Analyse Topologique des Données

Notes de Cours

Master M2 — 2025–2026

Yaë Ulrich Gaba

« La topologie est précisément la discipline mathématique qui permet le passage du local au global. »

— René Thom

25 mars 2026

Table des matières

Préface

L'analyse topologique des données (ATD, ou *Topological Data Analysis* — TDA en anglais) est un domaine relativement jeune, né au début des années 2000 à la confluence de la topologie algébrique, de la géométrie computationnelle et de la science des données. Son objectif fondamental est d'extraire des informations *qualitatives* et *structurelles* à partir de jeux de données complexes, en exploitant des invariants topologiques robustes au bruit et aux déformations continues.

Alors que les méthodes statistiques classiques reposent sur des modèles paramétriques ou sur des hypothèses de distribution, la TDA adopte une perspective radicalement différente : elle cherche à caractériser la *forme* des données. Cette approche s'est révélée particulièrement fructueuse dans des domaines aussi variés que la biologie moléculaire, les neurosciences, la science des matériaux, la finance quantitative et l'apprentissage profond.

Objectifs de ce cours

Ce cours est conçu pour des étudiants de niveau Master 2 ou doctorat, disposant d'une formation de base en :

- **Algèbre et topologie** : groupes, anneaux, modules ; espaces topologiques, compacité, connexité.
- **Analyse et probabilités** : espaces métriques, théorie de la mesure, convergence.
- **Algorithmique et programmation** : complexité, structures de données, Python.

À l'issue de ce cours, l'étudiant sera capable :

1. De construire et manipuler les principaux complexes simpliciaux (Vietoris–Rips, Čech, Alpha) associés à un nuage de points.
2. De calculer l'homologie simpliciale et l'homologie persistante, tant sur le plan théorique qu'algorithmique.
3. D'interpréter les diagrammes de persistance et les codes-barres, et d'en évaluer la stabilité.
4. D'appliquer l'algorithme Mapper pour la visualisation exploratoire.
5. D'intégrer les descripteurs topologiques dans des pipelines d'apprentissage automatique et d'apprentissage profond.
6. De mener un projet de recherche appliquant la TDA à des données réelles.

Organisation du cours

Le cours est structuré en onze chapitres, regroupés en quatre parties :

Partie I — Fondements (Chapitres 1–3) Motivations, complexes simpliciaux, homologie et introduction à l'homologie persistante.

Partie II — Théorie de la persistance (Chapitres 4–6) Diagrammes de persistance, théorèmes de stabilité et construction des complexes filtrés.

Partie III — Méthodes et algorithmes (Chapitres 7–8) Algorithme Mapper, distances entre diagrammes de persistance (Wasserstein, bottleneck).

Partie IV — TDA et apprentissage (Chapitres 9–11) Intégration avec le Machine Learning, applications concrètes et liens avec le Deep Learning.

Chaque chapitre contient des définitions rigoureuses, des théorèmes avec preuves (ou esquisses de preuves), des exemples détaillés, des implémentations en Python utilisant `gudhi`, `ripser` et `giotto-tda`, ainsi que des exercices de difficulté croissante.

Notations et conventions

Symbole	Signification
$\mathbb{R}^d$	Espace euclidien de dimension d
$\mathbb{N}, \mathbb{Z}, \mathbb{Q}$	Entiers naturels, entiers relatifs, rationnels
$\mathbb{F}$	Corps fini (souvent $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$)
K	Complexe simplicial abstrait
$\sigma = [v_0, \ldots, v_p]$	p-simplexe
$C_p(K)$	Groupe des p-chaînes de K
∂_p	Opérateur bord en dimension p
$H_p(K)$	p-ième groupe d'homologie de K
β_p	p-ième nombre de Betti ($= \dim H_p$)
$\mathrm{VR}(X, r)$	Complexe de Vietoris–Rips au paramètre r
$\check{C}(X, r)$	Complexe de Čech au paramètre r
Dgm_p	Diagramme de persistance en dimension p
d_B, W_p	Distance bottleneck, distance de Wasserstein d'ordre p
$\|\cdot\|$	Norme (euclidienne par défaut)
$\lvert S \rvert$	Cardinal de l'ensemble S
$\langle \cdot, \cdot \rangle$	Produit scalaire

Environnement logiciel

Les implémentations proposées dans ce cours utilisent **Python 3.10+** avec les bibliothèques suivantes :

- **GUDHI** (https://gudhi.inria.fr) : bibliothèque C++/Python développée par l'INRIA, offrant un large éventail de structures simpliciales, d'algorithmes de persistance et de fonctionnalités statistiques.
- **Ripser** (https://ripser.scikit-tda.org) : implémentation ultra-rapide de l'homologie persistante pour les complexes de Vietoris–Rips.
- **giotto-tda** (https://giotto-ai.github.io/gtda-docs) : bibliothèque orientée Machine Learning, compatible scikit-learn.
- **scikit-learn**, **NumPy**, **SciPy**, **Matplotlib** : prétraitement, calcul numérique et visualisation.
- **PyTorch** : pour les modèles d'apprentissage profond intégrant des couches topologiques.

Installation recommandée

Nous recommandons l'utilisation d'un environnement virtuel :

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

Repères historiques

L'analyse topologique des données puise ses racines dans des travaux fondateurs :

- **Années 1990** : Edelsbrunner, Letscher et Zomorodian introduisent les premières notions d'homologie persistante.
- **2002** : Zomorodian et Carlsson formalisent l'homologie persistante et son algorithme de calcul via la décomposition en modules gradués.
- **2005** : Cohen-Steiner, Edelsbrunner et Harer démontrent le théorème de stabilité fondamental pour les diagrammes de persistance.
- **2007** : Singh, Mémoli et Carlsson introduisent l'algorithme Mapper.
- **2009** : Chazal, Cohen-Steiner, Glisse, Guibas et Oudot étendent les résultats de stabilité aux distances de Wasserstein.
- **2015–présent** : Intégration croissante de la TDA avec le Machine Learning et le Deep Learning.

Remerciements

Ce cours doit beaucoup aux travaux pionniers de Gunnar Carlsson, Herbert Edelsbrunner, Frédéric Chazal, Steve Oudot, et de toute la communauté TDA. Nous remercions également les développeurs des bibliothèques GUDHI, Ripser et giotto-tda pour la qualité de leurs outils.

Intuition

La TDA repose sur une idée simple mais profonde : *la forme des données contient de l'information*. Les outils de la topologie algébrique permettent de quantifier cette forme de manière rigoureuse, stable et calculable. Ce cours vous donnera les clés pour exploiter cette idée dans vos propres recherches.

Les auteurs, mars 2026

Chapitre 1

Motivations et Vue d'Ensemble

Imaginez un nuage de points dans un espace de grande dimension. Les statistiques classiques vous donneront sa moyenne, sa variance, ses composantes principales. Mais elles ne vous diront pas s'il y a un « trou » au milieu, une boucle, ou deux amas séparés par un vide. Ces caractéristiques *topologiques* — la forme globale des données — sont pourtant souvent les plus informatives. C'est l'intuition fondatrice de l'*analyse topologique des données* (TDA), un domaine né au début des années 2000 des travaux de Gunnar Carlsson, Herbert Edelsbrunner et d'autres, à la frontière entre la topologie algébrique, la géométrie computationnelle et la science des données.

Intuition

Pourquoi avons-nous besoin de la topologie pour analyser des données ? Parce que la *forme* des données — leurs trous, leurs composantes connexes, leurs cavités — porte une information que les statistiques classiques ne capturent pas.

1.1 Le problème fondamental

Considérons un nuage de n points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$. Ce nuage est souvent un échantillon bruité d'un espace topologique sous-jacent $\mathcal{M}$ (une variété, une courbe, une surface…). Le problème central de la TDA est :

Question fondamentale

Comment récupérer les propriétés topologiques de $\mathcal{M}$ à partir de l'échantillon fini X ?

Exemple 1.1. Soit $\mathcal{M} = S^1$ le cercle unité dans $\mathbb{R}^2$. Un échantillon de 100 points bruités sur ce cercle forme un anneau. Topologiquement, S^1 possède une composante connexe ($\beta_0 = 1$) et un trou ($\beta_1 = 1$). La TDA permet de retrouver ces invariants à partir du nuage de points.

1.2 Limites des approches classiques

Les méthodes statistiques et d'apprentissage classiques présentent plusieurs limitations face aux données de forme complexe :

1. **Sensibilité au système de coordonnées** : les statistiques descriptives (moyenne, variance) dépendent du repère choisi. La topologie est invariante par homéomorphisme.

2. **Perte d'information structurelle** : le clustering identifie des composantes connexes mais ignore les trous et cavités.

3. **Fléau de la dimension** : en haute dimension, les distances euclidiennes deviennent peu informatives, mais les invariants topologiques restent significatifs.

4. **Absence de multi-échelle** : fixer un seuil de distance est arbitraire ; la persistance explore *toutes* les échelles simultanément.

1.3 La philosophie de la TDA

La TDA repose sur trois principes directeurs :

Définition 1.2 (Principes de la TDA)**.**

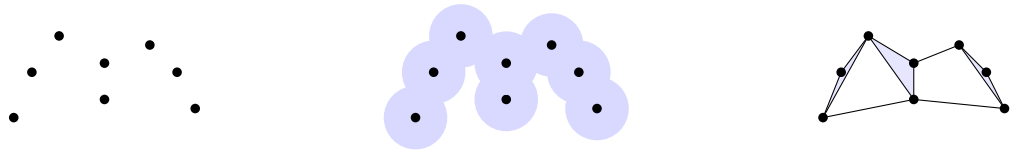
1. **Invariance par déformation** : les propriétés topologiques sont préservées par déformation continue (homéomorphisme, équivalence d'homotopie).

2. **Approche multi-échelle** : plutôt que de fixer un paramètre de résolution, on étudie l'évolution de la topologie sur tout le spectre de résolutions.

3. **Stabilité** : de petites perturbations des données produisent de petites perturbations des descripteurs topologiques.

1.4 Illustration visuelle : du nuage au complexe

La construction fondamentale de la TDA consiste à épaissir progressivement un nuage de points en un objet géométrique dont on peut calculer la topologie.



(a) Nuage de points X (b) Boules $B(x_i, r)$ (c) Complexe simplicial

1.5 Nombres de Betti : une première intuition

Les nombres de Betti sont les invariants topologiques les plus simples. Intuitivement :

Nombres de Betti

$$\beta_0 = \text{nombre de composantes connexes} \tag{1.1}$$
$$\beta_1 = \text{nombre de trous (boucles)} \tag{1.2}$$
$$\beta_2 = \text{nombre de cavités (vides enclos)} \tag{1.3}$$
$$\beta_p = \text{nombre de “trous” de dimension } p \tag{1.4}$$

Exemple 1.3.

- Un disque plein : $\beta_0 = 1$, $\beta_1 = 0$.
- Un cercle S^1 : $\beta_0 = 1$, $\beta_1 = 1$.
- Un tore T^2 : $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = 1$.
- Une sphère S^2 : $\beta_0 = 1$, $\beta_1 = 0$, $\beta_2 = 1$.

1.6 Pipeline typique de la TDA

Le processus général d'analyse topologique suit les étapes :

Pipeline TDA standard

1. **Données** : nuage de points $X \subset \mathbb{R}^d$ ou matrice de distances.
2. **Filtration** : construire une famille emboîtée de complexes simpliciaux $K_{r_1} \subseteq K_{r_2} \subseteq \cdots$.
3. **Homologie persistante** : calculer la naissance et la mort de chaque caractéristique topologique.
4. **Représentation** : encoder le résultat en diagramme de persistance ou code-barres.
5. **Vectorisation** : transformer en vecteur de caractéristiques pour l'apprentissage.
6. **Analyse/Prédiction** : appliquer des méthodes statistiques ou de ML.

