..., beta_dim over F_2."""
    simplices_by_dim = {}
    for s in K:
        d = len(s) - 1
        simplices_by_dim.setdefault(d, []).append(sorted(s))
    bettis = []
    for p in range(dim + 1):
        sp = simplices_by_dim.get(p, [])
        if p > 0:
            D_p = boundary_matrix(sp, simplices_by_dim.get(p-1, []))
            rk_p = np.linalg.matrix_rank(D_p) if D_p.size else 0
        else:
            rk_p = 0
        sp1 = simplices_by_dim.get(p+1, [])
        if sp1:
            D_p1 = boundary_matrix(sp1, sp)
            rk_p1 = np.linalg.matrix_rank(D_p1) if D_p1.size else 0
        else:
            rk_p1 = 0
        bettis.append(len(sp) - rk_p - rk_p1)
    return bettis
```

2.6 Computation Examples

Example 2.13 (Hollow Triangle)**.** Let K be the complex formed by three vertices $\{a, b, c\}$ and three edges $\{a, b\}, \{b, c\}, \{a, c\}$, *without* the 2-simplex $\{a, b, c\}$.

- $D_1 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ (over $\mathbb{F}_2$), rank 2.
- $\beta_0 = 3 - 2 = 1$ (one connected component).
- $\beta_1 = 3 - 2 - 0 = 1$ (one hole — the triangle is hollow).

Example 2.14 (Filled Triangle)**.** Add the 2-simplex $\{a, b, c\}$ to the previous complex. D_2 is the column vector $(1, 1, 1)^T$ over $\mathbb{F}_2$, rank 1.

- $\beta_0 = 3 - 2 = 1$.
- $\beta_1 = 3 - 2 - 1 = 0$ (the hole is filled).
- $\beta_2 = 1 - 1 = 0$.

2.7 Relative Homology and Exact Sequences

Definition 2.15 (Relative Homology)**.** Let $L \subseteq K$ be a subcomplex. The *relative homology* is:

$$H_p(K, L; \mathbb{F}) = \ker(\bar{\partial}_p)/\mathrm{im}(\bar{\partial}_{p+1})$$

where $\bar{\partial}_p$ is induced by ∂_p on the quotient $C_p(K)/C_p(L)$.

Theorem 2.16 (Long Exact Sequence of a Pair)**.** *There exists a long exact sequence:*

$$\cdots \to H_p(L) \xrightarrow{i_*} H_p(K) \xrightarrow{j_*} H_p(K, L) \xrightarrow{\partial_*} H_{p-1}(L) \to \cdots$$

2.8 Homology with GUDHI

Homology Computation with GUDHI

```python
import gudhi

# Build a simplicial complex
st = gudhi.SimplexTree()
st.insert([0, 1])
st.insert([1, 2])
st.insert([0, 2])
# Hollow triangle: no 2-simplex [0,1,2]

# Betti numbers
betti = st.betti_numbers()
print(f"Betti numbers: {betti}")
# Output: Betti numbers: {0: 1, 1: 1}
```

```python
# Now fill the triangle
st.insert([0, 1, 2])
betti = st.betti_numbers()
print(f"Betti numbers: {betti}")
# Output: Betti numbers: {0: 1}
```

2.9 Reduced Homology and Euler Characteristic

Definition 2.17 (Reduced Homology)**.** The *reduced homology* $\tilde{H}_p(K)$ is defined by augmenting the chain complex with the map $\varepsilon : C_0(K) \to \mathbb{F}$, $\varepsilon(\sum \lambda_i v_i) = \sum \lambda_i$. We have:

$$\tilde{H}_p(K) = \begin{cases} H_p(K) & \text{if } p > 0, \\ \ker(\varepsilon)/B_0(K) & \text{if } p = 0. \end{cases}$$

In particular, $\tilde{\beta}_0 = \beta_0 - 1$ for a non-empty complex.

2.10 Simplicial Maps and Functoriality

Definition 2.18 (Simplicial Map)**.** Let K and L be two simplicial complexes. A *simplicial map* $f : K \to L$ is a map $f : V_K \to V_L$ on vertices such that for every simplex $\sigma = \{v_0, \dots, v_p\} \in K$, we have $f(\sigma) = \{f(v_0), \dots, f(v_p)\} \in L$.

Proposition 2.19. Every simplicial map $f : K \to L$ induces a linear map $f_* : H_p(K) \to H_p(L)$ for all p. This assignment is functorial: $(g \circ f)_* = g_* \circ f_*$ and $(\mathrm{id}_K)_* = \mathrm{id}_{H_p(K)}$.

2.11 Exercises

Exercise 2.1. Compute by hand the Betti numbers of the simplicial complex formed by the faces of a tetrahedron (without the interior). Verify the Euler characteristic formula.

Exercise 2.2. Show that for a connected graph G with n vertices and m edges, $\beta_0 = 1$ and $\beta_1 = m - n + 1$.

Exercise 2.3. Implement in Python the construction of the boundary matrix D_1 for a graph given by its edge list. Verify that $D_0 \cdot D_1 = 0$ (over $\mathbb{F}_2$) on an example.

Exercise 2.4. Compute the homology of the triangulated Klein bottle (9 vertices, 27 edges, 18 triangles). Compare the results over $\mathbb{F}_2$ and over $\mathbb{Q}$.

Exercise 2.5. Show that if K is a cone (i.e., there exists a vertex v such that $\sigma \cup \{v\} \in K$ for all $\sigma \in K$), then $\tilde{H}_p(K) = 0$ for all p.

Chapter 3

Persistent Homology

Classical homology answers the question: "how many holes does this space have?" But in topological data analysis, the space itself is not given—we only have a point cloud, and the topology depends on the scale parameter chosen. At a small scale, every point is isolated; at a large scale, everything merges into a single connected component. The question becomes: "which topological features *persist* across scales?" This is the founding idea of persistent homology, developed in the early 2000s by Herbert Edelsbrunner, David Letscher, and Afra Zomorodian, then enriched by Gunnar Carlsson and his collaborators.

The key result is that persistence can be represented by a *persistence diagram*: a set of points in the plane, where each point (b, d) encodes the birth and death of a topological feature. Points far from the diagonal correspond to robust structures; those near the diagonal are noise. This representation, both simple and powerful, has become the central tool of TDA.

Intuition

Classical homology describes the topology of a fixed space. *Persistent* homology studies how this topology *evolves* as one traverses a nested family of complexes: it detects when a topological feature is born and when it dies.

3.1 Filtrations

Definition 3.1 (Filtration of a Simplicial Complex)**.** A *filtration* of a simplicial complex K is an increasing sequence of subcomplexes:

$$\emptyset = K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_m = K$$

where each K_i is a subcomplex of K.

Definition 3.2 (Function-Induced Filtration)**.** Equivalently, a filtration can be described by a function $f : K \to \mathbb{R}$ such that for every $\sigma \in K$ and every face $\tau \subseteq \sigma$, $f(\tau) \leq f(\sigma)$. The filtered complex at level r is:

$$K_r = f^{-1}((-\infty, r]) = \{\sigma \in K : f(\sigma) \leq r\}.$$

Example 3.3 (Vietoris–Rips Filtration)**.** Given a point cloud $X \subset \mathbb{R}^d$, the Vietoris–Rips filtration is:

$$\mathrm{VR}(X, r) = \big\{\sigma \subseteq X : \|x_i - x_j\| \leq 2r \text{ for all } x_i, x_j \in \sigma\big\}.$$

As r increases, more and more simplices appear.

$r = 0.3$, $\beta_0 = 4$ | $r = 0.7$, $\beta_0 = 2$ | $r = 1.0$, $\beta_0 = 1, \beta_1 = 1$ | $r = 1.5$, $\beta_0 = 1, \beta_1 = 0$

3.2 Persistent Homology: Definition

Definition 3.4 (Persistent Homology Groups)**.** Given a filtration $K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m$, the inclusion $\iota_{i,j} : K_i \hookrightarrow K_j$ $(i \leq j)$ induces a linear map in homology:

$$\iota_{i,j}^p : H_p(K_i) \to H_p(K_j).$$

The *p-th persistent homology group* from K_i to K_j is:

$$H_p^{i,j} = \mathrm{im}(\iota_{i,j}^p).$$

The *persistent Betti number* is $\beta_p^{i,j} = \dim H_p^{i,j}$.

Definition 3.5 (Birth and Death)**.** A homology class $\gamma \in H_p(K_i)$ is *born* at time i if $\gamma \notin \mathrm{im}(\iota_{i-1,i}^p)$. It *dies* at time $j > i$ if $\iota_{i,j-1}^p(\gamma) \neq 0$ but $\iota_{i,j}^p(\gamma) = 0$. Its *persistence* is $j - i$.

Persistent Betti Numbers

$$\beta_p^{i,j} = \dim \frac{Z_p(K_i)}{B_p(K_j) \cap Z_p(K_i)}$$

3.3 The Decomposition Theorem

Definition 3.6 (Persistence Module)**.** A *persistence module* (over $\mathbb{F}$) is a diagram of the form:

$$V_1 \xrightarrow{\varphi_1} V_2 \xrightarrow{\varphi_2} \cdots \xrightarrow{\varphi_{m-1}} V_m$$

where the V_i are finite-dimensional vector spaces and the φ_i are linear maps.

Theorem 3.7 (Decomposition — Crawley-Boevey, Gabriel)**.** *Every* pointwise finite-dimensional *persistence module over* $(\mathbb{R}, \leq)$ *decomposes uniquely (up to isomorphism and permutation) as a direct sum of interval modules:*

$$\mathbf{V} \cong \bigoplus_{k=1}^{N} \mathbb{I}[b_k, d_k)$$

where $\mathbb{I}[b, d)$ *equals* $\mathbb{F}$ *on* $[b, d)$ *and* 0 *elsewhere. The collection of pairs* $\{(b_k, d_k)\}$ *is the* persistence diagram.

Remark 3.8. This theorem is the persistence-module analogue of the structure theorem for modules over a PID (classification of graded modules over $\mathbb{F}[t]$ due to Zomorodian–Carlsson, 2005).

3.4 The Standard Algorithm

The boundary matrix reduction algorithm is the classical method for computing persistent homology.

Definition 3.9 (Filtered Boundary Matrix)**.** Order the simplices $\sigma_1, \ldots, \sigma_N$ of K so that $f(\sigma_i) \leq f(\sigma_j)$ for $i < j$ (where f is the filtration function), and faces appear before the simplices containing them. The *filtered boundary matrix* D is the $N \times N$ matrix whose entry (i, j) is the coefficient of σ_i in $\partial\sigma_j$.

Boundary Matrix Reduction

1. Let D be the filtered boundary matrix ($N \times N$).
2. Define $\text{low}(j)$ as the index of the lowest non-zero row in column j (or -1 if the column is zero).
3. For $j = 1, \ldots, N$:
   (a) While there exists $j' < j$ with $\text{low}(j') = \text{low}(j) \neq -1$:
   (b) Add column j' to column j (over $\mathbb{F}$).
4. After reduction:
   - If $\text{low}(j) = i$, then (σ_i, σ_j) forms a birth-death pair.
   - If column j is zero and j is not the image of any low, then σ_j is an essential generator (never dies).

Matrix Reduction in Python

