Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Υπολογιστική Γεωμετρία
Εργασία

Ονοματεπώνυμο: Γιάννης Ζερβάκης Αριθμός Μητρώου: 1115201800048

Στην υποχρεωτική εργασία του μαθήματος της Υπολογιστικής Γεωμετρίας, κληθήκαμε να υλοποιήσουμε αλγορίθμους και προγράμματα που στόχο έχουν την επίλυση υπολογιστικών προβλημάτων γεωμετρικού περιεχομένου. Στην παρούσα αναφορά (report) θα παρουσιαστούν, θα αναλυθούν, και τέλος θα σχολιαστούν τα αποτελέσματα των πειραμάτων που διεργενήθηκαν με βάση τα προβλήματα που ζητήθηκε να επιλύσουμε. Η εργασία υλοποιήθηκε σε γλώσσα `python` και για την απεικόνιση πολλών εκ των αποτελεσμάτων χρησιμοποιήθηκε η βιβλιοθήκη `matplotlib`. Επιπρόσθετα, ο κώδικας είναι κατάλληλα σχολιασμένος και ευανάγνωστος.

1 Αλγόριθμοι Εύρεσης Κυρτού Περιβλήματος στο Επίπεδο (ΚΠ2)

1.1 Εργαλεία – Βοηθητικός Κώδικας

Αρχικά θα ήταν χρήσιμο να πούμε δύο λόγια για κάποια εργαλεία που χρησιμοποιήθηκαν. Αυτά τα κομμάτια κώδικα βρίσκονται στο αρχείο `helpers.py`.

- Η κλάση `Point2D` δημιουργήθηκε και χρησιμοποιήθηκε για την αποθήκευση σημείων στο επίπεδο. Εμπεριέχει όλες τις μεθόδους που χρειάστηκαν προκειμένου να είναι συμβατή σε όλα τα σημεία του κώδικα.
- Οι συναρτήσεις `generate_random_2D_points` και `generate_random_2D_points` είναι γεννήτριες τυχαίων δισδιάστατων σημείων και μη συνευθειακών δισδιάστατων σημείων αντίστοιχα.
- Το κατηγόρημα του προσανατολισμού `ccw` το οποίο υπολογίζει την ορίζουσα: $\begin{vmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{vmatrix}$

  Στην πραγματικότητα μας ενδιαφέρει μόνο το πρόσημο της παραπάνω ορίζουσας.
- Τέλος, στο ίδιο αρχείο βρίσκονται και κάποιες μεθόδοι που έχουν να κάνουν με ευθείας και σημεία ως προς αυτές. Πιο συγκεκριμένα, έχουμε την `right_half_plane()` η οποία δέχεται δύο σημεία στο επίπεδο, τα οποία δημιουργούν μια ευθεία. Η μέθοδος αυτή υπολογίζει σημεία εκ των σημείων που δόθηκαν σαν τρίτο όρισμα, τα οποία βρίσκονται στο δεξί ημι-επίπεδο της ευθείας. Επίσης, έχουμε την `furthest_point_to_line()` η οποία δοθέντων δύο σημείων στο επίπεδο, υπολογίζει το μακρύτερο σημείο από ένα σύνολο σημείων που δόθηκαν ως όρισμα, ως προς την ευθεία που τα δύο σημεία ορίζουν.

1.2 Ανάλυση Αλγορίθμων – Συνευθειακά Σημεία

1.2.1 Αυξητικός Αλγόριθμος – Graham's Scan

Αρχικά υλοποιήσαμε τον αυξητικό αλγόριθμο. Ο κώδικας βασίστηκε εξ ολοκλήρου στις διαφάνειες του μαθήματος. Όσον αφορά τα συνευθειακά σημεία, ο αλγόριθμος των διαφανειών δεν τα λαμβάνουν υπόψιν τους. Πράγμα που σημαίνει πως υπολογίζει και τυχόν εσωτερικά σημεία συνευθειακών ευθύγραμμων τμημάτων.

