# Persistent Homology on Algebraic Varieties

Mason Boeman

University of Illinois at Chicago
boeman2@uic.edu

**Abstract**

*To what extent can persistent homology, which is a way of discovering topological features of point-cloud data, be used to reconstruct the topology of known algebraic varieties? Our hypothesis is that persistent homology will be a useful tool for studying the topology of solution sets of polynomial equations, something low dimensional examples have been suggestive of. Further application of persistent homology in higher dimension, and with more intricate varieties requires faster algorithms than what are currently available.*

## I. Introduction

The goal of this project is to produce a set of tools written in python which can compute the persistent homology of an approximation of a given algebraic variety. The homology of the variety can be inferred from the persistent homology of its approximation if the approximation has high accuracy. This process has two natural steps: sample points in a uniform way on an algebraic variety, and then compute the persistent homology of the point cloud data. This paper will focus on the latter of the two steps, and explain how and why each sub step is done. The sub steps are as follows:

1. Compute the neighborhood graph for some $\epsilon$, which will bound above the size of all edges in our approximation.

2. Compute the Vietoris Rips complex from the neighborhood graph, which will give us a set of simplices and an ordering on them. These simplices will be an approximation of the surface of our algebraic variety.

3. Compute the persistent homology of the Vietoris Rips complex

4. Display the barcode graph for analysis.

## II. Neighborhood Graph

### a. Definition

In order to do any computations with our sampled points, we need to turn them into a graph. This will allow us to store the spatial relationship between points in a convenient way for future computations. A neighborhood graph $G$ with some $\epsilon$ is a weighted undirected graph whose vertices are the sampled points on the variety, and with the property that any two vertices in $G$ are connected by an edge if and only if their distance is less than or equal to $\epsilon$ of each other (with respect to their ambient space) [4]. The result is a graph in which the neighborhood of a vertex v corresponds with the points inside the closed ball centered at the point v of radius $\epsilon$.

### b. Computation

We will store the neighborhood graph as a list of adjacencies. Our method of computing the adjacencies is recursive, and hinges on the following observation: if two points are within epsilon, then each of their corresponding coordinates are within epsilon of each other. Also, if two points in $\mathbb{R}^n$ lie on either side of a hyperplane of dimension n-1, then there is an epsilon edge connecting them only if they are both within epsilon of the hyperplane. Note that this condition is necessary but not suf-

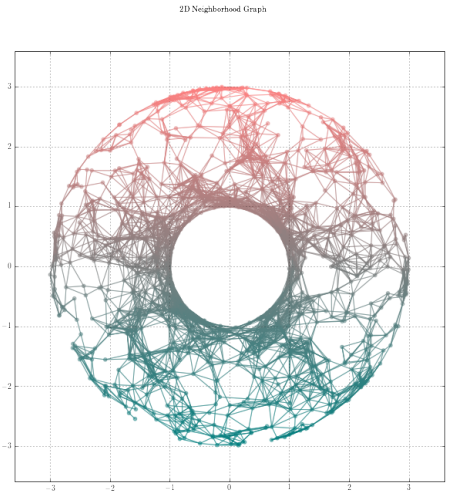
ficient for them to be within epsilon of each other. Therefore, the algorithm is as follows:

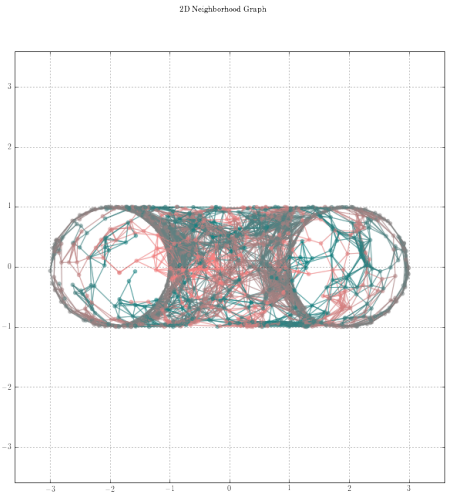
**Function** `NeighborhoodGraph`$(S, i, \epsilon)$
  **Data**: a set of vertices and an index of the coordinate to consider
  **Result**: report all edges shorter than $\epsilon$
  $m = median(\{s_i | s \in S\})$;
  $X = \{x \in S | x_i < m\}$;
  $Y = \{y \in S | y_i >= m\}$;
  $X_\epsilon = \{x \in X | x_i > m - \epsilon\}$;
  $Y_\epsilon = \{y \in Y | y_i < m + \epsilon\}$;
  **for** $x \in X_\epsilon$ **do**
    **for** $y \in Y_\epsilon$ **do**
      **if** $dist(x, y) < \epsilon$ **then**
        report edge $(x, y)$;
      **end**
    **end**
  **end**
  `NeighborhoodGraph`$(X, i, \epsilon)$;
  `NeighborhoodGraph`$(Y, i, \epsilon)$;
**end**