```python
def reduce_boundary_matrix(D):
    """Standard boundary matrix reduction over F_2.
    D: list of sets, D[j] = set of nonzero row indices of col j.
    Returns: pairs (birth, death) and essential generators.
    """
    n = len(D)
    pivot_col = {}  # maps pivot row -> column index
    pairs = []

    for j in range(n):
        while D[j] and max(D[j]) in pivot_col:
            j_prime = pivot_col[max(D[j])]
            D[j] = D[j].symmetric_difference(D[j_prime])

        if D[j]:
            i = max(D[j])
            pivot_col[i] = j
            pairs.append((i, j))

    return pairs
```

```
# Example: hollow triangle [a, b, c] with edges
# Order: a(0), b(1), c(2), ab(3), bc(4), ac(5)
D = [set(), set(), set(),
     {0, 1}, {1, 2}, {0, 2}]
pairs = reduce_boundary_matrix(D)
print("Pairs (birth, death):", pairs)
```

3.5 Algorithmic Complexity

Theorem 3.10 (Complexity of the Standard Algorithm)**.** *The boundary matrix reduction algorithm has worst-case complexity $O(N^3)$, where N is the total number of simplices in the filtration.*

Remark 3.11. In practice, the matrix is very sparse and the algorithm runs much faster. Optimizations like the *clearing optimization* and the *twist* considerably reduce computation time. The `ripser` implementation exploits these optimizations along with implicit matrix reduction.

3.6 Persistent Homology with Ripser

Computation with Ripser

```
import numpy as np
from ripser import ripser

# Generate points on a noisy circle
n = 100
theta = np.linspace(0, 2*np.pi, n, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])
X += 0.05 * np.random.randn(n, 2)

# Persistent homology up to dimension 1
result = ripser(X, maxdim=1)
dgm0 = result['dgms'][0]  # H_0
dgm1 = result['dgms'][1]  # H_1

print(f"H_0: {len(dgm0)} pairs")
print(f"H_1: {len(dgm1)} pairs")

# Most persistent pair in H_1
if len(dgm1) > 0:
    pers = dgm1[:, 1] - dgm1[:, 0]
    idx = np.argmax(pers)
    print(f"Most persistent H_1 pair: "
          f"birth={dgm1[idx,0]:.3f}, "
          f"death={dgm1[idx,1]:.3f}, "
          f"persistence={pers[idx]:.3f}")
```

3.7 Persistent Homology with GUDHI

Computation with GUDHI

```
import gudhi
import numpy as np

# Data: noisy circle
n = 100
theta = np.linspace(0, 2*np.pi, n, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])
X += 0.05 * np.random.randn(n, 2)

# Rips complex
rips = gudhi.RipsComplex(points=X, max_edge_length=2.0)
st = rips.create_simplex_tree(max_dimension=2)

# Persistence
persistence = st.persistence()
betti = st.betti_numbers()

print(f"Betti numbers: {betti}")
gudhi.plot_persistence_diagram(persistence)
```

3.8 Persistence Modules and Quiver Representations

Definition 3.12 (Quiver A_n)**.** The quiver A_n is the directed graph:

$$1 \to 2 \to 3 \to \cdots \to n.$$

A (discrete, length-n) persistence module is a representation of the quiver A_n over $\mathbb{F}$.

Theorem 3.13 (Gabriel, 1972)**.** *The quiver A_n is of finite representation type: its indecomposable representations are exactly the interval modules $\mathbb{I}[b, d]$, $1 \leq b \leq d \leq n$.*

Remark 3.14. This result underlies the decomposition theorem for persistence modules. For *multi-parameter* filtrations (2D or higher), Gabriel's theorem no longer applies and the classification of indecomposables is *wild* — this is one of the major challenges in multi-parameter TDA.

3.9 Extended Persistence

Definition 3.15 (Extended Persistence)**.** A class is said to be born at b and die at $d = +\infty$ if it survives in all complexes of the filtration. Such classes are called *essential*. The extended persistence diagram includes points $(b, +\infty)$.

Proposition 3.16. For a Vietoris–Rips filtration of a connected point cloud, there is always exactly one essential point in dimension 0, corresponding to the last connected component that never dies.

3.10 Exercises

Exercise 3.1. Compute by hand the persistent homology of the following filtration (over $\mathbb{F}_2$):

$$K_0 = \{a\} \subset K_1 = \{a, b\} \subset K_2 = \{a, b, ab\} \subset K_3 = \{a, b, c, ab, bc\} \subset K_4 = \{a, b, c, ab, bc, ac\} \subset K_5 = K_4$$

Give the birth-death pairs for H_0 and H_1.

Exercise 3.2. Implement the boundary matrix reduction algorithm over $\mathbb{F}_2$. Test it on the complex from the previous exercise.

Exercise 3.3. Generate 200 points on the figure-eight (two tangent circles) with noise. Compute persistent homology and interpret the persistence diagram.

Exercise 3.4. Show that if a simplex σ appears in the filtration at time t and creates a new cycle (i.e., its corresponding column reduces to zero), then the class of this cycle is born at time t.

Exercise 3.5. Compare the computation times of `gudhi` and `ripser` on point clouds of 500, 1000, and 2000 random points in $\mathbb{R}^3$. Discuss the differences.

Chapter 4

Barcodes and Persistence Diagrams

Persistent homology produces a considerable amount of information: for each dimension, a list of topological features with their birth and death times. How does one visualise and compare these results? Two equivalent representations have established themselves: the *barcode*, where each feature is a horizontal bar whose length measures its persistence, and the *persistence diagram*, where each feature is a point in the upper half-plane. These representations, introduced by Edelsbrunner, Letscher, and Zomorodian in the early 2000s, have become the standard interface between topological theory and applications.

Intuition

The persistence diagram and the barcode are two equivalent representations of the result of persistent homology. Each point (b, d) in the diagram represents a topological feature born at b and dying at d. The farther a point is from the diagonal, the more persistent — and thus more significant — the feature.

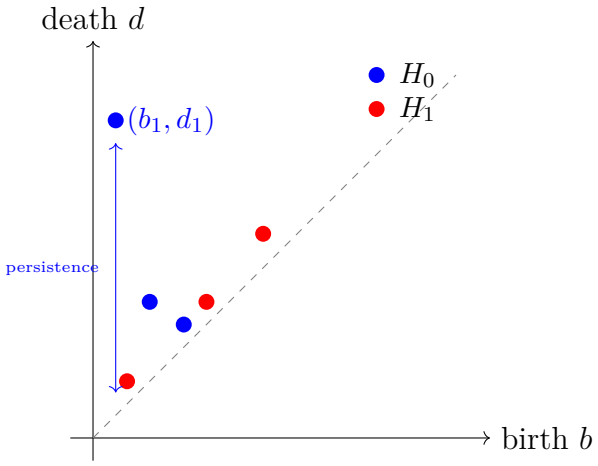
4.1 Persistence Diagrams

Definition 4.1 (Persistence Diagram)**.** Let $\{(b_k, d_k)\}_{k=1}^{N}$ be the set of birth-death pairs from persistent homology in dimension p. The *persistence diagram* is the multiset:

$$\mathrm{Dgm}_p = \{(b_k, d_k) \in \mathbb{R}^2 : b_k < d_k\} \cup \Delta$$

where $\Delta = \{(x, x) : x \in \mathbb{R}\}$ is the diagonal, equipped with infinite multiplicity.

Remark 4.2. Adding the diagonal with infinite multiplicity is essential for correctly defining distances between diagrams (Chapter 8).

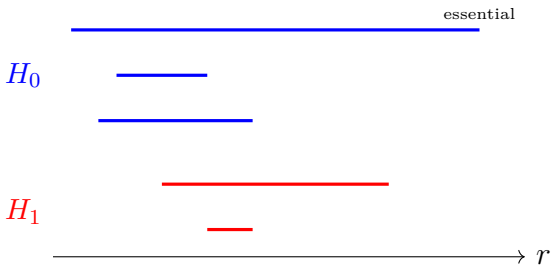


Definition 4.3 (Persistence)**.** The *persistence* of a point (b, d) is $\text{pers}(b, d) = d - b$. The *total persistence* is:

$$\text{Pers}_q(\text{Dgm}) = \left(\sum_{(b,d)\in \text{Dgm}} (d-b)^q \right)^{1/q}.$$

4.2 Barcodes

Definition 4.4 (Barcode)**.** The *barcode* is the equivalent representation where each pair (b_k, d_k) is drawn as a horizontal segment $[b_k, d_k)$. The segments are stacked vertically.



Remark 4.5. The barcode and the persistence diagram contain exactly the same information. The barcode is often more intuitive for visualization; the diagram is better suited for mathematical analysis (distances, stability).

4.3 Betti Functions

Definition 4.6 (Betti Function)**.** The *Betti function* $\beta_p : \mathbb{R} \to \mathbb{N}$ assigns to each $r \in \mathbb{R}$ the Betti number of the filtered complex at level r:

$$\beta_p(r) = \dim H_p(K_r).$$

It is a step function, piecewise non-decreasing.

Proposition 4.7. The Betti function $\beta_p(r)$ can be read directly from the barcode: $\beta_p(r)$ is the number of bars containing r in dimension p.

4.4 Persistence Landscapes

Definition 4.8 (Persistence Landscape)**.** Let $\mathrm{Dgm}_p = \{(b_i, d_i)\}$. For each point, define the tent function:

$$\Lambda_i(t) = \max\left(0,\, \min(t - b_i,\, d_i - t)\right).$$

The *k-th persistence landscape* is:

$$\lambda_k(t) = k\text{-th largest value of } \{\Lambda_i(t)\}_{i=1}^{N}.$$

Persistence Landscape

$$\lambda_k : \mathbb{R} \to \mathbb{R}_{\geq 0}, \quad \lambda_k(t) = \mathrm{kmax}\big\{\Lambda_i(t) : (b_i, d_i) \in \mathrm{Dgm}_p\big\}$$

where kmax denotes the k-th maximum.

Theorem 4.9 (Bubenik, 2015)**.** *Persistence landscapes are elements of a separable Banach space. They satisfy a strong law of large numbers and a central limit theorem.*

4.5 Persistence Images

Definition 4.10 (Persistence Image)**.** The *persistence image* is obtained by:

1. Transforming each point (b, d) to $(b, d - b)$ (birth-persistence coordinates).
2. Placing a Gaussian $\mathcal{N}((b_i, p_i), \sigma^2 I)$ centered at each transformed point.
3. Weighting by a weight function $w(b, p)$ (typically increasing in p).
4. Discretizing on a grid to obtain a matrix.

Persistence Image

$$\rho(x, y) = \sum_{i=1}^{N} w(b_i, p_i) \cdot \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x - b_i)^2 + (y - p_i)^2}{2\sigma^2}\right)$$

where $p_i = d_i - b_i$ is the persistence of the i-th point.

4.6 Implementations

Visualizing Diagrams and Barcodes

```python
import numpy as np
from ripser import ripser
from persim import plot_diagrams
import matplotlib.pyplot as plt

# Noisy circle
theta = np.linspace(0, 2*np.pi, 100, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])
```

```
X += 0.1 * np.random.randn(*X.shape)

result = ripser(X, maxdim=1)
dgms = result['dgms']

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Persistence diagram
plot_diagrams(dgms, ax=axes[0])
axes[0].set_title("Persistence Diagram")

# Barcode
for dim, dgm in enumerate(dgms):
    color = ['blue', 'red'][dim]
    offset = 0 if dim == 0 else len(dgms[0])
    for k, (b, d) in enumerate(dgm):
        if np.isinf(d):
            d = max(dgm[np.isfinite(dgm[:, 1]), 1].max(), b + 0.5)
        axes[1].plot([b, d], [k + offset, k + offset],
                     color=color, linewidth=2)
axes[1].set_title("Barcode")
axes[1].set_xlabel("$r$")

