Υπολογιστική Γεωμετρία
Πρώτη Εργασία

Σιώρος Βασίλειος - 1115201500144
Ανδρινοπούλου Χριστίνα - 1115201500006

Ιούνιος 2020

1. Compute Voronoi diagrams of different sets of vertices of your choice using the routine *Voronoi* (and its companion *voronoi_plot_2d* for visualization) from the module scipy.spatial. Plot your results.

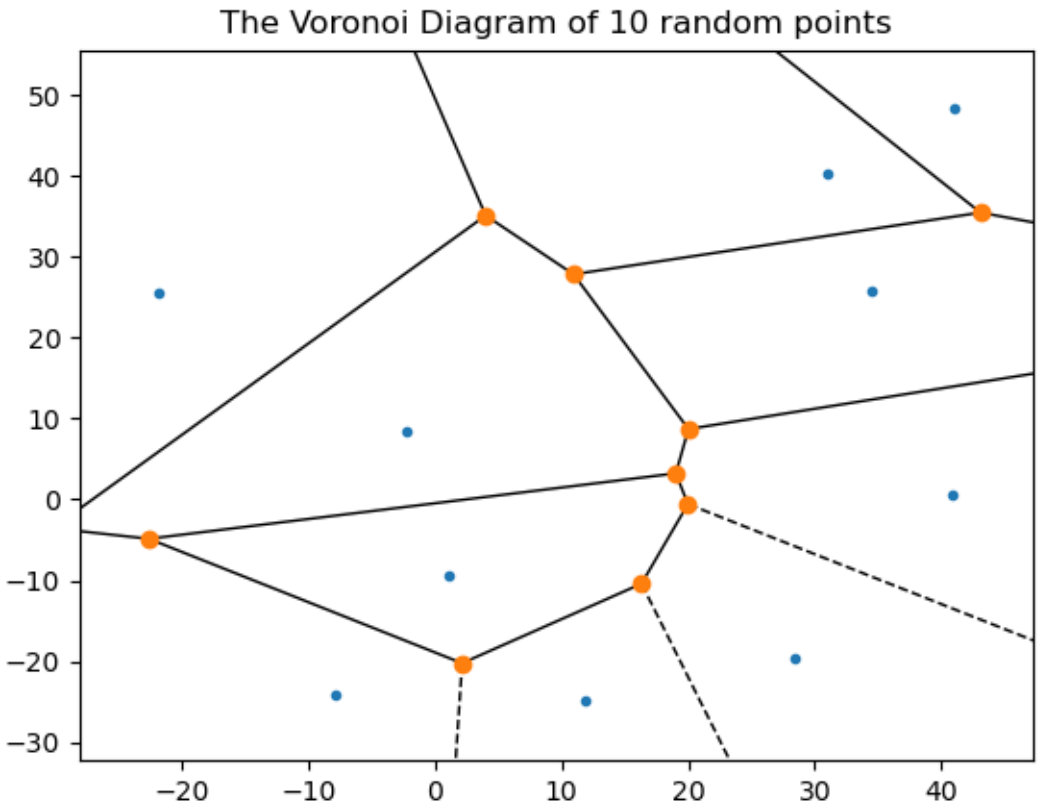

Figure 1: The Voronoi diagram of 10 random points with a random seed of 0

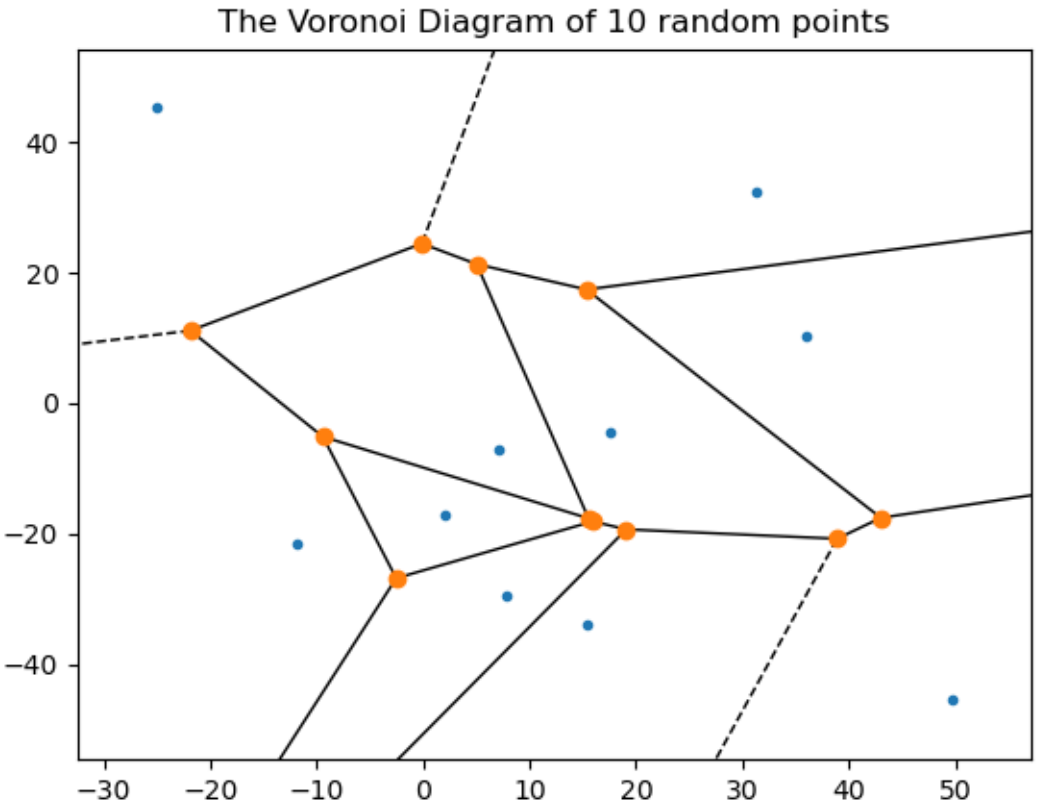

Figure 2: The Voronoi diagram of 10 random points with a random seed of 10

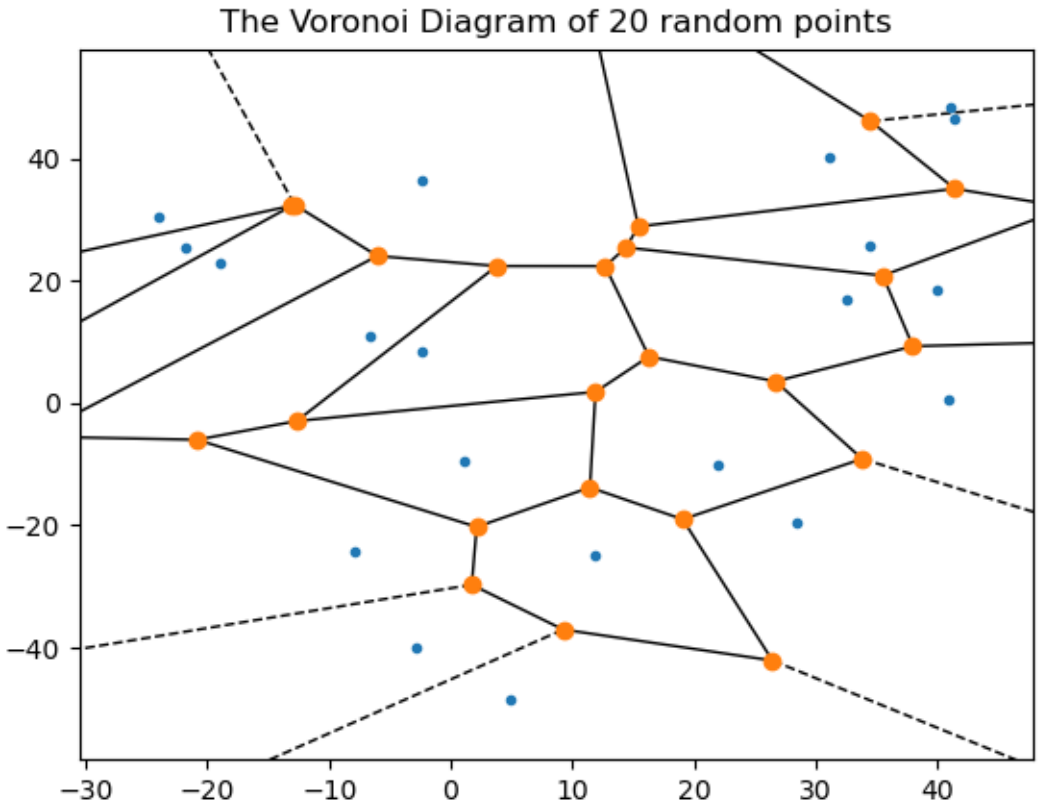

Figure 3: The Voronoi diagram of 20 random points with a random seed of 0

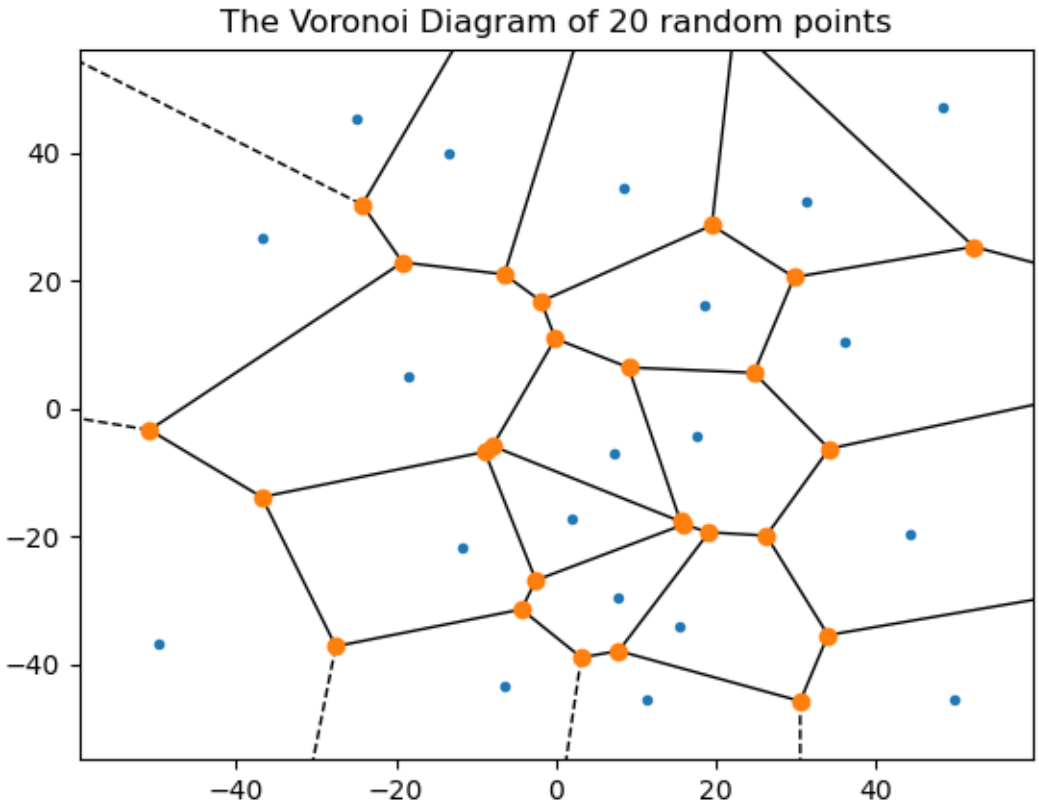

Figure 4: The Voronoi diagram of 20 random points with a random seed of 10

2. Using the routine *Delaunay* in the module scipy.spatial compute the Delaunay triangulation of different sets of vertices of your choice and plot your results.

