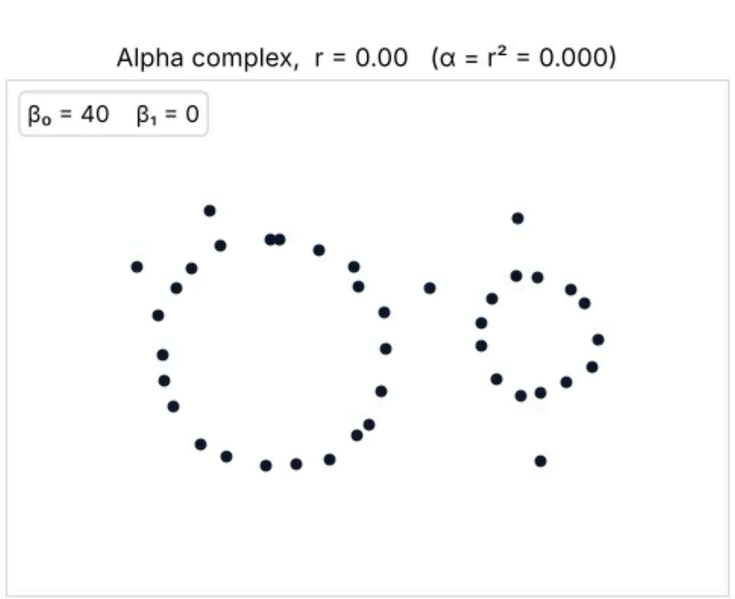
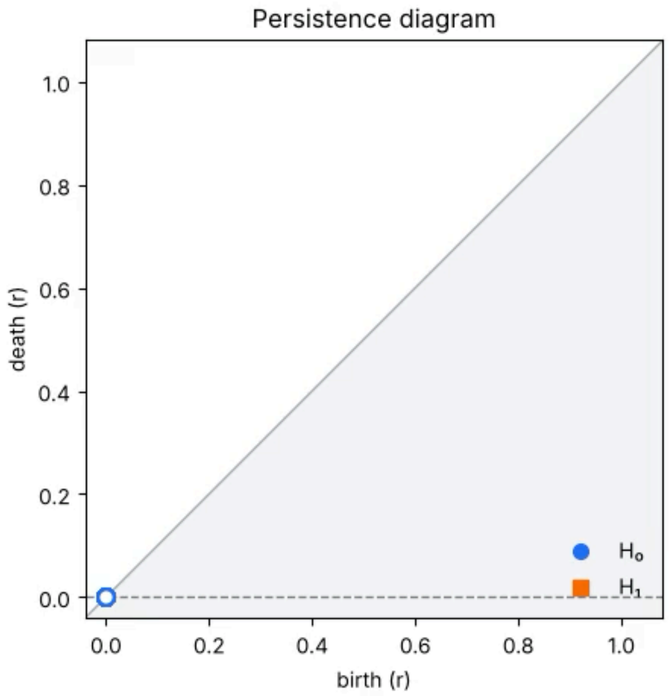
Computing Persistence

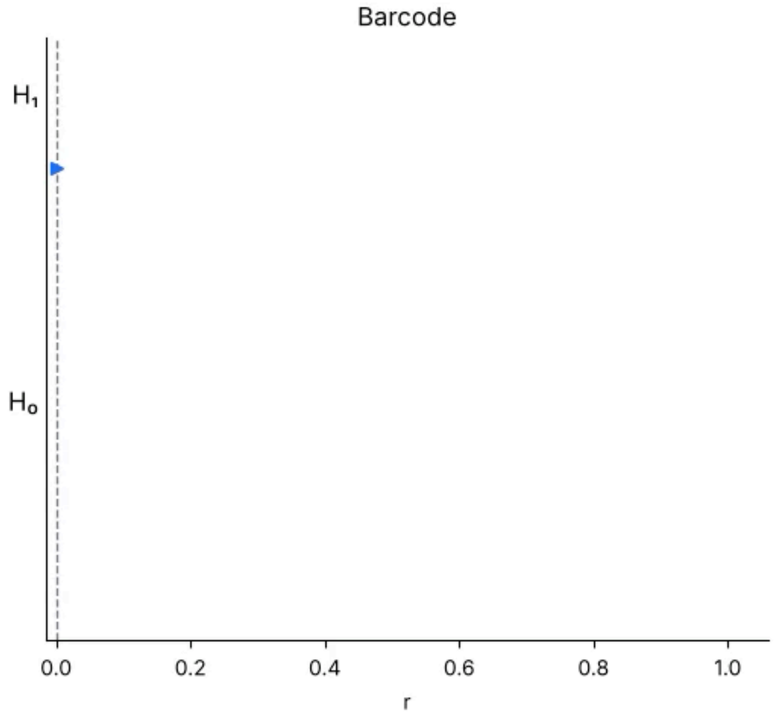
Elena Xinyi Wang

October 28, 2026

Everything up until now







But only mathematically... how about computationally?

Review of setup

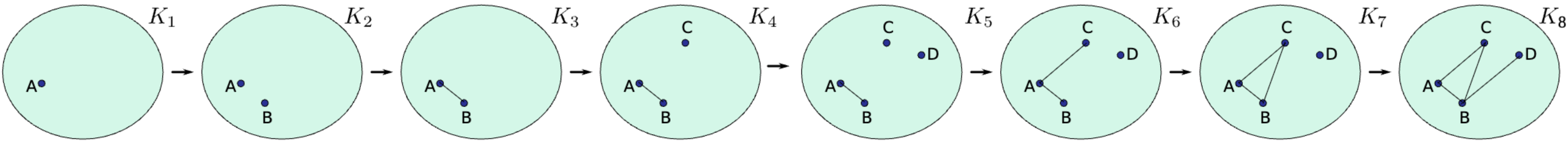
Simplicial Complex from a Filtration

A function $f : K \to \mathbb{R}$ is called monotone if for every $\tau \leq \sigma, f(\tau) \leq f(\sigma)$.

A simplex-wise filtration means that exactly one simplex is added at each step.

Fix values $a_0 \leq a_1 \leq \cdots \leq a_n$. Let $K_i = f^{-1}(-\infty, a_i\}$. The sublevelset filtration induced by this function is defined to be

$$\emptyset = K_0 \subseteq K_1 \subseteq K_2 \subseteq \cdots \subseteq K_n = K.$$



Persistence Module

Given a filtration, the p-dimensional *persistence module* is

$$H_p(\mathcal{F}) : 0 = H_p(K_0) \to H_p(K_1) \to H_p(K_2) \to \cdots \to H_p(K_n) = H_p(K)$$

with maps $h_p^{i,j} : H_p(K_i) \to H_p(K_j)$ induced by inclusion

The *pth persistent homology groups* are the images induced by inclusion

$$H_p^{i,j} = \mathrm{Im}\left(H_p(K_i) \to H_p(K_j)\right).$$

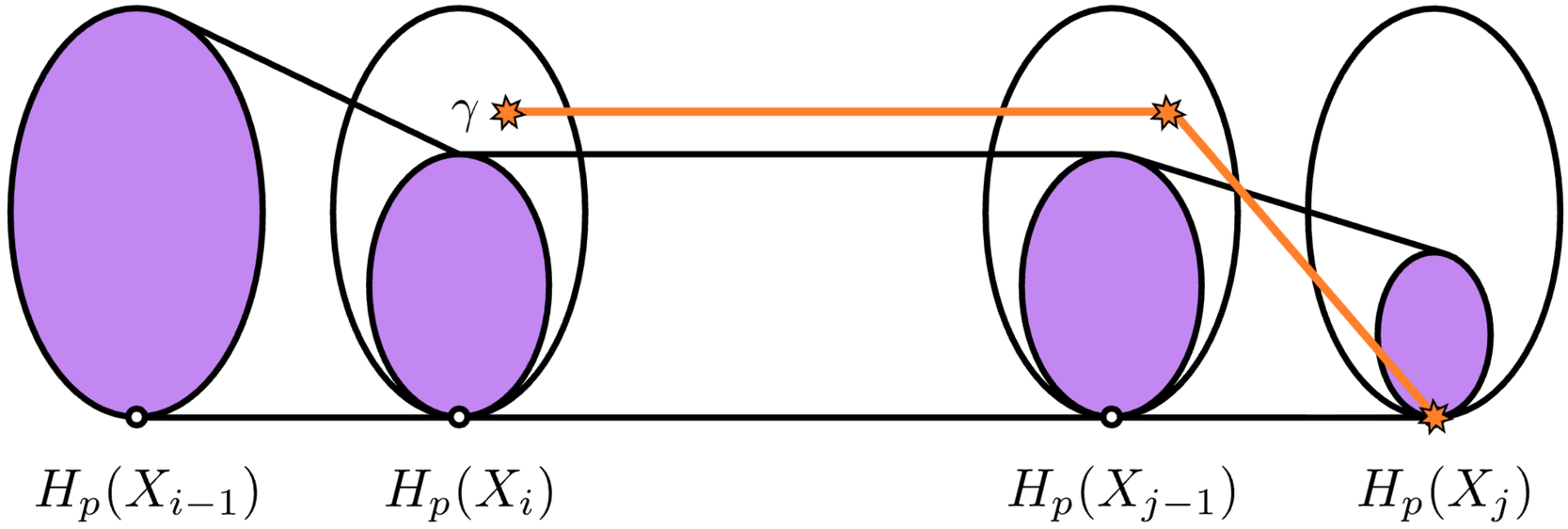
The pth persistent Betti numbers are the ranks

$$\beta_p^{i,j} = \mathrm{rank}\, H_p^{i,j}$$

Birth and Death (Book Version)