Premier exemple avec Ripser

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

# Générer un cercle bruité
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
# Calculer l'homologie persistante
result = ripser(X, maxdim=1)
diagrams = result['dgms']

# Visualiser
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].scatter(X[:, 0], X[:, 1], s=10)
axes[0].set_title("Nuage de points")
axes[0].set_aspect('equal')
plot_diagrams(diagrams, ax=axes[1])
axes[1].set_title("Diagramme de persistance")
plt.tight_layout()
plt.savefig("premier_exemple_tda.pdf")
```

1.7 Aperçu des applications

La TDA a démontré son utilité dans de nombreux domaines :

1.7.1 Biologie et médecine

- **Analyse de la forme des protéines** : les nombres de Betti persistants caractérisent les cavités et tunnels des structures moléculaires.
- **Génomique** : Mapper révèle des sous-groupes dans les données d'expression génique du cancer du sein.
- **Imagerie médicale** : détection de tumeurs via l'homologie persistante des images.

1.7.2 Neurosciences

- **Connectome cérébral** : analyse topologique des réseaux neuronaux.
- **Codes neuronaux** : l'homologie persistante révèle la structure des représentations spatiales (cellules de lieu).

1.7.3 Science des matériaux

- **Matériaux amorphes** : caractérisation de la structure à moyenne portée.
- **Matériaux poreux** : quantification de la porosité via les nombres de Betti.

1.7.4 Données temporelles et signaux

- **Détection d'anomalies** : les changements topologiques signalent des régimes anormaux.
- **Reconstruction de Takens** : plongement d'une série temporelle et analyse de la forme résultante.

1.8 Formalisme mathématique : un premier aperçu

Nous terminons ce chapitre d'introduction par un aperçu formel des objets principaux, qui seront développés dans les chapitres suivants.

Définition 1.4 (Filtration)**.** Une *filtration* est une famille de sous-espaces topologiques (ou de complexes simpliciaux) $\{K_r\}_{r\in\mathbb{R}}$ telle que :

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

Définition 1.5 (Module de persistance)**.** Un *module de persistance* est un foncteur $\mathbf{V} : (\mathbb{R}, \leq) \to \mathbf{Vec}_{\mathbb{F}}$ qui associe à chaque $r \in \mathbb{R}$ un espace vectoriel V_r et à chaque paire $r \leq s$ une application linéaire $v_r^s : V_r \to V_s$.

Théorème 1.6 (Décomposition des modules de persistance)**.** *Tout module de persistance* pointwise finite-dimensional *(p.f.d.) se décompose de manière unique (à isomorphisme près) en somme directe de modules d'intervalle :*

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

où $\mathbb{I}[b, d)$ est le module qui vaut $\mathbb{F}$ sur $[b, d)$ et 0 ailleurs, avec applications identité sur l'intervalle.

1.9 Pourquoi la topologie et pas seulement la géométrie ?

Remarque 1.7. La géométrie étudie des propriétés métriques (distances, courbures, angles) qui dépendent du plongement. La topologie étudie des propriétés invariantes par déformation continue : elle est plus grossière mais plus robuste. La TDA exploite cette robustesse pour extraire l'information essentielle.

Par exemple, une tasse et un beignet sont topologiquement identiques (tous deux ont un trou), bien qu'ils soient géométriquement très différents.

1.10 Exercices

Exercice 1.1. Générer un nuage de 200 points échantillonnés uniformément sur un tore dans $\mathbb{R}^3$ (paramétrisation standard avec $R = 2, r = 1$). Ajouter un bruit gaussien de variance $\sigma^2 = 0.05$. Calculer l'homologie persistante avec `ripser` et vérifier que l'on retrouve $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = 1$.

Exercice 1.2. Expliquer pourquoi les nombres de Betti seuls ne suffisent pas à distinguer tous les espaces topologiques. Donner un exemple de deux espaces ayant les mêmes nombres de Betti mais qui ne sont pas homéomorphes.

Exercice 1.3. Implémenter le calcul naïf de la matrice de distances deux à deux pour un nuage de n points dans $\mathbb{R}^d$. Quelle est la complexité ? Comment cette matrice est-elle utilisée dans la construction d'un complexe de Vietoris–Rips ?

Exercice 1.4. Rechercher dans la littérature un article récent (post-2020) utilisant la TDA dans un domaine d'application de votre choix. Résumer la méthodologie en une page : quelles données, quel complexe, quels descripteurs topologiques, quel résultat principal ?

CHAPITRE 1. MOTIVATIONS, APPLICATIONS ET VUE D'ENSEMBLE

Chapitre 2

Complexes Simpliciaux et Homologie

Comment compter les « trous » d'un espace ? La question peut sembler vague, mais la topologie algébrique — née des travaux de Poincaré à la fin du XIX^{e} siècle — y apporte une réponse précise : l'*homologie*. L'idée est de découper l'espace en briques simples (des triangles, des tétraèdres, leurs analogues en dimension supérieure) et d'appliquer de l'algèbre linéaire pour détecter les cycles qui ne sont pas des bords. Ce chapitre construit cette machinerie pas à pas.

Intuition

Les complexes simpliciaux sont les briques élémentaires de la TDA : ils généralisent les graphes en dimensions supérieures. L'homologie est l'outil algébrique qui permet de compter les trous de chaque dimension dans ces complexes.

2.1 Simplexes

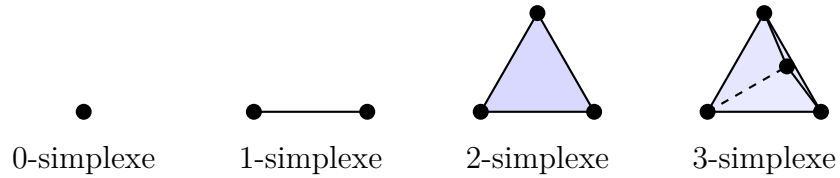
Définition 2.1 (Simplexe géométrique)**.** Un *p-simplexe géométrique* est l'enveloppe convexe de $p+1$ points *affinement indépendants* $v_0, \dots, v_p \in \mathbb{R}^d$:

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

L'entier p est la *dimension* de σ.

Exemple 2.2.

- 0-simplexe : un point (sommet).
- 1-simplexe : un segment (arête).
- 2-simplexe : un triangle (plein).
- 3-simplexe : un tétraèdre (plein).



Définition 2.3 (Face)**.** Une *face* d'un p-simplexe $\sigma = [v_0, \ldots, v_p]$ est tout simplexe obtenu en supprimant un ou plusieurs sommets. La *i-ème face opposée* est :

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

où $\hat{v}_i$ indique l'omission du sommet v_i.

2.2 Complexes simpliciaux abstraits

Définition 2.4 (Complexe simplicial abstrait)**.** Un *complexe simplicial abstrait* est une paire $K = (V, \Sigma)$ où V est un ensemble fini de sommets et $\Sigma \subseteq 2^V$ est une collection de sous-ensembles non vides de V telle que :

1. Pour tout $v \in V$, $\{v\} \in \Sigma$.
2. Si $\sigma \in \Sigma$ et $\tau \subseteq \sigma$ avec $\tau \neq \emptyset$, alors $\tau \in \Sigma$.

Les éléments de Σ sont appelés *simplexes* et la condition (2) est la *propriété de fermeture par faces*.

Définition 2.5 (Dimension)**.** La *dimension* d'un simplexe σ est $\dim \sigma = |\sigma| - 1$. La *dimension* du complexe est $\dim K = \max_{\sigma \in \Sigma} \dim \sigma$.

Exemple 2.6. Considérons $V = \{a, b, c, d\}$ et :

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

C'est un complexe simplicial de dimension 2. Il contient un 2-simplexe $\{a, b, c\}$ (triangle plein), quatre 1-simplexes (arêtes) et quatre 0-simplexes (sommets).

2.3 Chaînes, bords et cycles

Fixons un corps $\mathbb{F}$ (typiquement $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$).

Définition 2.7 (Groupe des chaînes)**.** Le *groupe des p-chaînes* de K à coefficients dans $\mathbb{F}$ est l'espace vectoriel :

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

Un élément de C_p est une combinaison linéaire formelle de p-simplexes.

Définition 2.8 (Opérateur bord)**.** L'*opérateur bord* $\partial_p : C_p(K) \to C_{p-1}(K)$ est défini sur les générateurs par :

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

Propriété fondamentale du bord

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

En d'autres termes, *le bord d'un bord est nul.*

Proposition 2.9. Pour tout $p \geq 0$, on a $\partial_{p-1} \circ \partial_p = 0$.

Démonstration. Soit $\sigma = [v_0, \ldots, v_p]$. Alors :

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

où $j' = j$ si $j < i$ et $j' = j - 1$ si $j > i$. Chaque terme apparaît exactement deux fois avec des signes opposés, donc la somme est nulle. □

Définition 2.10 (Cycles et bords)**.**

- Le groupe des *p-cycles* est $Z_p(K) = \ker \partial_p$.
- Le groupe des *p-bords* est $B_p(K) = \operatorname{im} \partial_{p+1}$.
- Comme $\partial_p \circ \partial_{p+1} = 0$, on a $B_p(K) \subseteq Z_p(K)$.

2.4 Homologie simpliciale

Définition 2.11 (Groupe d'homologie)**.** Le *p-ième groupe d'homologie* de K à coefficients dans $\mathbb{F}$ est le quotient :

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

Le *p-ième nombre de Betti* est $\beta_p = \dim_{\mathbb{F}} H_p(K; \mathbb{F})$.

Formule des nombres de Betti

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

Théorème 2.12 (Caractéristique d'Euler–Poincaré)**.** *Si K est un complexe simplicial de dimension d, sa* caractéristique d'Euler *vérifie :*

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

où Σ_p désigne l'ensemble des p-simplexes de K.

2.5 Calcul matriciel de l'homologie

En pratique, le calcul de l'homologie se ramène à de l'algèbre linéaire. Fixons un ordre sur les simplexes de chaque dimension. L'opérateur ∂_p est alors représenté par une matrice D_p à coefficients dans $\mathbb{F}$.

Calcul des nombres de Betti

1. Construire les matrices de bord D_p pour $p = 1, \ldots, d$.
2. Calculer $r_p = \text{rang}(D_p)$ (par réduction de Smith ou élimination de Gauss sur $\mathbb{F}$).
3. Les nombres de Betti sont : $\beta_p = |\Sigma_p| - r_p - r_{p+1}$.

Calcul de l'homologie avec des matrices

```python
import numpy as np

def boundary_matrix(simplices_p, simplices_pm1):
    """Matrice de bord sur 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):
    """Calcule beta_0, ..., beta_dim sur 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 Exemples de calcul

Exemple 2.13 (Triangle creux)**.** Soit K le complexe formé de trois sommets $\{a, b, c\}$ et trois arêtes $\{a, b\}, \{b, c\}, \{a, c\}$, *sans* le 2-simplexe $\{a, b, c\}$.

- $D_1 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ (sur $\mathbb{F}_2$), de rang 2.
- $\beta_0 = 3 - 2 = 1$ (une composante connexe).
- $\beta_1 = 3 - 2 - 0 = 1$ (un trou — le triangle est creux).

Exemple 2.14 (Triangle plein)**.** Ajoutons le 2-simplexe $\{a, b, c\}$ au complexe précédent. D_2 est le vecteur colonne $(1, 1, 1)^T$ sur $\mathbb{F}_2$, de rang 1.

- $\beta_0 = 3 - 2 = 1$.
- $\beta_1 = 3 - 2 - 1 = 0$ (le trou est rempli).
- $\beta_2 = 1 - 1 = 0$.

2.7 Homologie relative et suites exactes

Définition 2.15 (Homologie relative)**.** Soit $L \subseteq K$ un sous-complexe. L'*homologie relative* est :

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

où $\bar{\partial}_p$ est induit par ∂_p sur le quotient $C_p(K)/C_p(L)$.

Théorème 2.16 (Suite exacte longue du couple)**.** *Il existe une suite exacte longue :*

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

2.8 Homologie avec GUDHI

Calcul d'homologie avec GUDHI

```python
import gudhi

# Construire un complexe simplicial
st = gudhi.SimplexTree()
st.insert([0, 1])
st.insert([1, 2])
st.insert([0, 2])
# Triangle creux: pas de 2-simplexe [0,1,2]

# Nombres de Betti
betti = st.betti_numbers()
print(f"Nombres de Betti: {betti}")
# Sortie: Nombres de Betti: {0: 1, 1: 1}
```

```python
# Maintenant, remplir le triangle
st.insert([0, 1, 2])
betti = st.betti_numbers()
print(f"Nombres de Betti: {betti}")
# Sortie: Nombres de Betti: {0: 1}
```

2.9 Homologie réduite et caractéristique d'Euler

Définition 2.17 (Homologie réduite)**.** L'*homologie réduite* $\tilde{H}_p(K)$ est définie en augmentant le complexe de chaînes par l'application $\varepsilon : C_0(K) \to \mathbb{F}$, $\varepsilon(\sum \lambda_i v_i) = \sum \lambda_i$. On a :

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

En particulier, $\tilde{\beta}_0 = \beta_0 - 1$ pour un complexe non vide.

2.10 Applications simpliciales et fonctorialité

Définition 2.18 (Application simpliciale)**.** Soient K et L deux complexes simpliciaux. Une *application simpliciale* $f : K \to L$ est une application $f : V_K \to V_L$ sur les sommets telle que pour tout simplexe $\sigma = \{v_0, \ldots, v_p\} \in K$, on ait $f(\sigma) = \{f(v_0), \ldots, f(v_p)\} \in L$.

Proposition 2.19. Toute application simpliciale $f : K \to L$ induit une application linéaire $f_* : H_p(K) \to H_p(L)$ pour tout p. Cette assignation est fonctorielle : $(g \circ f)_* = g_* \circ f_*$ et $(\mathrm{id}_K)_* = \mathrm{id}_{H_p(K)}$.

2.11 Exercices

Exercice 2.1. Calculer à la main les nombres de Betti du complexe simplicial formé par les faces d'un tétraèdre (sans l'intérieur). Vérifier la formule de la caractéristique d'Euler.

Exercice 2.2. Montrer que pour un graphe connexe G avec n sommets et m arêtes, $\beta_0 = 1$ et $\beta_1 = m - n + 1$.

Exercice 2.3. Implémenter en Python la construction de la matrice de bord D_1 pour un graphe donné par sa liste d'arêtes. Vérifier que $D_0 \cdot D_1 = 0$ (sur $\mathbb{F}_2$) sur un exemple.

Exercice 2.4. Calculer l'homologie de la bouteille de Klein triangulée (9 sommets, 27 arêtes, 18 triangles). Comparer les résultats sur $\mathbb{F}_2$ et sur $\mathbb{Q}$.

Exercice 2.5. Montrer que si K est un cône (i.e., il existe un sommet v tel que $\sigma \cup \{v\} \in K$ pour tout $\sigma \in K$), alors $\tilde{H}_p(K) = 0$ pour tout p.

