# Topological Data Analysis on proteins structures

Edoardo Carroccetto, Andrea Monticone

**Contents**

## 1. Introduction

Proteins are complex biomolecules that play crucial roles in nearly all biological processes. Proteins are made up of smaller units called amino acids, which are linked together in long chains. The sequence of amino acids determines each protein's unique 3-dimensional structure and understanding it is vital for insights into their functions and interactions.

Proteins can have various levels of structure: primary, secondary, tertiary, and quaternary. The primary is the sequence of amino acids. Secondary structures are formed by hydrogen bonds between amino acids, creating alpha-helices and beta-sheets. Tertiary structure is the overall 3D shape of the protein, and quaternary structure arises when multiple protein chains (subunits) assemble into a larger molecule. In this project, we are going to use Topological Data Analysis algorithms as the mapper algorithm and Persistent Homology, firstly to represent the protein as a graph and eventually to underline some topological features as the presence of hole and concavity.

## 2. Simplicial Complexes

In order to give the definitions of Simplices and Simplicial Complexes we have to previously define some required objects. Let $u_0, u_1, ..., u_k$ be points in $\mathbb{R}^d$. A point $x = \sum_{i=0}^{k} \lambda_i u_i$ is an *affine combination* of the $u_i$ if the $\lambda_i$ sum to 1. The *affine hull* is the set of affine combinations.
The $k+1$ points are *affinely independent* iff the $k$ vectors $u_i - u_0$, for $1 \leq i \leq k$, are linearly independent. (In $\mathbb{R}^d$ we can have at most $d$ linearly independent vectors and therefore at most $d+1$ affinely independent points).
An affine combination, $x = \sum_{i=0}^{k} \lambda_i u_i$, is a *convex combination* if all $\lambda_i$ are non-negative. The *convex hull* is the set of convex combinations.
Now we can give the definition of k-simplex.

**Definition 2.1 (k-simplex).** *A k-simplex is the convex hull of $k+1$ affinely independent points, $\sigma = conv\{u_0, u_1, ..., u_k\}$*

We use special names for the first few dimensions, *vertex* for 0-simplex, *edge* for 1-simplex, *triangle* for 2-simplex, and *tetrahedron* for 3-simplex.
A *face* of $\sigma$ is the convex hull of a non-empty subset of the $u_i$ and it is *proper* if the subset is not the entire set. We sometimes write $\tau \leq \sigma$ if $\tau$ is a face and $\tau < \sigma$ if it is a proper face of $\sigma$. If $\tau$ is a (proper) face of $\sigma$ we call $\sigma$ a (proper) coface of $\tau$.
We are interested in sets of Simplices that are closed under taking faces and that have no improper intersections.

**Definition 2.2 (Simplicial Complex).** *A Simplicial Complex is a finite collection of simplices $K$ such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$, and $\sigma, \sigma_0 \in K$ implies $\sigma \cap \sigma_0$ is either empty or a face of both.*

The dimension of $K$ is the maximum dimension of any of its simplices. The *underlying space*, denoted as $|K|$, is the union

of its simplices together with the topology inherited from the ambient Euclidean space in which the simplices live.
It is often easier to construct a complex abstractly and to worry about how to put into Euclidean space later, if at all.

**Definition 2.3 (Abstract Simplicial Complex).** *An abstract simplicial complex is a pair $(V, \Sigma)$ where $V$ is a finite set and $\Sigma$ is a family of non-empty subset of $V$. The following properties must hold:*

$$\sigma \in \Sigma \quad \tau \subseteq \sigma \implies \tau \in \Sigma$$

The sets in $\Sigma$ are its simplices. Given a (geometric) simplicial complex $K$, we can construct an abstract simplicial complex $A$ by throwing away all simplices and retaining only their sets of vertices. We call $A$ a *vertex scheme* of $K$. Symmetrically, we call $K$ a *geometric realization* of $A$.
Constructing geometric realizations is surprisingly easy if the dimension of the ambient space is sufficiently high.

**Theorem 2.1 (Geometric Realization Theorem).** *Every abstract simplicial complex of dimensions $d$ has a geometric realization in $\mathbb{R}^{2d+1}$*

Eventually, we can give the definition of *Simplicial maps*. The natural equivalent of continuos maps between topological spaces are simplicial maps between simplicial complexes.
Every point $x \in |K|$ belong to the interior of exactly one simplex in $K$. Letting $\sigma = conv\{u_0, u_1, ..., u_k\}$ be this simplex, we have $x = \sum_{i=0}^{k} \lambda_i u_i$ with $\sum_{i=0}^{k} \lambda_i = 1$ and $\lambda_i > 0 \ \forall i$. Setting $b_i(x) = \lambda_i$ for $0 \leq i \leq k$ and $b_i(x) = 0$ for $k+1 \leq i \leq n$ we have $x = \sum_{i=0}^{n} b_i(x) u_i$ and we call $b_i(x)$ the *barycentric coordinates* of $x \in K$.
We use these coordinates to construct a piecewise linear continuous map from a particular kind of map between the vertices of two simplicial complexes.
A *vertex map* is a function $\phi : vert(K) \longrightarrow vert(L)$ with the property that the vertices of every simplex in $K$ map to vertices of a simplex in $L$. Then we can extend this concept:

**Definition 2.4 (Simplicial map).** *$\phi$ can be extended to a continuous map $f : |K| \longrightarrow |L|$ defined by:*

$$f(x) = \sum_{i=1}^{n} b_i(x) \phi(u_i)$$

*the simplicial map induced by $\phi$.*

### *2.1. Construction of Simplicial Complexes*

Simplicial Complexes often arise as intersection patterns of collection of sets. There is a standard way of construction simplicial complexes starting from some spaces: the Nerve of a covering. Let now $T$ a topological space and let $\mathcal{U} = \{U_a\}_{a \in A}$ a cover of $T$.

**Definition 2.5 (Nerve of a Covering).** *The nerve of $\mathcal{U}$, denoted* Nrv $F$*, is the abstract simplicial complex with vertex A, and $\{a_0, \ldots, a_n\} \in \Sigma$ iff $U_{a_0} \cap \cdots \cap U_{a_n} \neq \emptyset$.*

Intuitively we are creating a simplicial complex with the same shape of the covering. To visualize it take the classical any object embedded in a classical euclidean space and take as element of the cover classical balls. Suppose that the set $A$ coincide with the center of the balls. Then consider the union of all the balls and try to stretch it: the number of holes and cavities remain the same; thus we get the following theorem:

**Theorem 2.2 (Nerve theorem).** *Let $F$ be a finite collection of closed convex sets in euclidean space. Then the nerve of $F$ and the union of the sets in $F$ have the same homotopy type*

#### *2.1.1. Čech Complex*

Consider the special case in which the cover is made of geometric balls, all of the same radius, $r$.

