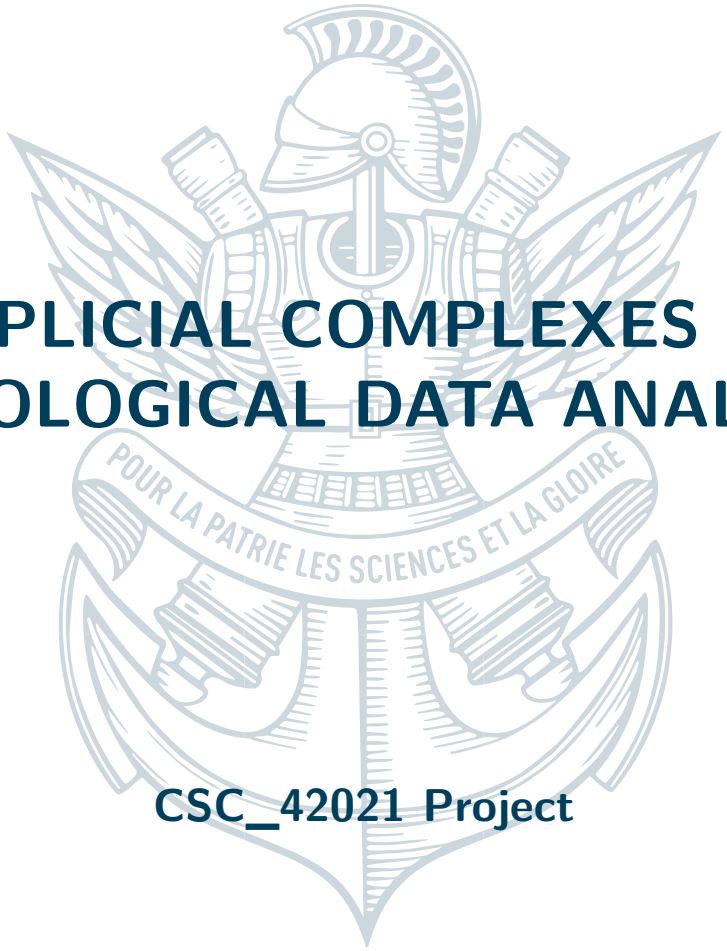

SIMPLICIAL COMPLEXES FOR TOPOLOGICAL DATA ANALYSIS

CSC_42021 Project

February 16, 2025

Salah CHIKHI - Baptiste ARNAUDO

CONTENTS

1 INTRODUCTION

1.1 Generalities

Data analysis is the process of inspecting data to extract useful information, identify patterns, and support decision making. It involves various techniques from statistics, machine learning, and computational methods to interpret data, detect trends, and draw meaningful conclusions. In an era of rapidly growing datasets, effective data analysis is crucial to extract meaningful insights and make informed decisions across various fields.

There exist a number of tools to analyze data. This project focuses on one aspect of data analysis known as **Topological Data Analysis** (TDA). It focuses on identifying robust topological features, such as connected components. A key concept in TDA is the use of simplicial complexes, which provide combinatorial representations of data using simplices (points, edges, triangles, tetrahedra, and higher-dimensional analogs known as k-simplexes). These structures enable the application of homology theory to quantify topological properties, with persistent homology being a central tool for tracking how features evolve as we vary the scale of analysis.

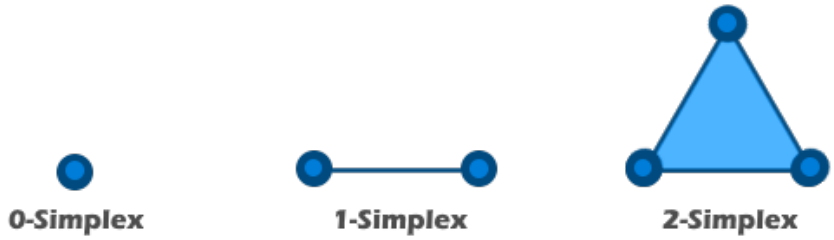

Figure 1: k-simplexes

From here, we may define simplicial complexes which are a generalization of graphs. In addition to vertices and edges, it may contain any k-simplex. This project studies two sorts of simplicial complexes: the **Čech complex** and the α**-complex**.

Definition 1. Čech complex

Let X be a set points in a metric space and $r \in \mathbb{R}^{+*}$. The **Čech complex** of X of parameter r, noted $\check{\mathrm{C}}_r(X)$, is defined as :

$$\check{\mathrm{C}}_r(X) = \left\{ \sigma \subset X, \quad \bigcap_{x \in \sigma} B(x, r) \neq \emptyset \right\}$$

Definition 2. α**-complex**

Its definition is similar to the Čech complex, but instead of having a ball around each point, we only keep the part of the ball that is in the Voronoi cell of this point, that is to say, the points of the space that are closer to the point than to any other point of the set.