1.2.2 Αλγόριθμος Περιτυλίγματος

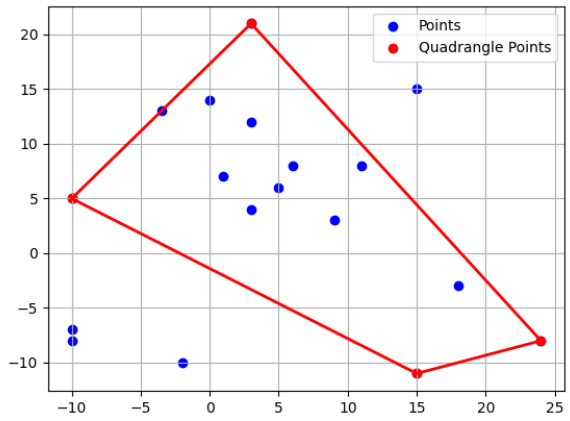
Στην συνέχεια υλοποιήσαμε τον αλγόριθμο περιτυλίγματος για την εύρεση κυρτού περιβλήματος. Ο κώδικας και πάλι βασίστηκε εξ ολοκλήρου στις διαφάνειες του μαθήματος. Στην περίπτωση των συνευθειακών σημείων, ο εν λόγω αλγόριθμος δεν συνυπολογίζει τα σημεία που είναι εσωτερικά του τμήματος που σχηματίζουν τρία συνευθειακά σημεία. Αυτό γίνεται με τον έλεγχο που υπάρχει αν το κατηγόρημα προσανατολισμού επιστρέψει την τιμή μηδέν. Στην περίπτωση αυτή ελέγχουμε αν το σημείο που είναι υποψήφιο για κορυφή του ΚΠ2 είναι εσωτερικό.

1.2.3 Αλγόριθμος Διαίρε και Βασίλευε

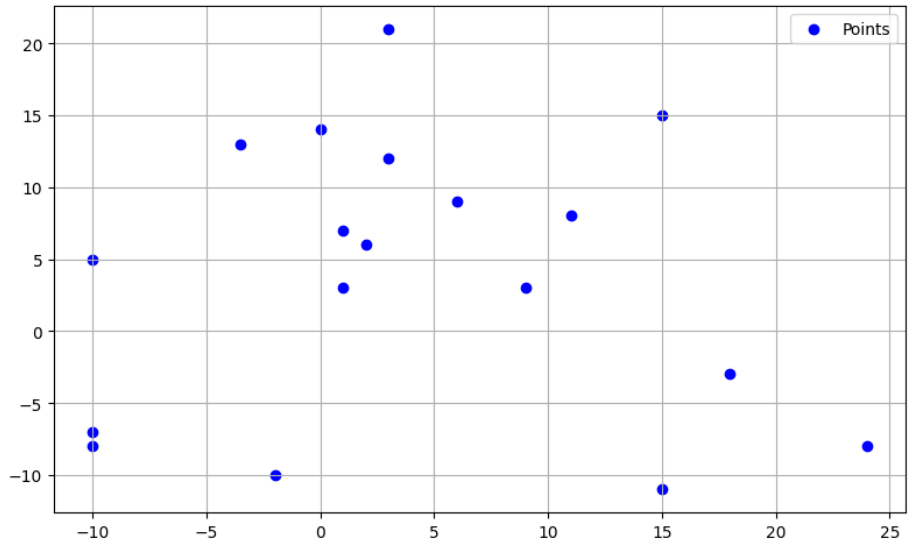
Υλοποιήσαμε επίσης τον αλγόριθμο εύρεσης ΚΠ2 Διαίρε και Βασίλευε. Ο βασικός κώδικας βασίστηκε στις διαφάνειες του μαθήματος, ενώ για τον κώδικα της συνάρτησης συνένωσης (`merge()`) ακολουθήσαμε εν μέρει τον αλγόριθμο που περιγράφεται σε αυτό το άρθρο. Γενικότερα, ο αλγόριθμος Διαίρε και Βασίλευε, συνυπολογίζει τα εσωτερικά σημεία συνευθειακών τμημάτων. Έτσι το αποτέλεσμα μοιάζει με εκείνο αυξητικού αλγόριθμου.