# Persistence landscape
t = np.linspace(0, 2.5, 500)
dgm1 = dgms[1]
dgm1_finite = dgm1[np.isfinite(dgm1[:, 1])]
tents = np.array([np.maximum(0, np.minimum(t - b, d - t))
                  for b, d in dgm1_finite])
if len(tents) > 0:
    landscape_1 = np.sort(tents, axis=0)[::-1]
    for k in range(min(3, len(landscape_1))):
        axes[2].plot(t, landscape_1[k], label=f"$\\lambda_{k+1}$")
    axes[2].legend()
axes[2].set_title("Persistence Landscapes ($H_1$)")

plt.tight_layout()
plt.savefig("persistence_representations.pdf")
```

Persistence Images with giotto-tda

```
from gtda.homology import VietorisRipsPersistence
from gtda.diagrams import PersistenceImage
import numpy as np

# Data
theta = np.linspace(0, 2*np.pi, 100, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])
X += 0.1 * np.random.randn(*X.shape)
```

```python
# giotto-tda pipeline
VR = VietorisRipsPersistence(homology_dimensions=[0, 1])
diagrams = VR.fit_transform(X[np.newaxis, :, :])

PI = PersistenceImage(sigma=0.1, n_bins=20)
images = PI.fit_transform(diagrams)
print(f"Persistence image shape: {images.shape}")
```

4.7 Persistence Entropy

Definition 4.11 (Persistence Entropy)**.** Let $\mathrm{Dgm} = \{(b_i, d_i)\}_{i=1}^{N}$ be a persistence diagram (finite points only). The *persistence entropy* is:

$$E(\mathrm{Dgm}) = -\sum_{i=1}^{N} p_i \log p_i, \quad p_i = \frac{d_i - b_i}{\sum_{j=1}^{N}(d_j - b_j)}.$$

It measures the complexity of the distribution of lifetimes.

4.8 Betti Curve

Definition 4.12 (Betti Curve)**.** The *Betti curve* is the function $\beta_p : \mathbb{R} \to \mathbb{N}$ defined by:

$$\beta_p(t) = \left|\{(b_i, d_i) \in \mathrm{Dgm}_p : b_i \leq t < d_i\}\right|.$$

This curve is directly related to the barcode: it counts the number of active bars at each time t.

4.9 Persistence Silhouettes

Definition 4.13 (Persistence Silhouette)**.** The *persistence silhouette* of order $q > 0$ is the weighted average of tent functions:

$$\phi_q(t) = \frac{\sum_i (d_i - b_i)^q \cdot \Lambda_i(t)}{\sum_i (d_i - b_i)^q}.$$

For $q = 1$, this is the arithmetic mean weighted by persistence.

4.10 Statistics on Diagrams

Theorem 4.14 (Fréchet Mean)**.** *Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a sample of persistence diagrams. The* Fréchet mean *is:*

$$\bar{D} = \arg\min_{D} \sum_{i=1}^{n} W_2(D, D_i)^2$$

where W_2 is the 2-Wasserstein distance. This minimum exists but is not necessarily unique.

Proposition 4.15. Persistence landscapes, being elements of a Banach space, admit a well-defined mean:

$$\bar{\lambda}_k(t) = \frac{1}{n} \sum_{i=1}^{n} \lambda_k^{(i)}(t).$$

4.11 Exercises

Exercise 4.1. Construct the persistence diagram and barcode for a Vietoris–Rips filtration of 6 points equally spaced on a circle.

Exercise 4.2. Implement in Python the computation of a persistence landscape from a persistence diagram. Test on a noisy circle.

Exercise 4.3. Compute the persistence image of a cloud of 200 points on a torus. Vary the parameter σ and observe the effect on the image.

Exercise 4.4. Show that persistence entropy is maximal when all lifetimes are equal, and minimal (zero) when only one pair has nonzero persistence.

Exercise 4.5. Compare the persistence landscapes of a circle, a torus, and a sphere sampled with noise. What differences do you observe?

Chapter 5

Stability Theorems

Persistent homology would be a mere mathematical curiosity if it were not *stable*: a small change in the data must produce a small change in the persistence diagram. This fundamental result, proved by David Cohen-Steiner, Herbert Edelsbrunner, and John Harer in 2007, is the pillar on which all applied TDA rests. It states that the bottleneck distance between two persistence diagrams is bounded by the L^∞ distance between the filtration functions. Without this theorem, the slightest noise in the data could completely overturn the topological descriptors, rendering them useless in practice.

Intuition

Stability theorems are the theoretical foundation of TDA: they guarantee that small perturbations of the data produce small changes in persistence diagrams. Without stability, topological descriptors would be useless in practice.

5.1 Distances between Diagrams: Review

Before stating the stability theorems, we recall the distances used. These will be developed in Chapter 8.

Definition 5.1 (Bottleneck Distance)**.** Let D_1, D_2 be two persistence diagrams. The *bottleneck distance* is:

$$d_B(D_1, D_2) = \inf_{\gamma} \sup_{x \in D_1} \|x - \gamma(x)\|_\infty$$

where the infimum is over all bijections $\gamma : D_1 \to D_2$ (recall that diagrams include the diagonal with infinite multiplicity).

Definition 5.2 (Wasserstein Distance)**.** The *q-Wasserstein distance* ($1 \leq q < \infty$) is:

$$W_q(D_1, D_2) = \left(\inf_{\gamma} \sum_{x \in D_1} \|x - \gamma(x)\|_\infty^q \right)^{1/q}.$$

Relationship between Distances

$$d_B(D_1, D_2) = W_\infty(D_1, D_2) \leq W_q(D_1, D_2) \leq W_p(D_1, D_2) \quad \text{for } p \leq q.$$

5.2 Bottleneck Stability Theorem

The fundamental theorem of TDA is due to Cohen-Steiner, Edelsbrunner, and Harer (2007).

Definition 5.3 (Tame Function)**.** A function $f : X \to \mathbb{R}$ is called *tame* if the sublevel sets $f^{-1}((-\infty, r])$ have finite-dimensional homology for all r, and the number of critical values is finite.

Theorem 5.4 (Bottleneck Stability — Cohen-Steiner, Edelsbrunner, Harer, 2007)**.** *Let $f, g : X \to \mathbb{R}$ be two tame functions on a triangulable topological space X. Then:*

$$d_B(\mathrm{Dgm}(f), \mathrm{Dgm}(g)) \leq \|f - g\|_\infty .$$

Proof sketch. The proof relies on three ingredients:

1. **Interpolation**: consider the family $f_t = (1-t)f + tg$ for $t \in [0, 1]$.

2. **Box lemma**: if the diagrams of f and g differ in a region, then $\|f - g\|_\infty$ is at least as large as the size of that region.

3. **Matching construction**: build a bijection γ between diagrams by tracking points through the interpolation.

See Cohen-Steiner, Edelsbrunner, Harer (2007) for the complete proof. □

Remark 5.5. This theorem is tight: there exist functions for which equality is achieved.

5.3 Stability for Point Clouds

Corollary 5.6 (Vietoris–Rips Stability)**.** *Let $X, Y \subset \mathbb{R}^d$ be two finite point clouds. Let $d_H(X, Y)$ be the Hausdorff distance. Then:*

$$d_B(\mathrm{Dgm}(\mathrm{VR}(X)), \mathrm{Dgm}(\mathrm{VR}(Y))) \leq 2\, d_H(X, Y).$$

Definition 5.7 (Hausdorff Distance)**.** The *Hausdorff distance* between two compact sets X, Y in a metric space (M, d) is:

$$d_H(X, Y) = \max\left(\sup_{x \in X} \inf_{y \in Y} d(x, y),\ \sup_{y \in Y} \inf_{x \in X} d(x, y) \right).$$

Proof. Let $f_X(z) = \inf_{x \in X} d(z, x)$ be the distance function to X, and similarly for f_Y. Then $\|f_X - f_Y\|_\infty = d_H(X, Y)$. The persistence diagrams of the sublevel set filtrations of f_X and f_Y coincide (up to constants) with those of the Rips filtrations. The result follows from the bottleneck stability theorem, with the factor 2 arising from the parametrization convention. □

5.4 Wasserstein Stability

Theorem 5.8 (Wasserstein Stability — Chazal et al., 2009)**.** *Let $f, g : X \to \mathbb{R}$ be two tame functions. For all $q \geq 1$ and $p \in \mathbb{N}$, if the diagrams $\mathrm{Dgm}_p(f)$ and $\mathrm{Dgm}_p(g)$ have at most N off-diagonal points, then:*

$$W_q(\mathrm{Dgm}_p(f), \mathrm{Dgm}_p(g)) \leq C(N, q) \cdot \|f - g\|_\infty$$

where $C(N, q) = N^{1/q}$ for the simplest bound.

Remark 5.9. Unlike bottleneck stability, Wasserstein stability depends on the number of points in the diagram. Sharper bounds exist under additional hypotheses.

5.5 Stability of Representations

Theorem 5.10 (Stability of Persistence Landscapes)**.** *Persistence landscapes are 1-Lipschitz with respect to the bottleneck distance:*

$$\left\| \lambda_k^{(1)} - \lambda_k^{(2)} \right\|_\infty \leq d_B(D_1, D_2)$$

for all $k \geq 1$.

Proposition 5.11 (Stability of Persistence Images)**.** Persistence images are Lipschitz with respect to the 1-Wasserstein distance: there exists $C > 0$ (depending on σ and the weight function) such that:

$$\|\mathrm{PI}(D_1) - \mathrm{PI}(D_2)\|_F \leq C \cdot W_1(D_1, D_2).$$

5.6 Generalized Stability Theorem

Definition 5.12 (Interleaving Distance)**.** Two persistence modules $\mathbf{U}$ and $\mathbf{V}$ are *ε-interleaved* if there exist natural morphisms $\varphi : \mathbf{U} \to \mathbf{V}[\varepsilon]$ and $\psi : \mathbf{V} \to \mathbf{U}[\varepsilon]$ such that the composites $\psi[\varepsilon] \circ \varphi$ and $\varphi[\varepsilon] \circ \psi$ coincide with the 2ε-shift structure maps.

The *interleaving distance* is:

$$d_I(\mathbf{U}, \mathbf{V}) = \inf\{\varepsilon \geq 0 : \mathbf{U} \text{ and } \mathbf{V} \text{ are } \varepsilon\text{-interleaved}\}.$$

Theorem 5.13 (Algebraic Stability Isometry)**.** *For any p.f.d. persistence modules:*

$$d_B(\mathrm{Dgm}(\mathbf{U}), \mathrm{Dgm}(\mathbf{V})) = d_I(\mathbf{U}, \mathbf{V}).$$

Remark 5.14. This theorem, due to Chazal, Cohen-Steiner, Glisse, Guibas, and Oudot (2009) and Bauer–Lesnick (2015), shows that the bottleneck distance is the "right" distance on persistence diagrams: it coincides exactly with the natural notion of proximity for persistence modules.

5.7 Applications of Stability

5.7.1 Topological Inference

Theorem 5.15 (Topological Inference — Niyogi, Smale, Weinberger, 2008)**.** *Let $\mathcal{M}$ be a compact submanifold of $\mathbb{R}^d$ with reach $\tau > 0$. If X is an ε-dense sample of $\mathcal{M}$ with $\varepsilon < \tau/2$, then the Čech filtration of X at parameter $r \in (\varepsilon, \tau - \varepsilon)$ has the same homotopy type as $\mathcal{M}$.*

5.7.2 Robustness to Noise

Proposition 5.16. If X is a point cloud sampled from a manifold $\mathcal{M}$ and X' is a noisy version with $d_H(X, X') \leq \delta$, then:

$$d_B(\mathrm{Dgm}(X), \mathrm{Dgm}(X')) \leq 2\delta.$$

Topological features with persistence $> 4\delta$ in the diagram of X' correspond to genuine features of $\mathcal{M}$.

5.8 Numerical Verification

Numerical Verification of Stability