Birth: A class $\gamma \in H_p(K_i)$ is born at K_i if it is not in $H_P^{i-1,i}$

Death: The class γ dies entering K_j if $\gamma \in H_p(X_{j-1})$ is not trivial, but $h_p^{i-1,\,j}(\gamma) = 0$.



Persistence Algorithm

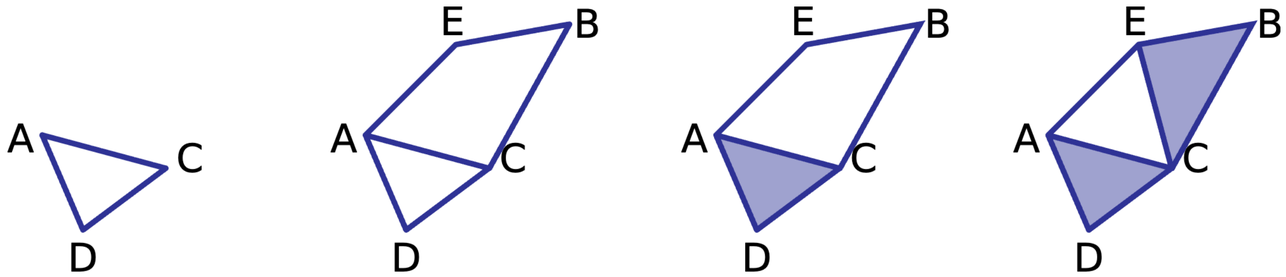
Compatible Filtration

Let $f : K \to \mathbb{N}$ give the index where any particular simplex appears in the filtration.

A compatible ordering of the simplices of K is a sequence $\sigma_1, \sigma_2, \ldots, \sigma_m$ such that

- $f(\sigma_i) \leq f(\sigma_j)$ implies $i < j$
- $\sigma_i \subsetneq \sigma_j$ implies $i < j$

(Could have just assumed that the filtration was simplex-wise to begin with)



K_0 K_1 K_2 K_3

Positive and Negative Simplices

Simplex-wise filtration:

When a p-simplex $\sigma_k = K_j \backslash K_{j-1}$ is added, exactly one of the following two possibilities occurs:

1. A non-boundary p-cycle c along with its classes $[c] + h$ for any class $h \in H_p(K_{j-1})$ are born (created). In this case, we call σ_j a *positive simplex* (also called a creator)

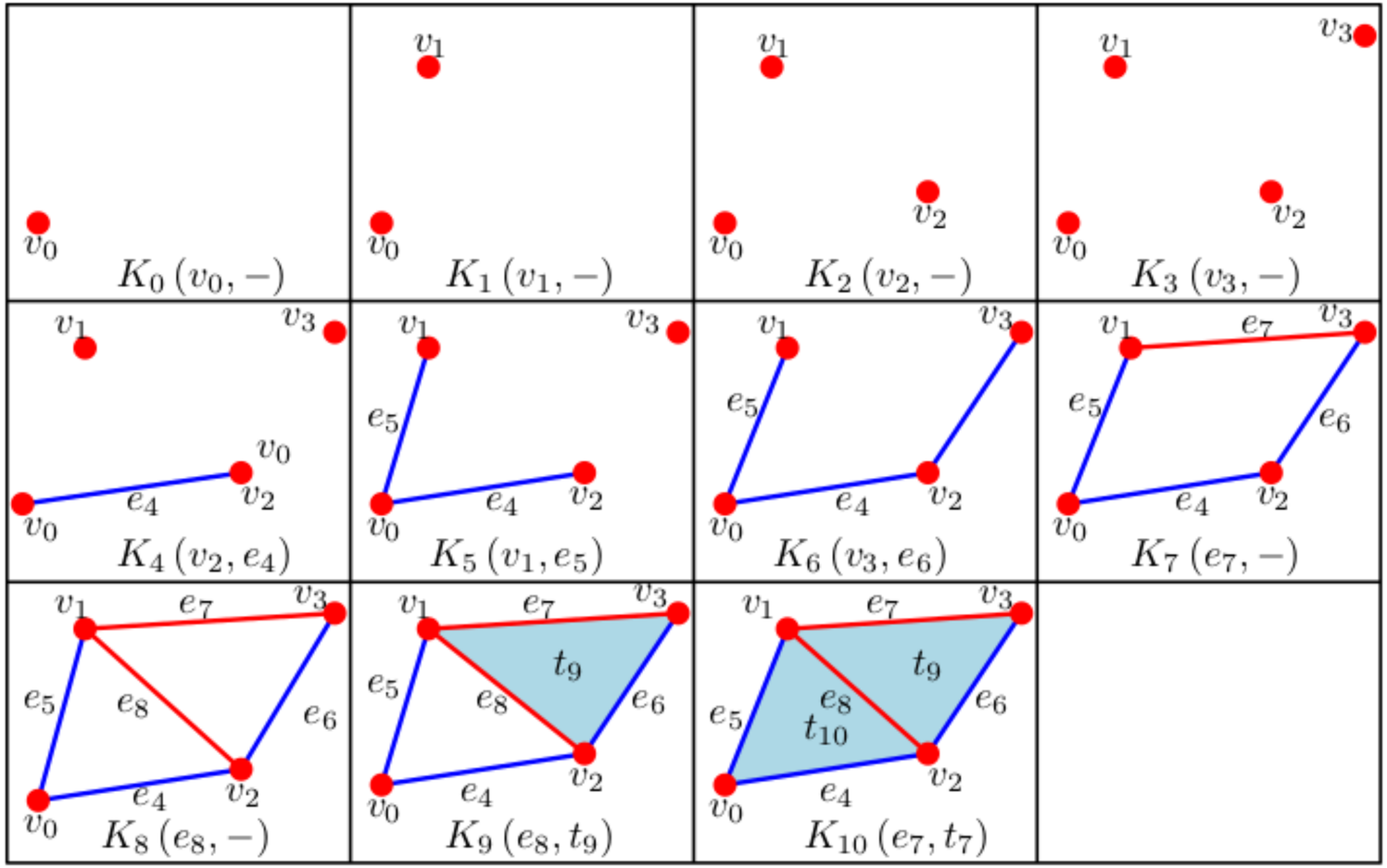
Positive and Negative Simplices

Simplex-wise filtration:

When a p-simplex $\sigma_k = K_j \backslash K_{j-1}$ is added, exactly one of the following two possibilities occurs:

1. A non-boundary p-cycle c along with its classes $[c] + h$ for any class $h \in H_p(K_{j-1})$ are born (created). In this case, we call σ_j a *positive simplex* (also called a creator)
2. An existing $(p-1)$-cycle c along with its classes $[c]$ dies (destroyed). In this case we call σ_j a *negative simplex* (also called a destructor).

