Dynamic persistent homology: Computation and implementation

Jānis Lazovskis

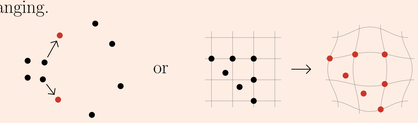
Institute of Clinical and Preventive Medicine
University of Latvia

Topical Workshop "Foundations of Computational Geometry and Topology"
ICERM, May 18-21, 2025, Providence, Rhode Island, USA

Funding provided by Activity 1.1.1.9 "Post-doctoral Research" of the Specific Objective 1.1.1 "Strengthening research and innovative capacities and introduction of advanced technologies in the common R&D system" of the European Union's Cohesion Policy Programme for 2021-2027 research application No 1.1.1.9/LZP/1/24/125 "Efficient topological signatures for representation learning in medical imaging"

What **dynamics** could I consider?

Yes: *The* ***data*** *is changing:* The relationship among the elements of the underlying data set changes, either from the elements themselves changing, or from the distance function changing.



Yes: *The* ***filtration*** *is changing:* The priority of the underlying data or associated structures changes. This may imply a change in the order of higher dimensional components, depending on the filtration.

Maybe: *The filtration is a* ***zigzag*** *:* Zigzag filtrations can encode both entrance and exit times. This allows for a single static filtration if the dynamics are known beforehand, but still requires updates if there are unknown changes to happen.

$$K_0 \xleftarrow{\sigma} K_1 \xrightarrow{\tau} K_2 \xleftarrow{\rho} K_3$$
$$\Downarrow$$
$$K_0 \xleftarrow{\sigma} K_1 \xrightarrow{\tau} K_2 \xleftarrow{\sigma} K_2' \xleftarrow{\rho} K_3'$$

Maybe: *The data comes from a* ***dynamical system*** *:* Dynamical systems encode change, and are related to TDA by decomposing a space into cells and by providing a filter function to a space. The change follows a particular pattern, and the dynamical system changes if the pattern changes.

What context is useful for dynamics in persistent homology?

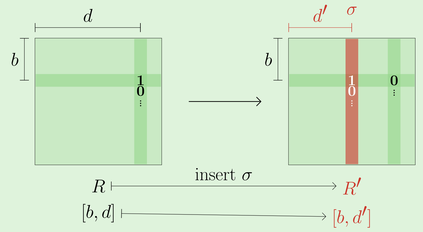
Efficiency: When change is localized, many steps of the initial computation are the same. Instead of recomputing from scratch, partial results may be reused.

ML pipelines: Efficiency at scale means similar inputs (for example, training data) with or without topological changes (for example, when validating) have minimal computational cost when producing persistence.

Inverse problems: Given a particular type of dynamics, classify the underlying changes that could have caused them. For example, all changes that cause a bar, or class of bars, to become shorter, by no more than a fixed amount.

How does **topology** see dynamics?

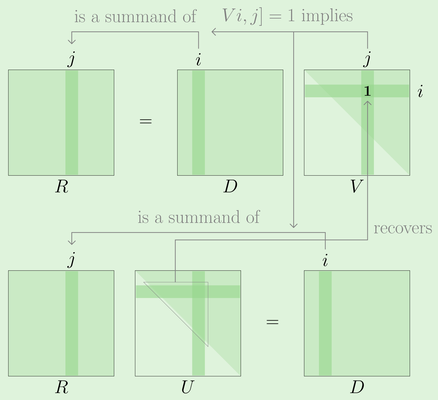
A change in the ***barcode*** *:* If the changes to the reduced boundary matrix produced by the dynamics result in pivot changes, then it may be possible to relate changes in the barcode to the dynamics. The relationship also may be measured by a distance on bars as elements of $\mathbb{R}^2$.



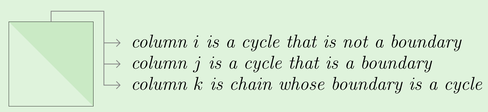
A change in the ***representative cycles*** *:* When two different reduction procedures produce the same barcode, the representative cycles may not be the same. Representatives of finite cycles may be retrieved from R, whereas representatives of the infinite cycles are not visible in R.

$$c_1 + \partial\left(\sigma_i + \cdots + \sigma_j - \cdots - \sigma_k\right) = c_2$$

Both the $R = DV$ or $RU = D$ matrix **factorization**, or decomposition, provide enough information to retrieve the representatives of the infinite classes. The matrices $U, V = U^{-1}$ are upper triangular if left-to-right column additions are used in the reduction of R.