Chapitre 3

Homologie Persistante

L'homologie classique répond à la question : « combien de trous a cet espace ? »Mais en analyse topologique des données, l'espace lui-même n'est pas donné : on n'a qu'un nuage de points, et la topologie dépend du paramètre d'échelle choisi. À petite échelle, chaque point est isolé ; à grande échelle, tout fusionne en une seule composante connexe. La question devient alors : « quelles caractéristiques topologiques *persistent* à travers les échelles ? »C'est l'idée fondatrice de l'homologie persistante, développée au début des années 2000 par Herbert Edelsbrunner, David Letscher et Afra Zomorodian, puis enrichie par Gunnar Carlsson et ses collaborateurs.

Le résultat clé est que la persistance peut être représentée par un *diagramme de persistance* : un ensemble de points dans le plan, où chaque point (b, d) encode la naissance et la mort d'une caractéristique topologique. Les points loin de la diagonale correspondent à des structures robustes ; ceux près de la diagonale sont du bruit. Cette représentation, à la fois simple et puissante, est devenue l'outil central de la TDA.

Intuition

L'homologie classique décrit la topologie d'un espace fixé. L'homologie *persistante* étudie comment cette topologie *évolue* lorsqu'on parcourt une famille emboîtée de complexes : elle détecte quand une caractéristique topologique naît, et quand elle meurt.

3.1 Filtrations

Définition 3.1 (Filtration d'un complexe simplicial)**.** Une *filtration* d'un complexe simplicial K est une suite croissante de sous-complexes :

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

telle que pour tout i, K_i est un sous-complexe de K.

Définition 3.2 (Filtration par valeurs)**.** De manière équivalente, une filtration peut être décrite par une fonction $f : K \to \mathbb{R}$ telle que pour tout $\sigma \in K$ et toute face $\tau \subseteq \sigma$, on a $f(\tau) \leq f(\sigma)$. Le complexe filtré au niveau r est alors :

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

Exemple 3.3 (Filtration de Vietoris–Rips)**.** Étant donné un nuage de points $X \subset \mathbb{R}^d$, la filtration de Vietoris–Rips est définie par :

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

Lorsque r croît, de plus en plus de simplexes apparaissent.

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

3.2 Homologie persistante : définition

Définition 3.4 (Groupes d'homologie persistante)**.** Étant donnée une filtration $K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m$, l'inclusion $\iota_{i,j} : K_i \hookrightarrow K_j$ $(i \leq j)$ induit une application linéaire en homologie :

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

Le *p-ième groupe d'homologie persistante* de K_i à K_j est :

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

Le *nombre de Betti persistant* est $\beta_p^{i,j} = \dim H_p^{i,j}$.

Définition 3.5 (Naissance et mort)**.** Une classe d'homologie $\gamma \in H_p(K_i)$ *naît* au temps i si $\gamma \notin \mathrm{im}(\iota_{i-1,i}^p)$. Elle *meurt* au temps $j > i$ si $\iota_{i,j-1}^p(\gamma) \neq 0$ mais $\iota_{i,j}^p(\gamma) = 0$. Sa *persistance* est $j - i$.

Nombres de Betti persistants

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

3.3 Le théorème de décomposition

Définition 3.6 (Module de persistance)**.** Un *module de persistance* (sur $\mathbb{F}$) est un diagramme de la forme :

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

où les V_i sont des espaces vectoriels de dimension finie et les φ_i sont des applications linéaires.

Théorème 3.7 (Décomposition — Crawley-Boevey, Gabriel)**.** *Tout module de persistance* pointwise finite-dimensional *sur* $(\mathbb{R}, \leq)$ *se décompose de manière unique (isomorphisme et permutation) en somme directe de modules d'intervalle :*

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

où $\mathbb{I}[b, d)$ *vaut* $\mathbb{F}$ *sur* $[b, d)$ *et* 0 *ailleurs. La collection de paires* $\{(b_k, d_k)\}$ *est le* diagramme de persistance.

Remarque 3.8. Ce théorème est l'analogue pour les modules de persistance du théorème de structure des modules sur un anneau principal (théorème de classification des modules gradués sur $\mathbb{F}[t]$ dû à Zomorodian–Carlsson, 2005).

3.4 L'algorithme standard

L'algorithme de réduction des matrices de bord est la méthode classique pour calculer l'homologie persistante.

Définition 3.9 (Matrice de bord filtrée)**.** Ordonnons les simplexes $\sigma_1, \ldots, \sigma_N$ de K de sorte que $f(\sigma_i) \leq f(\sigma_j)$ pour $i < j$ (où f est la fonction de filtration), et que les faces apparaissent avant les simplexes qui les contiennent. La *matrice de bord filtrée* D est la matrice $N \times N$ dont l'entrée (i, j) est le coefficient de σ_i dans $\partial\sigma_j$.

Réduction de la matrice de bord

1. Soit D la matrice de bord filtrée ($N \times N$).
2. Définir $\text{low}(j)$ comme l'indice de la ligne la plus basse non nulle de la colonne j (ou -1 si la colonne est nulle).
3. Pour $j = 1, \ldots, N$:
   (a) Tant qu'il existe $j' < j$ tel que $\text{low}(j') = \text{low}(j) \neq -1$:
   (b) Ajouter la colonne j' à la colonne j (sur $\mathbb{F}$).
4. Après réduction :
   - Si $\text{low}(j) = i$, alors (σ_i, σ_j) forme une paire naissance-mort.
   - Si la colonne j est nulle et j n'est l'image d'aucun low, alors σ_j est un générateur essentiel (qui ne meurt jamais).

Réduction de matrice en Python

```python
def reduce_boundary_matrix(D):
    """Réduction standard de la matrice de bord (sur F_2).
    D: liste de sets, D[j] = ensemble des indices non nuls de col j.
    Retourne: paires (birth, death) et générateurs essentiels.
    """
    n = len(D)
    low = {}  # low[j] = indice le plus bas non nul de col j
    pairs = []

    for j in range(n):
        # Réduire la colonne j
        while D[j] and max(D[j]) in low:
            j_prime = low[max(D[j])]
            D[j] = D[j].symmetric_difference(D[j_prime])

        if D[j]:
```

```
            i = max(D[j])
            low[j] = i  # confusion: should be low[j]=i => pivot
            # Correction: stocker dans un dict pivot -> colonne
            pairs.append((i, j))
        # sinon: colonne nulle => potentiel générateur essentiel

    return pairs

# Exemple: triangle creux [a, b, c] avec arêtes
# Ordre: a(0), b(1), c(2), ab(3), bc(4), ac(5)
D = [set(), set(), set(),  # colonnes des 0-simplexes
     {0, 1}, {1, 2}, {0, 2}]  # colonnes des 1-simplexes
pairs = reduce_boundary_matrix(D)
print("Paires (naissance, mort):", pairs)
```

3.5 Complexité algorithmique

Théorème 3.10 (Complexité de l'algorithme standard)**.** *L'algorithme de réduction de la matrice de bord a une complexité en $O(N^3)$ dans le pire cas, où N est le nombre total de simplexes dans la filtration.*

Remarque 3.11. En pratique, la matrice est très creuse et l'algorithme est bien plus rapide. Des optimisations comme la *clearing optimization* et le *twist* réduisent considérablement le temps de calcul. L'implémentation `ripser` exploite ces optimisations et la réduction implicite de la matrice.

3.6 Homologie persistante avec Ripser

Calcul avec Ripser en Python

```
import numpy as np
from ripser import ripser

# Générer des points sur un cercle bruité
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)

# Homologie persistante jusqu'en dimension 1
result = ripser(X, maxdim=1)

# Diagrammes de persistance
dgm0 = result['dgms'][0]  # H_0
dgm1 = result['dgms'][1]  # H_1

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

```
# La paire la plus persistante en H_1
if len(dgm1) > 0:
    pers = dgm1[:, 1] - dgm1[:, 0]
    idx = np.argmax(pers)
    print(f"Paire H_1 la plus persistante: "
          f"naissance={dgm1[idx,0]:.3f}, "
          f"mort={dgm1[idx,1]:.3f}, "
          f"persistance={pers[idx]:.3f}")
```

3.7 Homologie persistante avec GUDHI

Calcul avec GUDHI

```
import gudhi
import numpy as np

# Données: cercle bruité
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)

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

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

print(f"Nombres de Betti: {betti}")

# Diagramme de persistance
gudhi.plot_persistence_diagram(persistence)
```

3.8 Modules de persistance et représentations de carquois

Définition 3.12 (Carquois A_n)**.** Le carquois A_n est le graphe orienté :

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

Un module de persistance (discret, de longueur n) est une représentation du carquois A_n sur $\mathbb{F}$.

Théorème 3.13 (Gabriel, 1972)**.** *Le carquois A_n est de type fini de représentation : ses représentations indécomposables sont exactement les modules d'intervalle $\mathbb{I}[b, d]$, $1 \leq b \leq d \leq n$.*

Remarque 3.14. Ce résultat est à la base du théorème de décomposition des modules de persistance. Pour les filtrations *multi-paramétriques* (2D ou plus), le théorème de Gabriel ne s'applique plus et la classification des indécomposables est *sauvage* — c'est l'un des défis majeurs de la TDA multi-paramétrique.

3.9 Persistance étendue

Définition 3.15 (Persistance étendue)**.** On dit qu'une classe naît en b et meurt en $d = +\infty$ si elle survit dans tous les complexes de la filtration. Ces classes sont dites *essentielles*. Le diagramme de persistance étendu inclut les points $(b, +\infty)$.

Proposition 3.16. Pour une filtration de Vietoris–Rips d'un nuage connexe, il y a toujours exactement un point essentiel en dimension 0, correspondant à la dernière composante connexe qui ne meurt jamais.

3.10 Exercices

Exercice 3.1. Calculer à la main l'homologie persistante de la filtration suivante (sur $\mathbb{F}_2$) :

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

Donner les paires naissance-mort pour H_0 et H_1.

Exercice 3.2. Implémenter l'algorithme de réduction de matrice de bord sur $\mathbb{F}_2$. Tester sur le complexe de l'exercice précédent.

Exercice 3.3. Générer 200 points sur la figure 8 (deux cercles tangents) avec du bruit. Calculer l'homologie persistante et interpréter le diagramme de persistance.

Exercice 3.4. Montrer que si σ est un simplexe qui apparaît dans la filtration au temps t et crée un nouveau cycle (i.e., la colonne correspondante est réduite à zéro), alors la classe de ce cycle naît au temps t.

Exercice 3.5. Comparer les temps de calcul de `gudhi` et `ripser` sur des nuages de 500, 1000 et 2000 points aléatoires dans $\mathbb{R}^3$. Commenter les différences.

Chapitre 4

Codes-Barres et Diagrammes de Persistance

L'homologie persistante produit une quantité considérable d'information : pour chaque dimension, une liste de caractéristiques topologiques avec leurs dates de naissance et de mort. Comment visualiser et comparer ces résultats ? Deux représentations équivalentes se sont imposées : le *code-barres*, où chaque caractéristique est une barre horizontale dont la longueur mesure sa persistance, et le *diagramme de persistance*, où chaque caractéristique est un point dans le demi-plan supérieur. Ces représentations, introduites par Edelsbrunner, Letscher et Zomorodian au début des années 2000, sont devenues l'interface standard entre la théorie topologique et les applications.

Intuition

Le diagramme de persistance et le code-barres sont deux représentations équivalentes du résultat de l'homologie persistante. Chaque point (b, d) du diagramme représente une caractéristique topologique née en b et morte en d. Plus un point est loin de la diagonale, plus la caractéristique est persistante — et donc significative.

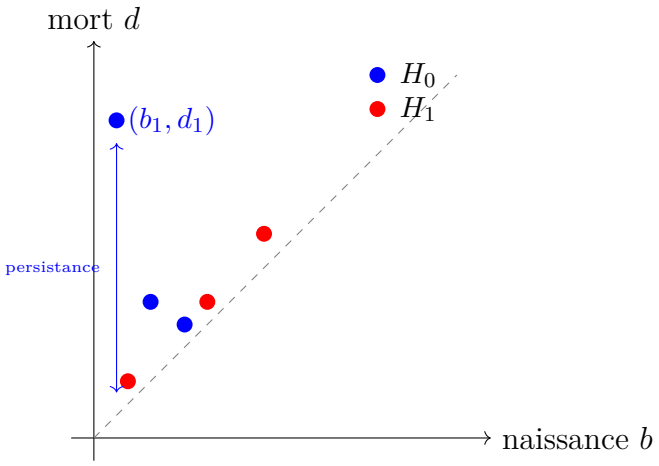
4.1 Diagrammes de persistance

Définition 4.1 (Diagramme de persistance)**.** Soit $\{(b_k, d_k)\}_{k=1}^{N}$ l'ensemble des paires naissance-mort issues de l'homologie persistante en dimension p. Le *diagramme de persistance* est le multi-ensemble :

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

où $\Delta = \{(x, x) : x \in \mathbb{R}\}$ est la diagonale, munie d'une multiplicité infinie.

Remarque 4.2. L'ajout de la diagonale avec multiplicité infinie est essentiel pour définir correctement les distances entre diagrammes (chapitre 8).

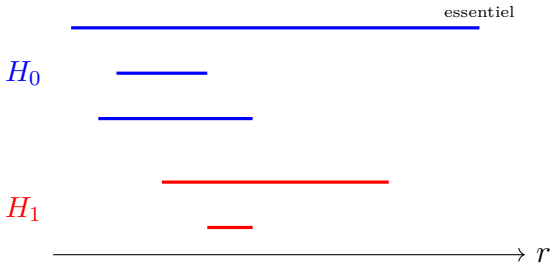


Définition 4.3 (Persistance)**.** La *persistance* d'un point (b, d) est $\text{pers}(b, d) = d - b$. La *persistance totale* est :

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

4.2 Codes-barres

Définition 4.4 (Code-barres)**.** Le *code-barres* (*barcode*) est la représentation équivalente où chaque paire (b_k, d_k) est dessinée comme un segment horizontal $[b_k, d_k)$. Les segments sont empilés verticalement.



Remarque 4.5. Le code-barres et le diagramme de persistance contiennent exactement la même information. Le code-barres est souvent plus intuitif pour la visualisation ; le diagramme est plus adapté à l'analyse mathématique (distances, stabilité).

4.3 Fonctions de Betti

Définition 4.6 (Fonction de Betti)**.** La *fonction de Betti* $\beta_p : \mathbb{R} \to \mathbb{N}$ associe à chaque $r \in \mathbb{R}$ le nombre de Betti du complexe filtré au niveau r :

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

C'est une fonction en escalier, croissante par morceaux.

Proposition 4.7. La fonction de Betti $\beta_p(r)$ peut se lire directement du code-barres : $\beta_p(r)$ est le nombre de barres qui contiennent r en dimension p.

4.4 Paysages de persistance

Définition 4.8 (Paysage de persistance)**.** Soit $\mathrm{Dgm}_p = \{(b_i, d_i)\}$. Pour chaque point, définir la fonction tente :

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

Le *k-ième paysage de persistance* est :

$$\lambda_k(t) = k\text{-ième plus grande valeur de } \{\Lambda_i(t)\}_{i=1}^{N}.$$

Paysage de persistance

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

où kmax dénote le k-ième maximum.

Théorème 4.9 (Bubenik, 2015)**.** *Les paysages de persistance sont des éléments d'un espace de Banach séparable. Ils vérifient une loi forte des grands nombres et un théorème central limite.*

4.5 Images de persistance

Définition 4.10 (Image de persistance)**.** L'*image de persistance* (*persistence image*) est obtenue en :

1. Transformant chaque point (b, d) en $(b, d - b)$ (coordonnées naissance-persistance).
2. Plaçant une gaussienne $\mathcal{N}((b_i, p_i), \sigma^2 I)$ centrée sur chaque point transformé.
3. Pondérant par une fonction de poids $w(b, p)$ (typiquement croissante en p).
4. Discrétisant sur une grille pour obtenir une matrice.

Image de persistance

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

où $p_i = d_i - b_i$ est la persistance du i-ème point.

4.6 Implémentation des représentations

Visualisation des diagrammes et codes-barres

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