Example

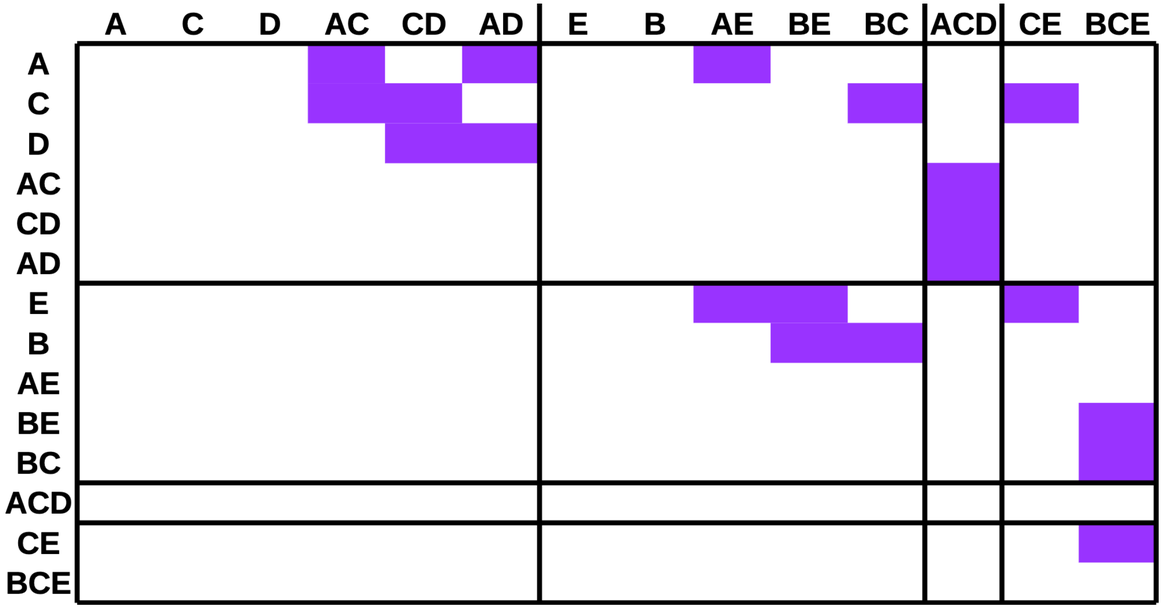

v_0
$K_0\,(v_0, -)$
v_1
v_0
$K_1\,(v_1, -)$
v_1
v_2
v_0
$K_2\,(v_2, -)$
v_3
v_1
v_2
v_0
$K_3\,(v_3, -)$
v_1
v_3
v_0
e_4
v_2
v_0
$K_4\,(v_2, e_4)$
v_1
v_3
e_5
e_4
v_2
v_0
$K_5\,(v_1, e_5)$
v_1
v_3
e_5
e_4
v_2
v_0
$K_6\,(v_3, e_6)$
v_1
e_7
v_3
e_5
e_6
e_4
v_2
v_0
$K_7\,(e_7, -)$
v_1
e_7
v_3
e_5
e_8
e_6
e_4
v_2
v_0
$K_8\,(e_8, -)$
v_1
e_7
v_3
t_9
e_5
e_8
e_6
e_4
v_2
v_0
$K_9\,(e_8, t_9)$
v_1
e_7
v_3
t_9
e_5
e_8
t_{10}
e_6
e_4
v_2
v_0
$K_{10}\,(e_7, t_7)$

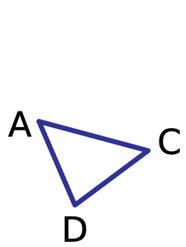
The Persistence Algorithm

Given the boundary matrix B with rows and columns in this total ordering. We will convert B to another matrix R using row/col operations.

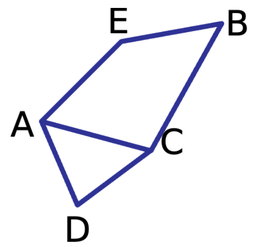
Result: $R = BV$ for some invertible V

- Let $\text{low}(j)$ be the row index of the lowest one in column j.
- If column j is entirely 0, $\text{low}(j)$ is NaN
- R is "reduced" if $\text{low}(j) \neq \text{low}(j')$ for any $j \neq j'$ which are not entirely zero columns.

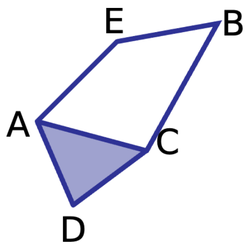




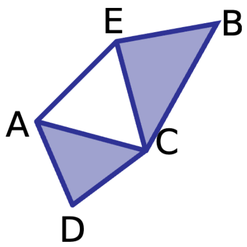
K_0



K_1



K_2



K_3

Idea

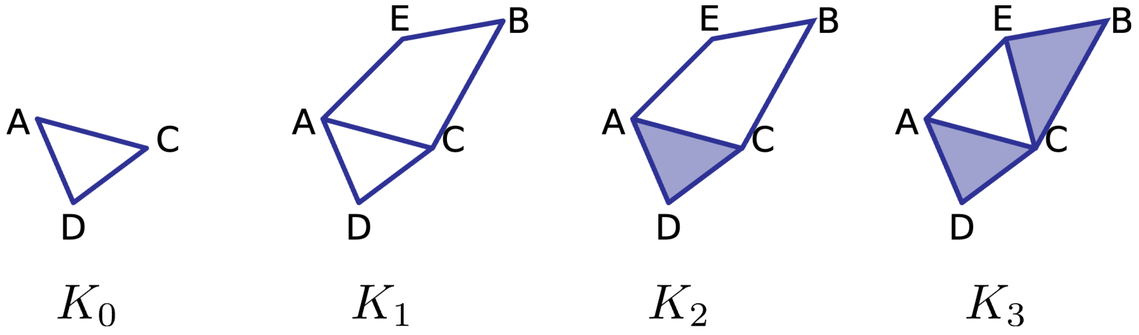
Persistence Algorithm

- R is the reduced boundary matrix, V records column operations
- B is upper triangular, only add from the right so it stays upper triangular
- Upper-left corner from (j, j) represents homology for the subcomplex stopping at σ_j
- If a column is entirely 0, the addition of that simplex created a new homology class
- If a column has a lowest one, then the addition of this simplex killed off a class that was there at the previous step.