**Definition 2.6 (Čech Complex).** *Let $S$ be a finite set of points in $\mathbb{R}^d$ and write $B_x(r) = x + r\mathcal{B}^d$ for the closed ball with center $x$ and radius $r$.*
*The Čech complex of $S$ and $r$ is isomorphic to the nerve of this collection of balls,*

$$\text{Čech}(r) = \{\sigma \subseteq S \mid \cap_{x \in \sigma} B_x(r) \neq 0\}$$

#### *2.1.2. Vietoris-Rips Complex*

Instead of checking all subcollections, we may just check pairs and add 2 and higher-dimensional Simplices whenever we can. This simplification leads to the *Vietoris-Rips complex* of $S$ and $r$ consisting of all subsets of diameter at most $2r$,

$$\text{Vietoris-Rips} = \{\sigma \subseteq S \mid \text{diam}(\sigma) \leq 2r\}$$

Clearly, the edge in the Vietoris-Rips Complex are the same as in the Čech Complex. Furthermore, Čech($r$) $\subseteq$ Vietoris-Rips($r$) because the latter contains every simplex warranted by the given edges.

**Lemma 2.1 (Vietoris-Rips Lemma).** *Letting $S$ be a finite set of points in some Euclidean space $r \geq 0$, we have:*

$$\text{Vietoris-Rips}(r) \subseteq \text{Čech}\left(\sqrt{2}r\right)$$

## 3. The Mapper Algorithm

The Mapper algorithm is a combination of dimensionality reduction, clustering and graph networks techniques used to get higher level understanding of the structure of data. It is mainly used for visualizing the shape of the data through a particular lens, detecting clusters and interesting topological structures which traditional methods fail to find and to select features that best discriminate data and for model interpretability.
Given a dataset of points, the basic steps behind Mapper are as follows:

- Map to a lower-dimensional space using a *filter function* $f$, or *lens*.
- Construct a cover $(U_i)_{i \in I}$ of the projected space typically in the form of a set of overlapping intervals which have constant length.

- For each interval $U_i$ cluster the points in the *preimage* $f^{-1}(U_i)$ into sets $C_{i,1}, ..., C_{i,k_i}$.
- Construct the graph whose vertices are the cluster sets and an edge exists between two vertices if two clusters share some points in common.

For the implementation, in python, of the mapper algorithm we create a subclass, named: "Mapper", of the class simplex with the following methods that we construct following the basics steps listed above.
For the first step, the choice of the lens is of great importance since the output graph will sensibly depend on that. Typical choices are standard dimensionality reduction algorithms like PCA, Isomap, MDS, t-SNE or UMAP, meanwhile density-based methods such as distance to the first k-nearest neighbors are often used in biological applications.
In this method we ask in input the data and a function, and the output will be a one dimensional reduction following the inserted function. Here we use the function eval() that allows us

**Algorithm 1** Function reduction

```
Require: function $f$
  for All points $x$ do
      Save in a list eval($f$, $x$)
  end for
```

to change the function in the input without any change in the code. We suppose that we a have a matrix with a set of coordinates. In this way for each point we are able to compute the function and save its value in a list. The reason why we save it in a list is because in this way, later, we will be able to perform the counter-image without any problem. For the second step, the construction of a cover, a standard choice is a collection of d-dimensional intervals of the same length pairwise overlapping with a specified percentage. Hence to construct the cover we take the maximum and the minimum. If the $r$ is the radius of the balls and $g$ is the overlap factor then the distance between the centers will be $d = 2r(1 - g)$. We set the first center in $m + d/2$ where $m$ is the minimum and the we compute the index of the balls in which the other points are starting from this. Notice that a point is in at least one or two covers, to take care of it we create a matrix in the first. If the point belong to a single ball then we rewrite the index twice, otherwise we write both the indexes. We resume this in this pseudocode: The advantages of this code is that is linear. We do not have to use a binary search or similar. For the third step, we have to choose the clustering technique. DBSCAN or hierarchical clustering are good picks in this context. We choose **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) that is a density-based unsupervised learning algorithm. It computes nearest neighbor graphs to find arbitrary-shaped clusters and outliers. Whereas the K-means clustering generates spherical-shaped clusters. We choose this cluster algorithm because does not require $k$ clusters initially. Instead, it requires two parameters: the radius of specific neighborhoods (if the distance between two points is less than the radius, it will be considered neighbors) and the minimum number of data points in a given neighborhood to form

**Algorithm 2** Covering

```
for Values of the points $y$ do
    $D = y - (m + d/2)$              ▷ distance from the first center
    $a = floor(D/d)$                 ▷ #centers between y and the first
    $b = (D - ad)/d$                 ▷ between two adjoin centers
    if $b < 0.5 - g/2$ then
        the point is in the cover $a1$
    else if $b > 0.5 + g/2$ then
        the point is in the cover $a + 1$
    else
        the point is in the cover $a$ and $a + 1$
    end if
end for
```

the clusters. Once we have done the clustering in each cover we proceed in create the graph. To do this we use the simplex class (explained later). Here we take as point of the simplex the centers of the clusters. In this way it will be more simple to represent it graphically. We follow the following pseudo-code

**Algorithm 3** Graph construction

```
for $c_1$ cluster do
    for $c_2$ cluster do
        if $c_1 \neq c_2$ then
            if $c_1 \cap c_2 \neq \emptyset$ then
                create and edge between $c_1$ and $c_2$
            end if
        end if
    end for
end for
```

## 3.1. Implementation

| **Mapper** | | |
|---|---|---|
| **Attributes** | | |
| simplex | ... | |
| **Method** | | |
| general_reduction | $O(nf)$ | Apply a one-dimensional dimensionality reduction. |
| Ball_covers | $O(n)$ | Cover the given one-dimensional space with balls. |
| split_cover | $O(n)$ | Split the data based on the cover information. |
| dbscan_sublists | $O(km^2)$ | Apply DBSCAN to each sub list in the given list. |
| create_simplex | $O(nkc)$ | Create the simplex given the results of the previous functions. |

Here $n$ is the number of points, $f$ is the complexity of the function $f$ we add during the one dimensional reduction. $k$ is the number of cover, $m$ is the number of element in a cover (the maximum) and $c$ is the number of clusters.

## 4. Persistent Homology

Let's recall the framework. We have a set of data $X$ that can be seen as a sample from the population $\mathbb{X}$. The main purpose of TDA is to infer the topological structure of $\mathbb{X}$ from our observation $X$. One can think, at first, to use tools of topology directly on the raw data $X$ but this is not very useful. The issue is that the data does not have a continuous structure: they have a discrete one that, from a topological point of view, is free of any useful significance. This problem can be solved with the use of simplicial complexes: the idea is to "approximate" the topological structure of $\mathbb{X}$ with a simplicial complex constructed starting from $X$. The way to construct a simplicial complex have been described before but now that we have the intuition some question can arise:

- How can I deduce topological features from the simplicial complex?
- If I infer some properties from the simplicial complex then do they remain valid for $\mathbb{X}$?