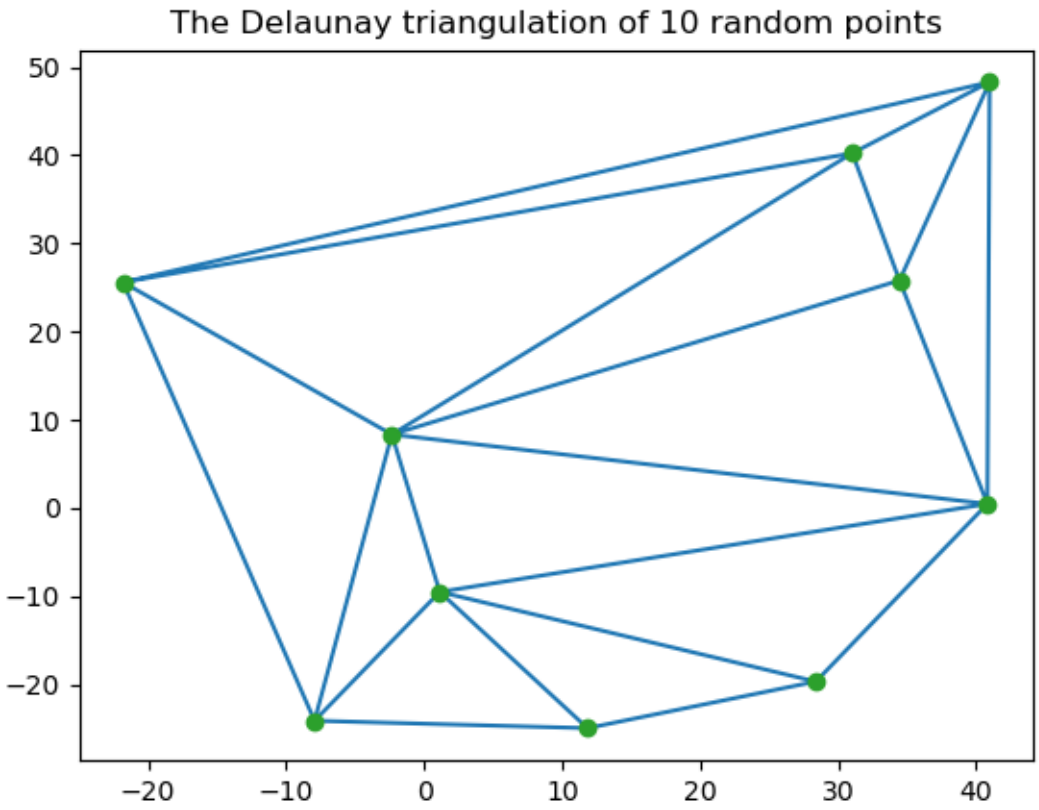

Figure 5: The Delaunay triangulation of 10 random points with a random seed of 0

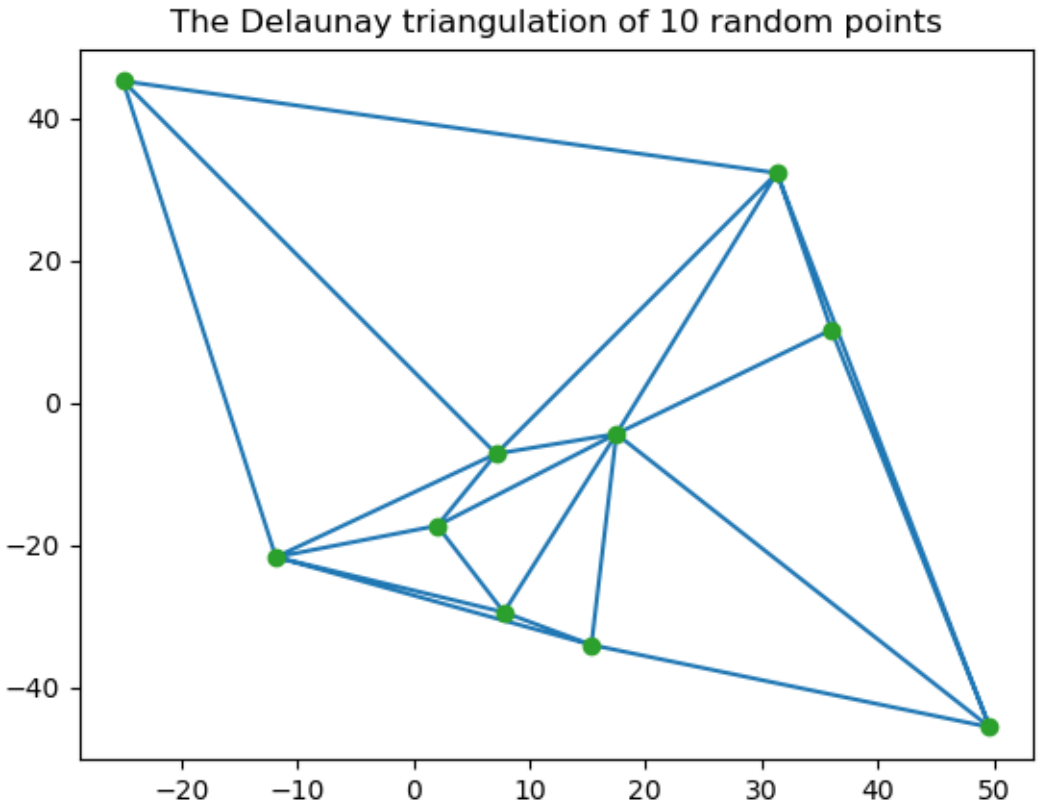

Figure 6: The Delaunay triangulation of 10 random points with a random seed of 10

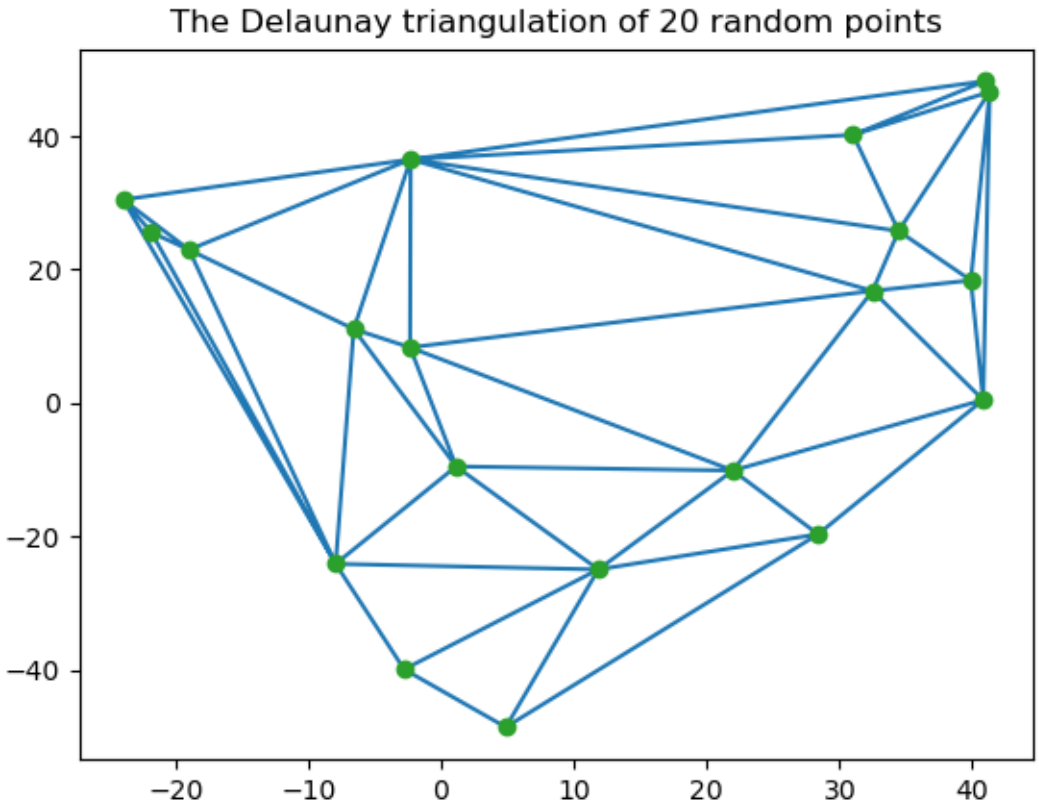

Figure 7: The Delaunay triangulation of 20 random points with a random seed of 0

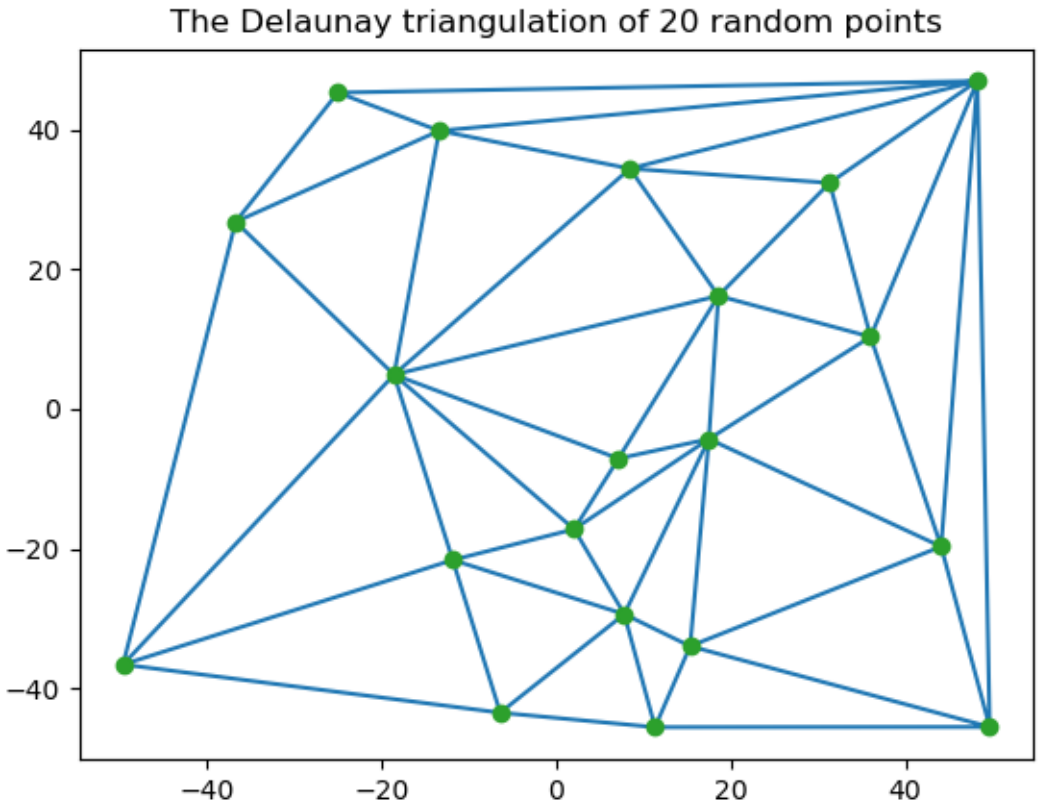

Figure 8: The Delaunay triangulation of 20 random points with a random seed of 10

3. Compute the shortest path of different set of vertices of your choice in a tri-angulation. By a path in this setting, we mean a chain of edges of this triangulation. Use the methods in the package scipy.sparse.csgraph.