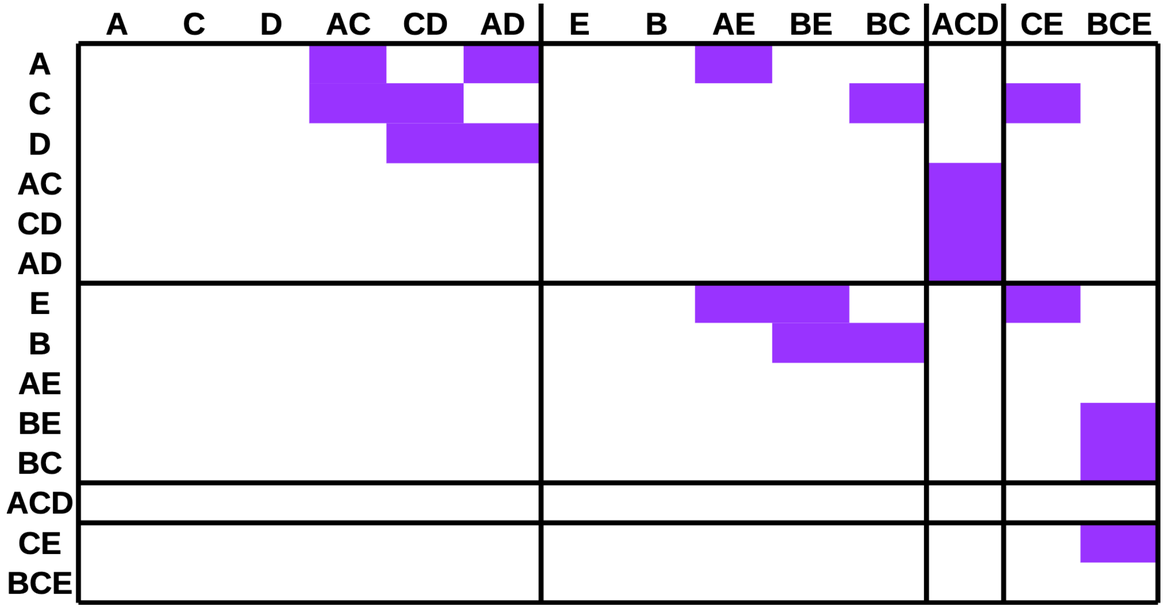
Persistence Algorithm

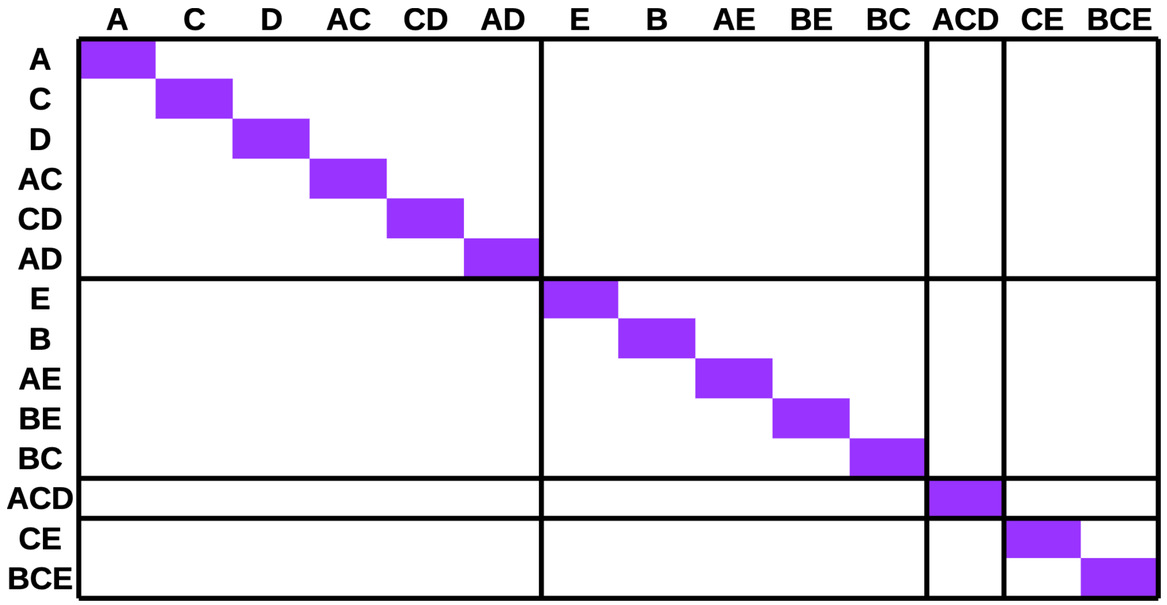
$R = B$

for $j = 1 \cdots m$ **do**
  while $\exists\, j' < j$ with $\text{low}(j') = \text{low}(j)$ **do**
    add column j' to column j
  end while
end for



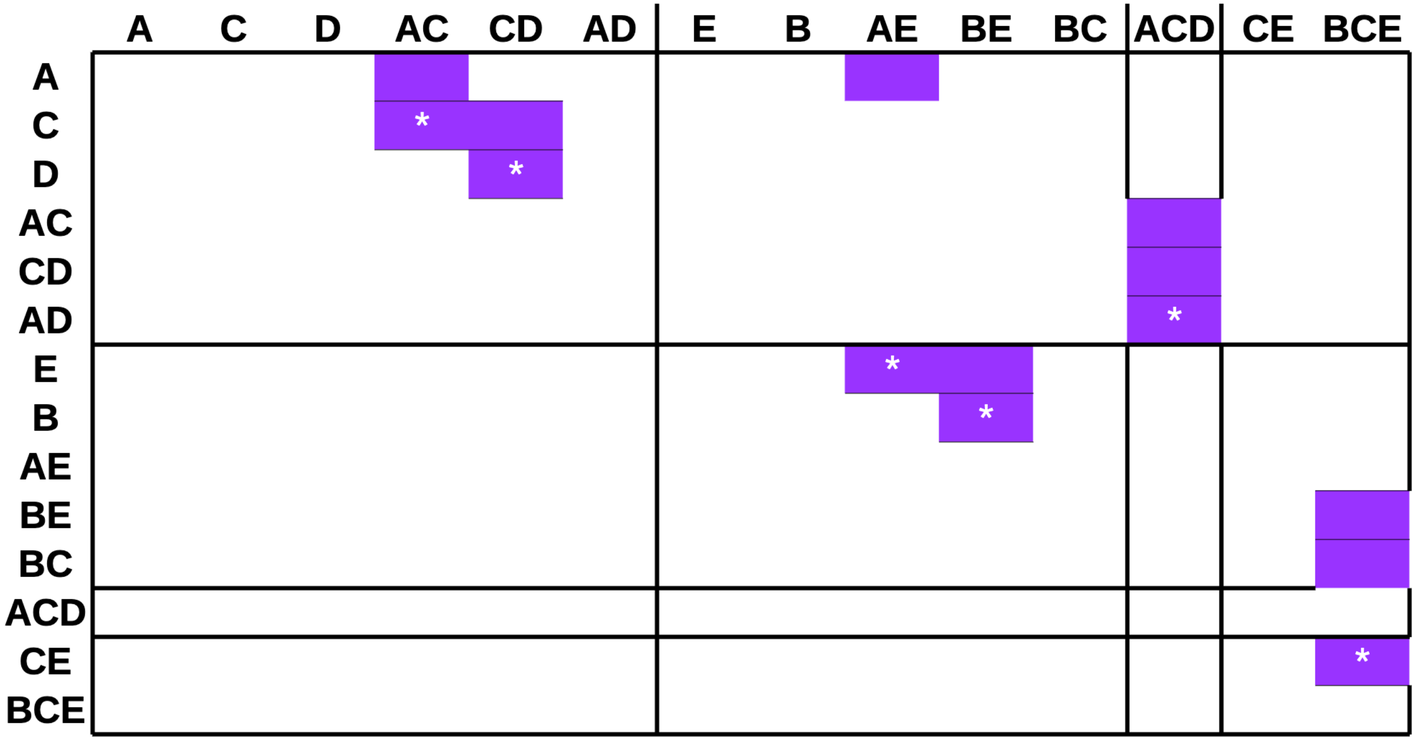
K_0 K_1 K_2 K_3

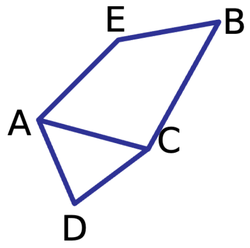




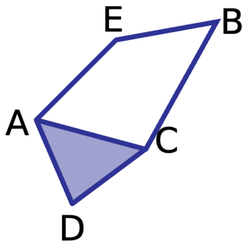
Pairing

- Every negative simplex must be paired with a previously entering positive simplex
- A negative simplex has a non-zero column, so it's paired with the simplex from the row of the lowest one