```python
import numpy as np
from ripser import ripser
from persim import bottleneck

# Exact circle
theta = np.linspace(0, 2*np.pi, 100, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])

# Noisy versions with increasing epsilon
epsilons = [0.01, 0.05, 0.1, 0.2, 0.5]
dgm_X = ripser(X, maxdim=1)['dgms']

for eps in epsilons:
    noise = eps * np.random.randn(*X.shape)
    X_noisy = X + noise
    dgm_Y = ripser(X_noisy, maxdim=1)['dgms']

    # Bottleneck distance in H_1
    d_bn = bottleneck(dgm_X[1], dgm_Y[1])

    # Hausdorff distance (approximation)
    from scipy.spatial.distance import directed_hausdorff
    d_h = max(directed_hausdorff(X, X_noisy)[0],
              directed_hausdorff(X_noisy, X)[0])

    print(f"eps={eps:.2f}: d_B={d_bn:.4f}, "
          f"d_H={d_h:.4f}, "
          f"ratio={d_bn/d_h:.4f}")
```

5.9 Limitations of Stability

Limitations

- Bottleneck stability is an *upper* bound: the diagram may change much less than the bound suggests.
- The Wasserstein bound depends on the number of points: for diagrams with many low-persistence points, the bound can be loose.
- Stability concerns the *diagrams*, not necessarily the *vectorized representations* (although these often inherit stability properties).

5.10 Exercises

Exercise 5.1. Show that the bottleneck distance satisfies the triangle inequality.

Exercise 5.2. Construct an example of two functions $f, g : [0,1] \to \mathbb{R}$ such that $d_B(\mathrm{Dgm}(f), \mathrm{Dgm}(g)) = \|f - g\|_\infty$ (show that the bound is tight).

Exercise 5.3. Numerically verify bottleneck stability by generating point clouds on a torus with different noise levels. Plot d_B versus d_H and verify linearity.

Exercise 5.4. Prove that if $\mathbf{U}$ and $\mathbf{V}$ are ε-interleaved, then $d_B(\mathrm{Dgm}(\mathbf{U}), \mathrm{Dgm}(\mathbf{V})) \leq \varepsilon$.

Exercise 5.5. Let X be a cloud of n i.i.d. points on a compact manifold $\mathcal{M}$. Show that $d_H(X, \mathcal{M}) \to 0$ almost surely as $n \to \infty$, and deduce the convergence of persistence diagrams.

Chapter 6

Vietoris-Rips, Čech, and Alpha Complexes

Persistent homology needs a *filtration* of simplicial complexes. But how does one build a complex from a point cloud? Three classical constructions dominate. The *Čech complex* adds a simplex whenever the balls of radius r centred at its vertices have a common intersection—it captures exactly the topology of the union of balls (nerve theorem). The *Vietoris-Rips complex* is simpler: a simplex is present whenever all pairs of vertices are within distance $\leq 2r$—it is much easier to compute, but only approximates the Čech. The *Alpha complex*, due to Edelsbrunner, restricts the Čech to Voronoï cells, producing a complex of linear size in the number of points.

Intuition

Constructing a simplicial complex from a point cloud is the fundamental step of TDA. Three constructions dominate: Vietoris–Rips (simple but large), Čech (exact but costly), and Alpha (small and exact in low dimensions).

6.1 Vietoris–Rips Complex

Definition 6.1 (Vietoris–Rips Complex)**.** Let $X = \{x_1, \ldots, x_n\}$ be a finite set in a metric space (M, d). The *Vietoris–Rips complex* at parameter $r \geq 0$ is:

$$\mathrm{VR}(X, r) = \big\{\sigma \subseteq X : d(x_i, x_j) \leq r \text{ for all } x_i, x_j \in \sigma\big\}.$$

Remark 6.2. The Vietoris–Rips complex is the *clique complex* (or *flag complex*) of the proximity graph $G_r = (X, E_r)$ where $\{x_i, x_j\} \in E_r$ iff $d(x_i, x_j) \leq r$. That is, $\sigma \in \mathrm{VR}(X, r)$ iff all pairs of vertices of σ are connected in G_r.

Proposition 6.3 (Properties)**.**

1. $\mathrm{VR}(X, r)$ is entirely determined by its 1-skeleton (the graph G_r).
2. $r \leq s \implies \mathrm{VR}(X, r) \subseteq \mathrm{VR}(X, s)$.
3. The maximum dimension of $\mathrm{VR}(X, r)$ can reach $n - 1$.

Combinatorial Explosion

For n points, the Vietoris–Rips complex can contain up to $2^n - 1$ simplices. In practice, one truncates the maximum dimension (typically ≤ 3).

6.2 Čech Complex

Definition 6.4 (Čech Complex)**.** The *Čech complex* at parameter r is:

$$\check{C}(X, r) = \{\sigma \subseteq X : \bigcap_{x_i \in \sigma} B(x_i, r) \neq \emptyset\}$$

where $B(x_i, r) = \{y \in M : d(x_i, y) \leq r\}$ is the closed ball of center x_i and radius r.

Theorem 6.5 (Nerve Theorem — Borsuk, 1948)**.** *Let $\mathcal{U} = \{U_i\}_{i \in I}$ be a finite cover of a topological space X by convex open sets. Then the* nerve *of $\mathcal{U}$:*

$$\mathcal{N}(\mathcal{U}) = \{\sigma \subseteq I : \bigcap_{i \in \sigma} U_i \neq \emptyset\}$$

has the same homotopy type as $\bigcup_{i \in I} U_i$.

Corollary 6.6. *In $\mathbb{R}^d$, $\check{C}(X, r)$ has the same homotopy type as $\bigcup_{i=1}^n B(x_i, r)$.*

Proposition 6.7 (Rips–Čech Relationship)**.** For any point cloud $X \subset \mathbb{R}^d$ and $r > 0$:

$$\check{C}(X, r) \subseteq \mathrm{VR}(X, 2r) \subseteq \check{C}\left(X, \frac{2r}{\sqrt{2}} \cdot \sqrt{\frac{d}{d+1}}\right).$$

In particular: $\check{C}(X, r) \subseteq \mathrm{VR}(X, 2r)$.

6.3 Alpha Complex

Definition 6.8 (Voronoï Diagram)**.** The *Voronoï diagram* of $X \subset \mathbb{R}^d$ is the partition of $\mathbb{R}^d$ into cells:

$$V_i = \{y \in \mathbb{R}^d : \|y - x_i\| \leq \|y - x_j\| \text{ for all } j\}.$$

Definition 6.9 (Delaunay Triangulation)**.** The *Delaunay triangulation* $\mathrm{Del}(X)$ is the simplicial complex dual to the Voronoï diagram: $\sigma = \{x_{i_0}, \ldots, x_{i_p}\} \in \mathrm{Del}(X)$ iff $\bigcap_{k=0}^p V_{i_k} \neq \emptyset$.

Definition 6.10 (Alpha Complex)**.** The *Alpha complex* at parameter r is:

$$\mathrm{Alpha}(X, r) = \{\sigma \in \mathrm{Del}(X) : \bigcap_{x_i \in \sigma} (B(x_i, r) \cap V_i) \neq \emptyset\}.$$

It is the subcomplex of the Delaunay triangulation formed by simplices whose balls restricted to Voronoï cells intersect.

Theorem 6.11 (Edelsbrunner, 1995)**.** *The Alpha complex satisfies:*

1. $\mathrm{Alpha}(X, r)$ *has the same homotopy type as $\bigcup_{i=1}^n B(x_i, r)$ (and hence as $\check{C}(X, r)$).*
2. $\dim(\mathrm{Alpha}(X, r)) \leq d$ *(the ambient dimension).*
3. *The total number of simplices is $O(n^{\lceil d/2 \rceil})$.*

Size Comparison

For n points in $\mathbb{R}^d$:

Complex	Max dimension	Nb simplices
Vietoris–Rips	$n-1$	$O(2^n)$
Čech	$n-1$	$O(2^n)$
Alpha	d	$O(n^{\lceil d/2 \rceil})$

6.4 Practical Construction

Rips Complex with GUDHI

```python
import gudhi
import numpy as np

# Point cloud
np.random.seed(42)
X = np.random.randn(50, 3)

# Rips complex
rips = gudhi.RipsComplex(points=X, max_edge_length=2.0)
st = rips.create_simplex_tree(max_dimension=3)

print(f"Number of simplices: {st.num_simplices()}")
print(f"Number of vertices: {st.num_vertices()}")
print(f"Dimension: {st.dimension()}")

# Persistence
pers = st.persistence()
gudhi.plot_persistence_diagram(pers)
```

Alpha Complex with GUDHI

```python
import gudhi
import numpy as np

# 2D point cloud
np.random.seed(42)
theta = np.linspace(0, 2*np.pi, 80, endpoint=False)
X = np.column_stack([np.cos(theta), np.sin(theta)])
X += 0.1 * np.random.randn(*X.shape)

# Alpha complex
alpha = gudhi.AlphaComplex(points=X)
st = alpha.create_simplex_tree()
```

```
print(f"Number of simplices: {st.num_simplices()}")
print(f"Dimension: {st.dimension()}")

# Persistence
pers = st.persistence()
gudhi.plot_persistence_diagram(pers)
```

Čech Complex Approximation

```
import gudhi
import numpy as np

# 2D point cloud
X = np.random.randn(30, 2)

# The Alpha complex is homotopy-equivalent to Čech
# and much smaller
alpha = gudhi.AlphaComplex(points=X)
st = alpha.create_simplex_tree()

# Filtration values are squared radii of the
# smallest enclosing ball of each simplex
for simplex, filt in st.get_filtration():
    if len(simplex) <= 3:
        print(f"  {simplex}: radius^2 = {filt:.4f}")
    if filt > 0.5:
        break
```

6.5 Sparse Rips and Weighted Rips

Definition 6.12 (Sparse Rips — Sheehy, 2013)**.** The *Sparse Rips complex* is an approximation of the Vietoris–Rips complex containing $O(n)$ simplices per dimension, instead of $O(2^n)$. For a parameter $\varepsilon > 0$, it produces a $(1+\varepsilon)$-approximation of the persistence diagrams.

Sparse Rips with GUDHI

```
import gudhi
import numpy as np

X = np.random.randn(500, 3)

# Sparse Rips (epsilon = 0.3)
sparse_rips = gudhi.RipsComplex(
    points=X, max_edge_length=2.0, sparse=0.3)
st = sparse_rips.create_simplex_tree(max_dimension=2)

print(f"Sparse Rips: {st.num_simplices()} simplices")
```

```
# Comparison with standard Rips
rips = gudhi.RipsComplex(points=X[:100], max_edge_length=2.0)
st_full = rips.create_simplex_tree(max_dimension=2)
print(f"Full Rips (100 pts): {st_full.num_simplices()} simplices")
```

6.6 Cubical Complexes

Definition 6.13 (Cubical Complex)**.** A *cubical complex* is the analogue of a simplicial complex where the building blocks are cubes (products of intervals) instead of simplices. It is particularly suited for grid data (images, 3D volumes).

Persistent Homology of an Image

```
import gudhi
import numpy as np

# Create a simple image (ring)
x = np.linspace(-2, 2, 50)
y = np.linspace(-2, 2, 50)
XX, YY = np.meshgrid(x, y)
image = np.sqrt(XX**2 + YY**2)  # distance to origin