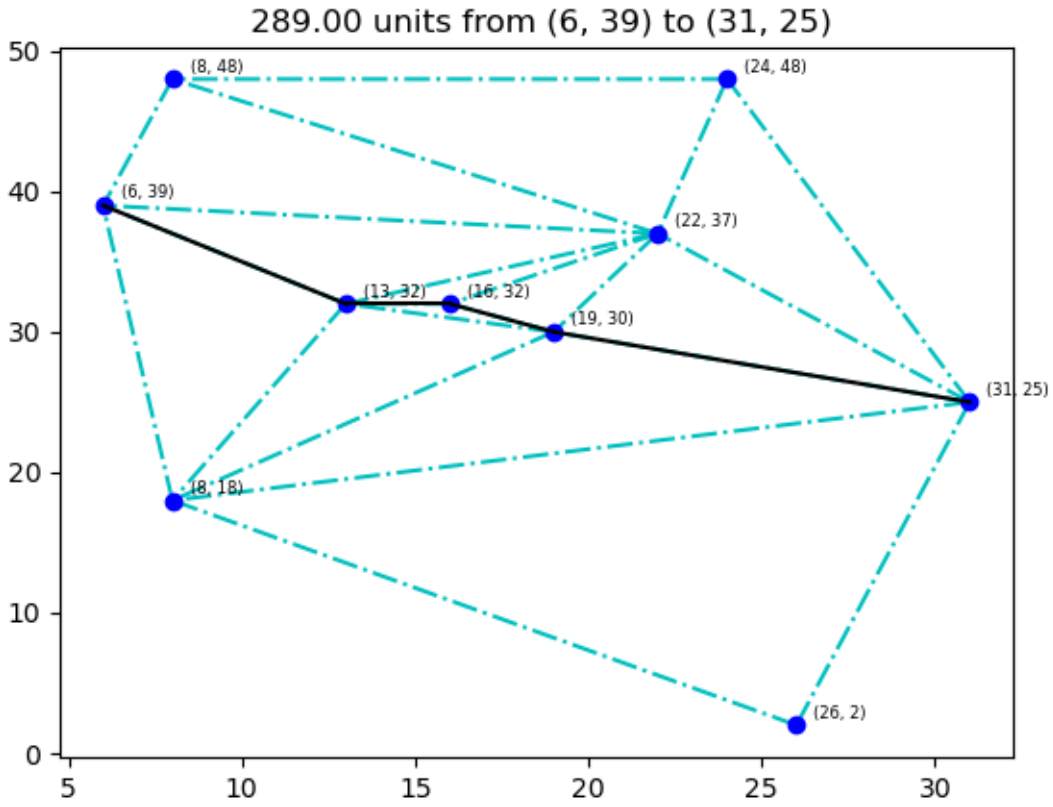

Figure 9: One of the shortest paths in a triangulation of 10 random points with a random seed of 0

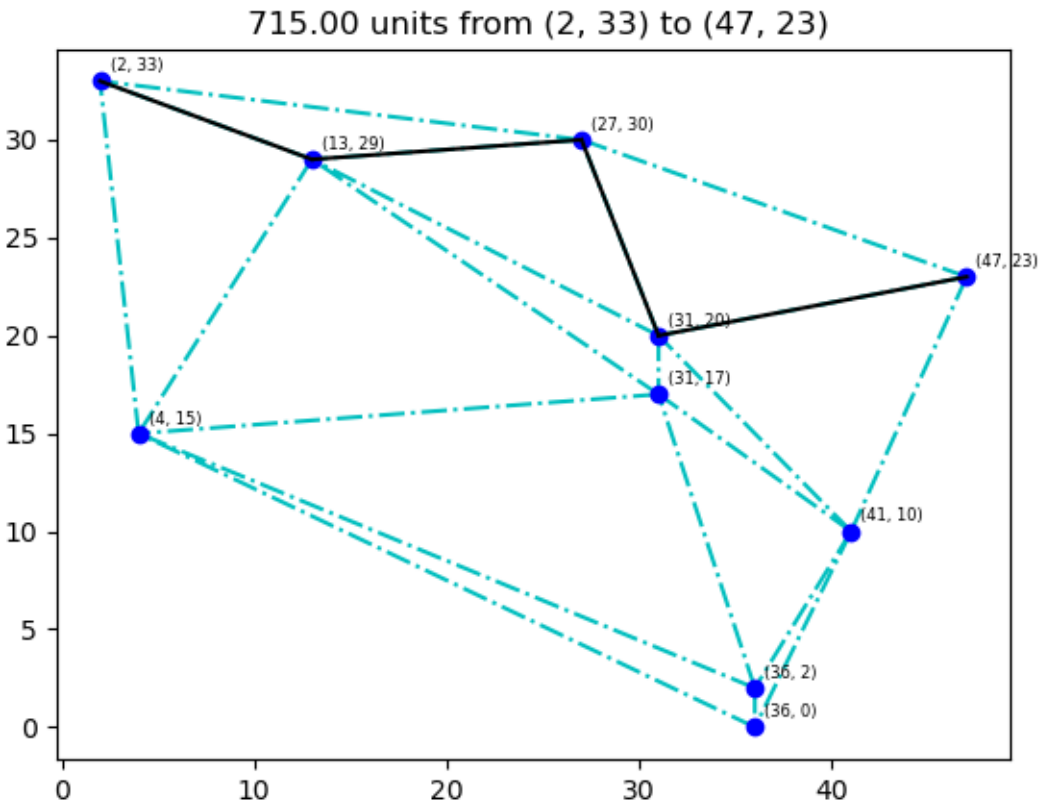

Figure 10: One of the shortest paths in a triangulation of 10 random points with a random seed of 10

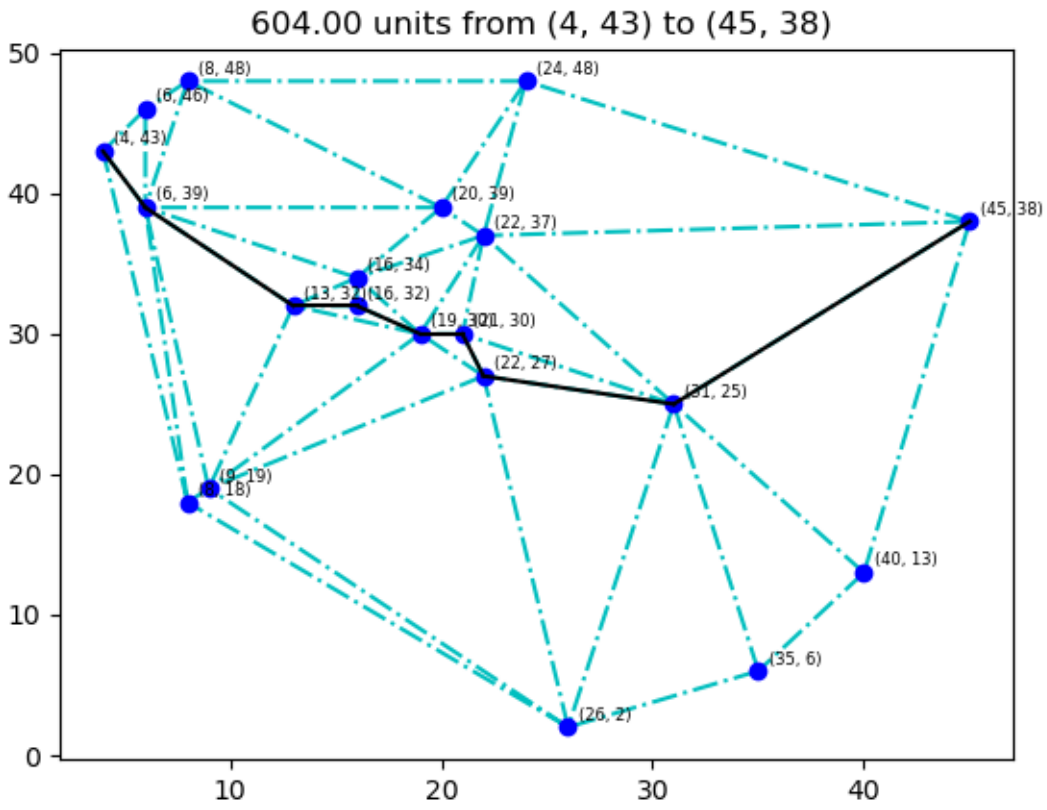

Figure 11: One of the shortest paths in a triangulation of 20 random points with a random seed of 0