```python
# Cercle bruité
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))

# Diagramme de persistance
plot_diagrams(dgms, ax=axes[0])
axes[0].set_title("Diagramme de persistance")

# Code-barres
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("Code-barres")
axes[1].set_xlabel("$r$")

# Paysage de persistance
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("Paysages de persistance ($H_1$)")

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

Images de persistance avec giotto-tda

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

# Données
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)

# Pipeline giotto-tda
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"Forme de l'image de persistance: {images.shape}")
```

4.7 Entropie de persistance

Définition 4.11 (Entropie de persistance)**.** Soit $\mathrm{Dgm} = \{(b_i, d_i)\}_{i=1}^{N}$ un diagramme de persistance (points finis seulement). L'*entropie de persistance* est :

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

Elle mesure la complexité de la distribution des persistances.

4.8 Courbe de Betti

Définition 4.12 (Courbe de Betti)**.** La *courbe de Betti* est la fonction $\beta_p : \mathbb{R} \to \mathbb{N}$ définie par :

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

Cette courbe est directement liée au code-barres : elle compte le nombre de barres actives à chaque instant t.

4.9 Silhouettes de persistance

Définition 4.13 (Silhouette de persistance)**.** La *silhouette de persistance* d'ordre $q > 0$ est la moyenne pondérée des fonctions tentes :

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

Pour $q = 1$, c'est la moyenne arithmétique pondérée par la persistance.

4.10 Statistiques sur les diagrammes

Théorème 4.14 (Moyenne de Fréchet)**.** *Soit $\mathcal{D} = \{D_1, \ldots, D_n\}$ un échantillon de diagrammes de persistance. La* moyenne de Fréchet *est :*

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

où W_2 est la distance de Wasserstein d'ordre 2. Ce minimum existe mais n'est pas nécessairement unique.

Proposition 4.15. Les paysages de persistance, étant des éléments d'un espace de Banach, admettent une moyenne bien définie :

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

4.11 Exercices

Exercice 4.1. Construire le diagramme de persistance et le code-barres pour une filtration de Vietoris–Rips de 6 points régulièrement espacés sur un cercle.

Exercice 4.2. Implémenter en Python le calcul d'un paysage de persistance à partir d'un diagramme de persistance. Tester sur un cercle bruité.

Exercice 4.3. Calculer l'image de persistance d'un nuage de 200 points sur un tore. Faire varier le paramètre σ et observer l'effet sur l'image.

Exercice 4.4. Montrer que l'entropie de persistance est maximale lorsque toutes les persistances sont égales, et minimale (nulle) lorsqu'une seule paire a une persistance non nulle.

Exercice 4.5. Comparer les paysages de persistance d'un cercle, d'un tore et d'une sphère échantillonnés avec du bruit. Quelles différences observe-t-on ?

Chapitre 5

Théorèmes de Stabilité

L'homologie persistante ne serait qu'une curiosité mathématique si elle n'était pas *stable* : un petit changement dans les données doit produire un petit changement dans le diagramme de persistance. Ce résultat fondamental, démontré par David Cohen-Steiner, Herbert Edelsbrunner et John Harer en 2007, est le pilier sur lequel repose toute la TDA appliquée. Il affirme que la distance bottleneck entre deux diagrammes de persistance est bornée par la distance L^∞ entre les fonctions de filtration. Sans ce théorème, le moindre bruit dans les données pourrait bouleverser complètement les descripteurs topologiques, les rendant inutilisables en pratique.

Intuition

Les théorèmes de stabilité sont le fondement théorique de la TDA : ils garantissent que de petites perturbations des données produisent de petits changements dans les diagrammes de persistance. Sans stabilité, les descripteurs topologiques seraient inutilisables en pratique.

5.1 Distances entre diagrammes : rappels

Avant d'énoncer les théorèmes de stabilité, rappelons les distances utilisées. Nous les développerons au chapitre 8.

Définition 5.1 (Distance bottleneck)**.** Soient D_1, D_2 deux diagrammes de persistance. La *distance bottleneck* est :

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

où l'infimum est pris sur toutes les bijections $\gamma : D_1 \to D_2$ (rappelons que les diagrammes incluent la diagonale avec multiplicité infinie).

Définition 5.2 (Distance de Wasserstein)**.** La *distance de Wasserstein d'ordre* q ($1 \leq q < \infty$) est :

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

Relation entre distances

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

5.2 Théorème de stabilité bottleneck

Le théorème fondamental de la TDA est dû à Cohen-Steiner, Edelsbrunner et Harer (2007).

Définition 5.3 (Fonction de Morse apprivoisée)**.** Une fonction $f : X \to \mathbb{R}$ est dite *apprivoisée* (*tame*) si les ensembles de sous-niveaux $f^{-1}((-\infty, r])$ ont une homologie de dimension finie pour tout r, et le nombre de valeurs critiques est fini.

Théorème 5.4 (Stabilité bottleneck — Cohen-Steiner, Edelsbrunner, Harer, 2007)**.** *Soient $f, g : X \to \mathbb{R}$ deux fonctions apprivoisées sur un espace topologique triangulable X. Alors :*

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

Esquisse de preuve. La preuve repose sur trois ingrédients :

1. **Interpolation** : considérer la famille $f_t = (1-t)f + tg$ pour $t \in [0, 1]$.
2. **Lemme de boîte** : si les diagrammes de f et g diffèrent dans une région, alors $\|f - g\|_\infty$ est au moins aussi grande que la taille de cette région.
3. **Construction d'un couplage** : on construit une bijection γ entre les diagrammes en suivant les points à travers l'interpolation.

Voir Cohen-Steiner, Edelsbrunner, Harer (2007) pour la preuve complète. □

Remarque 5.5. Ce théorème est optimal : il existe des fonctions pour lesquelles l'égalité est atteinte.

5.3 Stabilité pour les nuages de points

Corollaire 5.6 (Stabilité Vietoris–Rips)**.** *Soient $X, Y \subset \mathbb{R}^d$ deux nuages de points finis. Soit $d_H(X, Y)$ la distance de Hausdorff. Alors :*

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

Définition 5.7 (Distance de Hausdorff)**.** La *distance de Hausdorff* entre deux compacts X, Y dans un espace métrique (M, d) est :

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

Démonstration. Soit $f_X(z) = \inf_{x \in X} d(z, x)$ la fonction distance à X, et de même pour f_Y. Alors $\|f_X - f_Y\|_\infty = d_H(X, Y)$. Les diagrammes de persistance des filtrations de sous-niveaux de f_X et f_Y coïncident (aux constantes près) avec ceux des filtrations de Rips. Le résultat suit du théorème de stabilité bottleneck, avec le facteur 2 venant de la convention de paramétrisation. □

5.4 Stabilité de Wasserstein

Théorème 5.8 (Stabilité Wasserstein — Chazal et al., 2009)**.** *Soient $f, g : X \to \mathbb{R}$ deux fonctions apprivoisées. Pour tout $q \geq 1$ et tout $p \in \mathbb{N}$, si les diagrammes $\mathrm{Dgm}_p(f)$ et $\mathrm{Dgm}_p(g)$ ont au plus N points hors de la diagonale, alors :*

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

où $C(N, q) = N^{1/q}$ pour la borne la plus simple.

Remarque 5.9. Contrairement à la stabilité bottleneck, la stabilité de Wasserstein dépend du nombre de points dans le diagramme. Des bornes plus fines existent sous des hypothèses supplémentaires.

5.5 Stabilité des représentations

Théorème 5.10 (Stabilité des paysages de persistance)**.** *Les paysages de persistance sont* 1*-lipschitziens par rapport à la distance bottleneck :*

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

pour tout $k \geq 1$.

Proposition 5.11 (Stabilité des images de persistance)**.** Les images de persistance sont lipschitziennes par rapport à la distance de Wasserstein W_1 : il existe $C > 0$ (dépendant de σ et de la fonction de poids) tel que :

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

5.6 Théorème de stabilité généralisé

Définition 5.12 (Distance d'entrelacement)**.** Deux modules de persistance $\mathbf{U}$ et $\mathbf{V}$ sont *ε-entrelacés* s'il existe des morphismes naturels $\varphi : \mathbf{U} \to \mathbf{V}[\varepsilon]$ et $\psi : \mathbf{V} \to \mathbf{U}[\varepsilon]$ tels que les composées $\psi[\varepsilon] \circ \varphi$ et $\varphi[\varepsilon] \circ \psi$ coïncident avec les applications de structure de décalage 2ε.

La *distance d'entrelacement* est :

$$d_I(\mathbf{U}, \mathbf{V}) = \inf\{\varepsilon \geq 0 : \mathbf{U} \text{ et } \mathbf{V} \text{ sont } \varepsilon\text{-entrelacés}\}.$$

Théorème 5.13 (Isométrie algébrique de stabilité)**.** *Pour tout module de persistance p.f.d. :*

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

Remarque 5.14. Ce théorème, dû à Chazal, Cohen-Steiner, Glisse, Guibas et Oudot (2009) et Bauer–Lesnick (2015), montre que la distance bottleneck est la "bonne" distance sur les diagrammes de persistance : elle coïncide exactement avec la notion naturelle de proximité des modules de persistance.

5.7 Applications de la stabilité

5.7.1 Inférence topologique

Théorème 5.15 (Inférence topologique — Niyogi, Smale, Weinberger, 2008)**.** *Soit $\mathcal{M}$ une sous-variété compacte de $\mathbb{R}^d$ de nombre d'atteinte $\tau > 0$. Si X est un échantillon ε-dense de $\mathcal{M}$ avec $\varepsilon < \tau/2$, alors la filtration de Čech de X au paramètre $r \in (\varepsilon, \tau - \varepsilon)$ a le même type d'homotopie que $\mathcal{M}$.*

5.7.2 Robustesse au bruit

Proposition 5.16. Si X est un nuage de points échantillonné sur une variété $\mathcal{M}$ et X' est une version bruitée avec $d_H(X, X') \leq \delta$, alors :

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

Les caractéristiques topologiques de persistance $> 4\delta$ dans le diagramme de X' correspondent à de vraies caractéristiques de $\mathcal{M}$.

5.8 Démonstration illustrée

Vérification numérique de la stabilité

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

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

# Versions bruitées avec epsilon croissant
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']

    # Distance bottleneck en H_1
    d_bn = bottleneck(dgm_X[1], dgm_Y[1])

    # Distance de Hausdorff (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 Limites de la stabilité

Limites

- La stabilité bottleneck est une borne *sup* : le diagramme peut changer beaucoup moins que la borne.
- La borne de Wasserstein dépend du nombre de points : pour des diagrammes avec beaucoup de points de faible persistance, la borne peut être lache.
- La stabilité concerne les *diagrammes*, pas nécessairement les *représentations vectorielles* (bien que celles-ci héritent souvent de propriétés de stabilité).

5.10 Exercices

Exercice 5.1. Montrer que la distance bottleneck vérifie l'inégalité triangulaire.

Exercice 5.2. Construire un exemple de deux fonctions $f, g : [0,1] \to \mathbb{R}$ telles que $d_B(\mathrm{Dgm}(f), \mathrm{Dgm}(g)) = \|f - g\|_\infty$ (montrer que la borne est atteinte).

Exercice 5.3. Vérifier numériquement la stabilité bottleneck en générant des nuages de points sur un tore avec différents niveaux de bruit. Tracer d_B en fonction de d_H et vérifier la linéarité.

Exercice 5.4. Démontrer que si $\mathbf{U}$ et $\mathbf{V}$ sont ε-entrelacés, alors $d_B(\mathrm{Dgm}(\mathbf{U}), \mathrm{Dgm}(\mathbf{V})) \leq \varepsilon$.

Exercice 5.5. Soit X un nuage de n points i.i.d. sur une variété compacte $\mathcal{M}$. Montrer que $d_H(X, \mathcal{M}) \to 0$ presque sûrement quand $n \to \infty$, et en déduire la convergence des diagrammes de persistance.

Chapitre 6

Complexes de Vietoris-Rips, Čech et Alpha

L'homologie persistante a besoin d'une *filtration* de complexes simpliciaux. Mais comment construire un complexe à partir d'un nuage de points ? Trois constructions classiques dominent. Le complexe de *Čech* ajoute un simplexe dès que les boules de rayon r centrées sur ses sommets ont une intersection commune — il capture exactement la topologie de la réunion des boules (théorème du nerf). Le complexe de *Vietoris-Rips* est plus simple : un simplexe est présent dès que toutes les paires de sommets sont à distance $\leq 2r$ — il est bien plus facile à calculer, mais approxime seulement le Čech. Le complexe *Alpha*, dû à Edelsbrunner, restreint le Čech aux cellules de Voronoï, produisant un complexe de taille linéaire en le nombre de points.

Intuition

La construction d'un complexe simplicial à partir d'un nuage de points est l'étape fondamentale de la TDA. Trois constructions dominent : Vietoris–Rips (simple mais gros), Čech (exact mais coûteux), et Alpha (petit et exact en basse dimension).

6.1 Complexe de Vietoris–Rips

Définition 6.1 (Complexe de Vietoris–Rips)**.** Soit $X = \{x_1, \ldots, x_n\}$ un ensemble fini dans un espace métrique (M, d). Le *complexe de Vietoris–Rips* au paramètre $r \geq 0$ est :

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

Remarque 6.2. Le complexe de Vietoris–Rips est le *complexe de cliques* (ou *flag complex*) du graphe de proximité $G_r = (X, E_r)$ où $\{x_i, x_j\} \in E_r$ si et seulement si $d(x_i, x_j) \leq r$. Autrement dit, $\sigma \in \mathrm{VR}(X, r)$ si et seulement si toutes les paires de sommets de σ sont reliées dans G_r.

Proposition 6.3 (Propriétés)**.**

1. $\mathrm{VR}(X, r)$ est entièrement déterminé par son 1-squelette (le graphe G_r).
2. $r \leq s \implies \mathrm{VR}(X, r) \subseteq \mathrm{VR}(X, s)$.
3. La dimension maximale de $\mathrm{VR}(X, r)$ peut atteindre $n - 1$.

Explosion combinatoire

Pour n points, le complexe de Vietoris–Rips peut contenir jusqu'à $2^n - 1$ simplexes. En pratique, on tronque la dimension maximale (typiquement ≤ 3).

6.2 Complexe de Čech

Définition 6.4 (Complexe de Čech)**.** Le *complexe de Čech* au paramètre r est :

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

où $B(x_i, r) = \{y \in M : d(x_i, y) \leq r\}$ est la boule fermée de centre x_i et de rayon r.

Théorème 6.5 (Nerf — Borsuk, 1948)**.** *Soit $\mathcal{U} = \{U_i\}_{i \in I}$ un recouvrement fini d'un espace topologique X par des ouverts convexes. Alors le* nerf *de $\mathcal{U}$:*

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

a le même type d'homotopie que $\bigcup_{i \in I} U_i$.

Corollaire 6.6. *Dans $\mathbb{R}^d$, $\check{C}(X, r)$ a le même type d'homotopie que $\bigcup_{i=1}^n B(x_i, r)$.*

Proposition 6.7 (Relation Rips–Čech)**.** Pour tout nuage de points $X \subset \mathbb{R}^d$ et tout $r > 0$:

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

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

6.3 Complexe Alpha

Définition 6.8 (Diagramme de Voronoï)**.** Le *diagramme de Voronoï* de $X \subset \mathbb{R}^d$ est la partition de $\mathbb{R}^d$ en cellules :

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

Définition 6.9 (Triangulation de Delaunay)**.** La *triangulation de Delaunay* $\mathrm{Del}(X)$ est le complexe simplicial dual du diagramme de Voronoï : $\sigma = \{x_{i_0}, \ldots, x_{i_p}\} \in \mathrm{Del}(X)$ si et seulement si $\bigcap_{k=0}^p V_{i_k} \neq \emptyset$.

Définition 6.10 (Complexe Alpha)**.** Le *complexe Alpha* au paramètre r est :

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

C'est le sous-complexe de la triangulation de Delaunay formé des simplexes dont les boules restreintes aux cellules de Voronoï s'intersectent.

Théorème 6.11 (Edelsbrunner, 1995)**.** *Le complexe Alpha vérifie :*

1. $\mathrm{Alpha}(X, r)$ *a le même type d'homotopie que $\bigcup_{i=1}^n B(x_i, r)$ (et donc que $\check{C}(X, r)$).*
2. $\dim(\mathrm{Alpha}(X, r)) \leq d$ *(la dimension ambiante).*
3. *Le nombre total de simplexes est $O(n^{\lceil d/2 \rceil})$.*

Comparaison des tailles

Pour n points dans $\mathbb{R}^d$:

Complexe	Dimension max	Nb simplexes
Vietoris–Rips	$n-1$	$O(2^n)$
Čech	$n-1$	$O(2^n)$
Alpha	d	$O(n^{\lceil d/2 \rceil})$

6.4 Construction en pratique

Complexe de Rips avec GUDHI