The following definitions will be very useful in the following sections:

**Definition 4.1 (Homotopy).** *Let $X, Y$ be two topological spaces. An homotopy between two maps $f, g : X \to Y$ is a map $H : X \times [0,1] \to Y$ (here we used the product topology) such that $H(x,0) = f(x)$ and $H(x,1) = g(x)$. In this case we use the notation $f \simeq g$*

Intuitively we are saying that two maps are homotopic if there is a way to pass from $f$ to $g$ using continuous sample path: that is I want to be able to connect $f(x)$ to $g(x)$ using a path $\alpha$ such that $\alpha(0) = f(x)$ and $\alpha(1) = g(x)$; all this paths have to be nearby. Now we can extend this definition to two topological spaces:

**Definition 4.2.** *Let $X, Y$ be two topological spaces. Then they are said to be homotopic if there are functions $f : X \to Y$ and $g : Y \to X$ such that $f \circ g \simeq \mathrm{id}_Y$ and $g \circ f \simeq \mathrm{id}_X$.*

The homology is the study of the properties that are invariant respect to homotopic equivalence. These properties can be used also to classify topological spaces; but you have to remember that the notion of topological homotopy is weaker then the one of topological equivalence: so this classification is "roughly". Later, you will meet very often the term homological structure: when it happens you should remember that we are referring only on characteristics of topological spaces that are, in real life world, interpreted as components, holes, cavites... In simple terms, the homological structure of a topological spaces aims to describes some features using only the concepts itemized before. If, in addiction, we observe that the homological structure is described using algebraic structures and that there are "simple" theorems for classification this is even better. In a general framework the main line is:

1. I compute the homological structure, that is a family of algebraic structures (groups in general);
2. I use the theorem of classification of algebra to classify the topological spaces.

This is the same principle used in the theory of categories.

### *4.1. The homology of a simplicial complex*

On the base of what said before we want to find a procedure for classify simplexes using the homological properties. Suppose we are given $K = (V, \Sigma)$ a simplicial complex. Here we mean the combinatorial sense since is easier to work with, but remember that you can always construct a geometrical complex $|K|$ with abstract complex $K$ and so you can pass from one to the other every time you want. We can define the following subset:

$$\Sigma_k = \{\sigma \in \Sigma \mid \#\sigma = k+1\}$$

Thus, $\Sigma_k$ is the collection of all subset in $\Sigma$ that can be associated with a $k$-dimenisional geometrical object (remember that $\{a, b\}$, for example, is the line between $a$ and $b$, so a 1-dimensional object even if its cardinality is 2). If $\sigma \in \Sigma_k$ the $\sigma$ is called a $k$-simplex. This is a starting point for a richer structure that will help with our aim: counting the number of $k$-dimensional holes (intuitively: the 1-dimensional holes are the classical holes, 2-dimensional holes are the cavites,...). Take an element on $\Sigma_k$, this is a $k$-simplex, but $k$-simplexes never contain a $k$-dimensional hole, but they can be used to create the contour of a $k$-dimensional hole (in fact someone can say that $k$-dimensional holes are holes with a $k$-dimensional contour). So the next step is find a structure that contains also the "contour". A way to do the extensions is considering the problem with the respect to a formal point of view. If $A$ and $B$ are simplices then I can imagine $A + B$ as their concatenation. This construction is well known in algebra and it is called the free abelian group. But unfortunately, thought it is very intuitive, this is not the best way: instead we take the free vector space on the field $\mathbb{Z}_2$. We do this for three reason:

- It is more simple, since there are less elements
- Does not require the introduction of a order between the elements
- It brings some computational advantage since then we are able to use the tools of linear algebra (for example we can use the matrix to study the linar operators)

This choice and these points will be clear later. So we can define:

$$C_k = \mathrm{Free}_{\mathbb{Z}_2} \Sigma_k$$

So this is the set of all formal linear combination of the elements of $\Sigma_k$ using as coefficients elements of $\mathbf{Z}_2$: then, by definition, $\Sigma_k$ is a basis of this vector space (or the generator if you want to see this from the abelian group form). This will be useful later on. This is the group of the $k$-chains. Now that we have add all the linear combination we want to be able to make a distinction between the contour and the combination that are not linked to holes. We introduce so an important operator that takes an $k$-chains and then returns a $k-1$-chains. To define it, we impose its value if we take a vector from the (canonical) base:

$$\partial_k : \Sigma_k \to C_{k-1} \qquad \{c_1, \dots, c_n\} \to \sum_{i=1}^{n} \{c_1, \dots, \tilde{c}_i, \dots, c_n\}$$

The theory ensure that if a function is defined on a base then it can be extended linearly on the whole vector space, we denote this function with the same symbol. In the formula above $\tilde{c}_i$ means that in the set we are not including the element $c_j$. This operation is called boundary operator and the reason is quite intuitive: if we imagine to take an element in $\Sigma_k$ this function return the concatenations of all its $k-1$-simplices, that is, the contour. So if we take $c \in C_k$ it return the concatenation of contours, that is the contour of all the $k$-chain. With this operator we are able to find if a $k$-chain $\sigma$ is the contour of a hole: it is enough to compute $\partial_k\sigma$ and check if it is equal to zero. This is very good since now we can identify mathematically all holes, it is enough to take $\operatorname{Ker}\partial_k$ and we have found all contour of holes, doesn't it? Wrong, another step is required and it is based on this theorem that is built again on the idea that "the contour of a contour is void" or "the contour has no contour":

**Proposition 4.1.** *It holds that $\partial_k \circ \partial_{k+1} = 0$, where $0$ is the constant function in the zero of the space.*

This properties seems very similar to the criteria used before to identify holes, but it is not. It is correct to say that if an $k$-chain does not have a contour than there are some holes, but in general it is wrong to say that $k$-chain is the imagine of $k+1$-chain. And here this theorem simply say that if a $k$-chain is the imagine of a $k+1$-chain then it has some holes. We can resume the situation, up to now, in figure 1. From this perspective we

$$\cdots \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \xrightarrow{\partial_{n-1}} C_{n-2} \xrightarrow{\partial_{n-2}} \cdots$$