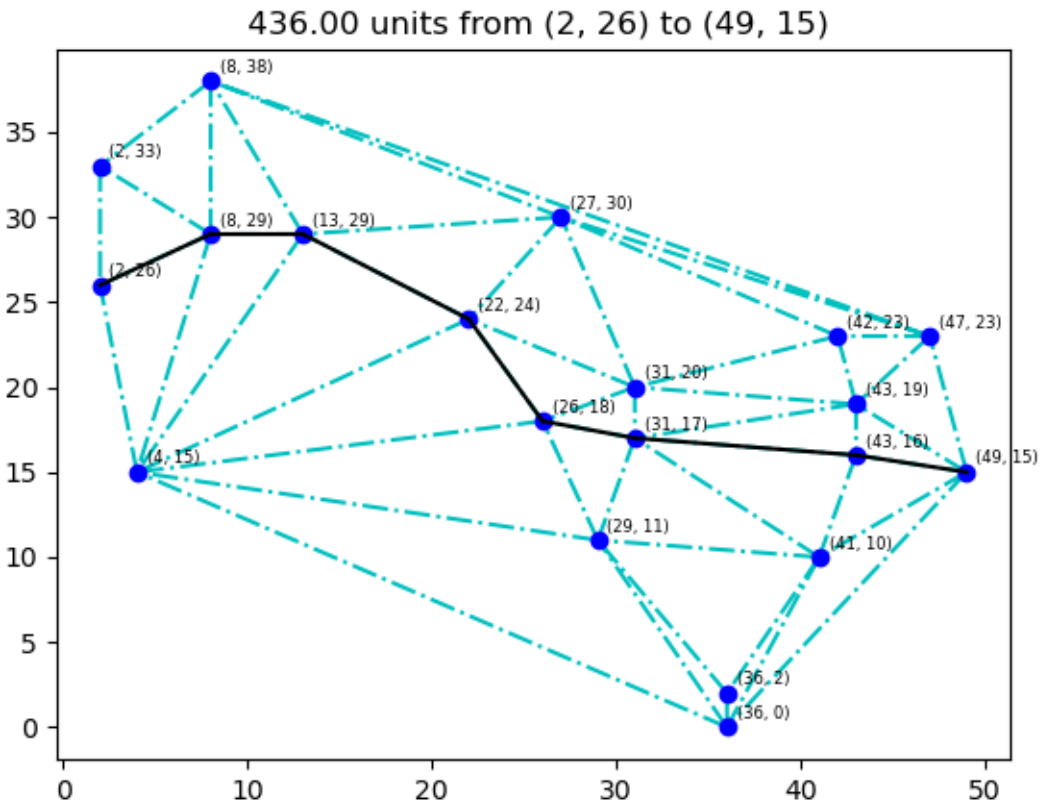

Figure 12: One of the shortest paths in a triangulation of 20 random points with a random seed of 10

4. Experiment yourself with the *.encloses_point* and *.encloses* methods of the sympy.geometry module using polygons or circles to check if they contain certain points of your choice. Do the same with *contains_point* or *contains* points from the Path class from the libraries of matplotlib.path.