In order to compute and visualize these complexes, we use concepts such as **LP-type algorithms** that will prove to be essential tools in this study.

1.2 Theoretical considerations

Before we study the algorithmic side of the project, let us first prove important theoretical results which we will use later on.

One of the main concepts which will serve as a building block during this project is the existence of a **Minimal Enclosing Ball** (MEB) of a set of points of $\mathbb{R}^d$. More generally, we prove the following theorem:

Theorem 1. Let E be a euclidean vector space over $\mathbb{R}$. Let $A \subset E$ be non-empty and bounded. Then there exists a **unique** ball of **minimal** radius that contains A. This immediately implies that the MEB of a finite set of points of $\mathbb{R}^d$ exists and is unique.

Proof. Let E be a euclidean vector space over $\mathbb{R}$. Let $A \subset E$ be non-empty and bounded. We first show that the MEB of A exists. Since A is bounded, for $a \in A$, we may define:

$$R_a = \sup_{x \in A} \|x - a\|$$

Of course, $A \subset \bar{B}(a, R_a)$ and R_a is the minimal enclosing ball of A with center equal to a. We may now define :

$$R = \inf_{a \in A} R_a.$$

There exists $(a_n) \in A^{\mathbb{N}}$ such that $R_{a_n} \to R$.

Now fix $x_0 \in A$. We have:

$$\|a_n\| \leq \|x_0\| + \|x_0 - a_n\| \leq \|x_0\| + R_{a_n}$$

Since the right hand side of this inequality converges, it is bounded hence (a_n) is bounded in the finite dimension vector space E. Thus, (a_n) has an adherence point. Without loss of generality, we may suppose that (a_n) converges to $a \in E$. Now fix $x \in A$. Since:

$$\|x - a_n\| \leq R_{a_n}$$

By letting n tend to infinity, we get :

$$\|x - a\| \leq R$$

Hence $A \subset \bar{B}(a, R)$ and by construction, $\bar{B}(a, R)$ is a MEB of A. Let us now prove the unicity of this ball. The key argument is that a euclidean norm statisfies the parallelogram identity. Suppose that $\bar{B}(a, R)$ and $\bar{B}(a', R)$ are two MEBs of A. Let $b = \dfrac{a' + a}{2}$. For all $x \in A$, the parallelogram identity yields:

$$\|x - b\|^2 + \left\| \frac{a - a'}{2} \right\|^2 = \frac{1}{2}(\|x - a\|^2 + \|x - a'\|^2) \leq R^2$$

Thus :

$$\left\| \frac{a - a'}{2} \right\|^2 \leq R^2 - \|x - b\|^2 \leq R^2 - R_b^2 \leq 0$$

Hence $a = a'$ proving the unicity of the MEB.

■

2
ANALYSIS OF THE WORK DONE IN EACH TASK

This section is dedicated to analyze the work done in each task. For each task, we explain the goal of the code provided, the reason as to why it works and analyze the code itself.

2.1 Task 1

2.1.1 • Computing the minimal enclosing ball

This task is dedicated to the core concept of the following algorithms which is the concept of the minimal enclosing ball. The algorithm takes in a set of points of $\mathbb{R}^d$ and returns its minimal enclosing ball. We know, by **Theorem 1** that this minimal enclosing ball exists and is unique.

In order to compute the minimal enclosing ball of a set of points, we first need to compute the circumcenter of a nondegenerate set of points $p_0, \dots, p_k$ for $k \leq d+1$.

Let $p_0, \dots, p_k$ be said points and g be their circumcenter.

There exists positive constants $\theta_0, \dots, \theta_k$ that sum to 1 such that

$$g = \sum_{i=0}^{k} \theta_i p_i$$

Now fix $0 \leq l \leq k$. Notice that

$$\|p_0 - g\| = \|p_l - g\|$$

We can rewrite this as :

$$\langle g, g\rangle + \langle p_l, p_l\rangle - 2\langle g, p_l\rangle = \langle g, g\rangle + \langle p_0, p_0\rangle - 2\langle g, p_0\rangle$$

i.e

$$\langle g, p_0 - p_l\rangle = \frac{\|p_0\|^2 - \|p_l\|^2}{2}$$

Hence:

$$\sum_{i=0}^{k} \theta_i \langle p_i, p_0 - p_l\rangle = \frac{\|p_0\|^2 - \|p_l\|^2}{2}$$

$$\sum_{i=0}^{k} \theta_i = 1$$

This defines a linear system in $(\theta_i)_{0\leq i\leq k}$ which we can solve numerically.