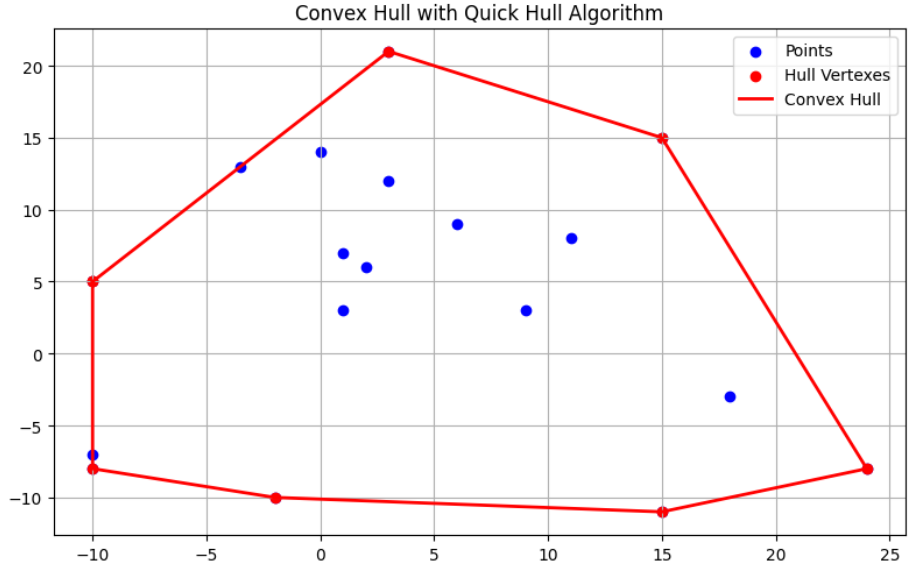
1.2.4 Quick Hull

Ο τελευταίος αλγόριθμος που υλοποιήσαμε για την εύρεση κυρτού περιβλήματος στο επίπεδο, είναι ο αλγόριθμος quick hull. Ο κώδικας βασίστηκε εξ ολοκλήρου στις διαφάνειες του μαθήματος. Στην τελευταία αυτή περίπτωση, πάλι δεν συνυπολογίζονται τα συνευθειακά σημεία. Εδώ διακρίνουμε δύο περιπτώσεις. Αρχικά τα συνευθειακά σημεία που προκύπτουν από τις τέσσερις πιο ακραίες αρχικές κορυφές του τετράπλευρου, τα οποία προφανώς δεν υπολογίζονται όπως φαίνεται στην πρώτη εικόνα. Η δεύτερη περίπτωση αφορά την κάτω αριστερά περιοχή του τετράπλευρου που βλέπουμε στην παρακάτω εικόνα.

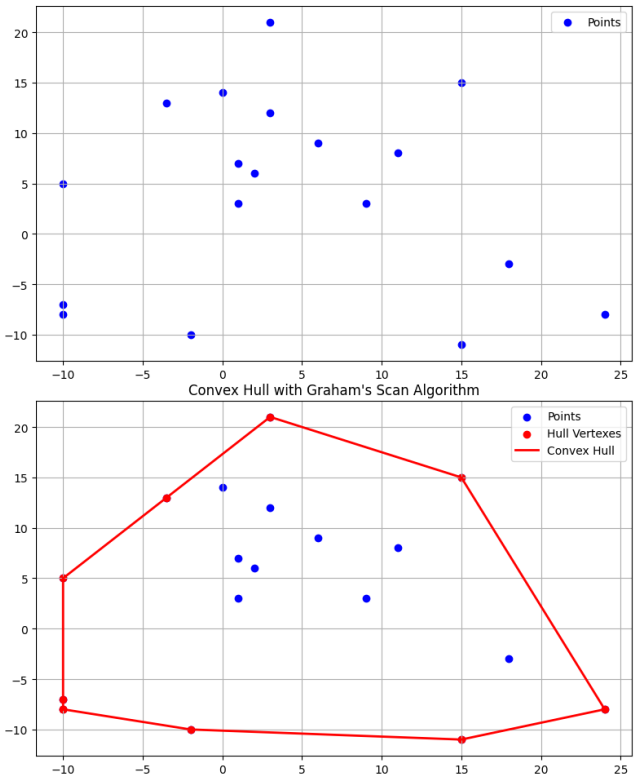


Εκεί, για $x = -10$ έχουμε τρία συνευθειακά σημεία. Από την ευθεία που σχηματίζει το αριστερότερο σημείο $A(-10, 5)$ με το χαμηλότερο $B(15, -11)$, το πιο μακρινό σημείο είναι το χαμηλότερο από τα τρία συνευθειακά που προαναφέραμε $C(-10, -8)$. Άρα το ημιεπίπεδο που δημιουργείται δεξία από την ευθεία που ορίζουν τα A, C, δεν περιέχει σημεία αφού το εσωτερικό των συνευθειακών είναι πάνω στην ευθεία AC. Το τελικό περίβλημα φαίνεται στην παρακάτω εικόνα.