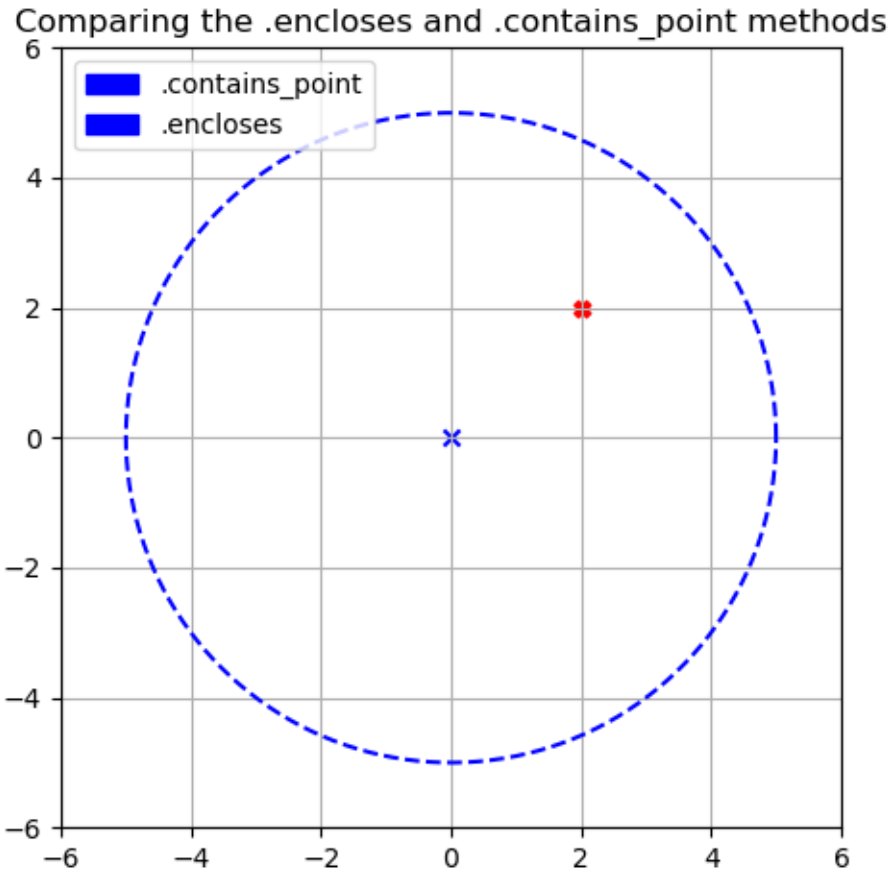

Figure 13: Using the *encloses* method

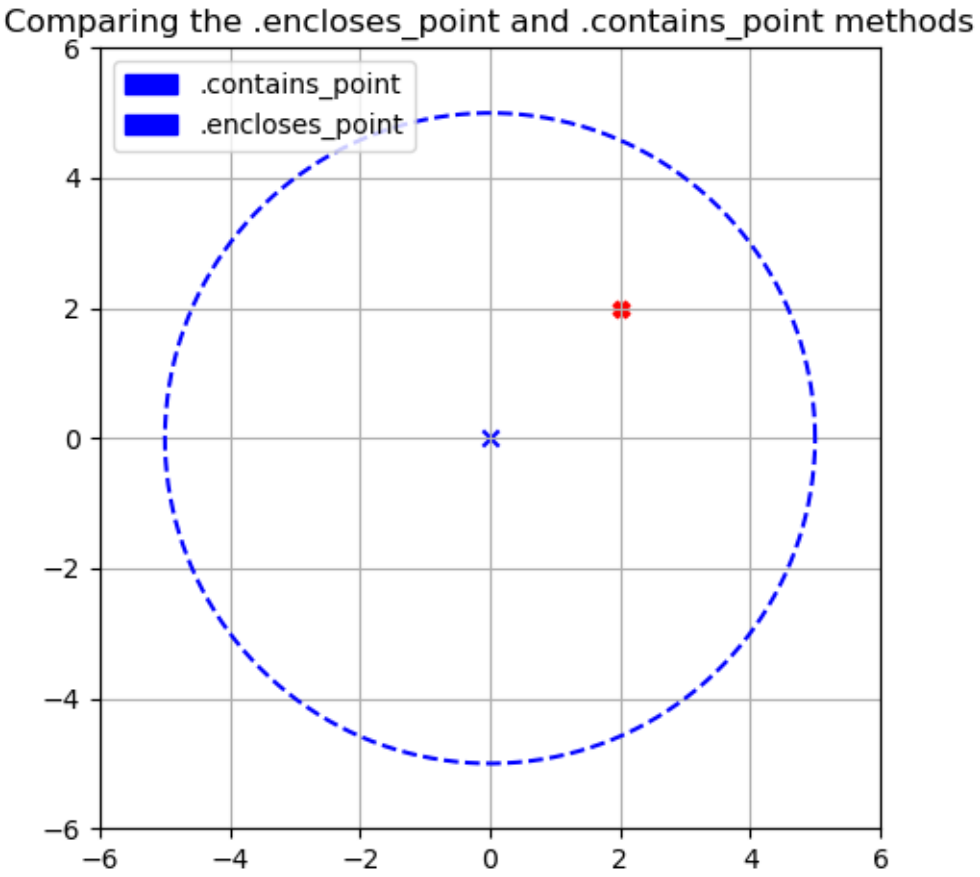

Figure 14: Using the *encloses_point* method

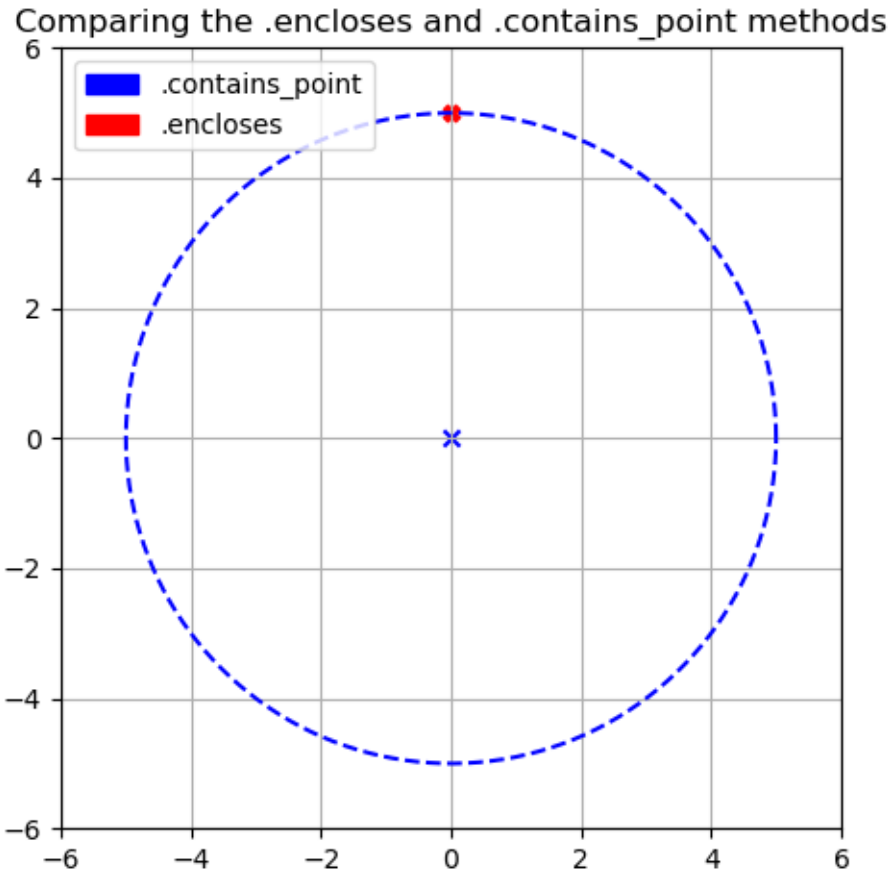

Figure 15: Comprtin the *encloses* and *contains* methods

5. The problem of finding the Voronoi cell that contains a given location is equivalent to the search for the nearest neighbor. We can always perform this search with a brute force algorithm, but in general there are more elegant and less complex approaches to this problem like the kd-trees. In the scipy use the class *KDTree* to perform some experiments of your choice.

Δεδομένου ενός σημείων ορίζεται το αντίστοιχο Voronoi διάγραμμα.