# Cubical complex
cc = gudhi.CubicalComplex(
    top_dimensional_cells=image.flatten(),
    dimensions=image.shape
)

# Persistence
pers = cc.persistence()
print("Image persistence:")
for dim, (b, d) in pers:
    if d - b > 0.5:
        print(f"  H_{dim}: ({b:.2f}, {d:.2f}), "
              f"pers={d-b:.2f}")
```

6.7 Choosing a Complex in Practice

Complex Selection Guide

- **Low-dimensional data** ($d \leq 3$): use the **Alpha** complex. It is small, exact, and fast to build.
- **High-dimensional or metric data**: use **Vietoris–Rips** (with `ripser` for speed). Truncate dimension to 2 or 3.

- **Grid data** (images, volumes): use a **cubical** complex.
- **Large number of points**: use **Sparse Rips** or subsample.

6.8 Exercises

Exercise 6.1. For 4 points at the vertices of a unit square, construct by hand the complexes $\mathrm{VR}(X, r)$ and $\check{C}(X, r)$ for $r = 0.5, 1.0, 1.5$. Verify the inclusion $\check{C}(X, r) \subseteq \mathrm{VR}(X, 2r)$.

Exercise 6.2. Construct the Voronoï diagram and Delaunay triangulation for 5 random points in $\mathbb{R}^2$ (possibly with `scipy.spatial.Delaunay`).

Exercise 6.3. Compare the computation times of Alpha and Rips complexes for point clouds of 100 to 1000 points in dimensions 2, 3, and 10.

Exercise 6.4. Show that the Vietoris–Rips complex of $n + 1$ points in general position in $\mathbb{R}^n$, for sufficiently large r, is the full simplex Δ^n.

Exercise 6.5. Compute the persistent homology of a binary image of your choice using a cubical complex with GUDHI.

Chapter 7

The Mapper Algorithm

In 2007, Gurjeet Singh, Facundo Mémoli, and Gunnar Carlsson published a paper that would transform applied TDA: the Mapper algorithm. The idea is remarkably simple: instead of computing the persistent homology of a point cloud (which produces numerical invariants but no direct visualization), Mapper constructs a *graph* that captures the global shape of the data. The procedure draws inspiration from the nerve theorem: one partitions the data into overlapping pieces using a filter function, clusters the points within each piece, then connects clusters that share points. The resulting graph is a "map" of the data, hence the name. Mapper has been used to discover a subgroup of breast cancer, analyse athletic performance, and explore genomic data — wherever visualizing the "shape" of data provides insight inaccessible to classical statistical methods.

Intuition

Mapper is a topological visualization algorithm that constructs a summary graph capturing the global structure of a high-dimensional dataset. Unlike persistent homology which computes numerical invariants, Mapper produces an interpretable geometric object.

7.1 Motivation

Dimensionality reduction techniques (PCA, t-SNE, UMAP) project data into a low-dimensional space but inevitably lose structural information. Mapper takes a different approach: it constructs a simplicial graph that preserves topological properties of the dataset.

Definition 7.1 (Reeb Graph — Underlying Idea)**.** Let $f : X \to \mathbb{R}$ be a continuous function on a topological space X. The *Reeb graph* $\mathcal{R}_f(X)$ is the quotient of X by the relation: $x \sim y$ iff $f(x) = f(y)$ and x, y are in the same connected component of $f^{-1}(f(x))$.

Mapper is a *discrete approximation* of the Reeb graph.

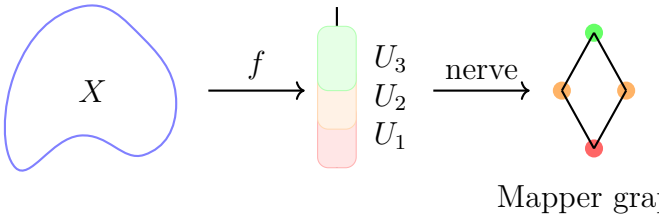
7.2 The Mapper Algorithm

Mapper Algorithm (Singh, Mémoli, Carlsson, 2007)

Inputs: Point cloud X, filter function $f : X \to \mathbb{R}^k$, cover $\mathcal{U}$ of $f(X)$, clustering algorithm $\mathcal{C}$.

1. **Filtering**: apply f to obtain $f(X) \subset \mathbb{R}^k$.
2. **Covering**: choose a cover $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ of $f(X)$ by open sets (typically overlapping intervals).
3. **Pullback and clustering**: for each U_α, compute $f^{-1}(U_\alpha) \cap X$ and apply clustering algorithm $\mathcal{C}$, yielding clusters $C_{\alpha,1}, \ldots, C_{\alpha,k_\alpha}$.
4. **Nerve construction**: build the simplicial complex whose vertices are the clusters and where $\{C_{\alpha_0,i_0}, \ldots, C_{\alpha_p,i_p}\}$ forms a simplex iff $C_{\alpha_0,i_0} \cap \cdots \cap C_{\alpha_p,i_p} \neq \emptyset$.

Output: the simplicial complex (often a graph, i.e., a 1-dimensional complex).



Mapper graph

7.3 Parameter Choices

7.3.1 Filter Function

Definition 7.2 (Common Filter Functions)**.**

- **Eccentricity**: $f(x) = \left(\frac{1}{n}\sum_{i=1}^{n} d(x, x_i)^p\right)^{1/p}$.
- **Density**: kernel density estimation (KDE), k-nearest neighbors.
- **Distance to k-th neighbor**: isolation measure.
- **Learned functions**: t-SNE components, UMAP, autoencoder.
- **Coordinate**: projection onto an axis, a principal component.

7.3.2 Cover

Definition 7.3 (Interval Cover)**.** For a filter function $f : X \to \mathbb{R}$, a standard cover is defined by:

- n_{int}: number of intervals.
- p_{ov}: overlap percentage ($0 < p_{\text{ov}} < 1$).

Each interval has width $w = \frac{\max f - \min f}{n_{\text{int}} - (n_{\text{int}}-1)p_{\text{ov}}}$ and the intervals are:

$$U_k = [\min f + k(1-p_{\text{ov}})w,\ \min f + k(1-p_{\text{ov}})w + w].$$

Parameter Sensitivity

Mapper's output depends strongly on the choice of filter function, number of intervals, overlap, and clustering algorithm. It is essential to vary these parameters and assess the robustness of the resulting graph.

7.4 Implementation with KeplerMapper

Mapper with KeplerMapper

```python
import numpy as np
import kmapper as km
from sklearn.datasets import make_circles
from sklearn.cluster import DBSCAN

# Data: two concentric circles
X, y = make_circles(n_samples=500, noise=0.05, factor=0.5)

# Initialize Mapper
mapper = km.KeplerMapper(verbose=1)

# Filter function: projection onto first coordinate
lens = mapper.fit_transform(X, projection=[0])

# Build Mapper graph
graph = mapper.map(
    lens, X,
    cover=km.Cover(n_cubes=15, perc_overlap=0.4),
    clusterer=DBSCAN(eps=0.3, min_samples=5)
)

# HTML visualization
mapper.visualize(
    graph,
    path_html="mapper_circles.html",
    title="Mapper: Concentric Circles",
    color_values=y
)
print(f"Number of nodes: {len(graph['nodes'])}")
print(f"Number of edges: {len(graph['links'])}")
```

7.5 Mapper with GUDHI

Manual Mapper Construction

```python
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KernelDensity

X = np.random.randn(200, 3)

# Filter: density
kde = KernelDensity(bandwidth=0.5)
kde.fit(X)
f_values = kde.score_samples(X)

# Cover
n_intervals = 10
overlap = 0.3
f_min, f_max = f_values.min(), f_values.max()
width = (f_max - f_min) / (n_intervals - (n_intervals-1) * overlap)

clusters_all = []
for k in range(n_intervals):
    start = f_min + k * (1 - overlap) * width
    end = start + width
    mask = (f_values >= start) & (f_values <= end)
    if mask.sum() < 2:
        continue
    X_local = X[mask]
    db = DBSCAN(eps=0.5, min_samples=3).fit(X_local)
    labels = db.labels_
    for l in set(labels):
        if l == -1:
            continue
        idx = np.where(mask)[0][labels == l]
        clusters_all.append(set(idx.tolist()))

# Build edges (non-empty intersection)
edges = []
for i in range(len(clusters_all)):
    for j in range(i+1, len(clusters_all)):
        if clusters_all[i] & clusters_all[j]:
            edges.append((i, j))

print(f"Nodes: {len(clusters_all)}, Edges: {len(edges)}")
```

7.6 Theoretical Foundations

Theorem 7.4 (Mapper Convergence — Carrière, Oudot, 2018)**.** *Under regularity conditions on f and X, as the number of points $n \to \infty$ and the cover parameters are chosen*

appropriately, the Mapper graph converges (in the functional interleaving distance) to the Reeb graph of (X, f).

Definition 7.5 (Functional Interleaving Distance)**.** Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two Reeb graphs. The *functional interleaving distance* is:

$$d_{\mathrm{FI}}(\mathcal{R}_1, \mathcal{R}_2) = \inf_{\varepsilon}\{\varepsilon \geq 0 : \mathcal{R}_1 \text{ and } \mathcal{R}_2 \text{ are } \varepsilon\text{-interleaved}\}.$$

7.7 Multidimensional Mapper

Definition 7.6 (2D Mapper)**.** When the filter function is $f : X \to \mathbb{R}^2$, the cover is a grid of overlapping rectangles in $\mathbb{R}^2$. The result is a simplicial complex of potentially dimension ≥ 2 (not just a graph).

2D Mapper with KeplerMapper

```python
import kmapper as km
import numpy as np
from sklearn.datasets import make_swiss_roll
from sklearn.cluster import DBSCAN

# Data: Swiss roll
X, color = make_swiss_roll(n_samples=1000, noise=0.5)

mapper = km.KeplerMapper()