Figure 1: Boundary operations

can state the proposition also as $\operatorname{Im}\partial_{n+1} \subseteq \operatorname{Ker}\partial_n$. And this justify the last step of our journey to construct the homological structure of a simplex, we define the $n$-th simplicial homology vector space of a simplex complex $K$ as:

$$H_n(K) = \operatorname{Ker}\partial_n / \operatorname{Im}\partial_{n+1}$$

The reason why we do this is simpler than it appears. Let us show you this simple example. To avoid to use annoying letters observe this simplicial complex (figure 2). Then the in $C_2$ the simplicial complex is seen in $C_2$ as figure 3a and in $C_1$ as 3b. In this last if we compute the Kernel then what we get is a subspace of dimension 2, since it identify two holes in the simplex. But one of them is a "false" one: the inside on the whole simplicial complex (figure 2) is filled. This difference between the two holes can be enlighten observing that one hole is in $\operatorname{Im}\partial_2$, the other not. So the idea is that there can be two type of holes: one those that are in $\operatorname{Im}\partial_n$ and those that are not, and the we like to not counting the former. A way to do this in our linear spaces

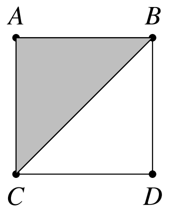
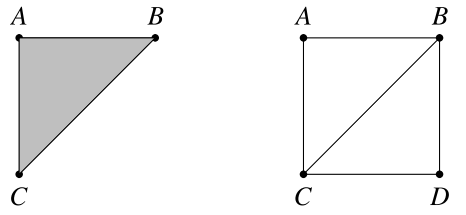


Figure 2: Our simplicial complex



(a) Our simplicial complex in $C_2$

(b) Our simplicial complex in $C_1$

Figure 3: The $k$-simplex of our simplicial complex

is identifying the elements inside $\operatorname{Im}\partial_n$ with the null element: that is doing a quotient operation on the space.

Now we are able to understand the points itemized before: the proposition 4.1 is the keystone of the whole system since it justify the passadge to the quotient. If we had imposed to $C_k$ to be the free abelian group, in order to obtain a similar results we should have introduced an order between the elements of $\Sigma_k$ and add a $(-1)^i$ after the summation of the formula above: this means more complexity on the space memory and time, so it is preferable to avoid it.

Now we states this theorem that ensure that this construction is well defined:

**Theorem 4.1.** *If $|K|$ and $|L|$ are homotopic, then $H_n(K)$ and $H_n(L)$ are isomorphic.*

This say that if we want to see if two simplicial complexes are homotopic it is sufficient to study the homological structures: if they are not isomorphic they are not homotopic, otherwise we cannot say more unfortunately. So we can study the homological structure of a simplex using linear spaces but here this classification is a roughly one. But there is a very great advantage: there is a easy criterion to establish if two linear spaces are isomorphic:

**Theorem 4.2.** *Two linear spaces are isomorphic if and only if they have the same dimension.*

So in order to study if there is an isomorphism between $H_n(K)$ and $H_n(L)$ it is enough to look at this sequence of numbers:

$$\beta_n(K) = \dim H_n(K)$$

They can be interpreted as the number of $k$-dimensional holes of $K$. Their name is Betti's number and it is all we need. The facto this is a great simplification and the problem of computing the Betti's number can be resolved using matrices.

## *4.2. Approximation of topological spaces thorugh simplicial complexes*

In a similar way we can define $H_n(T)$ on a general topological spaces $T$, and there are also similar results. Up to now the first question we proposed has been solved, we can focus on the second. To this question there is a theorem that comes in help, first we introduce this definition:

**Definition 4.3.** *Let X be a topological space and $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ be any cover of X. The nerve of $\mathcal{U}$, denoted by $\mathrm{N}\,\mathcal{U}$, is the abstract simplicial complex with vertex set A, and where a family $\{\alpha_0 \dots \alpha_n\}$ is an edge if and only if $U_{\alpha_0} \cap \cdots \cap U_{\alpha_n} \neq \emptyset$.*

This is a way to construct the simplicial complex from a cover. If we want to imagine it we can think as the sets of the covers as points, this points are connected if there a "share" of a part of the space between the two sets. This can be seen as an approximation of a topological space through a simplex with a given cover. The main theorem here is the following.

**Theorem 4.3 (Nerve theorem).** *Suppose that X and $\mathcal{U}$ are as above. Suppose further that*

- *The covering is composed by at most a countable quantity of open set;*
- *We have that, taken $S \subseteq A$, $\bigcup_{s \in S} U_s$ is either empty or contraible (that is homotopic to a point).*

*Then* $\mathrm{Nrv}\,\mathcal{U} \simeq X$.

As consequence all the homological groups (that is, our linear spaces) are isomorphic. This is the result we were seeking since it gives a sufficient condition to make true our question. And the best part is that since we are working especially with $V \subseteq \mathbb{R}^n$ we have already a method to generates this cover: the Čech complex attached to $V$ and $\varepsilon$. Here there is also a stronger result:

**Theorem 4.4.** *Let M be a compact Riemannian manifold. Then there is a positive number e so that $\check{\mathrm{C}}\mathrm{ech}(M, \epsilon)$ is homotopy equivalent to M whenever $\varepsilon \leq e$. Moreover, for every $\varepsilon \leq e$, there is a finite subset $V \subseteq M$ so that the subcomplex of $\check{\mathrm{C}}\mathrm{ech}(V, \varepsilon)$ is also homotopy equivalent to M.*

This result guarantees that if we want to compute an homological structure of a compact subset of $\mathbb{R}^n$ we can do it computing from a its finite subset. Unfortunately we cannot choose this finite subset as request in the theorem: we have got only a sample $X$ from $\mathbb{X}$ (our $M$) that is fixed. So even if this theorem ensures us that the approximation of the homological structure of $\mathbb{X}$ can be approximated by a good choice of $X \subseteq \mathbb{X}$ we are will never be able to find the exactly homological structure of $\mathbb{X}$ knowing a sample $X$. So the question ha to be reformulated in "how can I infer the homological structure of $\mathbf{X}$" having only $X$?". The answer is persistent homology and it will be discussed in the next section.

## *4.3. Homology inference*

Suppose now we have a set of points $X$ that were sampled from $\mathbb{X}$; using $X$ we want to say something about the topology of $\mathbb{X}$. Even if there is a theorem that ensures the existence, for a small value $\varepsilon$, of a good enough finite subset $\tilde{X}$ that can be used to approximate the homological structure of $\mathbb{X}$ we are not able to use it, but we can take some inspiration to achieve similar results. We can start considering a cover of $\mathbb{X}$, thus:

$$X_r = \bigcup_{x \in X} B_r(x)$$

Here some issues can arise, in particular:

1. We do not know the smallest value of $r$ such that $X_r$ is a cover of $\mathbb{X}$;
2. The Betti's numbers are quite unstable for little variation of $r$.

