Implementation of an algorithm to calculate Čech complexes in networks of neurons

Àlex Tudoras

Motivation

Networks of neurons can be interpreted as mathematical objects to be accurately analyzed. We present a mathematical model taking into consideration a set of different presynaptic cells, each of which emit circular idealized synaptic fields to postsynaptic cells. In particular, any postsynaptic cell will only propagate the electric impulse if it receives a minimum amount of inputs from different presynaptic cells. In this context, we can translate the network of presynaptic neurons to a mathematical graph, with individual neurons as vertices and intersection of synaptic fields emerging from them as edges. Hence, the network can be analyzed using the techniques provided by topological data analysis. In fact, Čech complexes are able to accurately capture the topological features of our network, especially in terms of homology, and due to the lack of a `Python` public code to construct them, we present a series of functions to do so.

Introduction

The Čech complex connects points in P with a simplex if the radius of the minimal enclosing sphere of the subset is less than ϵ [1, 2]. Computationally, its asymptotic worst case for the number of maximal simplices is determined. Indeed, the maximal simplices of the Cech complex correspond to maximal enclosing spheres which can be defined by up to $d+1$ points on them. That is, the number of maximal simplices is bounded by a polynomial of order $d+1$. Thus, a divide and conquer algorithm for the efficient computation of the Čech complex for a given radius ϵ of the covering balls results in an efficient construction [3].

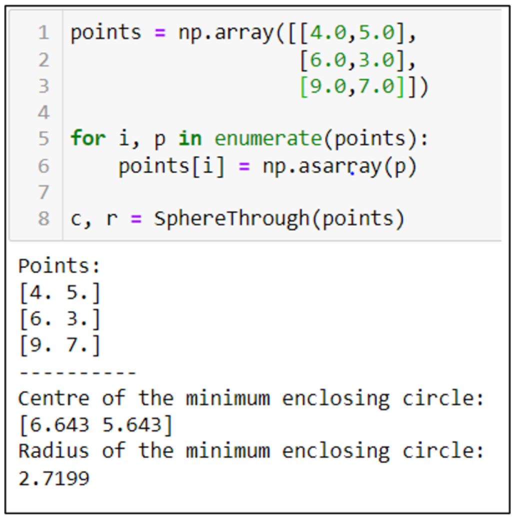
Construction

Note that we will use small letters to name a point, for example $p_i \in \mathbb{R}^d$, and capital letters to name sets of points. Let's consider a finite set P of n points $p_1, \dots, p_n \in \mathbb{R}^d$. We will name $B_\epsilon(p_i) = \{x \in \mathbb{R}^d \mid d(p,x) < \epsilon\}$ the open ball of radius ϵ centered around p_i. Then, we consider an open cover of P as the union of all $B_\epsilon(p)$, $\forall p \in P$. Hence, the Čech complex is $C_\epsilon(P) = \{S \subseteq P \mid \cap_{p \in S} B_\epsilon(p) \neq \varnothing\}$. Equivalently, we consider that $S \in C_\epsilon(P) \iff \exists q \in \mathbb{R}^d$ such that $S \in B_\epsilon(q)$.

Minimal closing sphere

We can find if a set S forms a simplex by calculating the unique minimal radius sphere that contains all the points in S, as `[c,r] := MinSphere(S)`.

This is a classical geometry problem that has been studied for centuries. The solution is unique for $d+1$ points in $\mathbb{R}^d$, since it is the unique d-dimensional sphere that passes through those points. In particular, if we are dealing with points in $\mathbb{R}^2$, we are able to trivially calculate the circle that passes through the set of 3 points $P = \{p_0, p_1, p_2\}$ with the function `[c,r] := SphereThrough(P)`.



```
points = np.array([[4.0,5.0],
                   [6.0,3.0],
                   [9.0,7.0]])

for i, p in enumerate(points):
    points[i] = np.asarray(p)

c, r = SphereThrough(points)
```

```
Points:
[4. 5.]
[6. 3.]
[9. 7.]
----------
Centre of the minimum enclosing circle:
[6.643 5.643]
Radius of the minimum enclosing circle:
2.7199
```