Θα επιχειρήσουμε να υπολογίσουμε τα Voronoi διαγράμματα διάφορων συνόλων σημείων στον δισδιάστατο χώρο, με τη βοήθεια του αλγορίθμου K κοντινότερων γειτόνων.

Δεδομένου ενός συνόλου εκπαίδευσης P, του οποίου το Voronoi διάγραμμα επιθυμούμε να υπολογίσουμε, ορίζουμε το meshgrid το οποίο ορίζεται από τα σημεία

$$(\min_{\forall p \in P} p.x), \min_{\forall p \in P} p.y))$$

$$(\max_{\forall p \in P} p.x), \max_{\forall p \in P} p.y))$$

Αφού εκπαιδεύσουμε το μοντέλο μας, κατηγοριοποιούμε κάθε σημείο που ανήκει στο meshgrid, έτσι ώστε να προκύψει το Voronoi διάγραμμα.

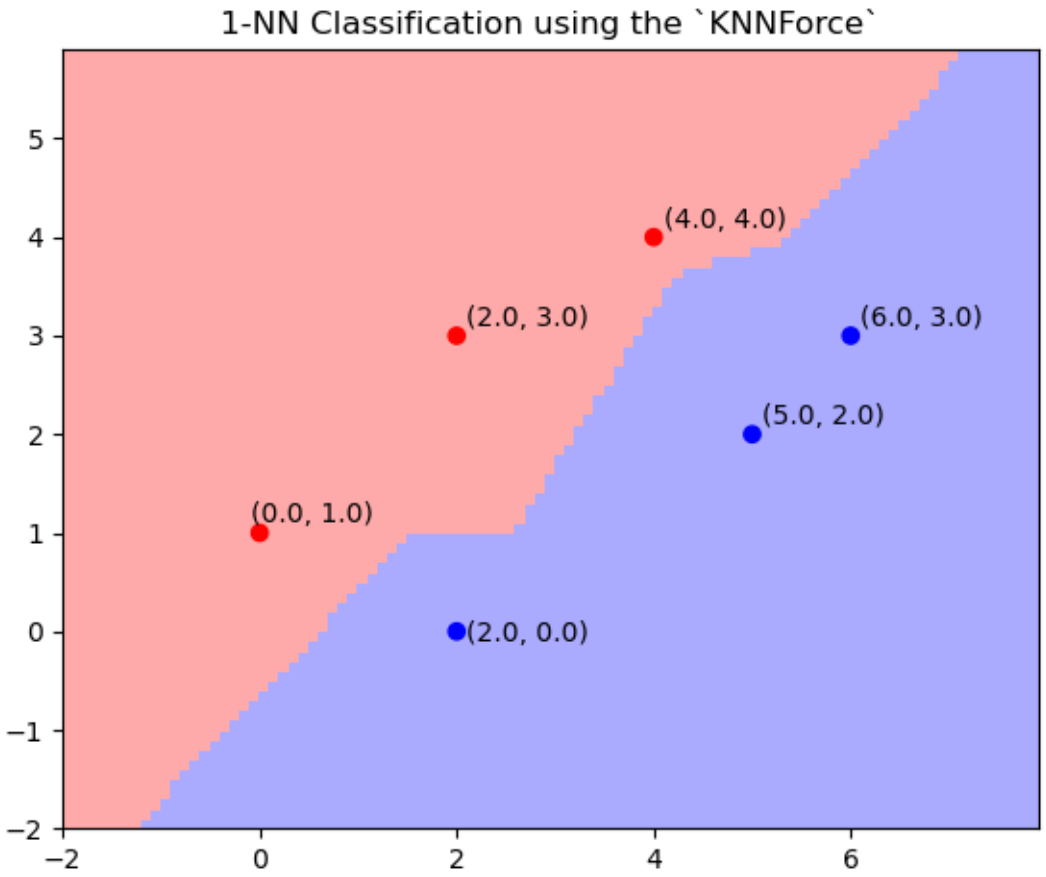

Figure 16: Using brute force in order to find the single closest neighbor of 80 points