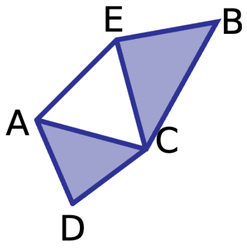




K_1

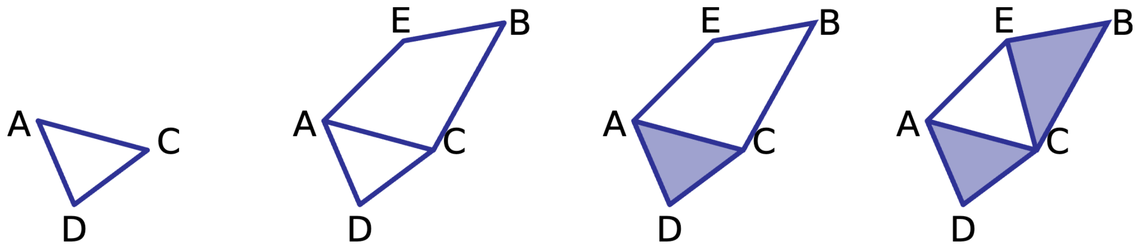


K_2

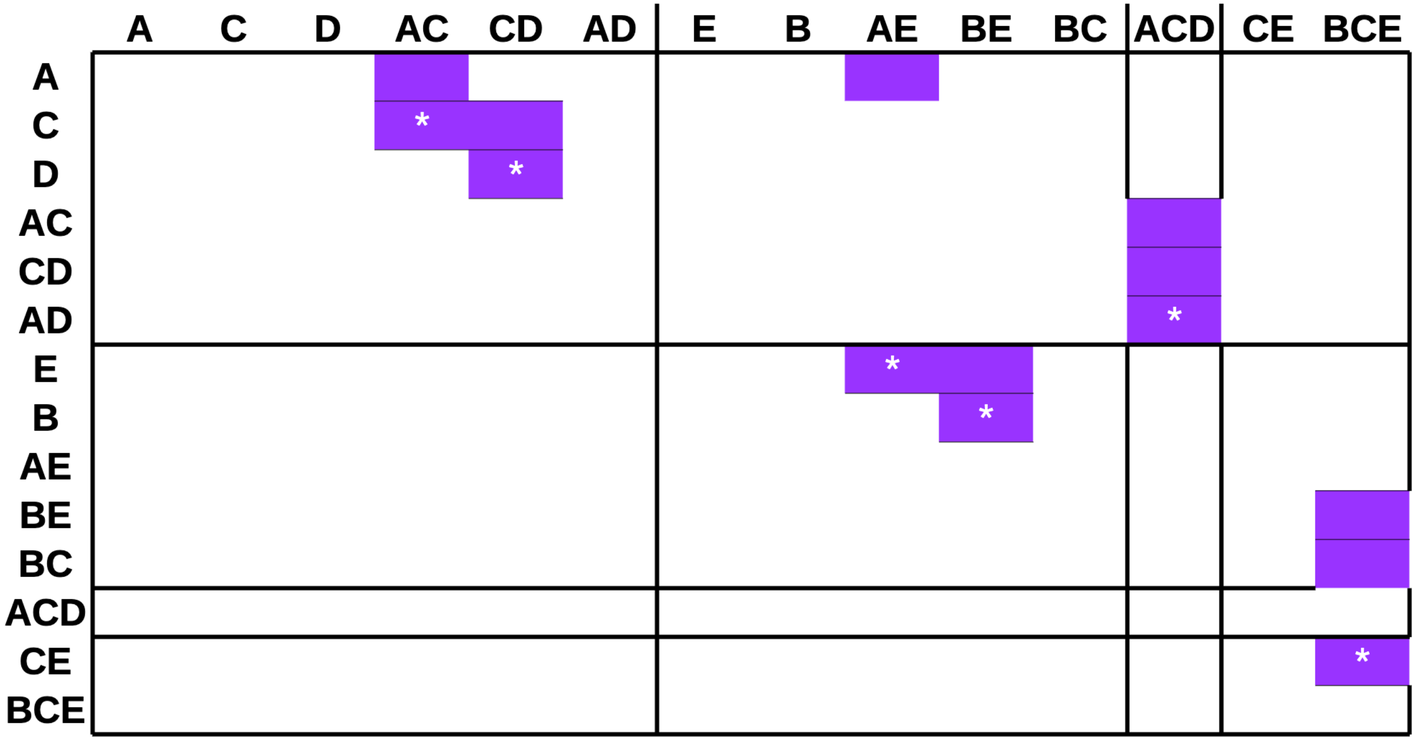


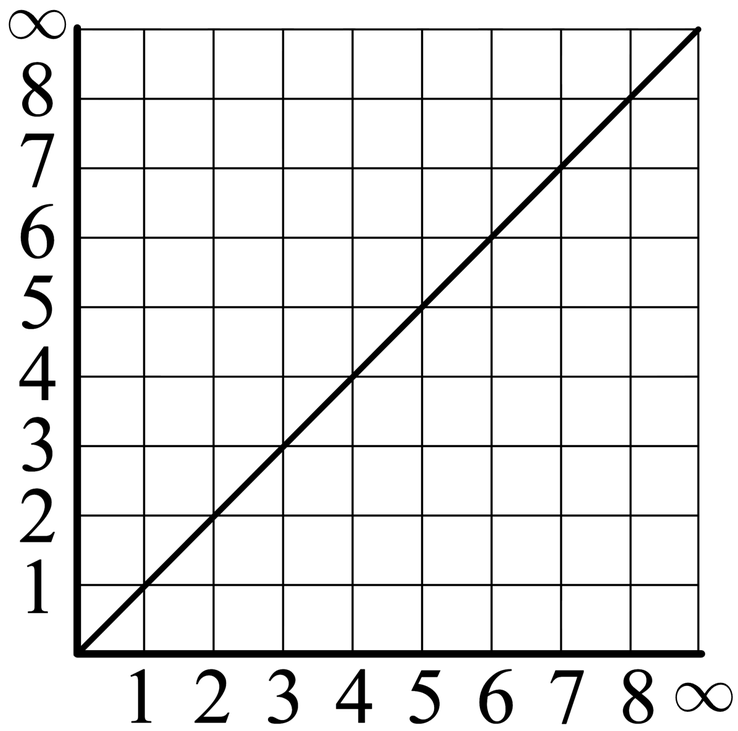
K_3

Reading off the persistence diagram



K_0 K_1 K_2 K_3





Complexity

- With m simplices, the matrix has size $m \times m$
- While loop: each such operation must move $\text{low}(j)$ up, but other columns can become 1 during the operation
- Worst case: $O(m)$ additions to cancel a row, each "add" takes $O(m)$ ➡ $O(m^3)$

$R = B$

for $j = 1 \cdots m$ **do**
    while $\exists\, j' < j$ with $\text{low}(j') = \text{low}(j)$ **do**
        add column j' to column j
    end while
end for

Improving this Runtime

- **Sparsity**: a simplex of dimension d has $d+1$ faces, hence the original boundary matrix is sparse
- **Efficient storage**: Sparse representations and bit operations can accelerate column additions in practice. (Practical speedup)
- The algorithm can also be reduced to Gaussian elimination: $O(m^{\omega})$, $2 \leq \omega \leq 2.25$ (October 22, 2026, but I can't verify if that's correct); I'm sure 2.373 is correct, though. (Theoretical worst-case bound)

More Speedups

Matrix algorithm:

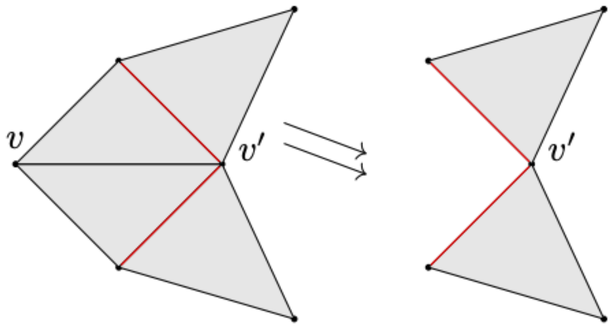
- Sweep columns left to right
- Every addition to col j moves low(j) higher
- If positive, it will zero it out and might take many column operations
- But once we zero it out, it's never used again!

Practical Improvement: *Clearing* [Chen and Kerber, 2011; Bauer et al, 2014]

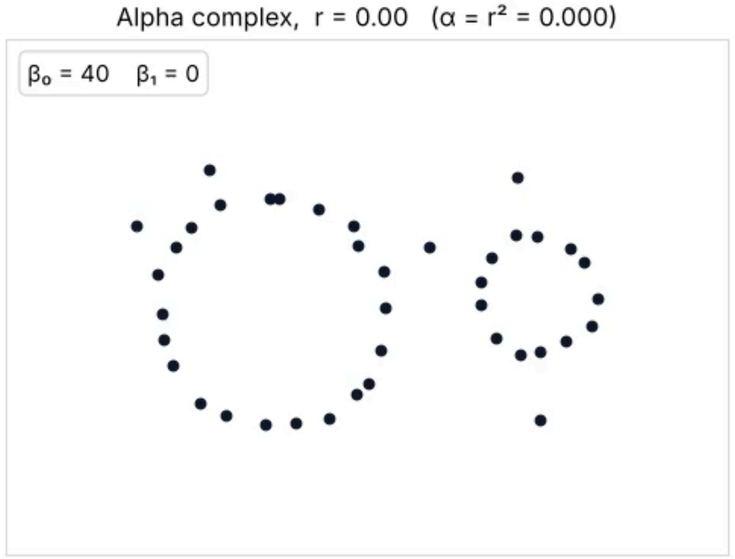
- Process simplices in decreasing dimension
- Then for pairs $(\sigma^{p-1}, \sigma^{p})$
- Result: we can skip earlier columns (since we know it will be 0's)