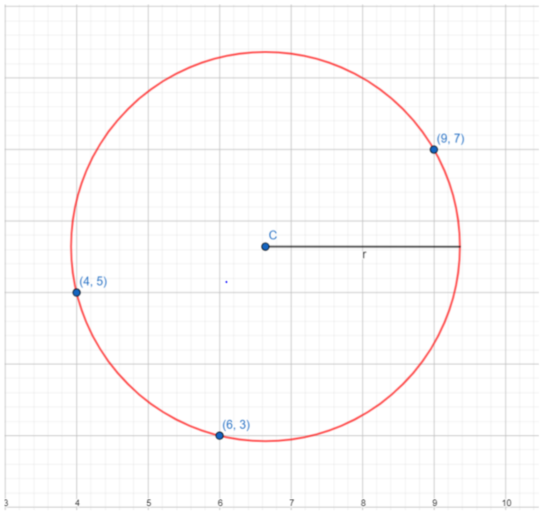
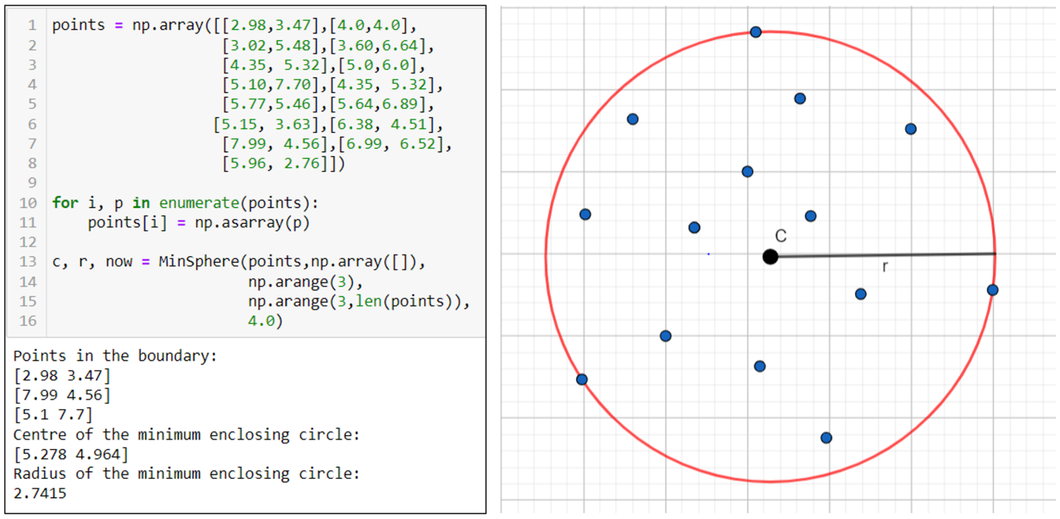


Figure 1: Example of `SphereThrough` to calculate the circle through a set of three points in the plane.

Additionally, as advised by Dantchev and Ivrissimtzis [3], we implement the solution described by [4] in a Python function to calculate, for points in $\mathbb{R}^2$, the smallest circle that encloses a set of points with a function `[c,r,boundary] := MinSphere(S)`.



```
points = np.array([[2.98,3.47],[4.0,4.0],
                   [3.02,5.48],[3.60,6.64],
                   [4.35, 5.32],[5.0,6.0],
                   [5.10,7.70],[4.35, 5.32],
                   [5.77,5.46],[5.64,6.89],
                  [5.15, 3.63],[6.38, 4.51],
                   [7.99, 4.56],[6.99, 6.52],
                   [5.96, 2.76]])

for i, p in enumerate(points):
    points[i] = np.asarray(p)

c, r, now = MinSphere(points,np.array([]),
                      np.arange(3),
                      np.arange(3,len(points)),
                      4.0)
```

```
Points in the boundary:
[2.98 3.47]
[7.99 4.56]
[5.1 7.7]
Centre of the minimum enclosing circle:
[5.278 4.964]
Radius of the minimum enclosing circle:
2.7415
```

Figure 2: Example of `MinSphere` to calculate the smallest circle that encloses a set of points in the plane.

Maximal simplices

In fact, we only need the maximal simplices to construct the Čech complex efficiently. Those will be the ones that cannot be extended to a valid simplex (according to our maximum edge length) by adding an extra point. For $Q \subseteq P$ we can check if it consistutes a maximal simplex `IsMaxSimplex(Q)` by cehcking that `MinSphere(Q)` has a radius $r_Q < \epsilon$ and, then, $\forall p_i \in P$, $p_i \notin Q$, the minimum sphere containing $Q \cup p_i$ has radius $r_i > \epsilon$.

In order to test the accuracy of the algorithm, we define a set of points in the plane as in the following image, and set a maximum radius of $r = 1.5$, which can be translated to the maximum extension of the synaptic field emitted by each of the cells that correspond to the points.

In our code, the relation between vertices and coordinates is given as $v_0 = [0,1]$, $v_1 = [1,0]$, $v_2 = [1,2]$, $v_3 = [2,1]$, $v_4 = [5,1]$, $v_5 = [6,2]$, $v_6 = [6,0]$. Then, by looking at the extension of each field and the intersections, we expect our algorithm to return `True` only in the three cases with the vertices' indices $\{0,1,2,3\}$, $\{3,4\}$ and $\{4,5,6\}$. Every other combination results in a simplex that is not maximal or in a combination of points that are too far from each other so that their fields have a common intersection.