```python
import gudhi
import numpy as np

# Nuage de points
np.random.seed(42)
X = np.random.randn(50, 3)

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

print(f"Nombre de simplexes: {st.num_simplices()}")
print(f"Nombre de sommets: {st.num_vertices()}")
print(f"Dimension: {st.dimension()}")

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

Complexe Alpha avec GUDHI

```python
import gudhi
import numpy as np

# Nuage de points en 2D
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)

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

```
print(f"Nombre de simplexes: {st.num_simplices()}")
print(f"Dimension: {st.dimension()}")

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

Complexe de Čech avec GUDHI

```
import gudhi
import numpy as np

# Nuage de points en 2D
X = np.random.randn(30, 2)

# Le complexe de Čech n'a pas de classe dédiée dans GUDHI
# mais on peut utiliser le complexe de Rips comme approximation
# ou utiliser miniball pour vérifier le critère de Čech

# Alternative: utiliser la filtration Alpha qui est
# homotopiquement équivalente au Čech
alpha = gudhi.AlphaComplex(points=X)
st = alpha.create_simplex_tree()

# Les valeurs de filtration Alpha sont les carrés des rayons
# de la plus petite boule englobante de chaque simplexe
for simplex, filt in st.get_filtration():
    if len(simplex) <= 3:
        print(f"  {simplex}: rayon^2 = {filt:.4f}")
    if filt > 0.5:
        break
```

6.5 Complexe de Rips pondéré et Rips spars

Définition 6.12 (Sparse Rips — Sheehy, 2013)**.** Le *Sparse Rips complex* est une approximation du complexe de Vietoris–Rips qui contient $O(n)$ simplexes dans chaque dimension, au lieu de $O(2^n)$. Pour un paramètre $\varepsilon > 0$, il produit une $(1+\varepsilon)$-approximation des diagrammes de persistance.

Sparse Rips avec 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()} simplexes")

# Comparaison avec Rips standard
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()} simplexes")
```

6.6 Complexes cubiques

Définition 6.13 (Complexe cubique)**.** Un *complexe cubique* est l'analogue d'un complexe simplicial où les briques élémentaires sont des cubes (produits d'intervalles) au lieu de simplexes. Il est particulièrement adapté aux données sur grille (images, volumes 3D).

Homologie persistante d'une image

```
import gudhi
import numpy as np

# Créer une image simple (anneau)
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 à l'origine

# Complexe cubique
cc = gudhi.CubicalComplex(
    top_dimensional_cells=image.flatten(),
    dimensions=image.shape
)

# Persistance
pers = cc.persistence()
print("Persistance de l'image:")
for dim, (b, d) in pers:
    if d - b > 0.5:  # filtrer le bruit
        print(f"  H_{dim}: ({b:.2f}, {d:.2f}), "
              f"pers={d-b:.2f}")
```

6.7 Choix du complexe en pratique

Guide de choix du complexe

- **Données en basse dimension** ($d \leq 3$) : utiliser le complexe **Alpha**. Il est petit, exact, et rapide à construire.
- **Données en haute dimension ou métriques** : utiliser **Vietoris–Rips** (avec `ripser` pour la vitesse). Tronquer la dimension à 2 ou 3.
- **Données sur grille** (images, volumes) : utiliser un complexe **cubique**.
- **Grand nombre de points** : utiliser **Sparse Rips** ou sous-échantillonner.

6.8 Exercices

Exercice 6.1. Pour 4 points aux sommets d'un carré unité, construire à la main les complexes $\mathrm{VR}(X, r)$ et $\check{C}(X, r)$ pour $r = 0.5, 1.0, 1.5$. Vérifier l'inclusion $\check{C}(X, r) \subseteq \mathrm{VR}(X, 2r)$.

Exercice 6.2. Construire le diagramme de Voronoï et la triangulation de Delaunay pour 5 points aléatoires dans $\mathbb{R}^2$ (éventuellement avec `scipy.spatial.Delaunay`).

Exercice 6.3. Comparer les temps de calcul des complexes Alpha et Rips pour des nuages de 100 à 1000 points en dimensions 2, 3 et 10.

Exercice 6.4. Montrer que le complexe de Vietoris–Rips d'un ensemble de $n+1$ points en position générale dans $\mathbb{R}^n$, pour r suffisamment grand, est le simplexe plein Δ^n.

Exercice 6.5. Calculer l'homologie persistante d'une image binaire de votre choix en utilisant un complexe cubique avec GUDHI.

Chapitre 7

L'Algorithme Mapper

En 2007, Gurjeet Singh, Facundo Mémoli et Gunnar Carlsson publient un article qui va transformer la TDA appliquée : l'algorithme Mapper. L'idée est d'une simplicité remarquable : au lieu de calculer l'homologie persistante d'un nuage de points (ce qui produit des invariants numériques mais pas de visualisation directe), Mapper construit un *graphe* qui capture la forme globale des données. Le procédé s'inspire du théorème du nerf : on découpe les données en morceaux chevauchants à l'aide d'une fonction filtre, on regroupe les points de chaque morceau par clustering, puis on relie les clusters qui partagent des points. Le graphe résultant est une « carte » des données, d'où le nom. Mapper a été utilisé pour découvrir un sous-groupe de cancer du sein, analyser la performance sportive, et explorer des données génomiques — partout où la visualisation de la « forme » des données apporte un éclairage inaccessible aux méthodes statistiques classiques.

Intuition

Mapper est un algorithme de visualisation topologique qui construit un graphe résumé capturant la structure globale d'un jeu de données de haute dimension. Contrairement à l'homologie persistante qui calcule des invariants numériques, Mapper produit un objet géométrique interprétable.

7.1 Motivation

Les techniques de réduction de dimension (PCA, t-SNE, UMAP) projettent les données dans un espace de faible dimension mais perdent inévitablement de l'information structurelle. Mapper adopte une approche différente : il construit un graphe simplicial qui préserve les propriétés topologiques du jeu de données.

Définition 7.1 (Reeb Graph — Idée sous-jacente)**.** Soit $f : X \to \mathbb{R}$ une fonction continue sur un espace topologique X. Le *graphe de Reeb* $\mathcal{R}_f(X)$ est le quotient de X par la relation : $x \sim y$ si et seulement si $f(x) = f(y)$ et x, y sont dans la même composante connexe de $f^{-1}(f(x))$.

Mapper est une *approximation discrète* du graphe de Reeb.

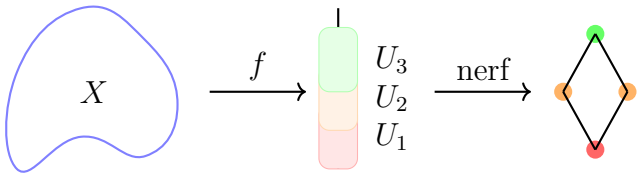
7.2 L'algorithme Mapper

Algorithme Mapper (Singh, Mémoli, Carlsson, 2007)

Entrées : Nuage de points X, fonction filtre $f : X \to \mathbb{R}^k$, recouvrement $\mathcal{U}$ de $f(X)$, algorithme de clustering $\mathcal{C}$.

1. **Filtrage** : appliquer f pour obtenir $f(X) \subset \mathbb{R}^k$.
2. **Recouvrement** : choisir un recouvrement $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ de $f(X)$ par des ouverts (typiquement des intervalles chevauchants).
3. **Préimage et clustering** : pour chaque U_α, calculer $f^{-1}(U_\alpha) \cap X$ et appliquer l'algorithme de clustering $\mathcal{C}$, obtenant des clusters $C_{\alpha,1}, \ldots, C_{\alpha,k_\alpha}$.
4. **Construction du nerf** : construire le complexe simplicial dont les sommets sont les clusters et où $\{C_{\alpha_0,i_0}, \ldots, C_{\alpha_p,i_p}\}$ forme un simplexe si et seulement si $C_{\alpha_0,i_0} \cap \cdots \cap C_{\alpha_p,i_p} \neq \emptyset$.

Sortie : le complexe simplicial (souvent un graphe, i.e., un complexe de dimension 1).



Graphe Mapper

7.3 Choix des paramètres

7.3.1 Fonction filtre

Définition 7.2 (Fonctions filtres courantes)**.**

- **Coordonnée** : projection sur un axe, une composante principale.
- **Excentricité** : $f(x) = \left(\frac{1}{n}\sum_{i=1}^{n} d(x,x_i)^p\right)^{1/p}$.
- **Densité** : estimation par noyau (KDE), k-plus proches voisins.
- **Distance au k-ème voisin** : mesure d'isolement.
- **Fonctions apprises** : composantes de t-SNE, UMAP, autoencodeur.

7.3.2 Recouvrement

Définition 7.3 (Recouvrement par intervalles)**.** Pour une fonction filtre $f : X \to \mathbb{R}$, un recouvrement standard est défini par :

- n_{int} : nombre d'intervalles.

- p_{ov} : pourcentage de chevauchement ($0 < p_{\text{ov}} < 1$).

La largeur de chaque intervalle est $w = \frac{\max f - \min f}{n_{\text{int}} - (n_{\text{int}} - 1)p_{\text{ov}}}$ et les intervalles sont :

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

Sensibilité aux paramètres

Le résultat de Mapper dépend fortement du choix de la fonction filtre, du nombre d'intervalles, du chevauchement et de l'algorithme de clustering. Il est essentiel de faire varier ces paramètres et d'évaluer la robustesse du graphe résultant.

7.4 Implémentation avec KeplerMapper

Mapper avec KeplerMapper

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

# Données: deux cercles concentriques
X, y = make_circles(n_samples=500, noise=0.05, factor=0.5)

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

# Fonction filtre: projection sur la première coordonnée
lens = mapper.fit_transform(X, projection=[0])

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

# Visualisation HTML
mapper.visualize(
    graph,
    path_html="mapper_circles.html",
    title="Mapper: Cercles Concentriques",
    color_values=y
)
print(f"Nombre de noeuds: {len(graph['nodes'])}")
print(f"Nombre d'arêtes: {len(graph['links'])}")
```