So we cannot choose for $r$ value as small as we want, moreover the second point highlights that a strategy based on the inference of $r$ does not work very well since it won't be consistent. Another approach can be to not consider a single approximation but the several ways that $X$ can approximate $\mathbb{X}$ trough simplicial complexes. From a computational point of view this increase the time complexity but also can be used to find some persistent features that are more likely to be true also in $\mathbb{X}$. So the main idea of persistent homology is to consider a particular evolution of a simplicial complexes as approximation of $\mathbb{X}$, this can be found with particular construction explained before, and then consider the persistent homological features that are more likely to occur also in $\mathbb{X}$. We have to formalize this framework, to do this we will consider the more general case and then we will focus on the particular cases of Čech and Viector-Rips.

### *4.3.1. The general framework*

First of all we need a mathematical object that formalize the evolution of approximation. We can use the concept of filtration:

**Definition 4.4.** *A filtration of a simplicial complex K is a collection of subcomplexes $K_i \subseteq K$ such that*

$$K_0 \subseteq K_1 \subseteq \cdots \subseteq K_N = K$$

Here, using the filtration, we are imposing an additive structure that was not explained before. We want several approximations of $\mathbb{X}$ but here we also impose that these should be "increasing": to construct $K_{n+1}$ we start from $K_n$ and we add some features. This seems to be a restriction that it is preferable to avoid in the most general case, and it is possible: though it complicates the things because the final order between the simplexes is not linear and for visualization (that is our main aim) is preferable to avoid. Moreover this is the exact thing that happen if we consider the Čech approximation letting $r$ varying: the approximation is increasing. An example of filtration of a simplicial complex $K$ can be visualized in the figure 4.

We want to see if there are persistent homological features, so we can for example consider the 1-dimensional holes in figure 4:

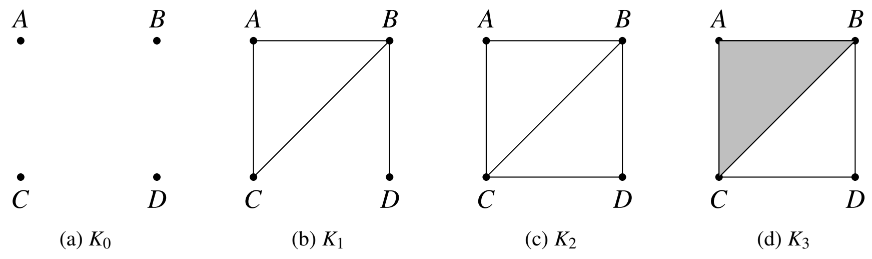


Figure 4: A filtration of $K = K_3$

1. In figure 4a there are no holes
2. In 4b an holes appears
3. In 4c there are two holes, but one was also in 4b
4. In 4d the hole in 4b is filled and that in 4c is again an hole.

We need a mathematical framework that can formalize the list above. We understand that we need something that is able to link the holes in the different simplicial approximation: a map between $H_p(K_m)$ and $H_p(K_n)$ when $m < n$. To do this we can start observing that since this is a filtration we can consider for each (increasing) pair of simplicial complexes the canonical inclusion map:

$$i_{m,n} : K_m \to K_n \qquad \text{for} \qquad m < n$$

This map can be used to induces a linear map between the spaces of $n$-dimensional holes, recalling that the element of $K_m$ are then a bases on the free linear space. So here we have a map:

$$i^{\star}_{m,n,p} : H_p(K_m) \to H_p(K_n) \qquad \text{for} \qquad m < n$$

This can map can be defined for each $p$ and each $m < n$: we have a family of maps. In particular:

- A new topological feature $[h] \neq [0]$ (that is a proper hole) in $H_p(K_n)$ appears can be formalized as $[h] \in H_p(K_n)$ but $[h] \notin \mathrm{Im}(i^{\star}_{n-1,n,p})$;
- A topological feature $[h] \in H_p(K_n)$ disappear at the next iteration can be formalize as $i^{\star}_{n,n+1,p}([h]) = [0]$

In particular, by what we have said in this list we can understand that an hole $[h] \in H_p(K_m)$ are preserved up to $H_p(K_n)$ if $i^{\star}_{m,n,p}([h]) \neq [0]$. By this observation we can introduce the following definition:

**Definition 4.5.** *The p-th persistent Betti's number associated to $i^{\star}_{m,n,p}$ is*

$$\beta^{m,n}_p = \dim \mathrm{Im}(i^{\star}_{m,n,p})$$

This is dimension can be interpreted as the number of $p$-dimensional holes that are persistent between $K_m$ and $K_n$. But to capture the evolution of the system we can also consider:

$$\mu^{m,n}_p = (\beta^{m,n-1}_n - \beta^{m,n}_n) - (\beta^{m-1,n-1}_n - \beta^{m-1,n}_n)$$
$$\mu^{m,\infty}_p = (\beta^{m,N}_n - \beta^{m,N}_n)$$

Where we recall that $K_N = K$. These numbers counts the number of classes $[h] \in H_p$ that born at time $m$ and die at time $n$. In particular, to visualize the evolution we can consider these two new structure can be found:

**Barcode** in dimesion $n$, here we interpret every non zero $\mu^{i,j}_n$ as the interval $[i, j)$, and stacking all the intervals on top of each other in the plane;

**Persistent diagram** in dimension $n$ is obtained by plotting $(i, j)$ (with multiplicity) in the plane for every non-zero $\mu^{i,j}_n$.

Compute this numbers seems very complicate. Fortunately there is a result that simplify the things. We need to follows this pipeline:

- Create the filtration of the simplicial complex.
- Create a base compatible with the filtration. This will be a matrix in $C_0 \oplus C_1 \oplus \cdots \oplus C_k$.
- Create the boundary matrix associated to this basis. If you notice it will be form by a concatenation of all the boundary matrices.
- Perform a column reduction via Gaussian eliminations
- Now observe the pivot element of each column of the matrix. If $\mathrm{low}(j) = i$ then the feature associated with position $i$ in the base dies when $j$ appears. If $\mathrm{low}(j)$ is undefined (there is no pivot) then the entrance of the feature associated to the element $j$ cause the birth of a feature in the filtration.