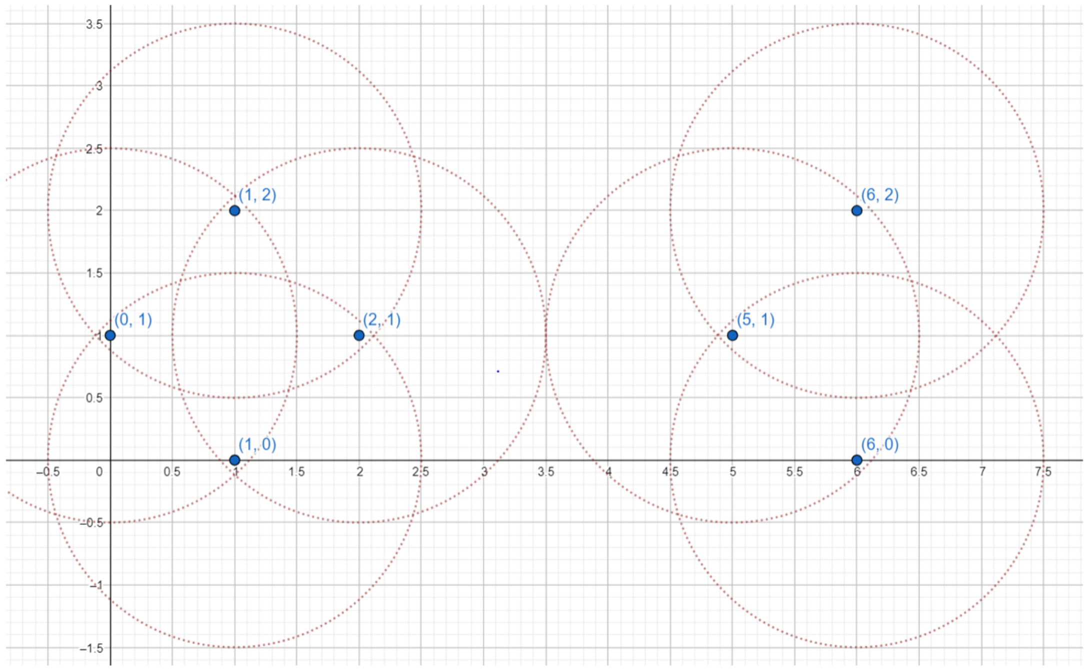
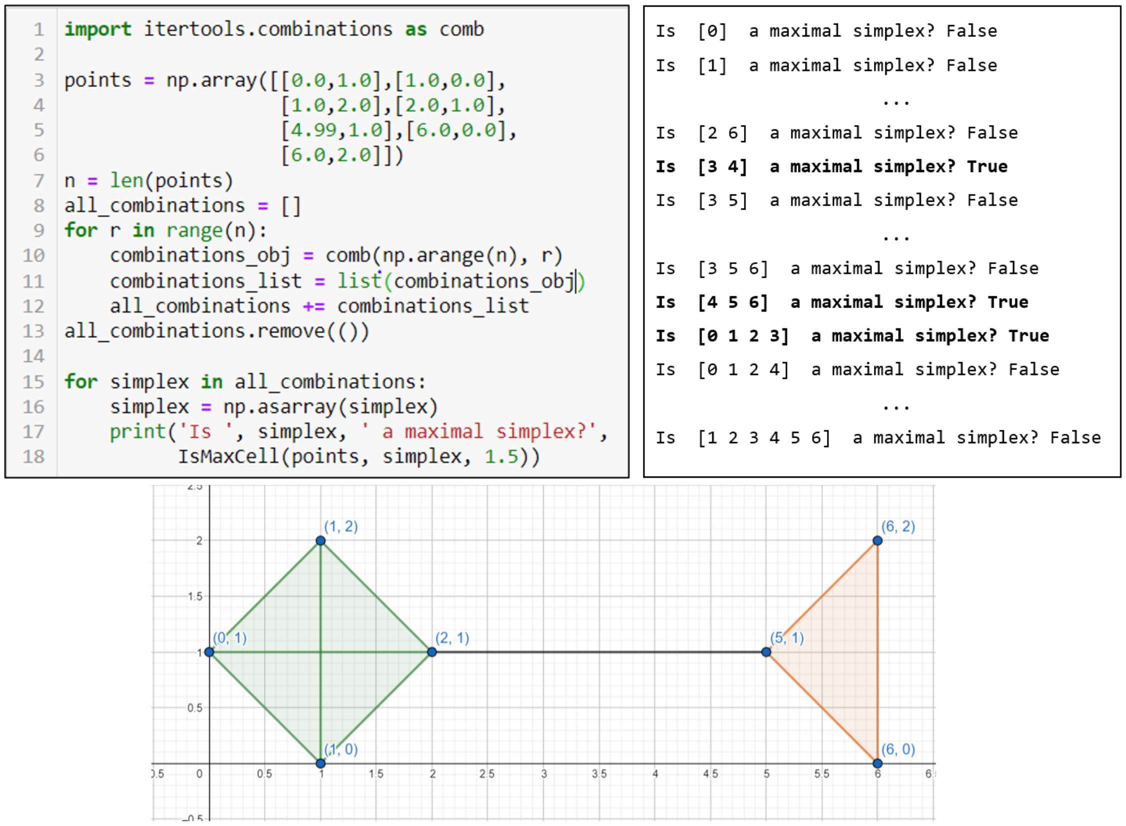