7.5 Mapper avec GUDHI

Mapper avec GUDHI

```
import gudhi
import numpy as np

# GUDHI offre une implémentation de Mapper via
# la classe MapperComplex (versions récentes)

# Exemple conceptuel: construction manuelle
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KernelDensity

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

# Filtre: densité
kde = KernelDensity(bandwidth=0.5)
kde.fit(X)
f_values = kde.score_samples(X)

# Recouvrement
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()))

# Construire les arêtes (intersection non vide)
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"Noeuds: {len(clusters_all)}, Arêtes: {len(edges)}")
```

7.6 Fondements théoriques

Théorème 7.4 (Convergence de Mapper — Carrière, Oudot, 2018). *Sous des conditions de régularité sur f et X, lorsque le nombre de points $n \to \infty$ et les paramètres de recouvrement sont choisis de manière adaptée, le graphe Mapper converge (au sens de la distance d'entrelacement fonctionnelle) vers le graphe de Reeb de (X, f).*

Définition 7.5 (Distance d'entrelacement fonctionnelle). Soient $\mathcal{R}_1$ et $\mathcal{R}_2$ deux graphes de Reeb. La *distance d'entrelacement fonctionnelle* est :

$$d_{\mathrm{FI}}(\mathcal{R}_1, \mathcal{R}_2) = \inf_{\varepsilon}\{\varepsilon \geq 0 : \mathcal{R}_1 \text{ et } \mathcal{R}_2 \text{ sont } \varepsilon\text{-entrelacés}\}.$$

7.7 Mapper multidimensionnel

Définition 7.6 (Mapper 2D). Lorsque la fonction filtre est $f : X \to \mathbb{R}^2$, le recouvrement est une grille de rectangles chevauchants dans $\mathbb{R}^2$. Le résultat est un complexe simplicial de dimension potentiellement ≥ 2 (pas juste un graphe).

Mapper 2D avec KeplerMapper

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

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

mapper = km.KeplerMapper()

# Filtre 2D: deux premières coordonnées
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 Applications célèbres de Mapper

7.8.1 Cancer du sein (Lum et al., 2013)

L'étude fondatrice de Mapper en biologie a révélé des sous-groupes dans les données d'expression génique du cancer du sein qui n'avaient pas été détectés par les méthodes de clustering classiques.

7.8.2 Analyse des joueurs de basketball (Lum et al., 2013)

Mapper a été appliqué aux statistiques de joueurs de NBA, révélant une structure en "Y" correspondant aux trois rôles principaux (meneur, ailier, pivot).

7.8.3 Diabète de type 2 (Li et al., 2015)

Mapper a identifié trois sous-types distincts de diabète de type 2 à partir de données cliniques, avec des implications thérapeutiques.

7.9 Coloration du graphe Mapper

Remarque 7.7. Le graphe Mapper est souvent coloré par une variable d'intérêt (réponse, label de classe, variable clinique) pour faciliter l'interprétation. Chaque nœud reçoit la valeur moyenne de la variable pour les points qu'il contient.

7.10 Ball Mapper

Définition 7.8 (Ball Mapper — Dłotko, 2019)**.** Le *Ball Mapper* est une variante simplifiée de Mapper où :

1. On fixe un rayon $\varepsilon > 0$.
2. On sélectionne un sous-ensemble de points-repères $L \subset X$ (par farthest point sampling).
3. Les nœuds sont les boules $B(l, \varepsilon)$ pour $l \in L$.
4. Deux nœuds sont reliés si leurs boules contiennent des points communs.

Avantage : pas besoin de choisir une fonction filtre.

7.11 Exercices

Exercice 7.1. Appliquer Mapper à un jeu de données iris (`sklearn.datasets.load_iris`) avec différentes fonctions filtres (PCA, excentricité, densité). Comparer les graphes résultants.

Exercice 7.2. Étudier l'effet du nombre d'intervalles et du pourcentage de chevauchement sur le graphe Mapper. Pour un cercle bruité, déterminer les paramètres qui produisent un graphe cyclique.

Exercice 7.3. Implémenter Ball Mapper en Python et l'appliquer à un nuage de points en 3D.

Exercice 7.4. Montrer que si le recouvrement est suffisamment fin et l'algorithme de clustering est exact, le graphe Mapper est une approximation du graphe de Reeb de (X, f).

Exercice 7.5. Télécharger un jeu de données réel (ex. : données génomiques d'expression) et appliquer Mapper pour identifier des sous-groupes. Interpréter le graphe résultant.

Chapitre 8

Distances entre Diagrammes

Pour que les diagrammes de persistance soient véritablement utiles, il faut pouvoir les *comparer*. Deux nuages de points ont-ils la même forme topologique ? Un signal bruité a-t-il la même structure qu'un signal propre ? Répondre à ces questions exige une notion de distance entre diagrammes. Deux familles dominent la théorie : la distance bottleneck, qui mesure le pire déplacement nécessaire pour apparier les points de deux diagrammes, et les distances de Wasserstein, qui mesurent le coût total du transport optimal. Le choix entre les deux n'est pas neutre : la distance bottleneck privilégie les structures les plus persistantes (les "grandes" barres), tandis que Wasserstein intègre l'information de toutes les barres, y compris les petites. Les algorithmes de calcul — problème d'affectation hongrois, enchères — et les propriétés théoriques de stabilité font l'objet de ce chapitre.

Intuition

Les diagrammes de persistance résumént l'information topologique d'un espace filtré. Pour pouvoir comparer deux espaces, il faut disposer de **distances** entre diagrammes. Le choix de la distance détermine quelles différences topologiques on privilégie : les structures les plus persistantes (bottleneck) ou l'ensemble du spectre (Wasserstein).

8.1 Rappels sur les diagrammes de persistance

Définition 8.1 (Diagramme de persistance)**.** Soit $\mathcal{F}$ une filtration d'un complexe simplicial. Le **diagramme de persistance** $\mathrm{Dgm}(\mathcal{F})$ est le multi-ensemble de points $(b_i, d_i) \in \mathbb{R}^2$ avec $b_i < d_i$, où b_i est la valeur de naissance et d_i la valeur de mort de la i-ème classe d'homologie persistante. On y adjoint la diagonale $\Delta = \{(x, x) : x \in \mathbb{R}\}$ avec multiplicité infinie.

Remarque 8.2. L'ajout de la diagonale avec multiplicité infinie permet de définir des bijections entre diagrammes de tailles différentes : un point du diagramme peut être apparié à un point de la diagonale, ce qui revient à le considérer comme du bruit.

8.2 La distance bottleneck

Définition 8.3 (Distance bottleneck)**.** Soient D_1 et D_2 deux diagrammes de persistance. La **distance bottleneck** est :

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

où l'infimum porte sur toutes les bijections $\gamma : D_1 \to D_2$ (en utilisant la diagonale pour compléter).

Remarque 8.4. La norme $\|\cdot\|_\infty$ utilisée ici est la norme du maximum : $\|(a,b) - (c,d)\|_\infty = \max(|a-c|, |b-d|)$. On peut aussi utiliser la distance à la diagonale : $\|(b,d)\|_\infty^\Delta = \frac{d-b}{2}$.

Intuition

La distance bottleneck mesure le « coût du pire appariement ». Elle est insensible au nombre de petites structures et ne capture que le décalage maximal entre les structures les plus importantes.

Exemple 8.5. Considérons $D_1 = \{(0,2), (1,3)\}$ et $D_2 = \{(0,2.5), (1,3.2)\}$. L'appariement naturel donne :

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

Proposition 8.6 (Métrique)**.** d_∞ est une métrique sur l'espace des diagrammes de persistance.

8.3 Les distances de Wasserstein

Définition 8.7 (Distance de Wasserstein)**.** Pour $p \geq 1$ et $q \geq 1$, la **distance de Wasserstein** $W_{p,q}$ entre deux diagrammes D_1, D_2 est :

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

où γ parcourt les bijections $D_1 \to D_2$. Le cas $q = \infty$ et $p = \infty$ redonne la distance bottleneck.

Conventions variables

La littérature utilise différentes conventions pour les indices de la distance de Wasserstein. Certains auteurs écrivent W_p avec $q = \infty$ implicite, d'autres utilisent $q = p$. Vérifiez toujours la convention utilisée dans chaque article.

Proposition 8.8 (Relation bottleneck–Wasserstein)**.** Pour tout $p \geq 1$:

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

De plus, $d_\infty(D_1, D_2) \leq W_{p,\infty}(D_1, D_2)$ pour tout p.

Résumé des 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 Le théorème de stabilité

Théorème 8.9 (Stabilité de la persistance — Cohen-Steiner, Edelsbrunner, Harer 2007)**.** *Soient $f, g : X \to \mathbb{R}$ deux fonctions continues sur un espace topologique triangulable X. Alors :*

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

Intuition

Le théorème de stabilité est le résultat le plus important de la TDA. Il garantit que de petites perturbations des données (au sens L^∞) ne peuvent produire que de petites perturbations du diagramme de persistance. C'est ce qui rend la persistance **robuste au bruit**.

Corollaire 8.10. *L'application $f \mapsto \mathrm{Dgm}(f)$ est 1-lipschitzienne de $\big(C(X, \mathbb{R}), \|\cdot\|_\infty\big)$ vers $\big(\mathcal{D}, d_\infty\big)$.*

Théorème 8.11 (Stabilité en Wasserstein)**.** *Sous des hypothèses de régularité supplémentaires (fonctions de Morse apprivoisées), on a pour tout $p \geq 1$:*

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

pour une constante C_p dépendant de X et p.

8.5 La distance d'entrelacement

Définition 8.12 (Modules de persistance)**.** Un **module de persistance** est un foncteur $\mathbb{V} : (\mathbb{R}, \leq) \to \mathbf{Vec}_k$, c'est-à-dire une famille d'espaces vectoriels $\{V_t\}_{t \in \mathbb{R}}$ avec des applications linéaires $v_{s,t} : V_s \to V_t$ pour $s \leq t$ vérifiant $v_{t,t} = \mathrm{id}$ et $v_{s,u} = v_{t,u} \circ v_{s,t}$ pour $s \leq t \leq u$.

Définition 8.13 (ε-entrelacement)**.** Deux modules de persistance $\mathbb{V}$ et $\mathbb{W}$ sont **ε-entrelacés** s'il existe des morphismes de modules décalés $\varphi : \mathbb{V} \to \mathbb{W}[\varepsilon]$ et $\psi : \mathbb{W} \to \mathbb{V}[\varepsilon]$ tels que :

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

où $\mathbb{W}[\varepsilon]_t = W_{t+\varepsilon}$ est le module décalé.