More precisely, define $\Theta = (\theta_0, \dots, \theta_k)^T$, $b = \left(\frac{||\ p_0\ ||^2 - ||\ p_0\ ||^2}{2}, \dots, \frac{||\ p_0\ ||^2 - ||\ p_k\ ||^2}{2}, 1\right)^T$. Then define a matrix A such that:

$$A_{ij} = \langle p_j, p_0 - p_{i+1}\rangle$$

for $0 \leq j \leq k$ and $0 \leq i \leq k-1$.

To satisfy the condition $\sum_{i=0}^{k} \theta_i = 1$, we extend A as follows:

$$\tilde{A} = \begin{bmatrix} A \\ \mathbf{1}^\top \end{bmatrix}$$

The linear system can now be written as :

$$\tilde{A}\Theta = b$$

which we solve numerically using **numpy**.

Note. We call **1** the vector with each entry being equal to 1.

We first tested the circumcenter function to make sure it worked before moving on to the MEB. It yielded the following satisfying results:

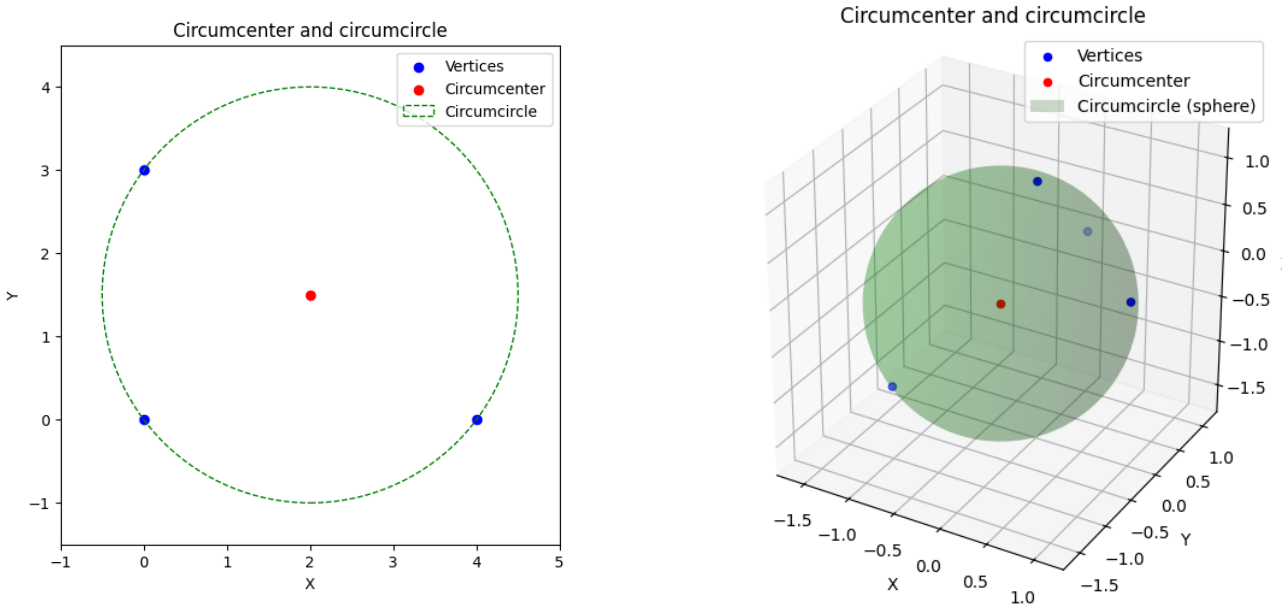

Figure 2: Examples of circumcenter for $d \in \{2, 3\}$

Note. We have also drawn the circumcircle for visualization purposes.

We now use this in order to compute the minimal enclosing ball (MEB). Given a set of points $S \subset \mathbb{R}^d$, for all $A \subset S$, we can define $f(A)$ to be radius of the minimal enclosing ball of the points of A. Let us show that this is indeed an LP-type problem.

- **Monotonicity :** It is clear that if $A \subset B \subset S$ then $f(A) \leq f(B) \leq f(S)$. Indeed, if $\mathcal{B}$ is a minimal enclosing ball of B then it is also an enclosing ball (though not necessarily minimal) of A. Hence the minimal enclosing ball of A is contained inside of that of B thus $f(A) \leq f(B)$. In particular $f(A) \leq f(B) \leq f(S)$.