### *4.3.2. Čech and Vietoris-Rips filtrations*

Here we focus more on Cech and Viectoris-Rips and their tractabiliy. We recall, firstly that $X_r = \bigcup_{x \in X} B_r(x)$. We recall also that the constructions of these two simplexes was done before. To create a filtration we can use an increasing sequence $(r_n)_n \subseteq \mathbb{R}$. Then our situation can be resumed in the diagram in the figure 5. The isomorphism are due to the Nerve theorem. It is possible to prove that the persistent Betti's number for the two different rows coincides, therefore the barcode and the persistent diagram must coincide. They can be interpreted as the capture of the evolution of the holes of a given dimension formed by the balls $B_r(P)$. Even if the Čech construction has a very simple geometrical interpretation it has a very big issues: in this approch one needs the coordinates of the points in $\mathbb{R}^d$ and this can be bad. In practice it is preferable to work with a distance matrix: sometimes we know only it and we do not have any coordinates. So a better filtration for persistent homology can be that of Viector Rips. There is a little issue: we know

$$\begin{tikzcd}
H_n(X_{r_0}) \arrow[r] \arrow[d, "\simeq"] & H_n(X_{r_1}) \arrow[r] \arrow[d, "\simeq"] & H_n(X_{r_2}) \arrow[r] \arrow[d, "\simeq"] & \cdots \arrow[r] & H_n(X_{r_m}) \arrow[d, "\simeq"] \\
H_n(\check{\mathrm{C}}\mathrm{ech}(X, r_0)) \arrow[r] & H_n(\check{\mathrm{C}}\mathrm{ech}(X, r_1)) \arrow[r] & H_n(\check{\mathrm{C}}\mathrm{ech}(X, r_2)) \arrow[r] & \cdots \arrow[r] & H_n(\check{\mathrm{C}}\mathrm{ech}(X, r_m))
\end{tikzcd}$$

Figure 5: Evolution of $H_n$ along the filtration

how the persistent homology work, but why can we choose this last filtration. The answer is based on:

$$\check{\mathrm{C}}\mathrm{ech}_r(X) \subseteq \text{Vietoris-Rips}_r(X) \subseteq \check{\mathrm{C}}\mathrm{ech}_{\sqrt{2}r}(P)$$

Applying homology we get the following graph (figure 6). This

$$\begin{tikzcd}
H_n(\check{\mathrm{C}}\mathrm{ech}_r(P)) \arrow[rr] \arrow[dr] & & H_n(\check{\mathrm{C}}\mathrm{ech}_{\sqrt{2}r}(P)) \arrow[dr] & \\
& H_n(\text{Vietoris-Rips}_r(P)) \arrow[ur] \arrow[rr] & & H_n(\text{Vietoris-Rips}_{\sqrt{2}r}(P))
\end{tikzcd}$$

Figure 6: Relation between Čech and Vietoris-Rips filtrations

diagram show that any class on $H_n(\text{Vietoris-Rips}_r(P))$ which has no zero class in $H_n(\text{Vietoris-Rips}_{\sqrt{2}r}(P))$ must be non zero in $H_n(\check{\mathrm{C}}\mathrm{ech}_{\sqrt{2}r}(P))$. Conversely the same reason can be done if we invert the roles. In poor words we can say that sufficiently long intervals in the barcode of the Cech filtration give rise to intervals in the barcode of the Vietoris-Rips filtration, and any sufficiently long interval in the barcode of the Vietoris-Rips corresponds to a true geometrical feature (recall the geometric interpretation of Čech). This, in addition to the two points of before, makes the Viectoris-Rips the main approch to construct filtrations.

## 5. Implementation of the persistent algoritmh

In this section we discuss the data type used and the more important algorithm. In particular we are focusing on the way some concept were implemented and in the way some algorithm were optimized.

### *5.1. Sparse Binary Matrix Class*

As we repeatedly said before we are going to use the tools of linear algebra: in particular we are going to use matrices with values in $\mathbb{Z}_2$ that have to be reduced. There are several ways to proceed to this goal: we can for example define a class for number in $\mathbb{Z}_2$ and its main operations and then use it with the numpy library. In this way the classical Gaussian Algorithm has a complexity of $O(n^3)$. Here is its pseudo-code:

**Algorithm 4** Matrix reduction algorithm

```
for $j = 1$ to $n$ do
    while there exists $i < j$ with low$(i)$ = low$(j) \neq -1$ do
        add column $i$ to column $j$
    end while
end for
```

Recall that from now on low($j$) always denotes the greatest non zero index in the $j$-th column. If the latter does not exist than it return $-1$. In the general case, with this approach, this operation has a linear time complexity. Moreover the sums of two vectors is an operation of linear time complexity. This is not optimal: recall that we are going to use matrices constructed from Vietoris-Rips simplicial complexes. The size, in the worst case, of such simplicial complex constructed starting from $n$ points in dimension 3 is $O(n^4)$. This will be the size of the matrices we are going to use: the final total complexity will be $O(n^{12})$ that is absurd. To face this problem we could choose another simplicial construction that approximate the Vietoris-Rips simplicial complex: there are construction that return a complexes of size $O(n)$. To recapitulate we can proceed in these ways:

- Use other construction that returns smaller simplicial complexes. In this way we lose some theoretical validation, but in the same time we optimize the algorithm.
- Use another way to represent the matrices that is optimized for this situations.

We proceed only with the second option, the way we optimized matrices worked so well that to have good result in few time it is possible use the Vietoris-Rips filtration. We observed that the matrices we use are always spare matrix: in a column of $O(n^4)$ rows there are at most $p << n$ indexes with a non null value. This lead to the following two observations:

- The summation for the most of the times sums two zeros, hence there are a lot of useless operations. We want to find a way to pass from a linear time complexity to a constant time complexity.
- The time complexity to compute the pivot is linear: as before since there a lot of zeros we want to make the computation constant in time complexity.

We want these two considerations at least for average case. Dictionaries come in aid. They allows to represent the matrix in a fancier and quick way. If we want represent a matrix with

$n$ columns the idea is to create a dictionaries (or also a list) in which each index associates a set containing all the indexes where the row is equal to one. With this framework the computations simplifies a lot. In fact we can assume that the number of element in a column that are equal to one remain constant, or at least when it became greater it happen in a very restrictive numbers of case. Then we get:

- The summation of two columns of the matrix can be replaced by the symmetric difference of two sets. Hence, by what we said so far, the time complexity is constant in the average case.
- The pivot computation can be done simply using the maximum of the set. Hence also here the time complexity is constant in the average case.

There is another little problem: the part that says while there exists $i < j$ with low($i$) = low($j$). We want to make also this part faster. To do this we use, again, dictionaries. We notice in fact that when the for cycle for the $i$-th column is done then its pivot does not change anymore. Hence the idea is to save in a dictionaries the values {low($i$) : $i$}. In this way, instead of comparing all the pivot values of all the columns it is enough to compute the pivot of the $j$-th column and then find the column with the same pivot using the dictionaries. This makes the computation even faster. Now we itemize the attribute and method of this ADT:

| **Sparse Binary Matrix** | | |
|---|---|---|
| **Attributes** | | |
| col | The number of column of the matrix | |
| matrix | The dictionaries of the matrix | |
| **Method** | | |
| len | $O(1)$ | Return the number of column of the matrix |
| put_one | $O(1)$ | Put a one in a specific row and column |
| put_one | $O(1)$ | Put a zero in a specific row and column |
| column_sum | $O(1)$ | Sum to a column another column of the matrix |
| pivot_element | $O(1)$ | Find the pivot of a specified column |
| reduction_Z2 | $O(n^2)$ | Reduce a matrix and return also the pivot of the columns |

Recall that the complexities are referred respected to a sparse binary matrix.