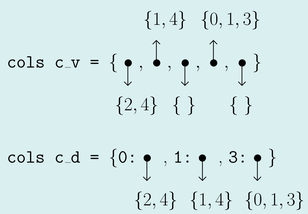
The upper triangular **chain matrix** M computes a consistent basis for the underlying chain complex. Each column is a chain, classified as one of three types.



How does **PH software** see what topology sees?

In the ***matrix*** *container:* Optimizing for sparse representation, often a vector (or dictionary) of (references to) vectors is used. If the containers are **column major**, then accessing columns is fast, but accessing rows is slow.

$$D = \begin{bmatrix} 0&0&0&1&0 \\ 0&1&0&1&0 \\ 1&0&0&0&0 \\ 0&0&0&1&0 \\ 1&1&0&0&0 \end{bmatrix}$$



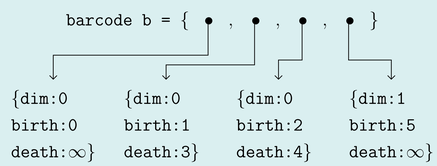
Knowing which matrices or rows need accessing allows optimization for faster **row access** of some rows. In the DV or RU factorizations, the matrix V or U may be stored with row access. Note that row i of V may be inferred by summing rows $i, i+1, \ldots$ of U.

$$\begin{bmatrix} 1&a&b&c \\ &1&d&e \\ &&1&f \\ &&&1 \end{bmatrix} \begin{bmatrix} 1&p&q&r \\ &1&s&t \\ &&1&u \\ &&&1 \end{bmatrix} = I \qquad \begin{aligned} p + a &= 0 \\ q + as + b &= 0 \\ r + at + bu + c &= 0 \end{aligned}$$

V U

The same holds analogously if a row major implementation is used. Column major containers are often used is PH software, as the main operations are column additions.

In the ***barcode*** *container:* Persistence information in some software is computed before being requested explicitly by the user, and is stored in a separate container.



Or, this information may be read off directly from the reduced boundary matrix, and never stored. In both cases, there is a surjective map from the simplex indices to the barcode indices.

By ***column operations*** *:* Mathematically obvious changes have to be specified. Column addition, rearrangement, insertion, or removal depends on the container used to hold the columns.

- `columns.erase(Index)` only erases one entry (one column), and does not adjust elements of other vectors
- if **several changes** happen at once, it is often more efficient to perform all index changes at once at the end, rather than at every intermediate step

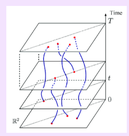
What **methods** are available for dynamic modification?

As all widely-used PH software has source code publicly available, technically everything is open to modification. But the developer(s) make explicit choices as to what functions or structures are implemented only for particular cases, and which are allowed to be used outside a predetermined context.

GUDHI: Has the vines-and-vineyards algorithm [C-SEM06] implemented, using the RU-decomposition. An implementation of SiRUP [GL26] for simplex removal is in preparation:

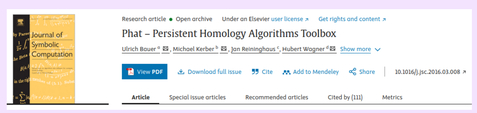
- `matrix.vine_swap(Index)`
- `matrix.insert_maximal_cell(Index, Boundary)`
- `matrix.remove_maximal_cell(Index)`
- `matrix.remove_maximal_cell_sirup(Index)`

The barcode dictionary cannot be directly modified, as it is private within a particular class.



Dionysus: Dionysus 1 has methods for creating a vineyard induced by changes in the vertex order, in a lower-star filtration. This has been generalized in the **PDB** package for Python, having persistence diagrams parametrized over a topological space, rather than just the interval:

- `swap_rows(i,j)`
- `swap_cols(i,j)`



PHAT: No inherent methods for dynamics, but variations in data structures and algorithms make it a rich test bed. Can be modified to track representative cycles of infinite classes.

References

[ELZ02] Edelsbrunner, Letscher, Zomorodian. *Topological Persistence and Simplification,* Discrete & Computational Geometry, 2002.

[C-SEM06] Cohen-Steiner, Edelsbrunner, Morozov. *Vines and Vineyards by Updating Persistence in Linear Time,* Symposium of Computational Geometry, 2006.

[MO15] Maria, Oudot. *Zigzag Persistence via Reflections and Transpositions,* Symposium on Discrete Algorithms, 2015.

[DH22] Dey, Hou. *Fast Computation of Zigzag Persistence,* European Symposium on Algorithms, 2022.

[GL26] Giunti, Lazovskis. *Pruning vineyards: Updating barcodes and representative cycles by removing simplices,* Foundations of Data Science, 2026.