- **Locality :** Let $A \subset B \subset S$ and $x \in S$. Suppose $f(A) = f(B) = f(A \cup \{x\})$. This means that x is either in the interior of the MEB of A or in its boundary. Since $f(A) = f(B)$ and $A \subset B$, x is also either in the interior of the MEB of B or in its boundary hence $f(A) = f(B \cup \{x\})$.

This shows that this, indeed, is an LP-type problem. We can now use typical algorithms that solve LP-type problems. We have decided to use Welzl's algorithm.

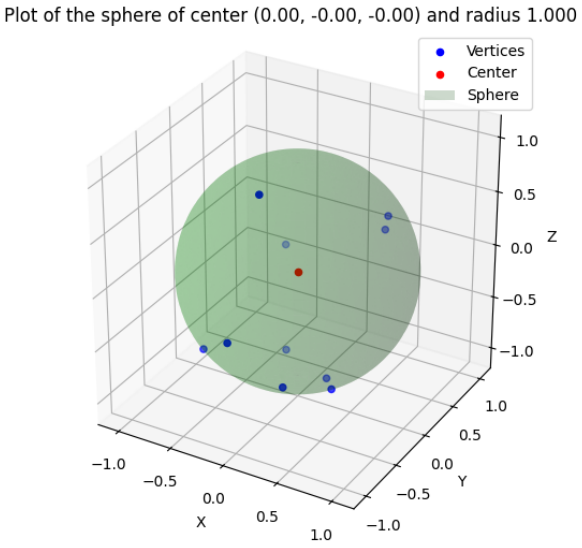

Figure 3: Result of our MEB algorithm for 10 random points

2.1.2 • Complexity analysis

Let n be the number of points we are considering and $\mathcal{P}$ be the set of these points. Welzl's algorithm has complexity $O(n \cdot d!)$ where d is the dimension of the space considered.

Having found a set $\mathcal{S}$ of $k+1 \leq d+2$ points such that the MEB of $\mathcal{S}$ is the MEB of $\mathcal{P}$, we compute its circumcenter which is done in $O(k^3) = O(d^3)$ (**numpy**'s system solver). Hence the complexity of the MEB algorithm:

$$O(d^3 + nd!) = O(nd!)$$

2.2 Task 2

2.2.1 • Enumerating simplexes

Our goal in this task is, given a set of n points in $\mathbb{R}^d$, to implement a naive algorithm that enumerates the simplexes of C^k and their filtration values.

This is done by generating all subsets of cardinal at most $k+1$ of our set of points using **backtracking**. Once that is done, we compute the filtration value of all non-empty subsets we've generated.

We compute each simplex of cardinal at most $k+1$, and for each of this simplex, we compute its MEB, the resulting complexity is proportional to :

$$\sum_{j=0}^{k+1} \binom{n}{j} d!j = d! \sum_{j=0}^{k} n \binom{n-1}{j-1}$$

which gives a complexity of $O(d!n2^n)$

2.3 Task 3

The key argument is the following : if a simplex is not in the complex, no simplex containing it can be in the complex.
We start off with tuples that have low filtration value and make our way up. If we ever meet a tuple $\mathcal{S}$ that has a filtration value larger than l, we do not consider any other tuple containing $\mathcal{S}$.

Hence if l is small, this will allow us to skip a large number of simplexes that have a filtration value larger than l. However, if l is large, we may end up computing most of the MEBs. Therefore, in the worst case, the complexity stays the same as that from task 2.
We can compare this algorithm to the naive algorithm which consists in generating all possible simplexes and only then filtering out the ones that have a filtration value larger than l.

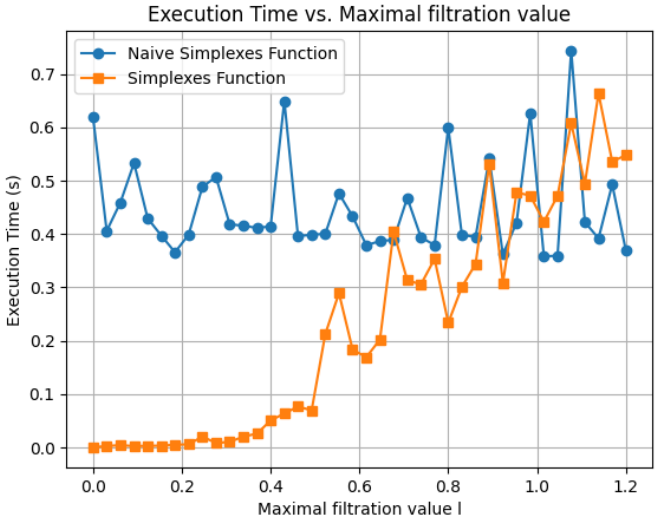

Figure 4: Runtime benchmark comparison between naive simplexes and simplexes functions

To make this figure, we generated 10 random points and ran the code for different values of $0 \leq l \leq 1.2$. The chosen dimension was $d = 3$.
Figure 4 clearly illustrates what was forecasted by theory: for small values of l, the non-naive version of simplexes runs a lot faster than the naive one. However, as soon as l grows large, the two algorithms become equivalent.