## 5.2. Simplex Class

In this class we represent simplicial complexes. Recalling what we have said before we are going to use the abstract definition of simplicial complex. Hence a simplicial complex can be represented as $(S, \Sigma)$, where $S$ is the set of the vertices and $\Sigma$ is such that if $\sigma \in \Sigma$ and $\pi \subseteq \sigma$, with $\pi \neq \emptyset$, then $\pi \in \Sigma$. Recall that:

$$C_n = \{\sigma \in \Sigma \mid \#\sigma = n + 1\}$$

This sets are very important since they are a base for their free vector space with coefficient in $\mathbf{Z}_2$. To simplify the computation what we can consider is to construct a list, in each position there is these sets. That is a simplex can be represented as:

$$\Sigma = [C_0, C_1, \ldots, C_n]$$

So now the problem is: how can I represent $C_i$? Fortunately Python has a structures called sets. A set in python is like a list, but here the order does not matter and moreover repetitions are not allowed. So $C_i$ can be represented as a set. Use this class has some computational advantages, since a set is a mutable object. There is an issues: a set can contain only Hash-able type, that is variable in which is possible assign a unique Hash number. For example a tuple, an immutable element, is Hash-able since it remain unchanged during its life: hence is possible to assign for all its life an unique number. A set is not Hash-able, since when we add an element to a set then we have to change also is Hash number. This can be a problem, because $C_i$ is a family of sets, hence we need another type of set that can be contained in a set. This is a frozenset that differently from the set is its immutable version. We can resume the class we are going to use:

**List** This is used to collect all the $C_i$ in a unique element. Here the order matters, hence we have to use a list or, if you prefer, a dictionary.

**Set** This is used to represent $C_i$, we can also use a frozenset, but in this way the method of this class are slower. In a set in fact we have the add method that makes the computation faster.

**Frozenset** This is used to represent the element of $C_i$, we use it because a set cannot contain a set, but it can contain a frozenset.

We describe the ADT:

| **Simplex** | | |
|---|---|---|
| **Attributes** | | |
| simplex | The list of set of before | |
| **Method** | | |
| add_dimension | $O(n)$ | Add a empty set a the bottom of the list $n$ times |
| simply_add_element | $O(1)$ | Insert an element specified by the user in the right place according its dimension |

## 5.3. Filtration Class

This is the last class needed. In this class we represent filtration of simplicial complexes, and we obtain the persistent results of before. To represent the filtration we can simply take a list whose element are element from simplex class. We used list since they can mutate, and there the order does matter. Now this is the pseudo-code we want to implement:

**Algorithm 5** Persistent Homology algorithm

```
If needed, compute the distance matrix
Compute the Vietoris Filtration given the distance matrix and a list of diameters
Create a Base compatible with the filtration
Create the matrix associated with the boundary application
Reduce the matrix
Using the reduction results compute the persistence
```

We procede by steps commenting the more relevant passages:

*5.3.1. Vietoris Filtration*

Here we do not report the pseudo-code because the algorithm has a lot of for cycle and try to represent it in a pseudo-code is not very helpful. We prefer to write its pipeline:

- For each simplex in the filtration add all the points.
- For each dimension $d$ take the element of the last simplicial complex (the biggest one) with dimension $d - 1$, and consider all the $d$-simplex by adding an element per time.
- For each new $d$-simplex compute the diameter and check the point in which it appears.
- Then from that point on add the $d$-simplex to the filtration.

One can ask the reason why do not we simply create the Vietoris-Rips simplicial complex for all the diameters instead of first create an $k$-simplex and then adding it to all the simplicial complex with the diameter great enough. If you make the computation the first case of before requires a $O(n^{d+1}d^2 f)$ computational complexity in the worst case, the second requires a $O(n^{d+1}d^2 f\log(f))$ since we have to add a binary search. Here $n$ is the number of point, $d$ is the dimension of the simplicial complex and $f$ is the length of the filtration. The reason why we prefer the second is because in most of the cases it is quicker, for example in the protein of this project the first with a low complexity require three minutes, the second twenty seconds. The reason is because in most of the case the features typically are added in the last part of the filtration, in the first part usually there are very few. Intuitively is just as the "real" complexity is $O(n^{d+1}d^2\log(f))$.

*5.3.2. Creation of a base*

Now that we have the simplicial complex we have to create a base that is compatible with the filtration. This means that:

- Firstly we take the points of the first simplex;
- Then all the pairs of the first simplex, then the triples and so on;
- Then we pass to the second simplex and we take all the pairs that are in the second simplex but that there are not in the first one; then we do the same for the triplets and so on;
- We continue in this way up to the last simplicial complex.

To do this we need a data type for deal also with a base. We can use lists, but it is better instead use dictionaries. In fact we can assign to each simplex the position in which they appear in the base. We do this for a main reason: find out if an element belongs to a dictionary is much faster. Hence our base takes a $k$-simplex and assigns its position in the base. In this way also the search of the position of an element is faster. Moreover we save in a list the index when we pass from a simplicial complex to the next of the filtration. This is useful later.

*5.3.3. Creation of the matrix*

Here we are going to use the Sparse binary class defined before. Using the base introduced before we are able to create the matrix very quickly. The algorithm is the following:

**Algorithm 6** Boundary matrix Algorithm

```
for $\sigma$ in Base.items() do
    $j \leftarrow$ Base[$\sigma$]
    for $s \in \sigma$ do
        $i \leftarrow$ Base[$\sigma \setminus \{s\}$]
        Put a one in the matrix in the position $(i, j)$
    end for
end for
```

Then we can get the reduction results using the algorithm of before.

*5.3.4. Persistence*

Here we want to save the pairs of the type:

(when the feature born, when the feature dies)

This will be saved in a list of list: the $i$-th index list contains all the birth-death of the feature of dimension $i$.

*5.3.5. Recapitulation*

Here we insert a recapitulation of what said so far.

| **Filtration** | |
|---|---|
| **Attributes** | |
| dimension_filtration | Dimension in which data live |
| distance_matrix | Distance matrix of the points |
| data | Coordinates of all points |
| index_filtration | List of all radius |
| base | Base compatible with the filtration |
| boundary_matrix | Boundary matrix |
| reduction_results | Reduced matrix and pivots |
| persistence_results | Pair in which a feature born or die |
| **Method** | |
| add_simplex_Vietoris | $O(n^{d+1} d^2 f \log(f))$ |
| create_basis | $O(m)$ |
| create_boundary_matrix | $O(md)$ |
| compute_reduction | $O(m^2)$ if we consider the dimension irrelevant |
| compute_persistence | $O(m \log(f))$ |

Here again $n$ is the number of points, $m$ is the number of feature in the last element of the filtration, $d$ is the dimension, $f$ is the length of the filtration.