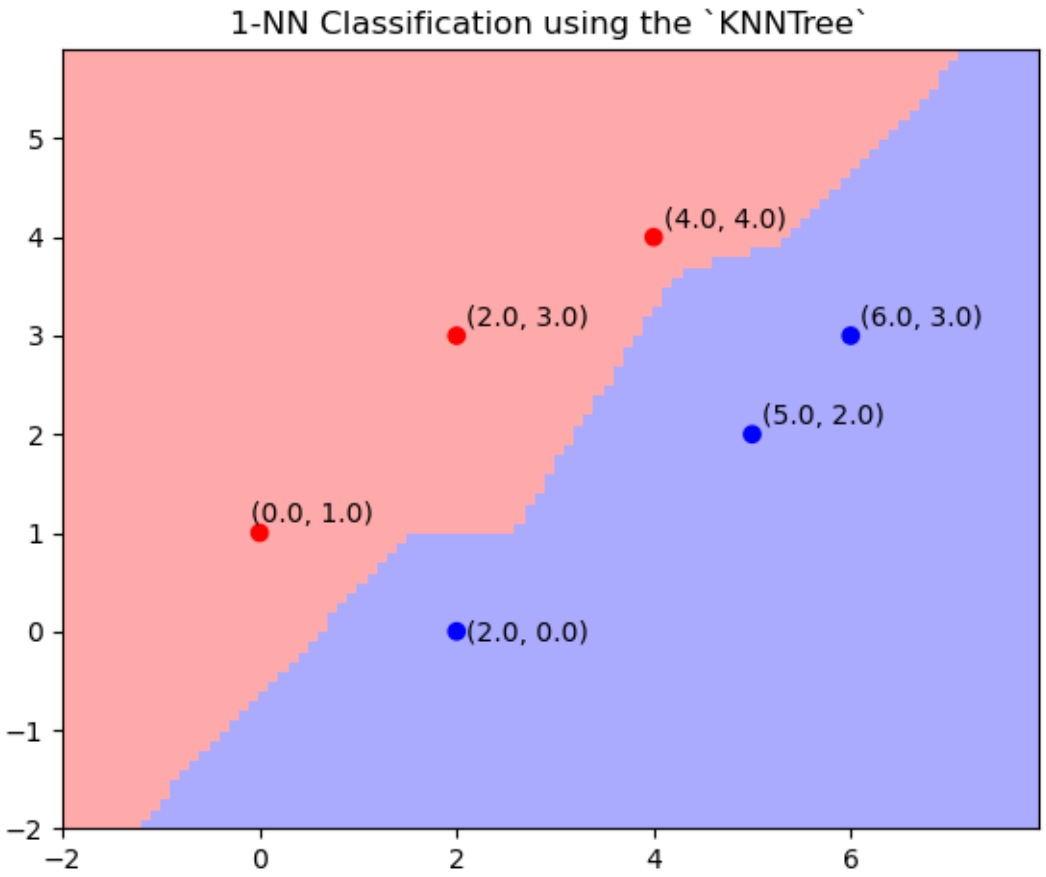

Figure 17: Using KD-Tree neighborhood look-up in order to find the single closest neighbor of 80 points

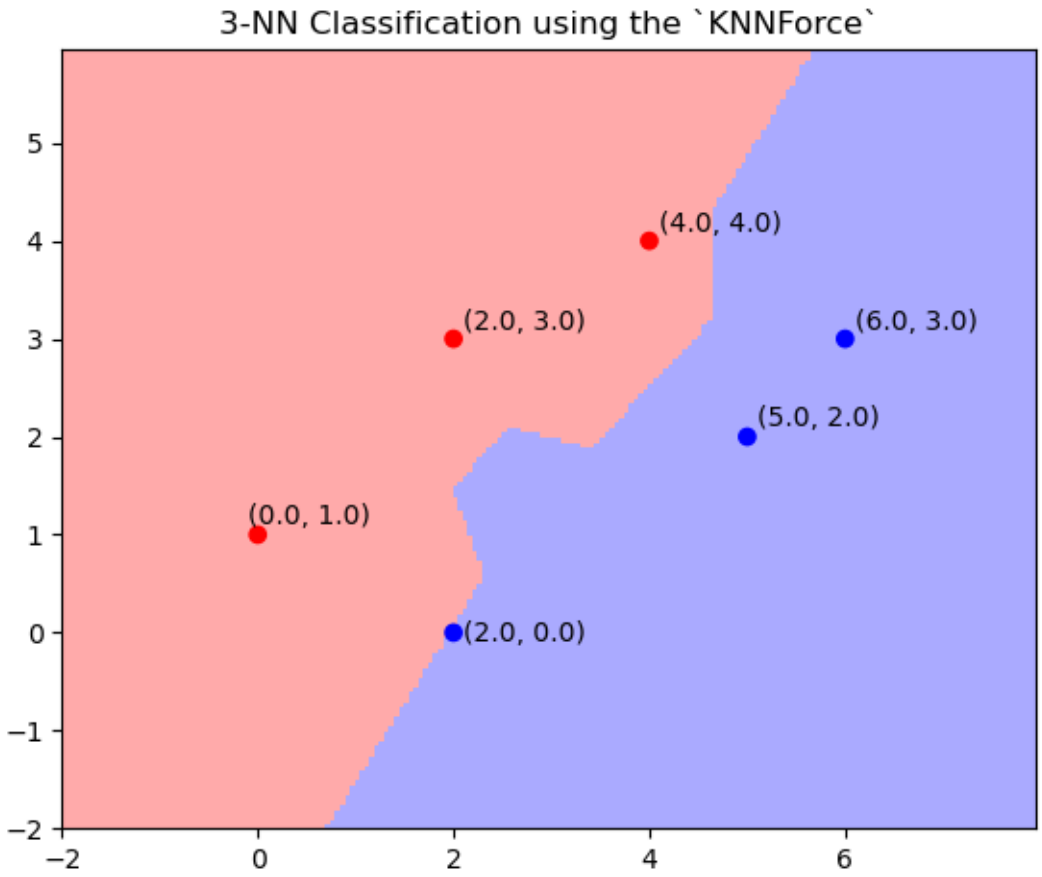

Figure 18: Using brute force in order to find the 3 closest neighbors of 160 points

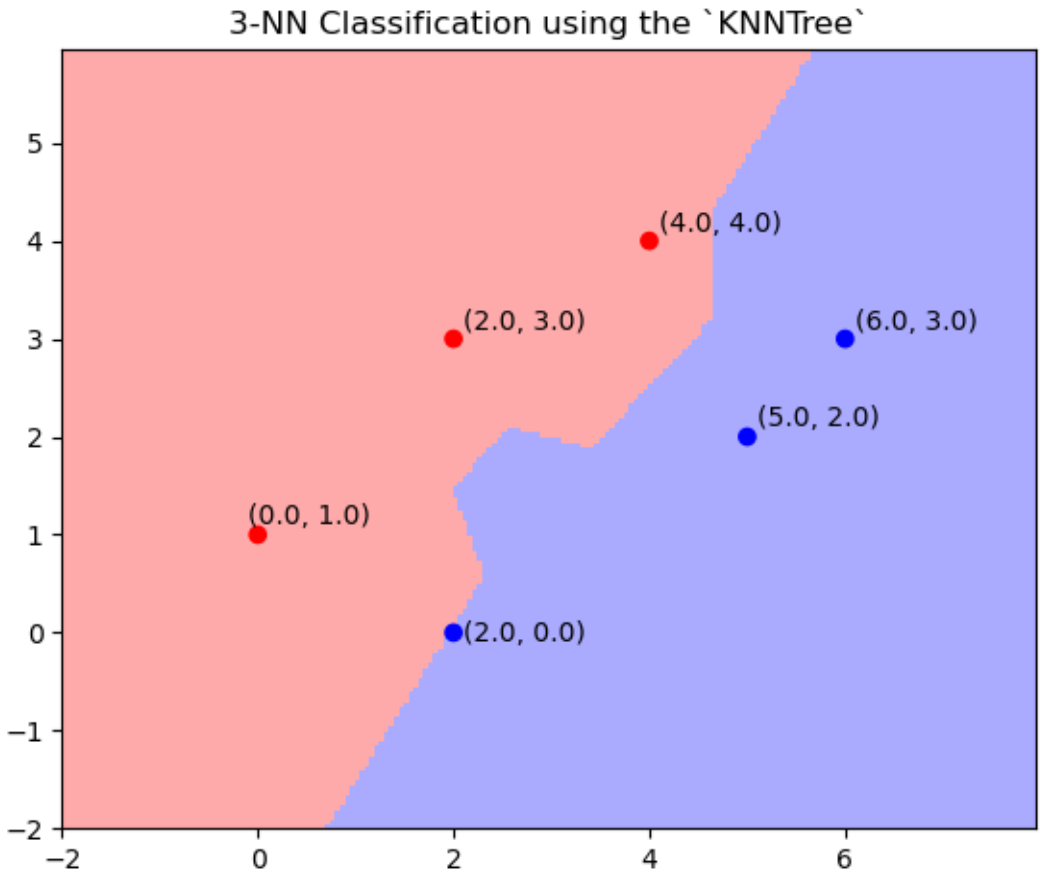

Figure 19: Using KD-Tree neighborhood look-up in order to find the 3 closest neighbors of 160 points