2.4 Task 4

In this task, we use a similar approach as the one used it task 1. Given $S \subset P \subset \mathbb{R}^d$, we need to find the minimal ball that has every point of S on its boundary, and no point in its strict interior. Formally, let us define:

$$f : A \subset P \mapsto \inf\{r \geq 0, \exists c \in \mathbb{R}^d, (\forall a \in A, ||a - c|| = r) \wedge (\forall p \in P, ||p - c|| \geq r)\} \in \mathbb{R} \cup \{\infty\}$$

The computation of f is indeed an LP-type problem because :

- **Monotonicity :** Clearly, it is the same argument as in task 1 (a ball having all points of B on its frontier has all points of A on it, and the constraints remain satisfied).
- **Locality :** Let $A \subset B \subset S$ and $x \in S$. Suppose $f(A) = f(B) = f(A \cup x)$. Then x is on the boundary of the ball that has every point of A on its boundary. But since $f(B) = f(A \cup x)$, x is on the boundary of the ball that has every point of B on its boundary, so $f(A) = f(B \cup x)$.

To compute f, we used a similar pipeline as in the MEB function, but we start Welzl's algorithm with the simplex as the basis. It ensures that at every recursive call, every vertex of the simplex remains on the boundary of the ball that is considered. Since, the algorithm is the same, the complexity is also the same.

2.5 Task 5

In this task, we implement an algorithm that enumerates the simplexes of dimension at most k and filtration value at most l of the α-complex and their filtration values.

To some extent, the idea is the same as the one from task 3. However, while considering temporary simplexes in our backtracking function, when we add new points, we have to make sure that the simplex stays inside the complex.

Using the given visualization tool, we plot randomly generated α-complexes and Čech complexes.

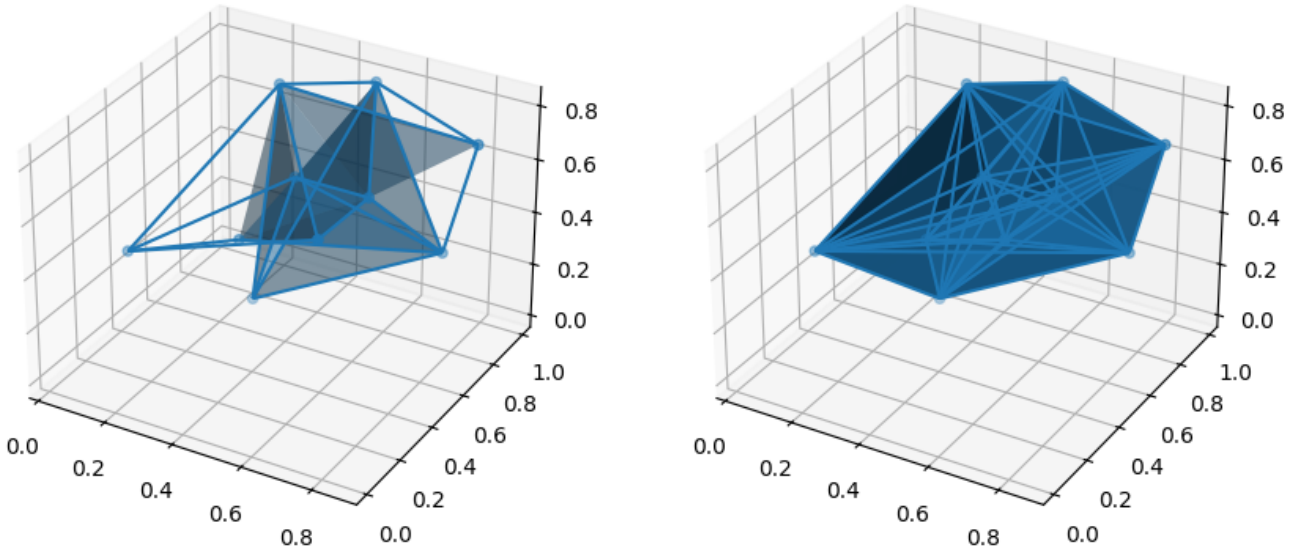

Figure 5: Examples of α-complex (on the left) and Čech complex (on the right) for $n = 10$, $d = 3$, $l_{\max} = 0.75$, $k = 5$

It is natural now to wonder how the number of simplexes evolves as k varies. Tests have yielded the following graphs:
As expected :

- The number of simplexes is stationary, as the more vertices a simplex has, the bigger is its filtration value. But we only take simplexes whose filtration value is at most 0.5.