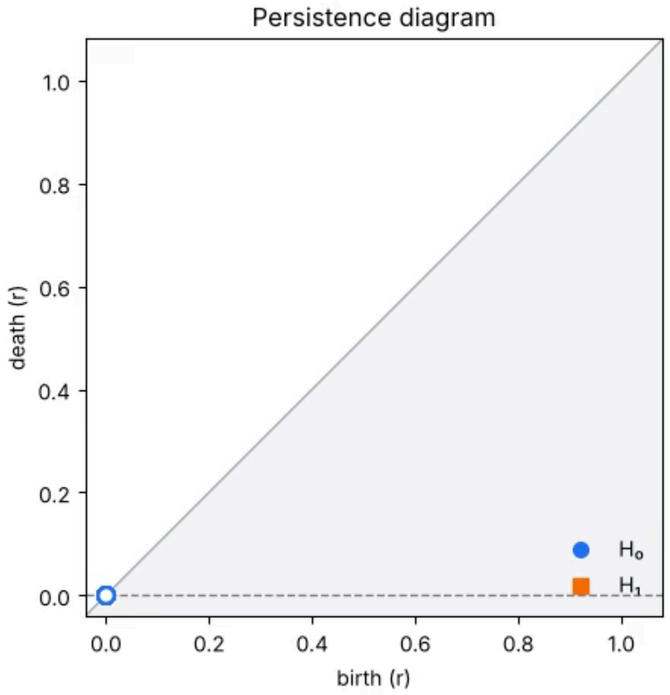
Even more speedups

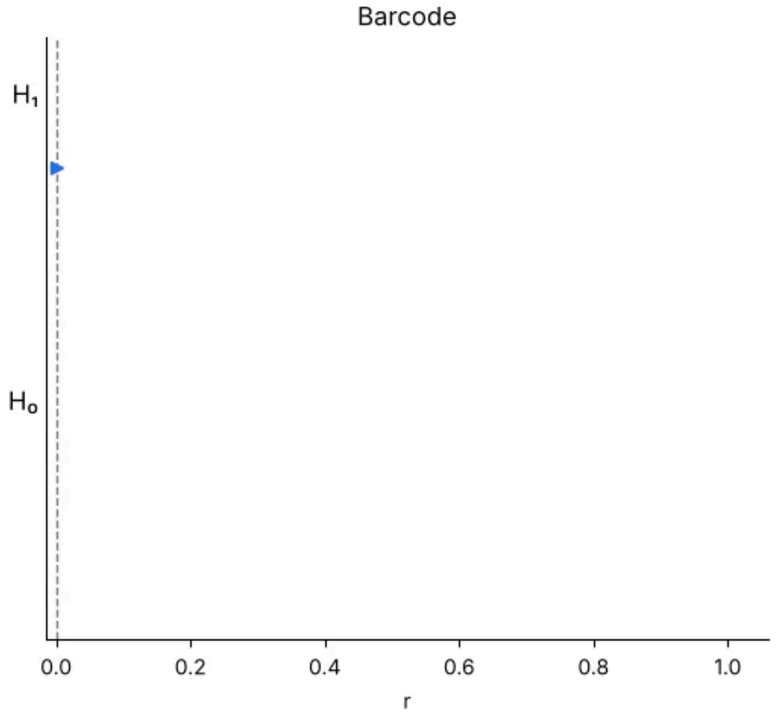
- Collapse [Boissonnet et al, 2018]
- Special case of 0-dimensional persistence (minimum spanning tree, union-find)
- Parallerization



What we have so far

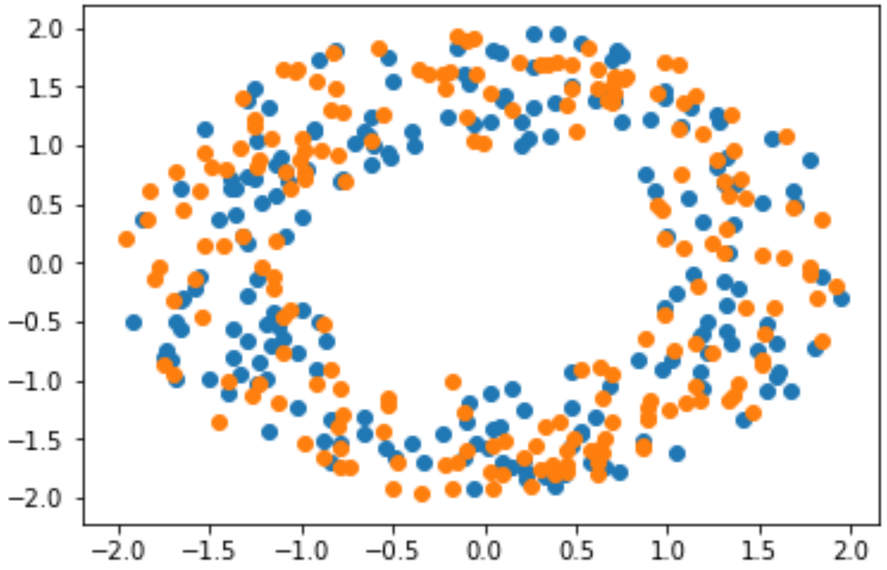


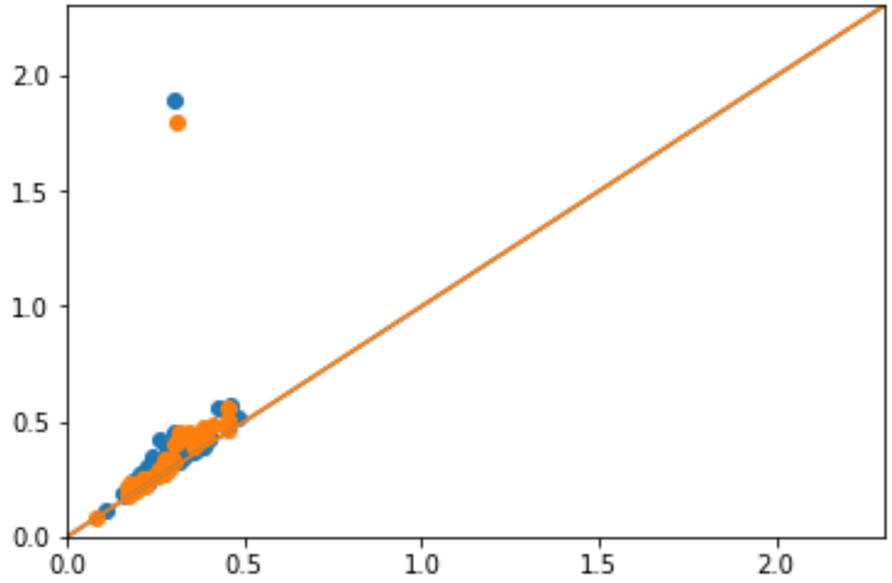




... now what?

Similar Persistence Diagrams





How different are they?

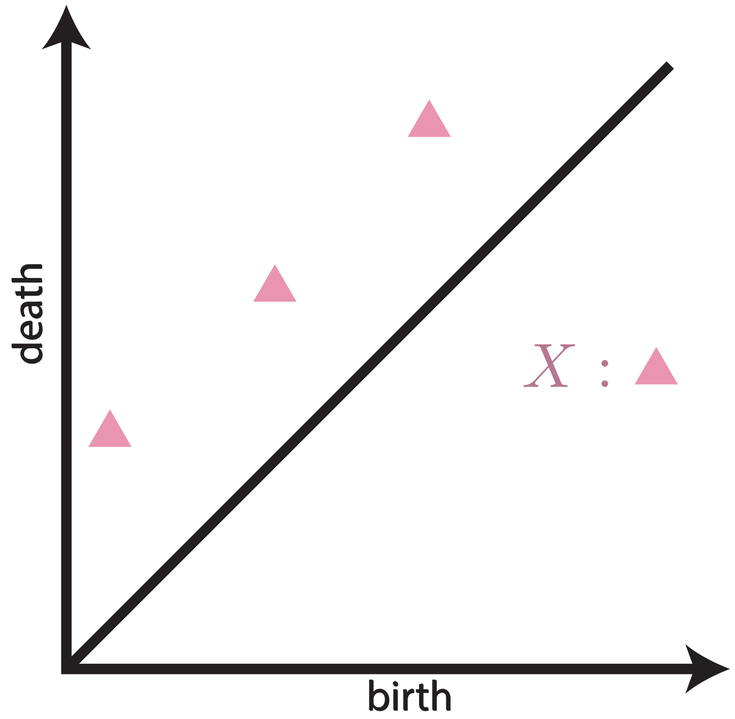
Distance Measures

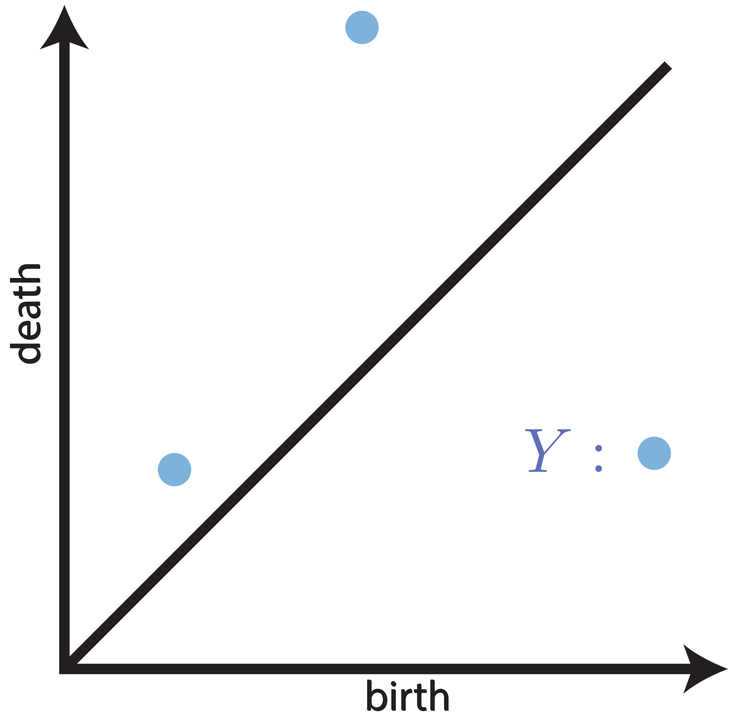
A distance on a set M is a function $d : M \to \mathbb{R}_{\geq 0}$ such that