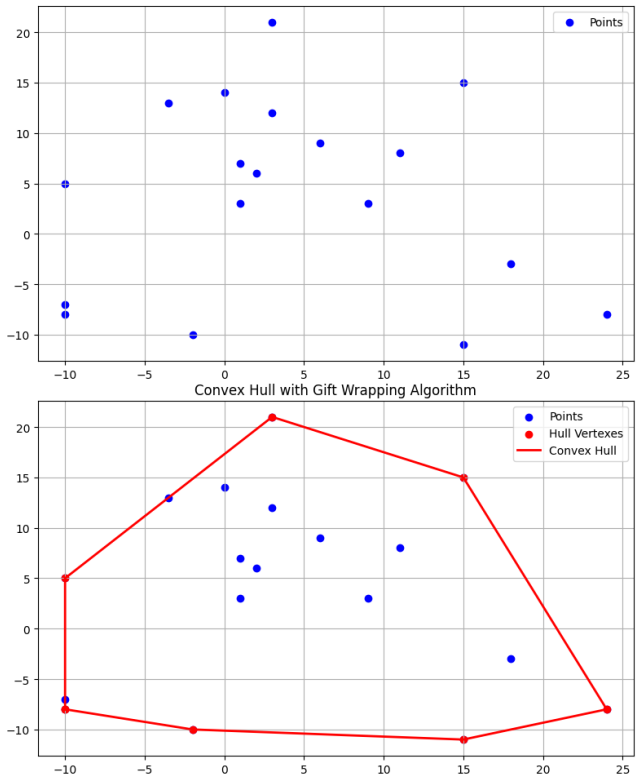




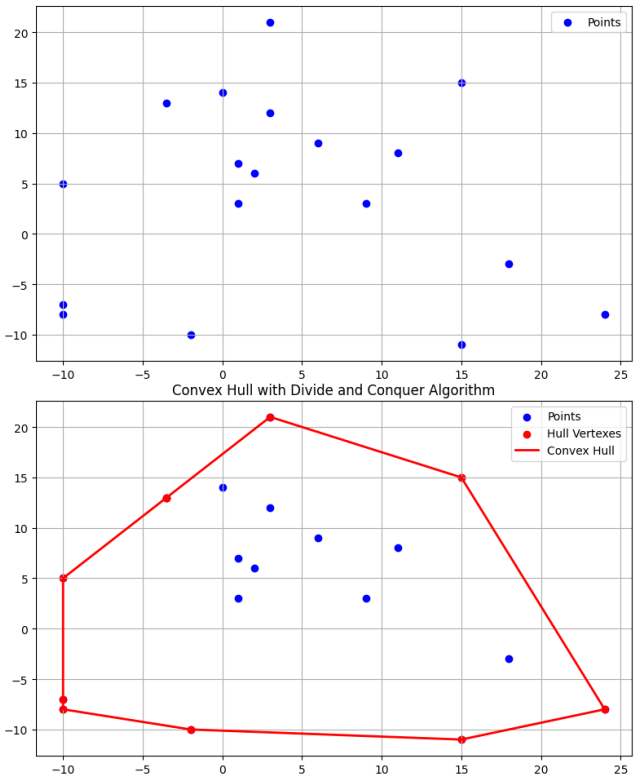
Συνολικά έχουμε για ένα μικρό παράδειγμα 19 δισδιάστατων σημείων τα εξής αποτελέσματα :