## 6. Application of TDA to protein Structures

The protein we choose is Deamino-Oxitocin a protein that is classified as Hormone. It is characterize by the low number of amino acids, which makes it ideal for us to verify our algorithms with our limited computational power.
With the use of the libraries nglview and BIO we was able to upload a file .pdb and to obtain the coordinates of all the amino acids of the protein.

### *6.1. Visualization of Protein Structures using the Mapper Algorithm*

Here for the reduction the choice of the function is very important also the cluster algoritmh must be well chosen. We tried several combinations:

| | |
|---|---|
| Fig. 7 | Function: $x_0$; $r = 0.5$, $g = 0.2$, $\varepsilon_{DBSCAN} = 0.25$, $Min_{DBSCAN} = 5$ |
| Fig. 8 | Function: $x_1$; $r = 0.5$, $g = 0.2$, $\varepsilon_{DBSCAN} = 0.25$, $Min_{DBSCAN} = 5$ |
| Fig. 9 | Function: $x_2$; $r = 0.5$, $g = 0.2$, $\varepsilon_{DBSCAN} = 0.25$, $Min_{DBSCAN} = 5$ |

Looking at the plot we can say that the protein has a very simple structure. Almost lineare without cavities or holes. It seems that there are some component and that the protein can be partitioned in several parts.

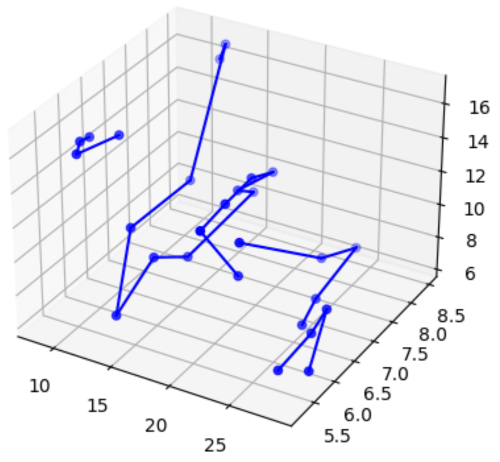


Figure 7: Mapper 1

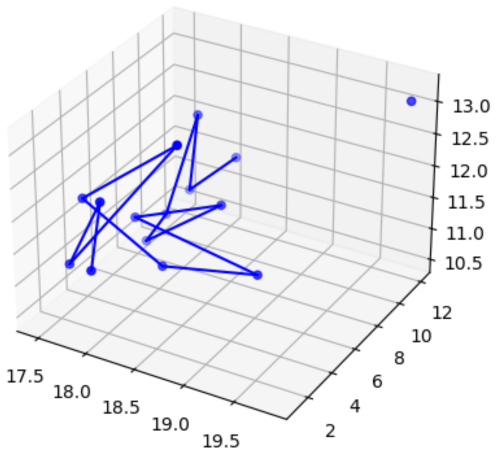


Figure 8: Mapper 2

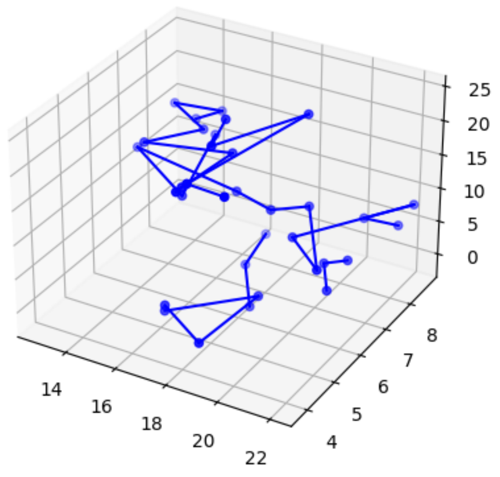


Figure 9: Mapper 3

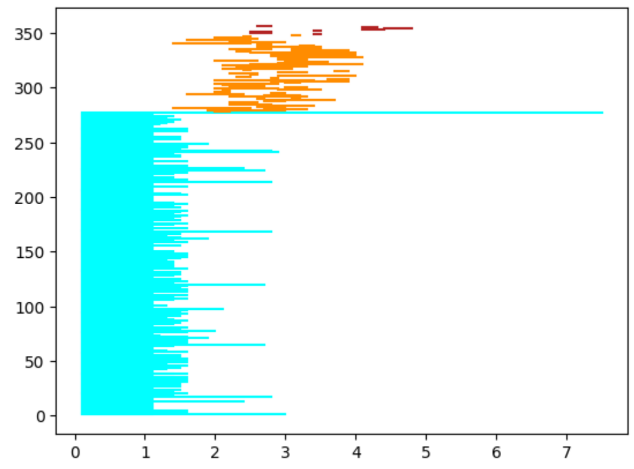


Figure 10: Barcode of the protein

### 6.2. *Identification of Topological Features in Protein*

If we use the classical euclidian distance and the Vietoris-Rips filtration what we get is the picture 10. We can see very clearly that there are no cavities since there are no very long red lines. For the circles the situations is different, we can see that there is at least one circle since it seems that an orange line is quite long. For the connected component instead we can see that there are, before disappear 9 features. This suggest us that the protein can be divided in 9 different parts. Of course we can just simply plot the protein in a graph but recall that this information can also be obtained using only the distance and without knowing the real position of the amino acids. This is the good part of this algorithm: it can be extended also to very abstract spaces and we are also able to obtain the shape of the dataset.

### 6.3. *Limitation*

The connection between amino acids, which is very important for our study, does not depend only on the distance between them. This makes the use of the euclidean distance, as we did in our algorithms, not accurate and this is why our graph does not represent the true structure of the protein. In order to achieve a great result should be used other types of distances as Chemical distance that does not consider just the physical proximity but also chemical interaction like hydrogen bonds, disulfide bridges or hydrophobic interaction.
We decide to stick on the euclidean distance because, its true that the connection between amino acids is characterize in the main part by chemical interaction but the physical proximity is not irrelevant and having this graph representation with the topological features could be also helpfull.

## 7. Summary and conclusions

In summary this was a simple introduction to the great world of topological data analysis where the principal aim is to understand how to infer the shape of some data. The abstrac approach makes this framework very flexible and with several applications. One of this the use of TDA to infer some spatial properties of proteins that from a biological point of view are very important.