- $d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$
- $d(x, y) = d(y, x)$
- $d(x, y = z) \leq d(x, y) + d(y, z)$

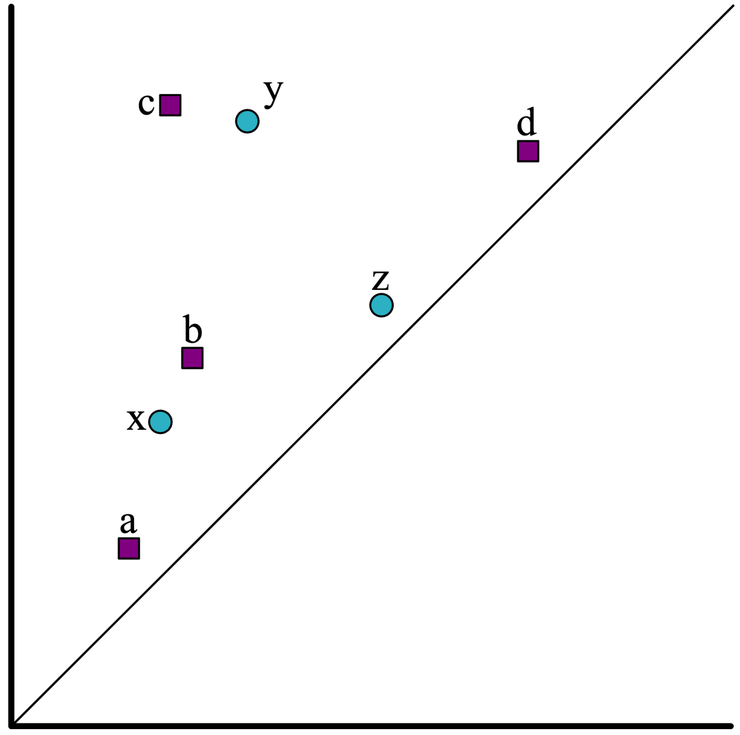
Ex. $\| (x_1, x_2) - (y_1, y_2) \|_{2'}$ $\| (x_1, x_2) - (y_1, y_2) \|_{\infty}$