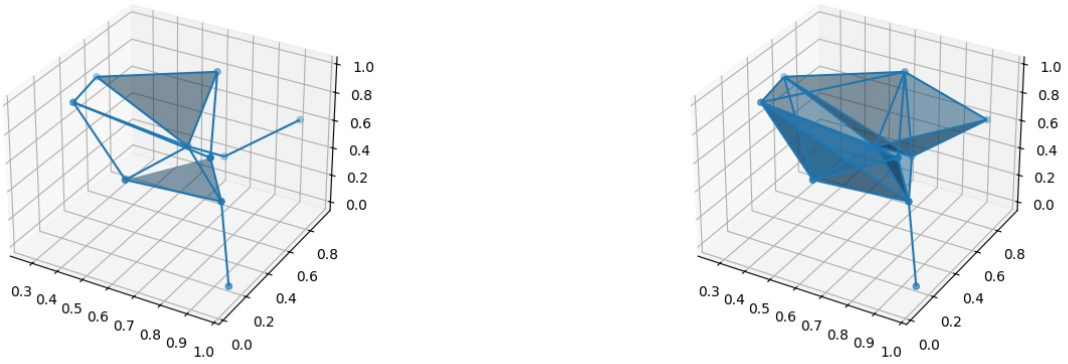

Figure 6: Examples of α-complex (on the left) and Čech complex (on the right) for $n = 10$, $d = 3$, $l_{\max} = 0.35$, $k = 5$

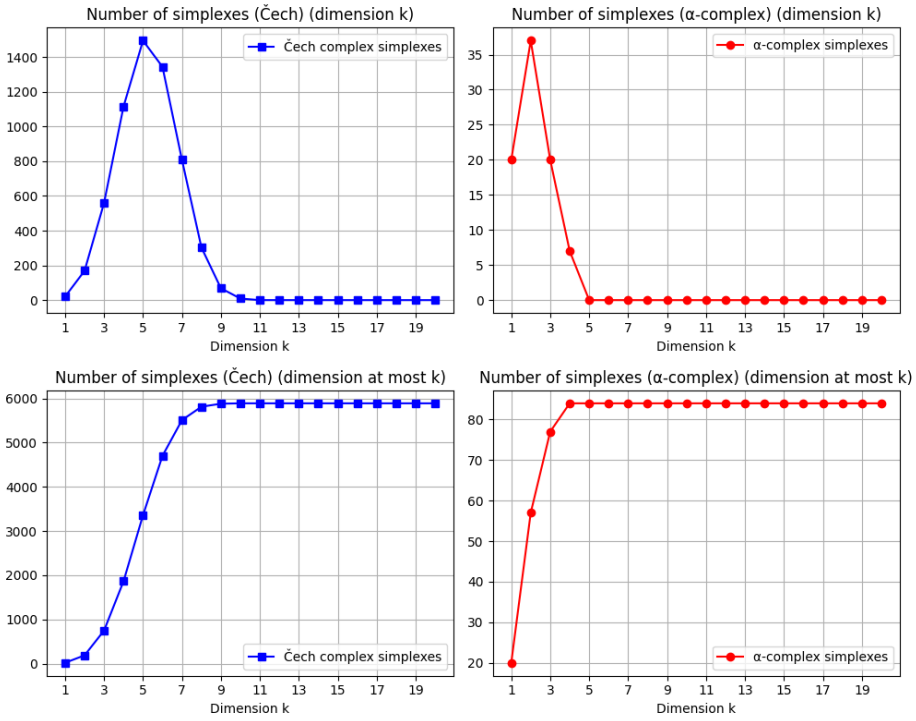

Figure 7: Evolution of the number of simplexes ($n = 20$,$l = 0.5$)

- The number of simplexes in the Čech complex is far larger than the number of simplexes belonging to the α-complex, which is a direct consequence of the definition of "belonging to the α-complex", which is way more restrictive.

We find the same kind of pattern as when the filtration value was fixed. As expected, the number of simplexes in the Čech complex is far larger than the number of simplexes in the α-complex.