**Algorithm 1:** Neighborhood Graph Computation

### c. Result

The result of the algorithm running on 1000 points sampled on a torus is the following graph (shown from two angles)




## III. Weighted Simplicial Complexes

### a. Definition

First, we'll introduce a few definitions. A $k$-simplex is a $k$ dimensional polytope which is the convex hull of $k+1$ points. These points are called the vertices of the $k$-simplex. A $(k-1)$-simplex $\sigma_{k-1}$ is the face of a $k$-simplex $\sigma_k$ if the vertices of $\sigma_k - 1$ are a subset of the vertices of $\sigma_k$. A sub-simplex is the same as a face, but can be any degree less than $k$ (instead of requiring degree $k-1$). A simplicial complex $C$ is a set of simplices with the following properties:

- if $\sigma_1$ is in $C$, and $\sigma_2$ is a face of $\sigma_1$, then $\sigma_2$ is in $C$.
- if $\sigma_1$ and $\sigma_2$ are in $C$, then $\sigma_1 \cap \sigma_2$ is a sub-simplex of both $\sigma_1$ and $\sigma_2$ (possibly the empty simplex).

A weighted simplicial complex is a simplicial complex with the added property that each simplex is assigned a real-valued weight. In particular we will be interested in the Vietoris-Rips complex, which is a weighted simplicial complex where the weight of a simplex is the largest distance between any two vertices of that simplex (the weight of zero-simplices is zero) [4].

### b. Computation

We store the Vietoris-Rips complex as a list of weighted simplices. To compute the list we use the Bron-Kerbosch algorithm to compute the maximal cliques of the neighborhood graph. Then, for every maximal clique, report every face of the simplex whose vertices are the points in the clique. The algorithm can be modified to only return simplices with dimension less than $k$ for some $k$, which improves running time from $O(2^n)$ to $O(n^k)$, which is significantly faster for smaller choices of $\epsilon$ in the neighborhood graph construction.

### c. Compatible Total Ordering

Later, in the persistent homology computation we will need a compatible total ordering on simplices. This is an order on simplicies with the property that every simplex is larger than all of its sub simplicies [1]. The order we will use is the following: if $weight(\sigma_1) \neq weight(\sigma_2)$ then we sort by weight. If $weight(\sigma_1) = weight(\sigma_2)$ then sort by the degree of the simplices. If this is also a tie, we need to break the tie, but how we do so does not matter. In our implementation, we compare the hashes of the simplex objects, but any method is fine, as long as it always returns the same answer.

Note that this is indeed a compatible total ordering of a Vietoris Rips complex, because for any face $\sigma_1$ of a simplex $\sigma_2$, the vertices of $\sigma_1$ are a subset of the vertices of $\sigma_2$, so the maximum distance between any two vertices in $\sigma_1$ is less than or equal to the maximum distance between any two vertices in $\sigma_2$. If they are equal, then the degree of a face is always less than the simplex it is a face of, so $\sigma_1$ is less than $\sigma_2$.

## IV. Persistent Homology

### a. Simplicial Homology

Intuitively, the simplicial homology is the number of k-dimensional holes in a given simplicial complex. Specifically, we will be using the construction (which uses $\mathbb{Z}/2\mathbb{Z}$ coefficients) which is detailed by Edelsbrunner [1].