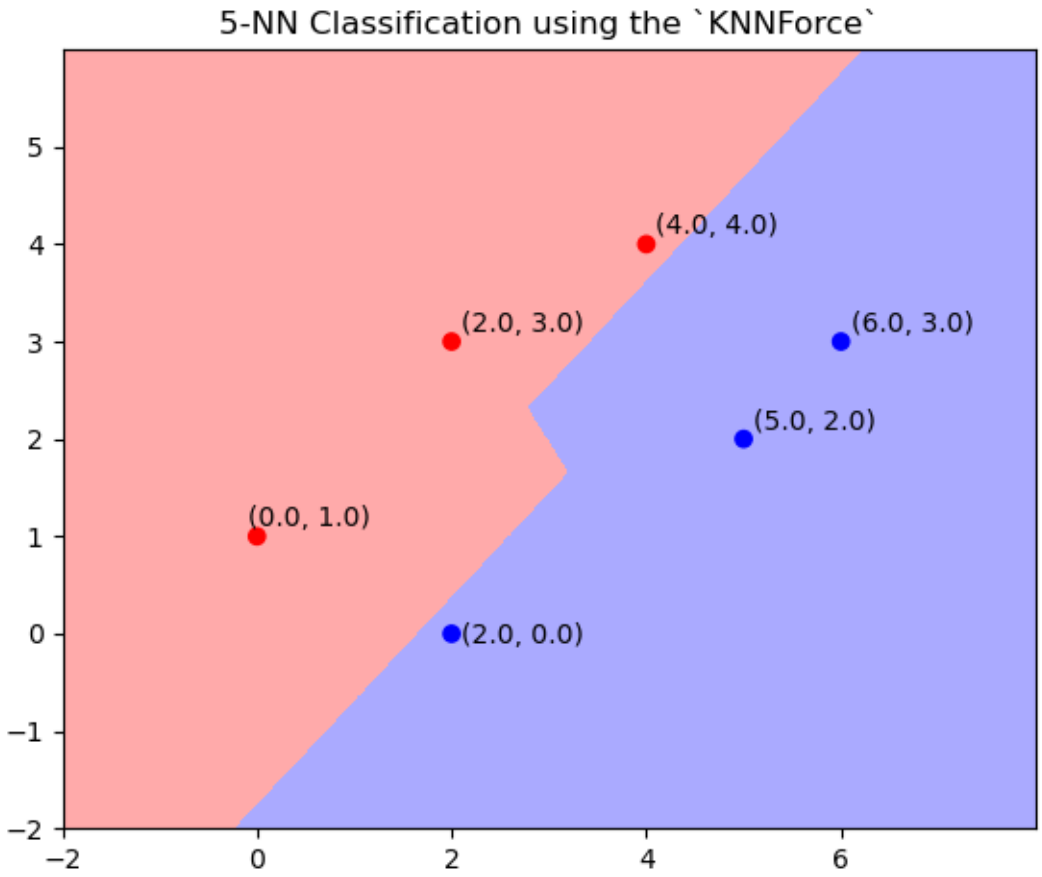

Figure 20: Using brute force in order to find the 5 closest neighbors of 1600 points

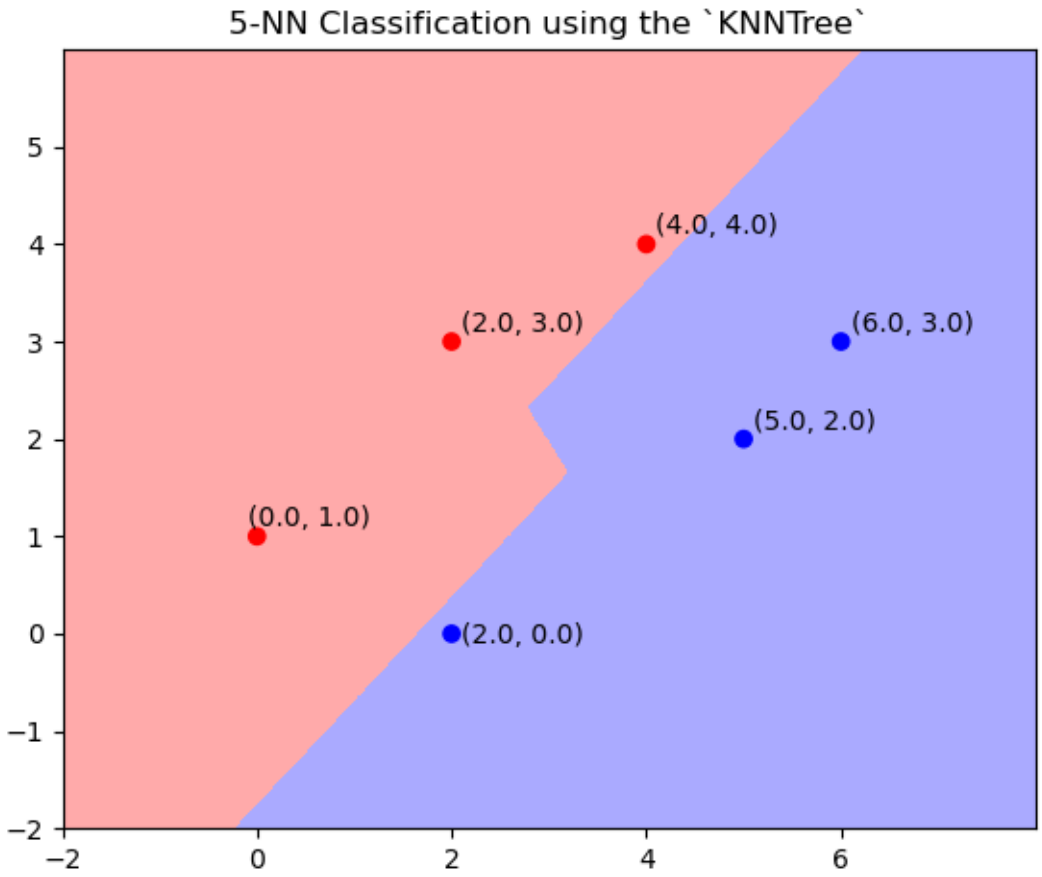

Figure 21: Using KD-Tree neighborhood look-up in order to find the 5 closest neighbors of 1600 points

Παρ' ότι ο μέθοδος των KD-Δέντρων είναι προσεγγιστική και δεν εγγυάται ότι θα βρεί τους Κ πραγματικά κοντινότερους γείτονες, στην προκείμενη περίπτωση η απόκλιση του τελικού αποτελέσματος σε σχέση με τη μέθοδο ωμής βίας είναι ουσιαστικά ανύπαρκτη.

Ωστόσο, αυτό που είναι σίγουρο είναι πώς η μέθοδος των KD-Δέντρων είναι σημαντικά γρηγορότερη, όπως φαίνεται και από τα παρακάτω δεδομένα.

Type	Neighbors	Predictions	Time	Efficiency
force	1	800	43271.45770	1
force	1	160	2870.635500	1
force	1	80	477.2699000	1
force	3	800	54532.69620	1
force	3	160	2157.242100	1
force	3	80	546.4535000	1
force	5	1600	252786.1562	1
force	5	800	63132.58790	1
tree	1	800	20512.63240	2.1095029080714185
tree	1	160	899.3549000	3.1918828707109950
tree	1	80	225.0998000	2.1202591028512687
tree	3	800	21523.35190	2.5336525859617620
tree	3	160	1015.666200	2.1239675988036226
tree	3	80	211.0624000	2.5890613392058460
tree	5	1600	103242.2123	2.4484767477226947
tree	5	800	21571.52390	2.9266633267388213