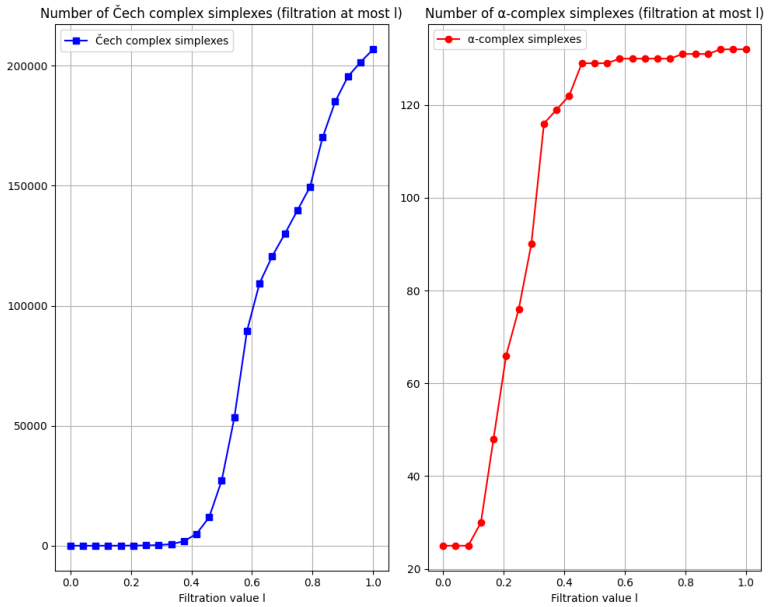

Figure 8: Evolution of the number of simplexes as l varies ($k = 5$, $n = 25$)

3 BONUS

In this section, we discuss the bonus tasks.

3.1 Bonus task 1

In this task, we use parallelism to compute filtration values more efficiently. More precisely, we compute all simplexes that are *potentially* part of the complex. After doing so, we compute the filtration values of these simplexes using parallelism (as computing the filtration value of each simplex is independent from one another).

But how can we find simplexes that are *potentially* part of the complex without computing their filtration values ? The key argument is that, given a set of points $\mathcal{P}$, the diameter of $\mathcal{P}$ divided by 2 is smaller than the filtration value of $\mathcal{P}$. Hence, during the backtracking, instead of comparing the filtration value of our temporary simplexes to l (which would be costly), we compare the diameter to $2l$ which we constantly update.
Whenever said diameter is larger than $2l$, we know that the filtration value must also be larger than l, thus we drop the temporary simplex.

Of course, this does not improve upon our complexity, but it does make the algorithm run faster for larger values of n.
We may notice that for small values of n, the simplexes function runs faster than parallel simplexes. Indeed, when n is small, computing MEBs is really efficient, hence parallelism does worse than the normal function. However, as n grows large, we can see that the parallel simplexes function runs a lot faster than the normal simplexes function.

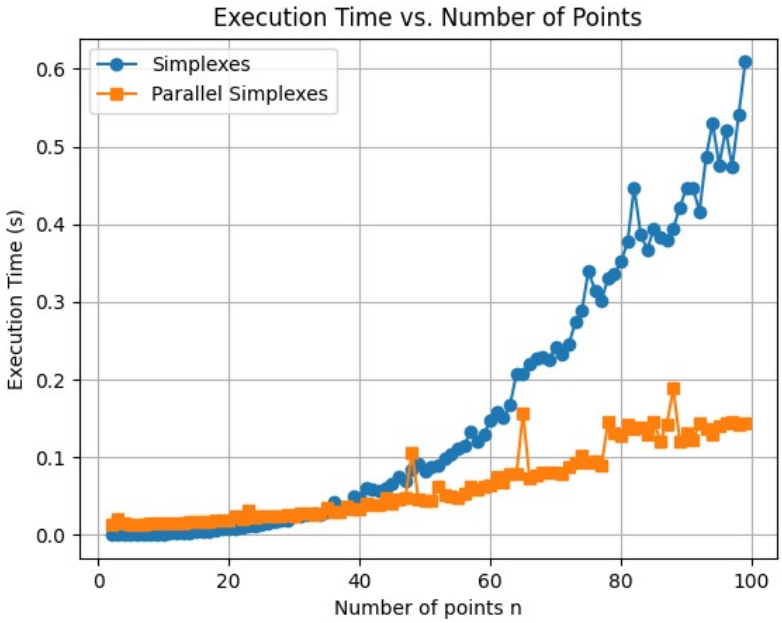

Figure 9: Simplexes vs Parallel simplexes

3.2 Bonus task 2

In this task, we notice some key arguments that allow us to speed things up for larger values of n.
More precisely, consider simplexes $\mathcal{S}$ that have a point p as vertex. The points of $\mathcal{P}$ that are at distance more than $2l$ of p cannot be part of the simplex $\mathcal{S}$. Furthermore, if they are at distance at most l of p, we know for a fact that they are part of $\mathcal{S}$. However, if they are at distance at least l and at most $2l$ of p, we have to check if they are or not part of $\mathcal{S}$. Given that many such points can exist and that checking if they are part of $\mathcal{S}$ or not can be done in independents steps, we use parallelism once more.

