An Algorithm for Computing Persistent Homology

Roadmap:

What is Topological Data Analysis?

- Analyzing data using topology to detect underlying structure/qualitative features of the dataset
- I found there are some genuinely fruitful uses of TDA (and Persistent Homology in particular). This is especially true in cases where we are analyzing some actual spatial structure, not just datasets:
  - Predicting financial crashes
  - Materials science (e.g. subcomponents of batteries)
  - Coverage in sensor networks
  - Pre-processing for machine learning, especially vision

What is Persistent Homology?

- Persistent homology uses triangulations (made up of simplices, e.g. triangles, tetrahedrons) to find holes (loops or spheres) in a dataset
- These loops or spheres are cycles that are not boundaries
    - Note that this distinction depends on structure outside of the candidate cycle itself
- To differentiate between the structures that emerge due to noise in the dataset vs genuine structural features, we expand a triangulation over time and see what features persist
    - This is called a filtration
- We then end up tracking the intervals for various homologies - recording when they enter our triangulation and when they become cycles

What does this project do?

- In my report, I explain a basic case of persistent homology fully, from the background theory needed to understand what persistent homology is to an explanation of the algorithm used to compute it, including implementation details
- There are thorough expositions of the PH algorithm, but they have incredibly high theoretical overhead (a general problem for TDA!)
    - “Computing Persistent Homology”, Zomorodian and Carlsson 2005
- There are also self-contained expositions of PH conceptually for non-mathematicians, but they don’t include discussion of the algorithm
    - “Persistent Homology: A Pedagogical Introduction with Biological Applications”, Kemme & Agyingi 2025
- The closest is a paper in 2002 introducing the basic algorithm, but it assumes some mathematical background I don’t, and doesn’t include analysis of algorithm implementation
    - “Topological Persistence and Simplification”, Edelsbrunner, Letscher & Zomorodian 2002

Algorithm Preliminary: An Observation

- Without getting into the underlying theory, this is really just an assertion, so you’ll have to trust me
- For any k-simplex, when it enters the filtration it either 1) “kills” a k-1-cycle (makes it into a boundary) or 2) “births” a k-cycle. (And not both.)

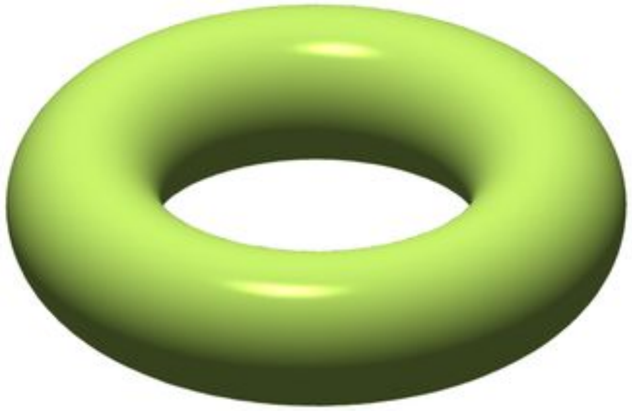
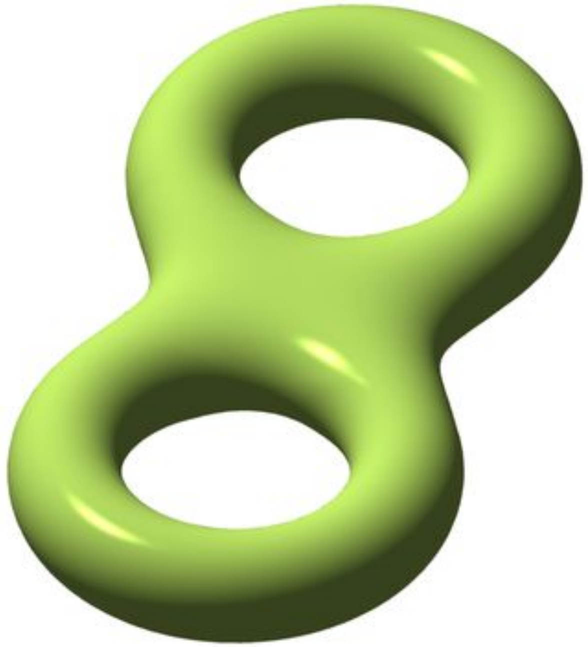
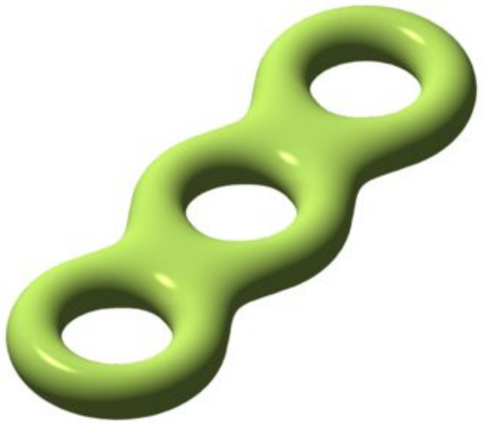
Algorithm sketch

- Maintain an array T where T[i] tells us what “kills” the cycle “birthed” by i
- Maintain a set of intervals (with information about the dimension of the intervals)
- Iterate through simplices in the order they enter the filtration, and check for any simplex j whether it is a killer or a creator
    - If it kills the cycle generated by simplex i: record it in T[i], and add an interval to our set of intervals, with the dimension corresponding the dimension of simplex i
    - If it’s a creator: mark it
- Afterwards, go through all marked simplices, and if they don’t have a killer record an interval from 0 to infinity
- Note: I implemented a simple version of the algorithm for the most common use case in applications, but the sketch here does generalize (implementing some of these steps just gets much gnarlier)

Testing

- It was fairly straightforward to test the correctness of my code
- I compared it to a TDA package Gudhi to make sure the interval sets look similar
- I generated point clouds out of shapes with simple/clear loops or spheres
  - Genus-g surfaces, or a bunch of spheres stuck together
  - I used the database EuLearn for generating genus-g surfaces

Genus-g surfaces

Images source: https://en.wikipedia.org/wiki/Genus_g_surface

Future directions

- The technical exposition of PH leans heavily on group theory, which is hard to understand if you don't already have background in it. Is there a way to rigorously present it that only uses set theory?
- Providing a proof of correctness for this algorithm that doesn't loop through even more complicated algebraic structures (rings, modules)
- Including an algorithm for computing the underlying filtration - I ended up using the Gudhi library for that part, and it would be nice to include implementation and more detailed exposition there

Questions?