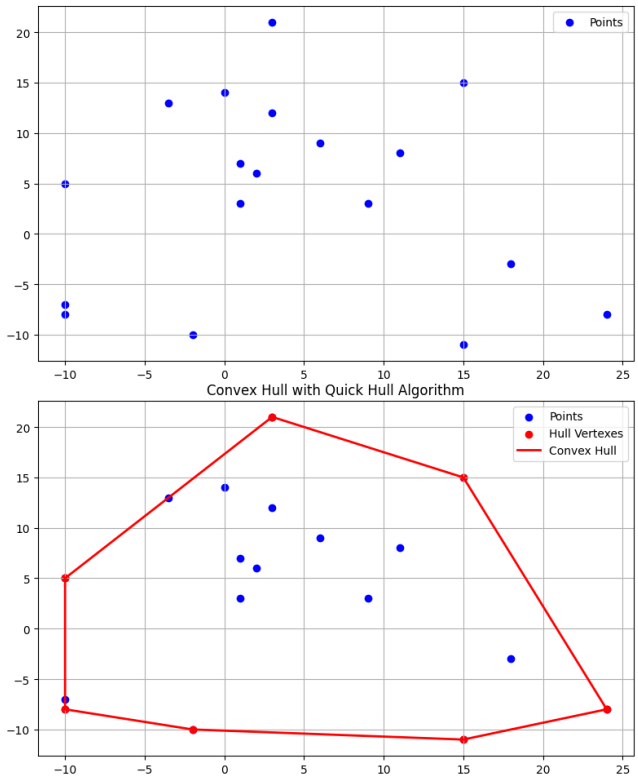
(α′) Graham's Scan



(β′) Gift Wrapping (Jarvis March)