Définition 8.14 (Distance d'entrelacement)**.** La **distance d'entrelacement** est :

$$d_I(\mathbb{V}, \mathbb{W}) = \inf\{\varepsilon \geq 0 : \mathbb{V} \text{ et } \mathbb{W} \text{ sont } \varepsilon\text{-entrelacés}\}.$$

Théorème 8.15 (Isometrie — Chazal et al. 2009)**.** *Pour des modules de persistance à paramètre réel décomposables en intervalles :*

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

Intuition

Le théorème d'isométrie montre que la distance bottleneck entre diagrammes coïncide exactement avec la distance d'entrelacement entre modules. Cela donne une interprétation **algébrique** de la distance bottleneck et fonde la stabilité sur des bases catégoriques solides.

8.6 Stabilité algébrique

Théorème 8.16 (Stabilité algébrique)**.** *Soit* $F : \mathbf{Top} \to \mathbf{Vec}_k^{(\mathbb{R},\leq)}$ *un foncteur d'homologie persistante. Pour toute paire d'espaces filtrés* (X, f) *et* (Y, g) *avec un morphisme* ε*-entrelacé entre les filtrations :*

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

Remarque 8.17. La stabilité algébrique généralise le théorème de stabilité classique au cadre des modules de persistance abstraits. Elle ne nécessite pas de fonctions continues : elle s'applique directement au niveau des filtrations et de l'algèbre des modules.

8.7 Calcul pratique des distances

Proposition 8.18 (Complexité)**.** La distance bottleneck entre deux diagrammes de n et m points peut être calculée en $\mathcal{O}(N^{1.5} \log N)$ où $N = n + m$, par réduction au problème d'appariement biparti. La distance de Wasserstein W_p se calcule en $\mathcal{O}(N^3)$ par l'algorithme hongrois.

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

# Deux diagrammes de persistance (naissance, mort)
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]])

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

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

8.8 Exercices

Exercice 8.1. Calculer à la main la distance bottleneck entre $D_1 = \{(0, 1), (0, 3), (2, 5)\}$ et $D_2 = \{(0, 1.5), (1, 4)\}$. *Indication* : essayez différents appariements et n'oubliez pas la diagonale.

Exercice 8.2. Montrer que $d_\infty(D_1, D_2) \leq W_{p,\infty}(D_1, D_2)$ pour tout $p \geq 1$.

Exercice 8.3. Soient $f(x) = \sin(x)$ et $g(x) = \sin(x) + 0.1$ sur $[0, 2\pi]$. Montrer que $d_\infty(\mathrm{Dgm}(f), \mathrm{Dgm}(g)) \leq 0.1$ en utilisant le théorème de stabilité.

Exercice 8.4 (Distance d'entrelacement)**.** Soient $\mathbb{V}$ le module k sur $[0, 2]$ et $\mathbb{W}$ le module k sur $[0.5, 2.5]$. Calculer $d_I(\mathbb{V}, \mathbb{W})$ directement à partir de la définition.

Exercice 8.5 (Implémentation)**.** En utilisant `gudhi`, comparer les distances bottleneck et Wasserstein-2 pour des diagrammes de persistance de deux nuages de points : un échantillon du cercle S^1 et un échantillon du tore T^2 plongé dans $\mathbb{R}^3$.

Chapitre 9

Apprentissage Machine avec la TDA

Les diagrammes de persistance capturent la "forme" des données, mais les algorithmes d'apprentissage machine — SVM, forêts aléatoires, réseaux de neurones — attendent des vecteurs de dimension fixe. Comment combler ce fossé ? C'est le problème de la *vectorisation*, l'un des défis centraux de la TDA appliquée. Peter Bubenik (2015) propose les *paysages de persistance*, des fonctions dans un espace de Banach ; Henry Adams et ses collaborateurs introduisent les *images de persistance* ; Mathieu Carrière et al. développent les noyaux de persistance. Plus récemment, les couches de persistance différentiables permettent d'intégrer la TDA directement dans les réseaux de neurones profonds, ouvrant la voie à un apprentissage topologique de bout en bout.

Intuition

Les diagrammes de persistance vivent dans un espace non vectoriel : on ne peut pas directement les additionner ni les multiplier par un scalaire. Pour les utiliser dans un pipeline d'apprentissage machine, il faut les **vectoriser**, c'est-à-dire les transformer en représentations vivant dans un espace de Hilbert ou un espace de features de dimension finie.

9.1 Paysages de persistance

Rappelons le paysage de persistance introduit au Chapitre 4 : pour un diagramme $D = \{(b_i, d_i)\}$, le k-ème paysage $\lambda_k(t)$ est la k-ème plus grande valeur des fonctions tentes $\Lambda_i(t)$. Les paysages vivent dans un espace de Banach, ce qui permet de calculer moyennes, variances et de réaliser des tests statistiques.

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

# Diagramme de persistance
dgm = np.array([[0.0, 1.5], [0.5, 2.0], [1.0, 3.0]])

# Calcul des 3 premiers paysages sur une grille
landscape = Landscape(num_landscapes=3, resolution=100)
L = landscape.fit_transform([dgm])
print(f"Shape: {L.shape}")  # (1, 300)
```

9.2 Images de persistance

Définition 9.1 (Image de persistance — Adams et al. 2017)**.** Soit D un diagramme de persistance. L'**image de persistance** est construite en trois étapes :

1. **Changement de coordonnées** : on transforme chaque (b_i, d_i) en $(b_i, d_i - b_i)$ (naissance, persistance).
2. **Pondération** : on applique une fonction de poids $w : \mathbb{R}^2 \to \mathbb{R}_+$ (typiquement linéaire en la persistance) pour atténuer les points proches de la diagonale.
3. **Lissage gaussien** : on place une gaussienne $\mathcal{N}((b_i, p_i), \sigma^2 I)$ sur chaque point et on discrétise sur une grille $N \times N$.

L'image résultante est :

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

Théorème 9.2 (Stabilité des images de persistance)**.** *Si w est lipschitzienne et bornée, l'application $D \mapsto \rho_D$ est lipschitzienne pour la distance de Wasserstein W_1 :*

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

9.3 Silhouettes de persistance

Rappelons la silhouette de persistance d'ordre q introduite au Chapitre 4 : c'est la moyenne pondérée (par $(d_i - b_i)^q$) des fonctions tentes. Pour q grand, elle donne plus de poids aux structures persistantes. La silhouette est un élément de $L^p(\mathbb{R})$, donc directement utilisable comme feature dans un pipeline d'apprentissage.

9.4 Méthodes à noyaux

Définition 9.3 (Noyau de persistance)**.** Un **noyau de persistance** est un noyau défini positif $K : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ sur l'espace des diagrammes de persistance.

Définition 9.4 (Noyau à échelle de persistance — Reininghaus et al. 2015)**.**

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

où $\bar{q} = (d_q, b_q)$ est le point q réfléchi par rapport à la diagonale.

Proposition 9.5 (Défini positivité)**.** Le noyau k_σ est défini positif et stable : $|k_\sigma(D_1, D_1') - k_\sigma(D_2, D_2')| \leq C_\sigma(W_1(D_1, D_2) + W_1(D_1', D_2'))$.

Définition 9.6 (Noyau de Fisher — Le, Yamada 2018)**.** Le **noyau de Fisher** modélise chaque diagramme comme une distribution et utilise la distance de Fisher entre distributions :

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

9.5 Intégration dans un pipeline ML

Exemple 9.7 (Pipeline complet de classification).

1. **Données** : nuages de points ou séries temporelles.
2. **Filtration** : Rips, Alpha ou sublevel set.
3. **Persistance** : calcul des diagrammes via `gudhi` ou `ripser`.
4. **Vectorisation** : images de persistance, paysages ou noyaux.
5. **Modèle** : SVM, Random Forest ou régression logistique.

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

# Pipeline scikit-learn avec features topologiques
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)
```

Choix de la vectorisation

- Les **paysages** préservent la structure de Banach et permettent des tests statistiques.
- Les **images** sont plus adaptées aux CNN car elles produisent des représentations 2D.
- Les **noyaux** sont préférables avec des SVM en petite dimension.

9.6 Scikit-TDA et écosystème logiciel

Remarque 9.8 (Écosystème Python pour la TDA).

- **GUDHI** : bibliothèque C++/Python complète (filtrations, persistance, représentations).
- **Ripser** : calcul rapide de la persistance de Rips.
- **scikit-tda** : méta-paquet intégrant Ripser, Persim (distances), KeplerMapper.
- **giotto-tda** : pipeline TDA compatible scikit-learn avec accélération C++.

9.7 Exercices

Exercice 9.1. Calculer les deux premiers paysages de persistance du diagramme $D = \{(0,3),(1,4),(2,5)\}$ et les tracer.

Exercice 9.2. Montrer que la silhouette de persistance d'ordre $q = 1$ est la moyenne pondérée (par la persistance) des fonctions tentes.

Exercice 9.3 (Implémentation)**.** Construire un pipeline complet qui classifie des nuages de points échantillonnés sur le cercle vs. la figure en huit, en utilisant des images de persistance et un SVM.

Exercice 9.4. Montrer que le noyau de Reininghaus est invariant par réflexion diagonale : $k_\sigma(D, D') = k_\sigma(D', D)$.

Exercice 9.5 (Comparaison expérimentale)**.** Comparer les performances de classification (accuracy, temps de calcul) entre paysages, images et noyaux de persistance sur un jeu de données de votre choix.

Exercice 9.6 (Stabilité des paysages de persistance)**.** Soient D_1 et D_2 deux diagrammes de persistance et $\lambda_k^{(1)}$, $\lambda_k^{(2)}$ leurs k-èmes paysages respectifs.

1. Montrer que pour chaque point (b, d) du diagramme, la fonction tente $\Lambda(t) = \max(0, \min(t - b, d - t))$ est 1-lipschitzienne.

2. En utilisant le fait que l'opération kmax est 1-lipschitzienne en norme ℓ^∞ sur les vecteurs triés, montrer que :

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

3. Généraliser au cas L^p : montrer que $\left\| \lambda_k^{(1)} - \lambda_k^{(2)} \right\|_{L^p} \leq C_p \cdot W_p(D_1, D_2)^{1/p}$ et discuter la constante C_p.

Exercice 9.7 (Pipeline SVM à noyau avec features de persistance)**.** On souhaite classifier des nuages de points en trois classes : cercle, tore (section 2D) et sphère (échantillonnée en 3D).

1. Pour chaque nuage, calculer le diagramme de persistance en homologie de degrés 0 et 1 via un complexe de Vietoris–Rips.

2. Construire la matrice de Gram du noyau de Reininghaus : $G_{ij} = k_\sigma(D_i, D_j)$ pour un paramètre σ choisi par validation croisée.

3. Entraîner un SVM à noyau précalculé (`kernel='precomputed'`) et rapporter l'accuracy en validation croisée 5-folds.

4. Comparer avec un pipeline utilisant des images de persistance concaténées (degrés 0 et 1) et un SVM RBF. Discuter les avantages et inconvénients de chaque approche.

Chapitre 10

Applications

La TDA est née dans les départements de mathématiques, mais elle a prouvé sa valeur dans les laboratoires, les hôpitaux et les entreprises. En 2011, Lum et al. utilisent Mapper pour découvrir un sous-groupe de cancer du sein invisible aux méthodes statistiques classiques. En neurosciences, Giusti et al. (2015) détectent des structures topologiques dans l'activité neuronale du cortex visuel. En matériaux, Kramar et al. analysent la structure granulaire sous compression. En finance, Gidea et Katz (2018) utilisent l'homologie persistante pour détecter les signaux précurseurs de krachs boursiers. Ce chapitre passe en revue ces applications, montrant comment les invariants topologiques apportent un éclairage complémentaire aux méthodes statistiques et géométriques traditionnelles.

Intuition

La TDA a trouvé des applications dans de nombreux domaines scientifiques. Sa force réside dans sa capacité à extraire des informations **qualitatives** (nombre de trous, de composantes, de cavités) à partir de données quantitatives, et ce de manière **robuste au bruit** et **invariante par déformation**.

10.1 Analyse de formes

Définition 10.1 (Descripteur topologique de forme)**.** Soit $\mathcal{S} \subset \mathbb{R}^3$ une surface triangulée. On peut construire plusieurs filtrations :

1. **Filtration géodésique** : $f(x) = d_{\mathcal{S}}(x, x_0)$ pour un point base x_0.
2. **Filtration par courbure** : $f(x) = \kappa(x)$ où κ est la courbure gaussienne.
3. **Heat kernel signature** : $f(x) = h_t(x, x)$ la diagonale du noyau de la chaleur à temps t.

Le diagramme de persistance de chaque filtration donne un descripteur topologique de la forme.

Exemple 10.2 (Classification de formes 3D)**.** On considère une base de données de modèles 3D (SHREC, ModelNet). Pour chaque objet :

1. On calcule la filtration par la distance géodésique depuis plusieurs points base.
2. On obtient un ensemble de diagrammes de persistance en dimensions 0 et 1.

3. On vectorise (images de persistance) et on classifie (SVM).

Cette approche est naturellement invariante par isométrie.

Proposition 10.3 (Invariance)**.** Les descripteurs topologiques issus de filtrations géodésiques sont invariants par isométrie de la surface. Ceux issus de filtrations par courbure sont invariants par mouvements rigides.