We use **scipy's KDTree** class to find points at a certain distance of p inside of $\mathcal{P}$.
As we can see for large values of n, while the initial version of the algorithm becomes hardly feasible in reasonable time, the KDTree optimization improves upon the parallel version of the algorithm.

Eventually, we plot hereafter the comparison between the optimized version of the function that enumerates the α-subsimplexes of a set of points and its naive version.

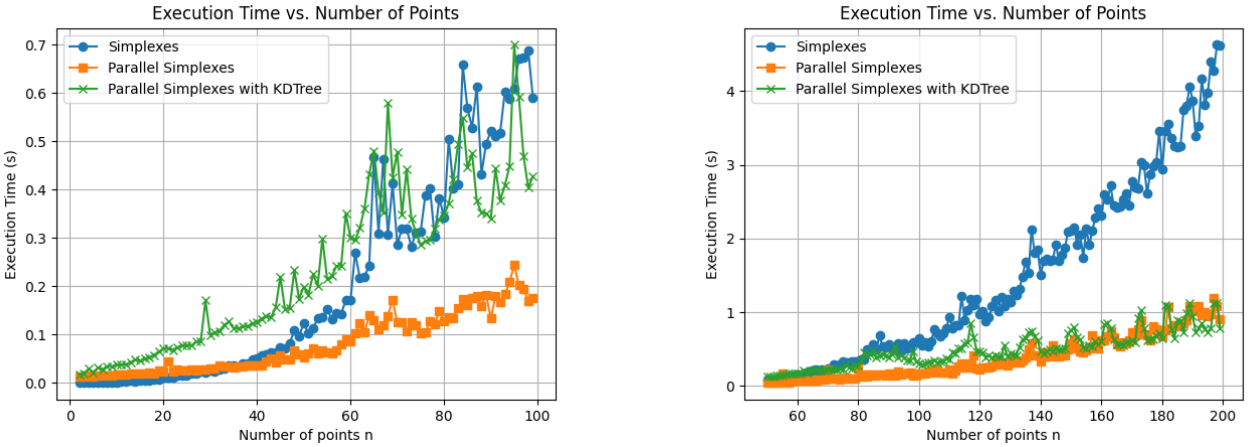

Figure 10: For small n (left), the KDTree optimization does poorly. For medium values (right) it improves.

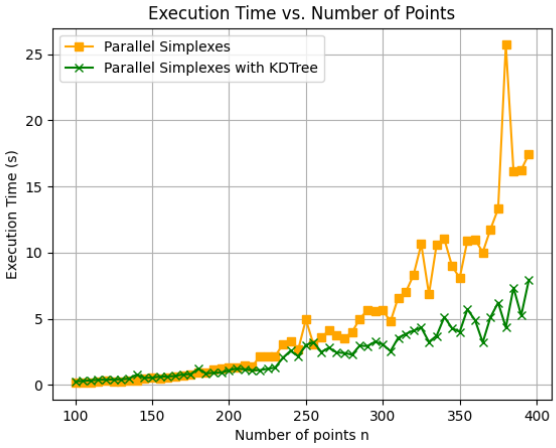

Figure 11: For large values of n, the KDTree optimization becomes much better

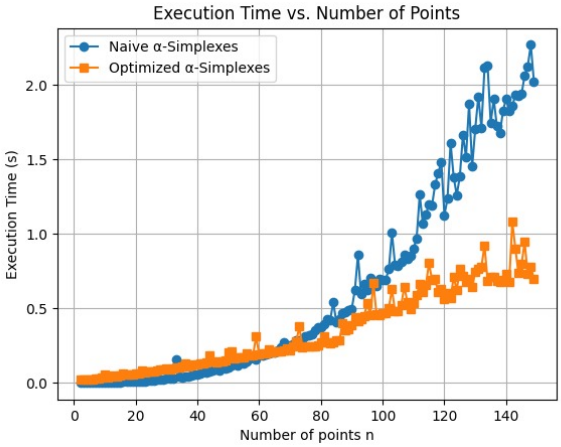

Figure 12: Optimization of our α-complex function