(γ′) Divide And Conquer



(δ′) Quick Hull

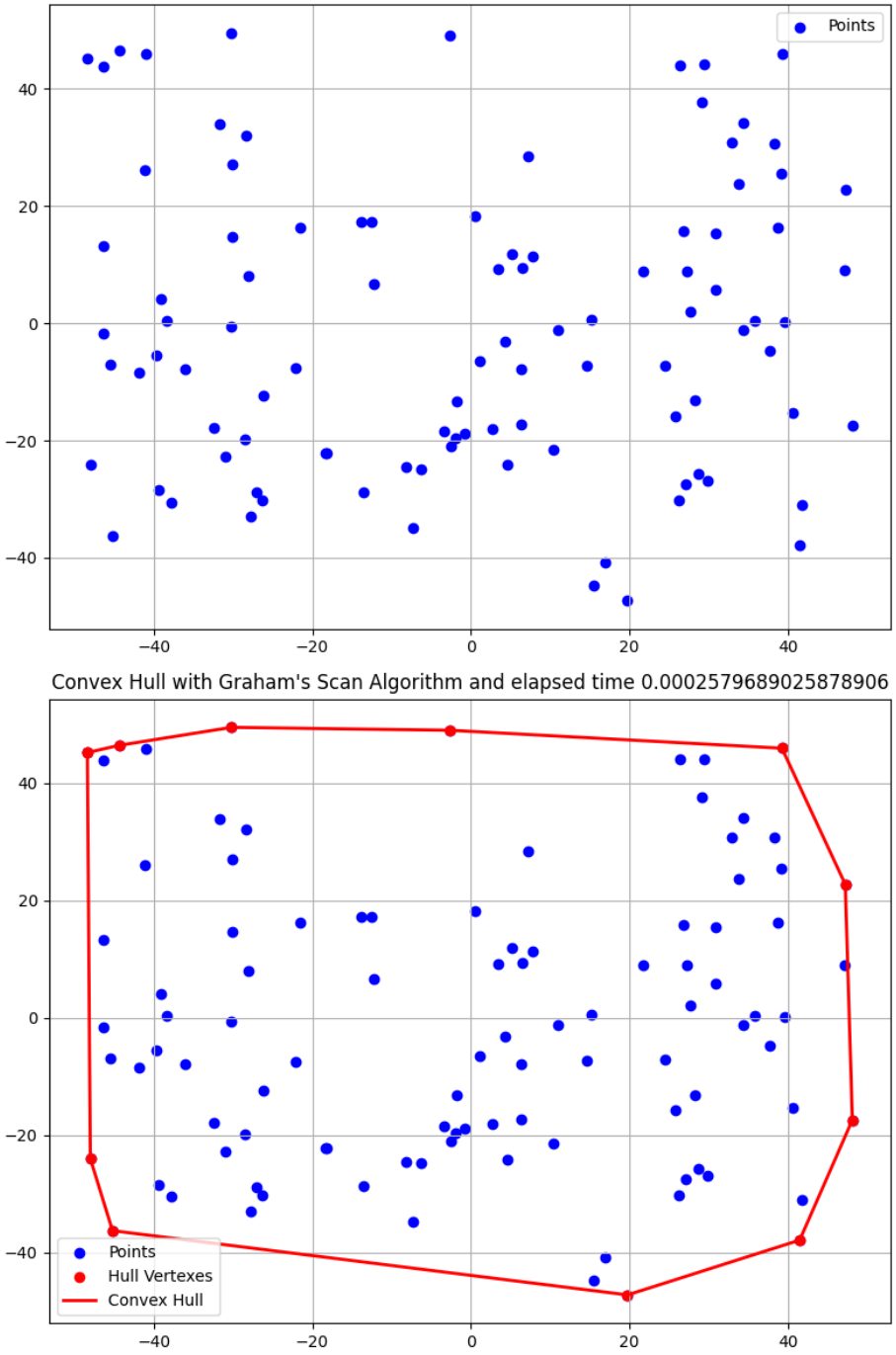
Αποτελέσματα για 19 σημεία στο επίπεδο

1.3 Εκτέλεση Αλγορίθμων – Παρουσίαση αποτελεσμάτων

Στο σημείο αυτό θα παρουσιάσουμε τα αποτελέσματα της εκτέλεσης των τεσσάρων αλγορίθμων για 100 μη-συνευθειακά σημεία στο επίπεδο.