# 2D filter: first and third coordinates
lens = mapper.fit_transform(X, projection=[0, 2])

graph = mapper.map(
    lens, X,
    cover=km.Cover(n_cubes=10, perc_overlap=0.3),
    clusterer=DBSCAN(eps=2.0, min_samples=5)
)

mapper.visualize(graph, path_html="mapper_swiss_roll.html",
                 color_values=color, title="Swiss Roll Mapper")
```

7.8 Famous Applications of Mapper

7.8.1 Breast Cancer (Lum et al., 2013)

The seminal biological study using Mapper revealed subgroups in breast cancer gene expression data that had not been detected by classical clustering methods.

7.8.2 Basketball Player Analysis (Lum et al., 2013)

Mapper was applied to NBA player statistics, revealing a "Y"-shaped structure corresponding to the three main roles (guard, forward, center).

7.8.3 Type 2 Diabetes (Li et al., 2015)

Mapper identified three distinct subtypes of type 2 diabetes from clinical data, with therapeutic implications.

7.9 Ball Mapper

Definition 7.7 (Ball Mapper — Dłotko, 2019)**.** *Ball Mapper* is a simplified variant of Mapper where:

1. Fix a radius $\varepsilon > 0$.
2. Select a subset of landmark points $L \subset X$ (by farthest point sampling).
3. Nodes are the balls $B(l, \varepsilon)$ for $l \in L$.
4. Two nodes are connected if their balls share common data points.

Advantage: no need to choose a filter function.

7.10 Exercises

Exercise 7.1. Apply Mapper to the iris dataset (`sklearn.datasets.load_iris`) with different filter functions (PCA, eccentricity, density). Compare the resulting graphs.

Exercise 7.2. Study the effect of the number of intervals and overlap percentage on the Mapper graph. For a noisy circle, determine parameters that produce a cyclic graph.

Exercise 7.3. Implement Ball Mapper in Python and apply it to a 3D point cloud.

Exercise 7.4. Show that if the cover is sufficiently fine and the clustering algorithm is exact, the Mapper graph is an approximation of the Reeb graph of (X, f).

Exercise 7.5. Download a real-world dataset (e.g., genomic expression data) and apply Mapper to identify subgroups. Interpret the resulting graph.

Chapter 8

Distances between Diagrams

For persistence diagrams to be truly useful, one must be able to *compare* them. Do two point clouds have the same topological shape? Does a noisy signal have the same structure as a clean one? Answering these questions requires a notion of distance between diagrams. Two families dominate the theory: the bottleneck distance, which measures the worst displacement needed to match the points of two diagrams, and the Wasserstein distances, which measure the total optimal transport cost. The choice is consequential: the bottleneck distance privileges the most persistent features (the "tall" bars), while Wasserstein integrates information from all bars, including the small ones. The computational algorithms — Hungarian assignment, auction methods — and the theoretical stability properties are the subject of this chapter.

Intuition

Persistence diagrams summarize the topological information of a filtered space. To compare two spaces, we need **distances** between diagrams. The choice of distance determines which topological differences we emphasize: the most persistent features (bottleneck) or the entire spectrum (Wasserstein).

8.1 Review of persistence diagrams

Definition 8.1 (Persistence diagram)**.** Let $\mathcal{F}$ be a filtration of a simplicial complex. The **persistence diagram** $\mathrm{Dgm}(\mathcal{F})$ is the multiset of points $(b_i, d_i) \in \mathbb{R}^2$ with $b_i < d_i$, where b_i is the birth value and d_i the death value of the i-th persistent homology class. We adjoin the diagonal $\Delta = \{(x, x) : x \in \mathbb{R}\}$ with infinite multiplicity.

Remark 8.2. Adding the diagonal with infinite multiplicity allows us to define bijections between diagrams of different sizes: a point in one diagram can be matched to a point on the diagonal, effectively treating it as noise.

8.2 The bottleneck distance

Definition 8.3 (Bottleneck distance)**.** Let D_1 and D_2 be two persistence diagrams. The **bottleneck distance** is:

$$d_\infty(D_1, D_2) = \inf_{\gamma} \sup_{x \in D_1} \|x - \gamma(x)\|_\infty$$

where the infimum is over all bijections $\gamma : D_1 \to D_2$ (using the diagonal to complete).

Intuition

The bottleneck distance measures the "worst-case matching cost." It is insensitive to many small features and only captures the maximum displacement among the most significant structures.

Example 8.4. Consider $D_1 = \{(0, 2), (1, 3)\}$ and $D_2 = \{(0, 2.5), (1, 3.2)\}$. The natural matching gives:

$$d_\infty(D_1, D_2) = \max\big(\|(0, 2) - (0, 2.5)\|_\infty,\ \|(1, 3) - (1, 3.2)\|_\infty\big) = \max(0.5, 0.2) = 0.5.$$

Proposition 8.5 (Metric property)**.** d_∞ is a metric on the space of persistence diagrams.

8.3 Wasserstein distances

Definition 8.6 (Wasserstein distance)**.** For $p \geq 1$ and $q \geq 1$, the **Wasserstein distance** $W_{p,q}$ between two diagrams D_1, D_2 is:

$$W_{p,q}(D_1, D_2) = \left(\inf_{\gamma} \sum_{x \in D_1} \|x - \gamma(x)\|_q^p \right)^{1/p}$$

where γ ranges over all bijections $D_1 \to D_2$. The case $q = \infty$, $p = \infty$ recovers the bottleneck distance.

Varying conventions

The literature uses different conventions for the Wasserstein distance indices. Some authors write W_p with $q = \infty$ implicit, others use $q = p$. Always check the convention in each paper.

Proposition 8.7 (Bottleneck–Wasserstein relationship)**.** For all $p \geq 1$:

$$d_\infty(D_1, D_2) = \lim_{p \to \infty} W_{p,\infty}(D_1, D_2).$$

Moreover, $d_\infty(D_1, D_2) \leq W_{p,\infty}(D_1, D_2)$ for all p.

Summary of distances

$$d_\infty(D_1, D_2) = \inf_{\gamma} \sup_{x \in D_1} \|x - \gamma(x)\|_\infty \tag{8.1}$$

$$W_{p,q}(D_1, D_2) = \left(\inf_{\gamma} \sum_{x \in D_1} \|x - \gamma(x)\|_q^p \right)^{1/p} \tag{8.2}$$

8.4 The stability theorem

Theorem 8.8 (Persistence stability — Cohen-Steiner, Edelsbrunner, Harer 2007)**.** *Let $f, g : X \to \mathbb{R}$ be continuous functions on a triangulable topological space X. Then:*

$$d_\infty\big(\mathrm{Dgm}(f),\ \mathrm{Dgm}(g)\big) \leq \|f - g\|_\infty .$$

Intuition

The stability theorem is the most important result in TDA. It guarantees that small perturbations of the data (in the L^∞ sense) can only produce small perturbations of the persistence diagram. This is what makes persistence **robust to noise**.

Corollary 8.9. *The map $f \mapsto \mathrm{Dgm}(f)$ is 1-Lipschitz from $\big(C(X,\mathbb{R}), \|\cdot\|_\infty\big)$ to $\big(\mathcal{D}, d_\infty\big)$.*

Theorem 8.10 (Wasserstein stability)**.** *Under additional regularity assumptions (tame Morse functions), for all $p \geq 1$:*

$$W_{p,\infty}(\mathrm{Dgm}(f), \mathrm{Dgm}(g)) \leq C_p \cdot \|f-g\|_\infty^{1-1/p} \cdot \|f-g\|_p^{1/p}$$

for a constant C_p depending on X and p.

8.5 The interleaving distance

Definition 8.11 (Persistence modules)**.** A **persistence module** is a functor $\mathbb{V} : (\mathbb{R}, \leq) \to \mathbf{Vec}_k$, i.e., a family of vector spaces $\{V_t\}_{t\in\mathbb{R}}$ with linear maps $v_{s,t} : V_s \to V_t$ for $s \leq t$ satisfying $v_{t,t} = \mathrm{id}$ and $v_{s,u} = v_{t,u} \circ v_{s,t}$ for $s \leq t \leq u$.

Definition 8.12 (ε-interleaving)**.** Two persistence modules $\mathbb{V}$ and $\mathbb{W}$ are **ε-interleaved** if there exist shifted module morphisms $\varphi : \mathbb{V} \to \mathbb{W}[\varepsilon]$ and $\psi : \mathbb{W} \to \mathbb{V}[\varepsilon]$ such that:

$$\psi[\varepsilon] \circ \varphi = v_{2\varepsilon}, \qquad \varphi[\varepsilon] \circ \psi = w_{2\varepsilon}$$

where $\mathbb{W}[\varepsilon]_t = W_{t+\varepsilon}$ is the shifted module.

Definition 8.13 (Interleaving distance)**.** The **interleaving distance** is:

$$d_I(\mathbb{V}, \mathbb{W}) = \inf\{\varepsilon \geq 0 : \mathbb{V} \text{ and } \mathbb{W} \text{ are } \varepsilon\text{-interleaved}\}.$$

Theorem 8.14 (Isometry theorem — Chazal et al. 2009)**.** *For interval-decomposable real-parameter persistence modules:*

$$d_I(\mathbb{V}, \mathbb{W}) = d_\infty(\mathrm{Dgm}(\mathbb{V}), \mathrm{Dgm}(\mathbb{W})).$$

Intuition

The isometry theorem shows that the bottleneck distance between diagrams coincides exactly with the interleaving distance between modules. This gives an **algebraic** interpretation of the bottleneck distance and grounds stability on solid categorical foundations.

8.6 Algebraic stability

Theorem 8.15 (Algebraic stability)**.** *Let $F : \mathbf{Top} \to \mathbf{Vec}_k^{(\mathbb{R},\leq)}$ be a persistent homology functor. For any pair of filtered spaces (X, f) and (Y, g) with an ε-interleaving morphism between the filtrations:*

$$d_I(H_*(X,f),\ H_*(Y,g)) \leq \varepsilon.$$

Remark 8.16. Algebraic stability generalizes the classical stability theorem to the setting of abstract persistence modules. It does not require continuous functions: it applies directly at the level of filtrations and module algebra.

8.7 Practical computation

Proposition 8.17 (Complexity)**.** The bottleneck distance between two diagrams of n and m points can be computed in $\mathcal{O}(N^{1.5} \log N)$ where $N = n + m$, by reduction to bipartite matching. The Wasserstein distance W_p can be computed in $\mathcal{O}(N^3)$ using the Hungarian algorithm.

```python
import numpy as np
from gudhi.wasserstein import wasserstein_distance
from gudhi.bottleneck import bottleneck_distance

# Two persistence diagrams (birth, death)
dgm1 = np.array([[0.0, 2.0], [1.0, 3.0], [2.0, 2.5]])
dgm2 = np.array([[0.0, 2.1], [1.1, 3.2]])

# Bottleneck distance
d_bn = bottleneck_distance(dgm1, dgm2)
print(f"Bottleneck: {d_bn:.4f}")

# Wasserstein distance (p=2)
d_w2 = wasserstein_distance(dgm1, dgm2, order=2)
print(f"Wasserstein-2: {d_w2:.4f}")
```

8.8 Exercises

Exercise 8.1. Compute by hand the bottleneck distance between $D_1 = \{(0,1), (0,3), (2,5)\}$ and $D_2 = \{(0, 1.5), (1,4)\}$. *Hint*: try different matchings and do not forget the diagonal.

Exercise 8.2. Show that $d_\infty(D_1, D_2) \leq W_{p,\infty}(D_1, D_2)$ for all $p \geq 1$.

Exercise 8.3. Let $f(x) = \sin(x)$ and $g(x) = \sin(x) + 0.1$ on $[0, 2\pi]$. Show that $d_\infty(\mathrm{Dgm}(f), \mathrm{Dgm}(g)) \leq 0.1$ using the stability theorem.

Exercise 8.4 (Interleaving distance)**.** Let $\mathbb{V}$ be the module k on $[0,2]$ and $\mathbb{W}$ the module k on $[0.5, 2.5]$. Compute $d_I(\mathbb{V}, \mathbb{W})$ directly from the definition.

Exercise 8.5 (Implementation)**.** Using `gudhi`, compare the bottleneck and Wasserstein-2 distances for persistence diagrams of two point clouds: a sample from the circle S^1 and a sample from the torus T^2 embedded in $\mathbb{R}^3$.

Chapter 9

Machine Learning with TDA

Persistence diagrams capture the "shape" of data, but machine learning algorithms — SVMs, random forests, neural networks — expect fixed-dimensional vectors. How does one bridge this gap? This is the problem of *vectorization*, one of the central challenges of applied TDA. Peter Bubenik (2015) proposed *persistence landscapes*, functions in a Banach space; Henry Adams and collaborators introduced *persistence images*; Mathieu Carrière et al. developed persistence kernels. More recently, differentiable persistence layers allow integrating TDA directly into deep neural networks, opening the way to end-to-end topological learning.

Intuition

Persistence diagrams live in a non-vectorial space: one cannot directly add them or multiply them by a scalar. To use them in a machine learning pipeline, we must **vectorize** them, i.e., transform them into representations living in a Hilbert space or a finite-dimensional feature space.

9.1 Persistence landscapes

Recall the persistence landscape introduced in Chapter 4: for a diagram $D = \{(b_i, d_i)\}$, the k-th landscape $\lambda_k(t)$ is the k-th largest value of the tent functions $\Lambda_i(t)$. Landscapes live in a Banach space, which allows computing means, variances, and performing statistical tests.