### b. Persistent Homology

Persistent homology is a tool for inferring the continuous from the discrete [2]. In our particular case, "the continuous" refers to the homology of our original algebraic variety, and "the discrete" refers to our input points, which were sampled on the variety. Let $C$ be a Vietoris-Rips complex with a compatible total ordering. Let $C_\sigma = \{\tau \in C \mid \tau < \sigma\}$. $C_\sigma$ is still a Vietoris-Rips complex because the definition of Vietoris-Rips complex is implied by the definition of "$<$" on simplices, so we can compute the simplicial homology of each $C_\sigma$ and compare them to each other.

The introduction of $\sigma$ to the complex can do two things to the simplicial homology: add a single element of homology, or remove a single element. For example, adding an edge between two points can either reduce the number of connected components by one or add a loop to the homology, but not both. Therefore each simplex can be labeled "positive" or "negative," depending on whether it introduced new homology or eliminated existing homology [1,3]. These positive and negative elements can be paired, so that each pair create and destroy the same element of homology, and this pairing is called the persistence pairing.

### c. Persistence Pairing

It is possible to map positive/negative pairs of simplices to homology classes of cycles, and Zomorodian gives an algorithm to do this [3]. The reason you would want such a map is because the difference in the weights between the positive and negative simplices in a pair indicates how long the corresponding topological attribute persists. A longer lifespan indicates that the attribute is more significant [1,3].

### d. Persistence Algorithm

The following is the algorithm given by Zomorodian in [3].

Our data will be stored in a sparse $n \times n$ matrix, where $n$ is the number of simplices in our Vietoris-Rips complex, and whose entries are in $\mathbb{Z}/2\mathbb{Z}$. The rows and columns are both labeled by the simplices, in increasing order along both the rows and columns. We will call this matrix $M$, and $M[i, j]$ is the element in the $i^{th}$ column and the $j^{th}$ row. We initialize the matrix with all zeros, then for each entry $M[a, b]$ in $M$, set $M[a, b] = 1$ if and only if $\sigma_b$ is a face of $\sigma_a$.

Let $low(\sigma)$ be the function which gives the simplex corresponding to the row containing the lowest nonzero entry in the column corresponding to $\sigma$. $low(\sigma) = 0$ if and only if there are no nonzero entries in the column indexed by $\sigma$.

Now the matrix is reduced according to the following algorithm due to Edelsbrunner [1]:

**Data**: $M$ a matrix
**Result**: $M$ is now reduced
**for** $j = 1$ *to* $n$ **do**
  **while** $\exists \sigma_k < \sigma_j \mid low(\sigma_k) = low(\sigma_j)$ **do**
    add column $\sigma_k$ to $\sigma_j$;
  **end**
**end**

**Algorithm 2:** Persistence

An example of the matrix $M$ before and after this algorithm is run can be found in figure 1.

After running the algorithm, a negative simplex $\sigma^-$ and a positive simplex $\sigma^+$ are paired if and only if $\sigma^+ = low(\sigma^-)$. Additionally, a positive simplex can be paired with no negative simplex, implying that that element of homology persists at least as far as our current choice of $\epsilon$ in the neighborhood graph construction.

## V. Barcode Graph

The barcode graph is the most straightforward method of displaying the information computed by the persistence algorithm. The barcode graph is essentially a graph with the $x$ axis as $\epsilon$ from the neighborhood graph, and the $y$ axis as homology. To produce this, we draw horizontal lines which start and end at x coordinates equal to the weights of the paired positive and negative simplices. As long as the lines are staggered in the y direction so they can be distinguished, the y coordinates of the segments do not matter. We sort by dimension, and color the segments according to the dimension of homology the segment represents. Figures 2 and 3 are examples of what this program outputs.

## VI. Conclusion

Unfortunately, these algorithms run slowly on large sample sizes, which are very important for accuracy of the topological data that can be computed from points sampled on an algebraic variety. The way to get around this speed limit is to reduce the $\epsilon$ from neighborhood graph so our initial graph is much less dense. Figures 2 and 3 show the difference between the complete graph and a much more modest $\epsilon$ cutoff. Both pictures are suggestive of a single cycle, a single connected component, and no other homology, so you can still draw conclusions from larger sample sizes.