10.2 Structure des protéines

Définition 10.4 (Représentation topologique d'une protéine)**.** Une protéine est modélisée comme un nuage de points dans $\mathbb{R}^3$ (les positions des atomes C_α de la chaîne principale). La filtration de Rips–Vietoris sur ce nuage capture :

- H_0 : la connectivité de la chaîne.
- H_1 : les boucles et les structures secondaires (α-hélices, feuillets β).
- H_2 : les cavités et poches.

Exemple 10.5 (Détection de poches de liaison)**.** Les poches de liaison d'une protéine (sites où se fixent les ligands) correspondent à des cavités topologiques (H_2). Les générateurs persistants en H_2 identifient ces poches : un point (b, d) avec une grande persistance $d - b$ signale une cavité **stable** à travers les échelles.

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

# Positions des C-alpha (extrait d'un fichier PDB)
coords = np.loadtxt("protein_ca.xyz")

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

# Filtrer H2 (cavites)
h2 = [(b, d) for dim, (b, d) in dgm if dim == 2 and d - b > 1.0]
print(f"Nombre de poches significatives: {len(h2)}")
```

10.3 Analyse de séries temporelles

Définition 10.6 (Plongement de Takens)**.** Soit $(x_t)_{t=0}^{T}$ une série temporelle. Le **plongement de Takens** avec dimension d et délai τ est :

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

Théorème 10.7 (Takens, 1981)**.** *Si la série temporelle est issue d'un système dynamique sur une variété $\mathcal{M}$ de dimension m, alors pour $d > 2m$ génériquement, le plongement de Takens est un plongement de $\mathcal{M}$ dans $\mathbb{R}^d$.*

Intuition

L'idée est de reconstruire l'espace des phases du système dynamique à partir de la seule observation scalaire. Ensuite, la TDA sur le nuage de points reconstruit détecte la topologie de l'attracteur : un cycle limite donne $\beta_1 = 1$, un tore donne $\beta_1 = 2, \beta_2 = 1$, etc.

Exemple 10.8 (Détection de périodicité)**.** Pour une série périodique $x_t = \sin(2\pi t/T)$, le plongement de Takens avec $d = 2$ trace une ellipse dans $\mathbb{R}^2$. Le diagramme H_1 contient un point très persistant, confirmant la périodicité.

```python
import numpy as np
from gudhi.representations import PersistenceImage
from gtda.time_series import SingleTakensEmbedding

# Serie temporelle periodique
t = np.linspace(0, 10 * np.pi, 1000)
x = np.sin(t) + 0.1 * np.random.randn(len(t))

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

10.4 Analyse d'images

Définition 10.9 (Filtration de sous-niveau pour images)**.** Soit $I : \{0, \dots, W\} \times \{0, \dots, H\} \to \mathbb{R}$ une image en niveaux de gris. La **filtration de sous-niveau** est :

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

La persistance de cette filtration capture les composantes connexes sombres (H_0) et les régions claires entourées de sombre (H_1).

Exemple 10.10 (Segmentation topologique)**.** En imagerie médicale, les tumeurs apparaissent comme des régions d'intensité différente. Les composantes persistantes en H_0 de la filtration de sous-niveau identifient ces régions de manière robuste, sans seuillage arbitraire.

10.5 Réseaux de capteurs

Définition 10.11 (Couverture topologique)**.** Un réseau de n capteurs avec portée r couvre un domaine $\Omega \subset \mathbb{R}^2$. Le complexe de Čech à rayon r construit sur les positions des capteurs permet de détecter les **trous de couverture** : $\beta_1 > 0$ signale un trou dans la couverture.

Théorème 10.12 (De Silva–Ghrist, 2007)**.** *Si le complexe de Rips* Rips_{2r} *des positions de capteurs a une homologie réduite triviale et si les capteurs ont une portée de détection* $r_s \geq 2r$*, alors le domaine est couvert.*

Remarque 10.13. L'avantage de cette approche est qu'elle ne nécessite pas de connaître les positions exactes des capteurs (seulement les distances), et elle fonctionne sans coordonnées.

10.6 Science des matériaux

Exemple 10.14 (Analyse de matériaux poreux)**.** Les matériaux poreux (zéolithes, aérogels) sont caractérisés par leur structure de pores. La TDA sur la représentation atomique capture :

- H_0 : les grains et agrégats.
- H_1 : les canaux et tunnels.
- H_2 : les cavités fermées.

La persistance de ces structures est corrélée aux propriétés macroscopiques (perméabilité, surface spécifique).

Exemple 10.15 (Verres métalliques amorphes)**.** Nakamura et al. (2015) ont utilisé la persistance pour distinguer les verres métalliques des liquides : les diagrammes H_1 et H_2 montrent des structures persistantes dans le verre absentes dans le liquide, sans nécessiter de paramètre d'ordre.

10.7 Exercices

Exercice 10.1. Calculer le diagramme de persistance H_0 et H_1 d'une série temporelle $x_t = \sin(t) + 0.5\sin(3t)$ après plongement de Takens avec $d = 3$. Interpréter les résultats.

Exercice 10.2. Pour une image binaire 100×100 contenant trois disques et un anneau, prédire le diagramme de persistance de la filtration de sous-niveau, puis vérifier avec `gudhi`.

Exercice 10.3 (Réseau de capteurs)**.** Générer aléatoirement 50 capteurs dans le carré unité. Utiliser la persistance H_1 pour détecter les trous de couverture en fonction du rayon r. Tracer la courbe du nombre de trous en fonction de r.

Exercice 10.4 (Projet)**.** Appliquer la TDA à un jeu de données réel de votre choix (par exemple : signaux EEG, données financières, images satellitaires). Présenter la filtration choisie, les diagrammes obtenus et leur interprétation.

Chapitre 11

TDA et Apprentissage Profond

Intuition

L'intégration de la TDA dans l'apprentissage profond pose un défi fondamental : le calcul de la persistance n'est a priori **pas différentiable**. Les travaux récents ont montré comment rendre la persistance différentiable, permettant d'entraîner des réseaux de neurones de bout en bout avec des objectifs topologiques.

11.1 Persistance différentiable

Définition 11.1 (Persistance comme fonction des filtrations)**.** Considérons un complexe simplicial K avec une filtration paramétrique $f_\theta : K \to \mathbb{R}$ dépendant de paramètres $\theta \in \mathbb{R}^p$. Le diagramme de persistance $\mathrm{Dgm}(f_\theta)$ est une fonction de θ.

Théorème 11.2 (Différentiabilité — Gameiro et al. 2016, Poulenard et al. 2018)**.** *Soit K un complexe simplicial fini et $f : K \to \mathbb{R}$ une filtration générique (valeurs de filtration distinctes). Les coordonnées de naissance $b_i(\theta)$ et de mort $d_i(\theta)$ dans le diagramme de persistance sont des fonctions différentiables de θ presque partout. Plus précisément :*

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

où σ_{b_i} et σ_{d_i} sont les simplexes qui créent et tuent la classe i.

Points de non-différentiabilité

La différentiabilité échoue quand deux valeurs de filtration coïncident (les simplexes de naissance et de mort peuvent changer). En pratique, on ajoute une petite perturbation ou on utilise des sous-gradients.

Gradient de la persistance

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

```
import torch
```

```python
from topologylayer.nn import RipsLayer

# Nuage de points differentiable
X = torch.randn(50, 2, requires_grad=True)

# Couche de persistance differentiable
rips_layer = RipsLayer(maxdim=1, sublevel=False)
dgm = rips_layer(X)

# Loss basee sur la persistance H1
loss = -dgm[1][:, 1].sum()  # maximiser les persistances H1
loss.backward()
print(f"Gradient shape: {X.grad.shape}")
```

11.2 PersLay : couche de persistance générique

Définition 11.3 (PersLay — Carrière et al. 2020)**. PersLay** est une couche neuronale qui transforme un diagramme de persistance $D = \{(b_i, d_i)\}_{i=1}^n$ en un vecteur de dimension fixe via :

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

où :

- $w : \mathbb{R}^2 \to \mathbb{R}_+$ est une fonction de poids (atténuation des points proches de la diagonale).
- $\phi : \mathbb{R}^2 \to \mathbb{R}^q$ est une fonction de représentation point par point (apprise ou fixée).
- op est une opération d'agrégation (somme, max, top-k, attention).

Proposition 11.4 (Universalité de PersLay)**.** Avec des choix adéquats de w, ϕ et op, PersLay peut approcher uniformément toute fonction continue et invariante par permutation sur les diagrammes de persistance.

Remarque 11.5. PersLay unifie les représentations classiques :

- **Paysages** : ϕ est la fonction tente, op est top-k.
- **Images** : ϕ est une gaussienne, op est la somme.
- **Silhouettes** : ϕ est la fonction tente, op est la moyenne pondérée.

11.3 Régularisation topologique

Définition 11.6 (Perte topologique)**.** Une **perte topologique** est un terme ajouté à la fonction de coût pour encourager ou pénaliser certaines structures topologiques. Formellement :

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

où $\mathcal{L}_{\text{topo}}$ est calculée à partir du diagramme de persistance.

Exemple 11.7 (Perte topologique pour la segmentation)**.** En segmentation d'images, on peut pénaliser les petits trous parasites dans la prédiction :

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

qui force le réseau à produire des régions simplement connexes.

Théorème 11.8 (Convergence de la régularisation topologique)**.** *Sous des conditions de régularité (filtration générique, Lipschitz), la descente de gradient stochastique avec perte topologique converge vers un point critique de* $\mathcal{L}_{\text{total}}$.

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

class TopologicalLoss(nn.Module):
    """Penalise les trous parasites en segmentation."""
    def __init__(self, lam=1.0):
        super().__init__()
        self.lam = lam
        # rips_layer defini avec une couche differentiable

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

11.4 Auto-encodeurs topologiques

Définition 11.9 (Auto-encodeur topologique — Moor et al. 2020)**.** Un **auto-encodeur topologique** est un auto-encodeur (Enc, Dec) dont la perte inclut un terme topologique :

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

où d_W est une distance de Wasserstein entre les diagrammes de persistance de l'espace d'entrée X et de l'espace latent $\text{Enc}(X)$.

Intuition

L'idée est de préserver la **topologie** des données lors de la réduction de dimension. Un auto-encodeur classique (ou un VAE) peut écraser des boucles ou fusionner des composantes connexes. Le terme topologique pénalise ces distorsions.

Proposition 11.10 (Préservation topologique)**.** Si $\mathcal{L}_{\text{topo}} = 0$, alors les nombres de Betti de X et de $\text{Enc}(X)$ coïncident pour la filtration de Rips utilisée.

11.5 Réseaux de neurones sur graphes et homologie persistante

Définition 11.11 (Filtration apprise sur un graphe)**.** Soit $G = (V, E)$ un graphe avec des features de nœuds $h_v \in \mathbb{R}^d$. On définit une filtration apprise :

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

où $g_\theta : \mathbb{R}^d \to \mathbb{R}$ est un réseau de neurones. Le diagramme de persistance de cette filtration fournit des features topologiques du graphe.

Exemple 11.12 (Classification de graphes)**.** Pour la classification de graphes moléculaires :

1. Un GNN produit des embeddings de nœuds h_v.
2. Une filtration apprise f_θ est calculée.
3. Le diagramme de persistance est vectorisé via PersLay.
4. Le vecteur résultant est concaténé au readout du GNN.

Cette combinaison améliore l'expressivité au-delà du test de Weisfeiler-Leman.

Théorème 11.13 (Expressivité topologique — Horn et al. 2022)**.** *Il existe des paires de graphes non distinguables par le test de Weisfeiler-Leman de niveau k mais distinguables par la persistance de la filtration induite par un GNN de message passing.*

11.6 Implémentation avec PyTorch

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

class TopoGNN(nn.Module):
    """GNN avec features topologiques."""
    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)
        # perslay: couche de vectorisation apprise
        # self.perslay = PersLay(...)

    def forward(self, x, edge_index, batch):
        h = torch.relu(self.conv1(x, edge_index))
        h = torch.relu(self.conv2(h, edge_index))
        # Readout classique
        graph_emb = global_mean_pool(h, batch)
        # Filtration apprise
```

```
        filt_vals = self.filtration(h).squeeze()
        # ... calcul de persistance et PersLay ...
        # topo_features = self.perslay(dgm)
        # out = self.classifier(
        #     torch.cat([graph_emb, topo_features], dim=-1)
        # )
        return graph_emb  # simplifie
```

11.7 Exercices

Exercice 11.1. Calculer à la main le gradient de la persistance totale $\sum_i (d_i - b_i)$ par rapport aux valeurs de filtration pour le complexe $\{a, b, c, ab, bc\}$ avec $f(a) = 0, f(b) = 1, f(c) = 2, f(ab) = 1, f(bc) = 2$.

Exercice 11.2. Montrer que PersLay avec $\phi(b, d) = (b, d - b)$, $w \equiv 1$ et op = sum revient à calculer $(\sum b_i, \sum (d_i - b_i))$.

Exercice 11.3 (Implémentation)**.** Implémenter un auto-encodeur topologique simple sur un jeu de données en forme de cercle. Comparer l'espace latent avec et sans le terme topologique.

Exercice 11.4. Donner un exemple de deux graphes non isomorphes indistinguables par 1-WL mais distinguables par la persistance H_0 d'une filtration par degré.

Exercice 11.5 (Projet)**.** Entraîner un réseau de segmentation d'image (U-Net) avec et sans régularisation topologique sur un jeu de données de vaisseaux sanguins. Comparer les nombres de Betti des prédictions.

Bibliographie

[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.