```python
from gudhi.representations import Landscape
import numpy as np

# Persistence diagram
dgm = np.array([[0.0, 1.5], [0.5, 2.0], [1.0, 3.0]])

# Compute first 3 landscapes on a grid
landscape = Landscape(num_landscapes=3, resolution=100)
L = landscape.fit_transform([dgm])
print(f"Shape: {L.shape}")  # (1, 300)
```

9.2 Persistence images

Definition 9.1 (Persistence image — Adams et al. 2017)**.** Let D be a persistence diagram. The **persistence image** is constructed in three steps:

1. **Coordinate change**: transform each (b_i, d_i) into $(b_i, d_i - b_i)$ (birth, persistence).
2. **Weighting**: apply a weight function $w : \mathbb{R}^2 \to \mathbb{R}_+$ (typically linear in persistence) to attenuate points close to the diagonal.
3. **Gaussian smoothing**: place a Gaussian $\mathcal{N}((b_i, p_i), \sigma^2 I)$ on each point and discretize on an $N \times N$ grid.

The resulting image is:

$$\rho(x, y) = \sum_{i=1}^{n} w(b_i, p_i) \cdot \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x - b_i)^2 + (y - p_i)^2}{2\sigma^2}\right).$$

Theorem 9.2 (Stability of persistence images)**.** *If w is Lipschitz and bounded, the map $D \mapsto \rho_D$ is Lipschitz with respect to the Wasserstein distance W_1:*

$$\|\rho_{D_1} - \rho_{D_2}\|_\infty \leq C \cdot W_1(D_1, D_2).$$

9.3 Persistence silhouettes

Recall the persistence silhouette of order q introduced in Chapter 4: it is the weighted average (by $(d_i - b_i)^q$) of the tent functions. For large q, it gives more weight to persistent features. The silhouette is an element of $L^p(\mathbb{R})$, hence directly usable as a feature in a machine learning pipeline.

9.4 Kernel methods

Definition 9.3 (Persistence kernel)**.** A **persistence kernel** is a positive definite kernel $K : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ on the space of persistence diagrams.

Definition 9.4 (Persistence scale kernel — Reininghaus et al. 2015)**.**

$$k_\sigma(D_1, D_2) = \frac{1}{8\pi\sigma} \sum_{p \in D_1} \sum_{q \in D_2} \left(e^{-\frac{\|p-q\|^2}{8\sigma}} - e^{-\frac{\|p-\bar{q}\|^2}{8\sigma}} \right)$$

where $\bar{q} = (d_q, b_q)$ is the point q reflected across the diagonal.

Proposition 9.5 (Positive definiteness)**.** The kernel k_σ is positive definite and stable: $|k_\sigma(D_1, D_1') - k_\sigma(D_2, D_2')| \leq C_\sigma(W_1(D_1, D_2) + W_1(D_1', D_2'))$.

Definition 9.6 (Fisher kernel — Le, Yamada 2018)**.** The **Fisher kernel** models each diagram as a distribution and uses the Fisher distance between distributions:

$$K_{\text{Fisher}}(D_1, D_2) = \exp\left(-\frac{1}{2t} d^2_{\text{Fisher}}(D_1, D_2)\right).$$

9.5 Integration into an ML pipeline

Example 9.7 (Complete classification pipeline).

1. **Data**: point clouds or time series.
2. **Filtration**: Rips, Alpha, or sublevel set.
3. **Persistence**: compute diagrams via `gudhi` or `ripser`.
4. **Vectorization**: persistence images, landscapes, or kernels.
5. **Model**: SVM, Random Forest, or logistic regression.

```python
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from gudhi.representations import (
    PersistenceImage, Landscape, DiagramSelector
)

# scikit-learn pipeline with topological features
pipe = Pipeline([
    ("selector", DiagramSelector(use=True)),
    ("persistence_image", PersistenceImage(
        bandwidth=0.1, resolution=[20, 20]
    )),
    ("classifier", SVC(kernel="rbf", C=10.0)),
])

# pipe.fit(X_train_diagrams, y_train)
# y_pred = pipe.predict(X_test_diagrams)
```

Choosing a vectorization

- **Landscapes** preserve Banach space structure and allow statistical tests.
- **Images** are better suited for CNNs since they produce 2D representations.
- **Kernels** are preferable with SVMs in low dimension.

9.6 Scikit-TDA and software ecosystem

Remark 9.8 (Python ecosystem for TDA).

- **GUDHI**: complete C++/Python library (filtrations, persistence, representations).
- **Ripser**: fast Rips persistence computation.
- **scikit-tda**: meta-package integrating Ripser, Persim (distances), KeplerMapper.
- **giotto-tda**: TDA pipeline compatible with scikit-learn, with C++ acceleration.

9.7 Exercises

Exercise 9.1. Compute the first two persistence landscapes of the diagram $D = \{(0, 3), (1, 4), (2, 5)\}$ and plot them.

Exercise 9.2. Show that the persistence silhouette of order $q = 1$ is the persistence-weighted average of the tent functions.

Exercise 9.3 (Implementation)**.** Build a complete pipeline that classifies point clouds sampled from a circle vs. a figure eight, using persistence images and an SVM.

Exercise 9.4. Show that the Reininghaus kernel is symmetric: $k_\sigma(D, D') = k_\sigma(D', D)$.

Exercise 9.5 (Experimental comparison)**.** Compare classification performance (accuracy, computation time) across landscapes, images, and persistence kernels on a dataset of your choice.

Exercise 9.6 (Persistence landscape stability)**.** Let D_1 and D_2 be two persistence diagrams and $\lambda_k^{(1)}$, $\lambda_k^{(2)}$ their respective k-th landscapes.

1. Show that for each diagram point (b, d), the tent function $\Lambda(t) = \max(0, \min(t - b, d - t))$ is 1-Lipschitz.
2. Using the fact that the kmax operation is 1-Lipschitz in ℓ^∞ norm on sorted vectors, show that:
$$\left\|\lambda_k^{(1)} - \lambda_k^{(2)}\right\|_{L^\infty} \leq W_\infty(D_1, D_2).$$
3. Generalize to the L^p case: show that $\left\|\lambda_k^{(1)} - \lambda_k^{(2)}\right\|_{L^p} \leq C_p \cdot W_p(D_1, D_2)^{1/p}$ and discuss the constant C_p.

Exercise 9.7 (Kernel SVM pipeline with persistence features)**.** The goal is to classify point clouds into three classes: circle, torus (2D cross-section), and sphere (sampled in 3D).

1. For each point cloud, compute the persistence diagram in homology degrees 0 and 1 via a Vietoris–Rips complex.
2. Build the Gram matrix of the Reininghaus kernel: $G_{ij} = k_\sigma(D_i, D_j)$ for a parameter σ chosen by cross-validation.
3. Train a kernel SVM with precomputed kernel (`kernel='precomputed'`) and report accuracy under 5-fold cross-validation.
4. Compare with a pipeline using concatenated persistence images (degrees 0 and 1) and an RBF SVM. Discuss the advantages and disadvantages of each approach.

Chapter 10

Applications

Intuition

TDA has found applications across numerous scientific domains. Its strength lies in its ability to extract **qualitative** information (number of holes, connected components, cavities) from quantitative data, in a manner that is **robust to noise** and **invariant under deformation**.

10.1 Shape analysis

Definition 10.1 (Topological shape descriptor)**.** Let $\mathcal{S} \subset \mathbb{R}^3$ be a triangulated surface. Several filtrations can be constructed:

1. **Geodesic filtration**: $f(x) = d_{\mathcal{S}}(x, x_0)$ for a base point x_0.
2. **Curvature filtration**: $f(x) = \kappa(x)$ where κ is the Gaussian curvature.
3. **Heat kernel signature**: $f(x) = h_t(x, x)$, the diagonal of the heat kernel at time t.

The persistence diagram of each filtration yields a topological descriptor of the shape.

Example 10.2 (3D shape classification)**.** Consider a database of 3D models (SHREC, ModelNet). For each object:

1. Compute the geodesic distance filtration from several base points.
2. Obtain a set of persistence diagrams in dimensions 0 and 1.
3. Vectorize (persistence images) and classify (SVM).

This approach is naturally invariant under isometry.

Proposition 10.3 (Invariance)**.** Topological descriptors from geodesic filtrations are invariant under isometry of the surface. Those from curvature filtrations are invariant under rigid motions.

10.2 Protein structure

Definition 10.4 (Topological representation of a protein)**.** A protein is modeled as a point cloud in $\mathbb{R}^3$ (the positions of the C_α atoms along the backbone). The Rips–Vietoris filtration on this point cloud captures:

- H_0: chain connectivity.
- H_1: loops and secondary structures (α-helices, β-sheets).
- H_2: cavities and pockets.

Example 10.5 (Binding pocket detection)**.** Protein binding pockets (sites where ligands bind) correspond to topological cavities (H_2). Persistent generators in H_2 identify these pockets: a point (b, d) with large persistence $d - b$ signals a **stable** cavity across scales.

```python
import numpy as np
from gudhi import RipsComplex

# C-alpha positions (extracted from a PDB file)
coords = np.loadtxt("protein_ca.xyz")

# Rips complex
rips = RipsComplex(points=coords, max_edge_length=12.0)
simplex_tree = rips.create_simplex_tree(max_dimension=3)
dgm = simplex_tree.persistence()

# Filter H2 (cavities)
h2 = [(b, d) for dim, (b, d) in dgm if dim == 2 and d - b > 1.0]
print(f"Number of significant pockets: {len(h2)}")
```

10.3 Time series analysis

Definition 10.6 (Takens embedding)**.** Let $(x_t)_{t=0}^{T}$ be a time series. The **Takens embedding** with dimension d and delay τ is:

$$\mathbf{x}_t = (x_t, x_{t+\tau}, x_{t+2\tau}, \ldots, x_{t+(d-1)\tau}) \in \mathbb{R}^d.$$

Theorem 10.7 (Takens, 1981)**.** *If the time series comes from a dynamical system on a manifold $\mathcal{M}$ of dimension m, then for $d > 2m$ generically, the Takens embedding is an embedding of $\mathcal{M}$ into $\mathbb{R}^d$.*

Intuition

The idea is to reconstruct the phase space of the dynamical system from a single scalar observation. Then, TDA on the reconstructed point cloud detects the topology of the attractor: a limit cycle gives $\beta_1 = 1$, a torus gives $\beta_1 = 2, \beta_2 = 1$, etc.

Example 10.8 (Periodicity detection)**.** For a periodic series $x_t = \sin(2\pi t/T)$, the Takens embedding with $d = 2$ traces an ellipse in $\mathbb{R}^2$. The H_1 diagram contains a highly persistent point, confirming periodicity.