## References

[1] Herbert Edelsbrunner and John Harer. Persistent homology—a survey. In *Surveys on discrete and computational geometry*, volume 453 of *Contemp. Math.*, pages 257–282. Amer. Math. Soc., Providence, RI, 2008.

[2] Shmuel Weinberger. What is...persistent homology? *Notices Amer. Math. Soc.*, 58(1):36–39, 2011.

[3] Afra J. Zomorodian. *Topology for computing*, volume 16 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2009. Reprint of the 2005 original [MR2111929].

[4] Afra J. Zomorodian. Fast construction of the vietoris-rips complex. *Computers & Graphics*, 34(3):263 – 271, 2010.

$$\begin{bmatrix}
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

**Figure 1:** *The matrix in the persistence algorithm, before (top) and after (bottom) reduction. The input is the complete graph on four vertices.*

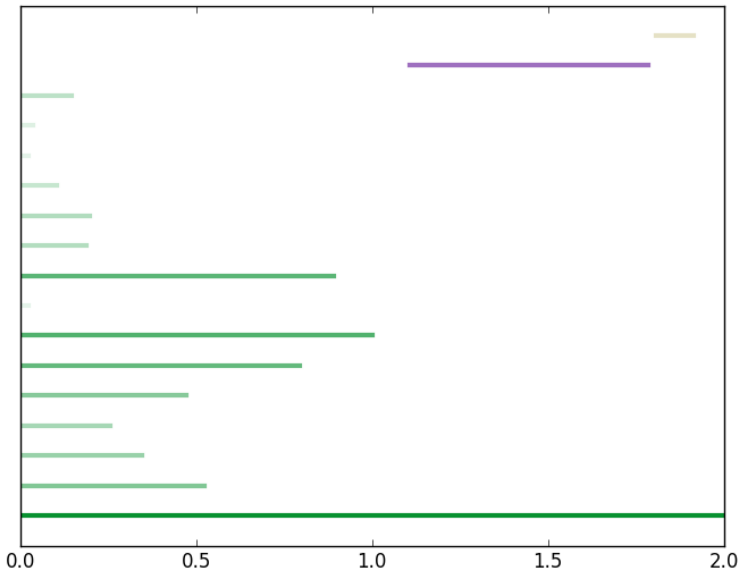



**Figure 2:** *15 points randomly chosen on the unit circle. neighborhood graph has an $\epsilon$ of* 2*. Total runtime was 11.7 seconds.*

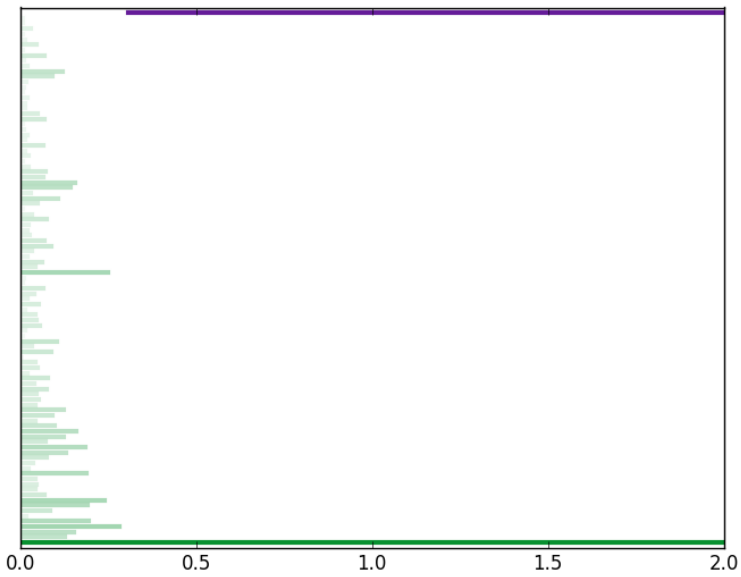



**Figure 3:** *100 points randomly chosen on the unit circle. neighborhood graph has an $\epsilon$ of* .5*. Total runtime was 2 minutes 33 seconds.*