Figure 3: Set of points in the Euclidean plane to calculate the Čech complex formed by them.

At the present moment, this function is not fully optimized for large amounts of points, but is effective for sets of the order of 100 points.



```python
import itertools.combinations as comb

points = np.array([[0.0,1.0],[1.0,0.0],
                   [1.0,2.0],[2.0,1.0],
                   [4.99,1.0],[6.0,0.0],
                   [6.0,2.0]])
n = len(points)
all_combinations = []
for r in range(n):
    combinations_obj = comb(np.arange(n), r)
    combinations_list = list(combinations_obj)
    all_combinations += combinations_list
all_combinations.remove(())

for simplex in all_combinations:
    simplex = np.asarray(simplex)
    print('Is ', simplex, ' a maximal simplex?',
          IsMaxCell(points, simplex, 1.5))
```

```
Is  [0]  a maximal simplex? False
Is  [1]  a maximal simplex? False
                   ...
Is  [2 6]  a maximal simplex? False
Is  [3 4]  a maximal simplex? True
Is  [3 5]  a maximal simplex? False
                   ...
Is  [3 5 6]  a maximal simplex? False
Is  [4 5 6]  a maximal simplex? True
Is  [0 1 2 3]  a maximal simplex? True
Is  [0 1 2 4]  a maximal simplex? False
                   ...
Is  [1 2 3 4 5 6]  a maximal simplex? False
```

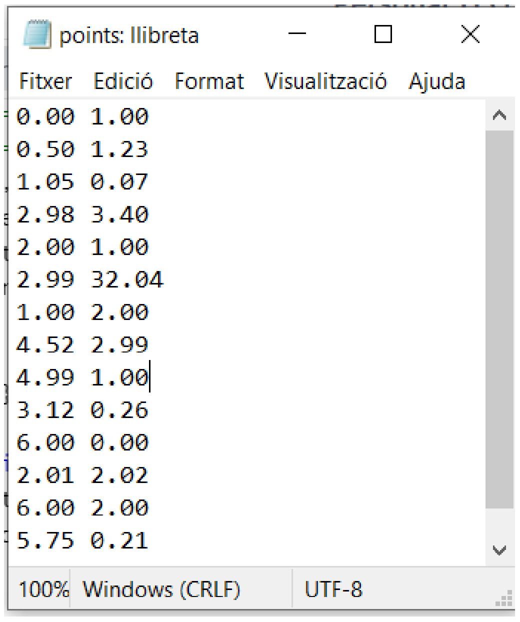
Figure 4: Input, output and graphical representation of testing all the combinations of a set of points in the function `IsMaxSimplex`.

Tutorial

In order to find the Čech complex given a set of coordinates in the plane with our functions, we need to follow the instructions as described in this section.

- Upload a file `points.txt` with the set of points in the following format

- Upload a file `parameters.txt` only containing, in each row:
  - Number with the maximum extension of the synaptic field, i.e., the radius of the circles around each point in the space.
  - Number with the minimum simplex to be highlighted in the geometrical representation
  - Number with the maximum size of the simplices to be calculated, this parameter is important because we need to limit the size of the simplices to be calculated in order to have a reasonable time of computation.


points: llibreta
Fitxer Edició Format Visualització Ajuda
100% Windows (CRLF) UTF-8

```
0.00 1.00
0.50 1.23
1.05 0.07
2.98 3.40
2.00 1.00
2.99 32.04
1.00 2.00
4.52 2.99
4.99 1.00
3.12 0.26
6.00 0.00
2.01 2.02
6.00 2.00
5.75 0.21
```

Figure 5: Example of the input file `points.txt`.

- Run the code `Cech_complex_from_cloud_points_file.py`

As an output, you will get a `Results` folder with the following files.

- `Result_SC.txt`: Text file containing all the simplices and general information about the dimensions of the generated simplicial complex
- `Geometrical_SC.png`: An image with the geometrical representation of the resulting simplicial complex, generated with tools from the `Gudhi` and `Networkx` libraries
- `Persistence_barcode.png`: An image with the persistence barcode calculated with the library `Gudhi`