```python
import numpy as np
from gtda.time_series import SingleTakensEmbedding

# Periodic time series
t = np.linspace(0, 10 * np.pi, 1000)
x = np.sin(t) + 0.1 * np.random.randn(len(t))

# Takens embedding
embedder = SingleTakensEmbedding(
    time_delay=10, dimension=3, parameters_type="fixed"
)
X_embedded = embedder.fit_transform(x)
```

10.4 Image analysis

Definition 10.9 (Sublevel set filtration for images)**.** Let $I : \{0, \dots, W\} \times \{0, \dots, H\} \to \mathbb{R}$ be a grayscale image. The **sublevel set filtration** is:

$$I^{-1}(-\infty, t] = \{(i, j) : I(i, j) \leq t\}.$$

The persistence of this filtration captures dark connected components (H_0) and bright regions surrounded by dark ones (H_1).

Example 10.10 (Topological segmentation)**.** In medical imaging, tumors appear as regions of different intensity. Persistent components in H_0 of the sublevel set filtration identify these regions robustly, without arbitrary thresholding.

10.5 Sensor networks

Definition 10.11 (Topological coverage)**.** A network of n sensors with range r covers a domain $\Omega \subset \mathbb{R}^2$. The Čech complex at radius r built on sensor positions allows detection of **coverage holes**: $\beta_1 > 0$ signals a hole in the coverage.

Theorem 10.12 (De Silva–Ghrist, 2007)**.** *If the Rips complex* Rips_{2r} *of the sensor positions has trivial reduced homology and the sensors have detection range* $r_s \geq 2r$*, then the domain is covered.*

Remark 10.13. The advantage of this approach is that it does not require exact sensor positions (only distances), and it works without coordinates.

10.6 Materials science

Example 10.14 (Porous materials analysis)**.** Porous materials (zeolites, aerogels) are characterized by their pore structure. TDA on the atomic representation captures:

- H_0: grains and aggregates.
- H_1: channels and tunnels.

- H_2: enclosed cavities.

The persistence of these features correlates with macroscopic properties (permeability, specific surface area).

Example 10.15 (Amorphous metallic glasses)**.** Nakamura et al. (2015) used persistence to distinguish metallic glasses from liquids: the H_1 and H_2 diagrams show persistent structures in the glass that are absent in the liquid, without requiring an order parameter.

10.7 Exercises

Exercise 10.1. Compute the H_0 and H_1 persistence diagrams of a time series $x_t = \sin(t) + 0.5\sin(3t)$ after Takens embedding with $d = 3$. Interpret the results.

Exercise 10.2. For a 100×100 binary image containing three disks and one annulus, predict the persistence diagram of the sublevel set filtration, then verify with `gudhi`.

Exercise 10.3 (Sensor network)**.** Randomly generate 50 sensors in the unit square. Use H_1 persistence to detect coverage holes as a function of the radius r. Plot the number of holes as a function of r.

Exercise 10.4 (Project)**.** Apply TDA to a real-world dataset of your choice (e.g., EEG signals, financial data, satellite images). Present the chosen filtration, the obtained diagrams, and their interpretation.

Chapter 11

TDA and Deep Learning

Intuition

Integrating TDA into deep learning poses a fundamental challenge: the computation of persistence is a priori **not differentiable**. Recent work has shown how to make persistence differentiable, enabling end-to-end training of neural networks with topological objectives.

11.1 Differentiable persistence

Definition 11.1 (Persistence as a function of filtrations)**.** Consider a simplicial complex K with a parametric filtration $f_\theta : K \to \mathbb{R}$ depending on parameters $\theta \in \mathbb{R}^p$. The persistence diagram $\mathrm{Dgm}(f_\theta)$ is a function of θ.

Theorem 11.2 (Differentiability — Gameiro et al. 2016, Poulenard et al. 2018)**.** *Let K be a finite simplicial complex and $f : K \to \mathbb{R}$ a generic filtration (distinct filtration values). The birth coordinates $b_i(\theta)$ and death coordinates $d_i(\theta)$ in the persistence diagram are differentiable functions of θ almost everywhere. More precisely:*

$$\frac{\partial b_i}{\partial \theta_j} = \frac{\partial f_\theta(\sigma_{b_i})}{\partial \theta_j}, \qquad \frac{\partial d_i}{\partial \theta_j} = \frac{\partial f_\theta(\sigma_{d_i})}{\partial \theta_j}$$

where σ_{b_i} and σ_{d_i} are the simplices that create and kill class i.

Non-differentiability points

Differentiability fails when two filtration values coincide (the birth and death simplices may change). In practice, one adds a small perturbation or uses subgradients.

Persistence gradient

$$\frac{\partial \mathrm{pers}_i}{\partial \theta} = \frac{\partial d_i}{\partial \theta} - \frac{\partial b_i}{\partial \theta} = \nabla_\theta f(\sigma_{d_i}) - \nabla_\theta f(\sigma_{b_i})$$

```
import torch
from topologylayer.nn import RipsLayer
```

```python
# Differentiable point cloud
X = torch.randn(50, 2, requires_grad=True)

# Differentiable persistence layer
rips_layer = RipsLayer(maxdim=1, sublevel=False)
dgm = rips_layer(X)

# Loss based on H1 persistence
loss = -dgm[1][:, 1].sum()  # maximize H1 persistences
loss.backward()
print(f"Gradient shape: {X.grad.shape}")
```

11.2 PersLay: a generic persistence layer

Definition 11.3 (PersLay — Carriere et al. 2020)**. PersLay** is a neural layer that transforms a persistence diagram $D = \{(b_i, d_i)\}_{i=1}^n$ into a fixed-dimension vector via:

$$\text{PersLay}(D) = \text{op}(w(p_i) \cdot \phi(p_i))_{i=1}^n$$

where:

- $w : \mathbb{R}^2 \to \mathbb{R}_+$ is a weight function (attenuating points near the diagonal).
- $\phi : \mathbb{R}^2 \to \mathbb{R}^q$ is a pointwise representation function (learned or fixed).
- op is a permutation-invariant aggregation (sum, max, top-k, attention).

Proposition 11.4 (Universality of PersLay)**.** With suitable choices of w, ϕ, and op, PersLay can uniformly approximate any continuous, permutation-invariant function on persistence diagrams.

Remark 11.5. PersLay unifies classical representations:

- **Landscapes**: ϕ is the tent function, op is top-k.
- **Images**: ϕ is a Gaussian, op is sum.
- **Silhouettes**: ϕ is the tent function, op is the weighted average.

11.3 Topological regularization

Definition 11.6 (Topological loss)**.** A **topological loss** is a term added to the cost function to encourage or penalize certain topological structures:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda \cdot \mathcal{L}_{\text{topo}}$$

where $\mathcal{L}_{\text{topo}}$ is computed from the persistence diagram.

Example 11.7 (Topological loss for segmentation)**.** In image segmentation, one can penalize spurious small holes in the prediction:

$$\mathcal{L}_{\text{topo}} = \sum_{(b_i,d_i)\in H_1} (d_i - b_i)^2$$

which forces the network to produce simply connected regions.

Theorem 11.8 (Convergence of topological regularization)**.** *Under regularity conditions (generic filtration, Lipschitz), stochastic gradient descent with topological loss converges to a critical point of* $\mathcal{L}_{\text{total}}$*.*

```python
import torch
import torch.nn as nn

class TopologicalLoss(nn.Module):
    """Penalize spurious holes in segmentation."""
    def __init__(self, lam=1.0):
        super().__init__()
        self.lam = lam

    def forward(self, pred, target, dgm_h1):
        ce_loss = nn.functional.cross_entropy(pred, target)
        # H1 persistences = d - b
        pers = dgm_h1[:, 1] - dgm_h1[:, 0]
        topo_loss = (pers ** 2).sum()
        return ce_loss + self.lam * topo_loss
```

11.4 Topological autoencoders

Definition 11.9 (Topological autoencoder — Moor et al. 2020)**.** A **topological autoencoder** is an autoencoder (Enc, Dec) whose loss includes a topological term:

$$\mathcal{L} = \|x - \text{Dec}(\text{Enc}(x))\|^2 + \lambda \cdot d_W(\text{Dgm}(X),\ \text{Dgm}(\text{Enc}(X)))^2$$

where d_W is a Wasserstein distance between the persistence diagrams of the input space X and the latent space $\text{Enc}(X)$.

Intuition

The idea is to preserve the **topology** of the data during dimensionality reduction. A standard autoencoder (or VAE) may collapse loops or merge connected components. The topological term penalizes such distortions.

Proposition 11.10 (Topological preservation)**.** If $\mathcal{L}_{\text{topo}} = 0$, then the Betti numbers of X and $\text{Enc}(X)$ coincide for the Rips filtration used.

11.5 Graph neural networks and persistent homology

Definition 11.11 (Learned graph filtration)**.** Let $G = (V, E)$ be a graph with node features $h_v \in \mathbb{R}^d$. Define a learned filtration:

$$f_\theta(v) = g_\theta(h_v), \qquad f_\theta(e_{uv}) = \max(f_\theta(u), f_\theta(v))$$

where $g_\theta : \mathbb{R}^d \to \mathbb{R}$ is a neural network. The persistence diagram of this filtration provides topological features of the graph.

Example 11.12 (Graph classification)**.** For molecular graph classification:

1. A GNN produces node embeddings h_v.
2. A learned filtration f_θ is computed.
3. The persistence diagram is vectorized via PersLay.
4. The resulting vector is concatenated with the GNN readout.

This combination improves expressiveness beyond the Weisfeiler-Leman test.

Theorem 11.13 (Topological expressiveness — Horn et al. 2022)**.** *There exist pairs of non-isomorphic graphs indistinguishable by the k-WL test but distinguishable by the persistence of the filtration induced by a message-passing GNN.*

11.6 Implementation with PyTorch

```python
import torch
import torch.nn as nn
from torch_geometric.nn import GCNConv, global_mean_pool

class TopoGNN(nn.Module):
    """GNN with topological features."""
    def __init__(self, in_dim, hidden_dim, out_dim):
        super().__init__()
        self.conv1 = GCNConv(in_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.filtration = nn.Linear(hidden_dim, 1)
        self.classifier = nn.Linear(hidden_dim + 16, out_dim)

    def forward(self, x, edge_index, batch):
        h = torch.relu(self.conv1(x, edge_index))
        h = torch.relu(self.conv2(h, edge_index))
        # Standard readout
        graph_emb = global_mean_pool(h, batch)
        # Learned filtration
        filt_vals = self.filtration(h).squeeze()
        # ... persistence computation and PersLay ...
        return graph_emb  # simplified
```

11.7 Exercises

Exercise 11.1. Compute by hand the gradient of the total persistence $\sum_i (d_i - b_i)$ with respect to filtration values for the complex $\{a, b, c, ab, bc\}$ with $f(a) = 0, f(b) = 1, f(c) = 2, f(ab) = 1, f(bc) = 2$.

Exercise 11.2. Show that PersLay with $\phi(b, d) = (b, d-b)$, $w \equiv 1$, and op = sum reduces to computing $(\sum b_i, \sum (d_i - b_i))$.

Exercise 11.3 (Implementation)**.** Implement a simple topological autoencoder on a circle-shaped dataset. Compare the latent space with and without the topological term.

Exercise 11.4. Give an example of two non-isomorphic graphs indistinguishable by 1-WL but distinguishable by the H_0 persistence of a degree-induced filtration.

Exercise 11.5 (Project)**.** Train an image segmentation network (U-Net) with and without topological regularization on a blood vessel dataset. Compare the Betti numbers of the predictions.

Bibliography

[1] Edelsbrunner, H. and Harer, J.L., *Computational Topology*, AMS, 2010.

[2] Carlsson, G., Topology and Data, *Bulletin of the AMS*, 46(2):255–308, 2009.

[3] Oudot, S.Y., *Persistence Theory: From Quiver Representations to Data Analysis*, AMS, 2015.

[4] Chazal, F. and Michel, B., *An Introduction to Topological Data Analysis*, Frontiers in AI, 2021.