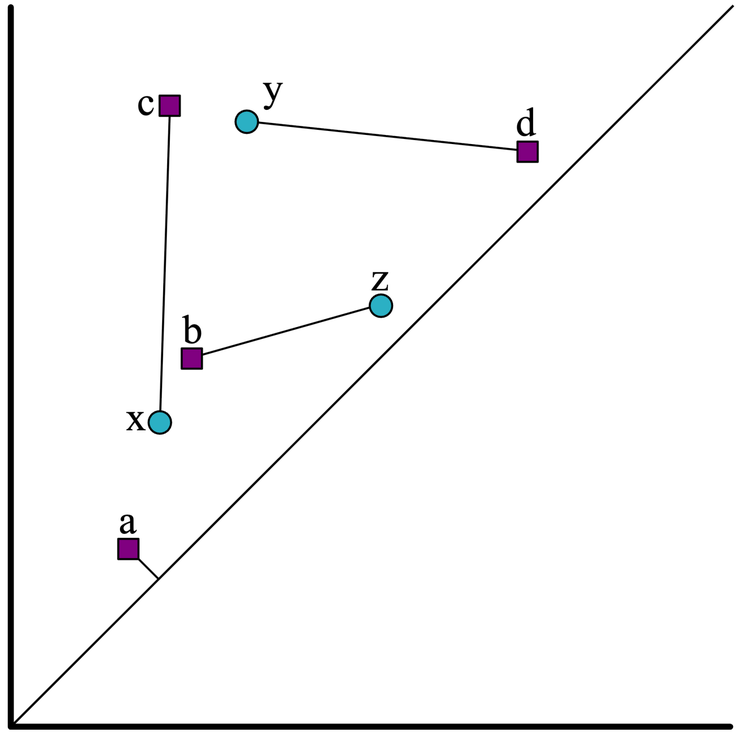
Comparing two Persistence Diagrams

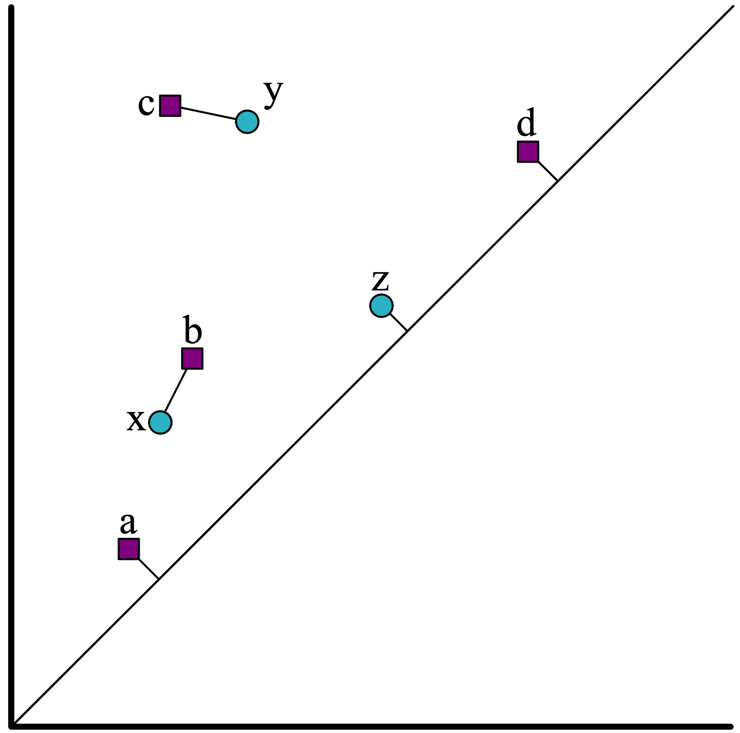




Find Matchings





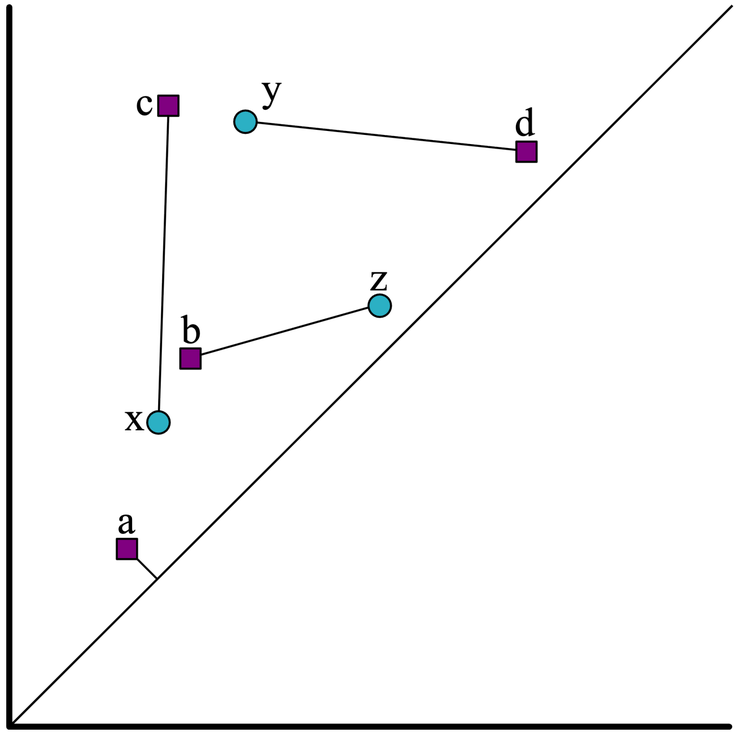


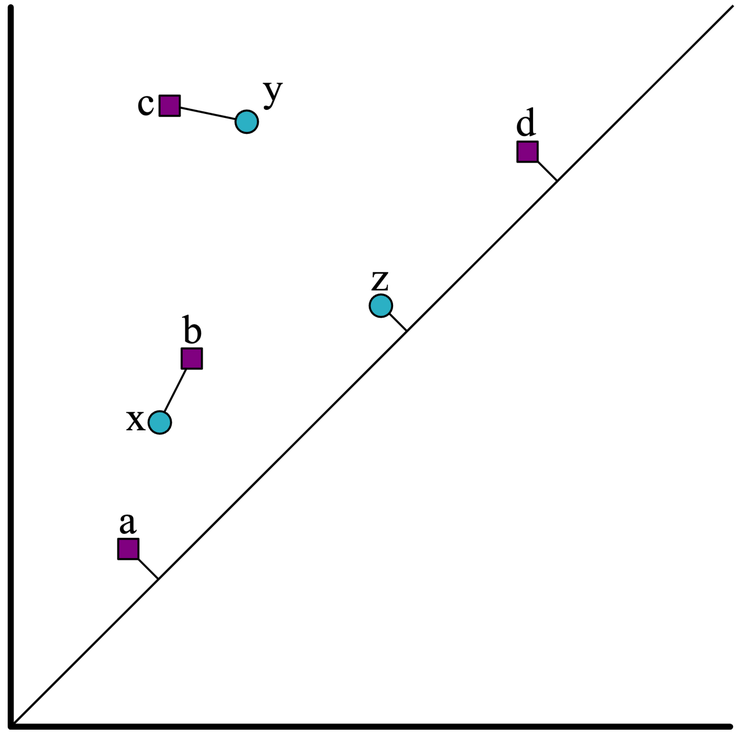
Bottleneck Distance, Version 1

- Given two persistence diagrams X, Y, the **bottleneck distance** is

$$d_B(X, Y) = \inf_{\eta: X \to Y} \sup_{x \in X} \|x - \eta(x)\|_\infty$$

- $\eta : X \to Y$ ranges over all bijections

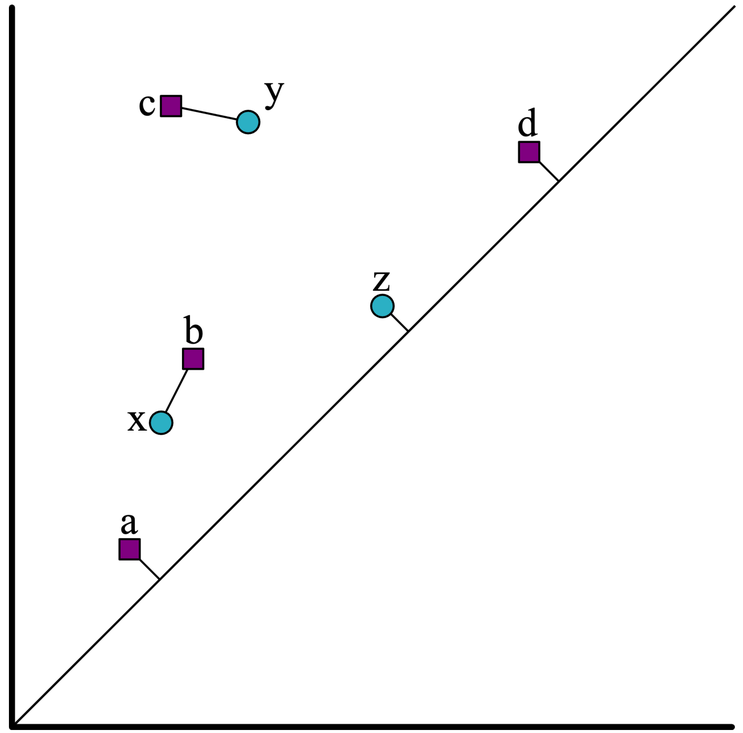




Bottleneck Distance, Version 2

Partial Matching

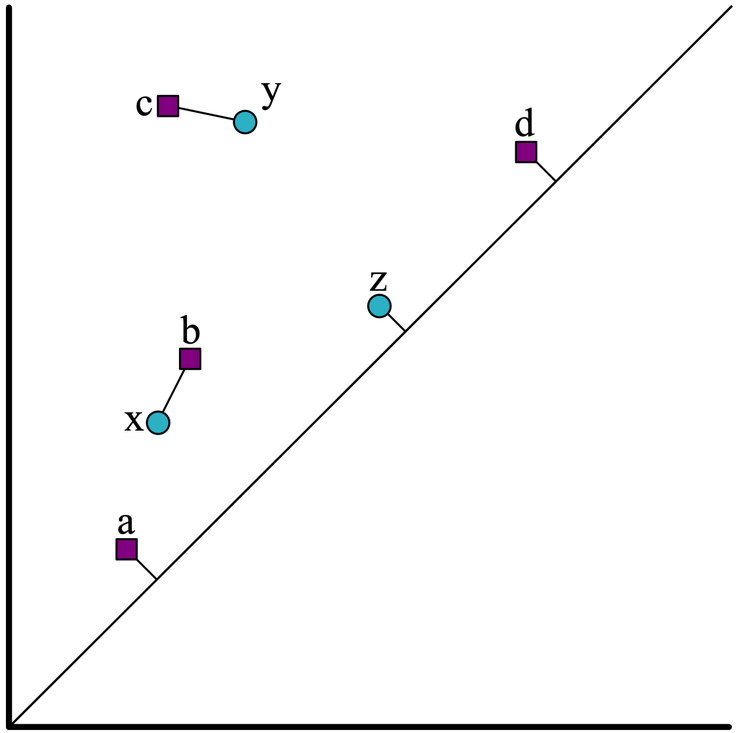
A partial matching between X and Y is a bijection on a subset of the off-diagonal points of the two diagrams $M : X' \to Y'$ for $X' \subseteq X$ and $Y' \subseteq Y$



Cost of a matching

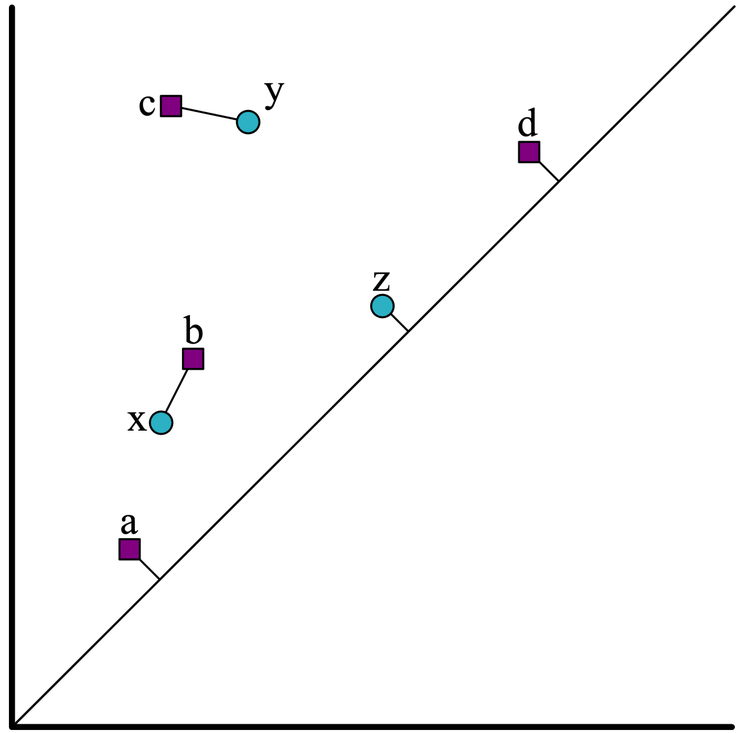
The cost of this partial matching is given by

$$c(M) = \sup\left(\{\|x - M(x)\|_\infty \mid x \in X'\} \cup \left\{\frac{1}{2}|x_1 - x_2| \mid (x_1, x_2) \in (X \setminus X') \cup (Y \setminus Y')\right\}\right)$$



Bottleneck Distance, V2

The bottleneck distance between X and Y is $d_B(X, Y) = \inf_M c(M)$, where the infinum runs over all possible partial matchings



Homework Presentation 1

Computing H_0 with union-find

- Explain the union-find data structure, including the operations MakeSet, Find, and Union.
- Apply union-find to this filtration, tracking connected components.
- Draw the persistence diagrams for H_0

$$
\begin{aligned}
K_0 &= \{A, B, C, D\}, \\
K_1 &= K_0 \cup \{AB, BC\}, \\
K_2 &= K_1 \cup \{AC, CD\}, \\
K_3 &= K_2 \cup \{ABC\}.
\end{aligned}
$$

Homework Presentation 2

Persistence Matrix Reduction

- Define a compatical ordering, then construct the boundary matrix
- Apply the persistence algorithm and identify all positive and negative simplicies
- Draw the persistence diagrams for H_0 and H_1

$$
\begin{aligned}
K_0 &= \{A, B, C, D\}, \\
K_1 &= K_0 \cup \{AB, BC\}, \\
K_2 &= K_1 \cup \{AC, CD\}, \\
K_3 &= K_2 \cup \{ABC\}.
\end{aligned}
$$