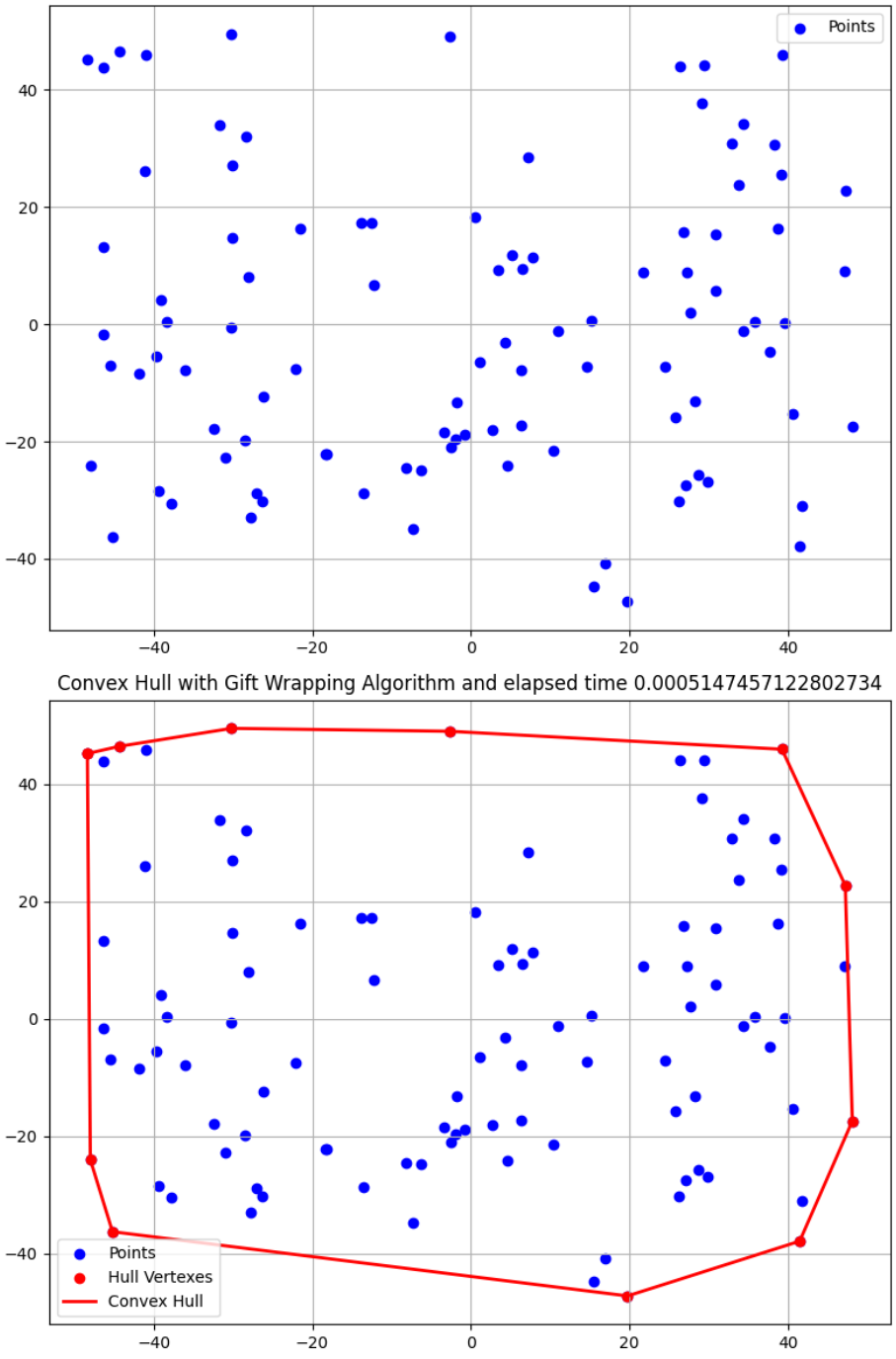
1.3.1 Ορθότητα - ΚΠ2

1.3.1.1 Αυξητικός Αλγόριθμος – Graham's Scan



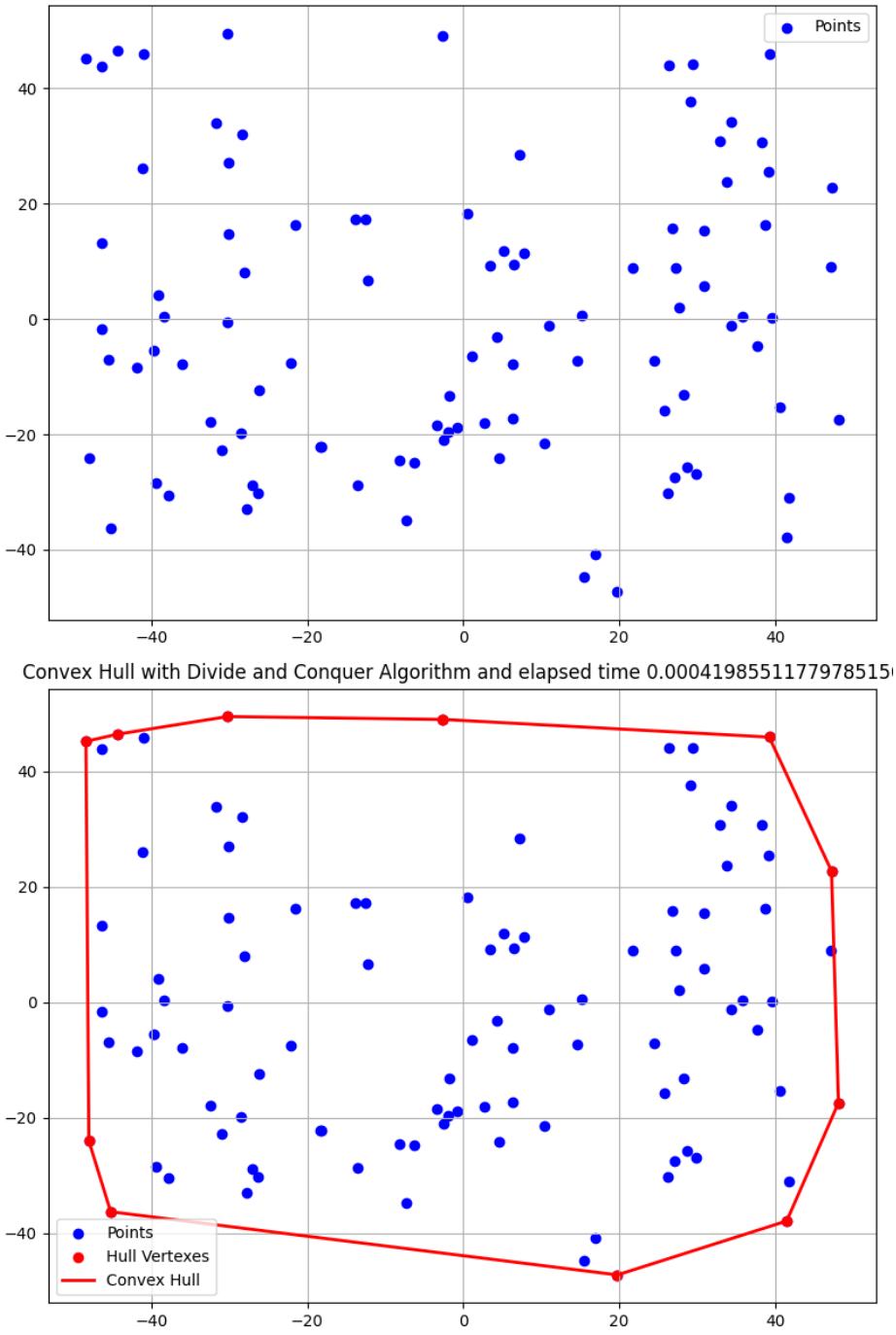
Graham's Scan

1.3.1.2 Αλγόριθμος Περιτυλίγματος – Gift Wrapping - Jarvis March



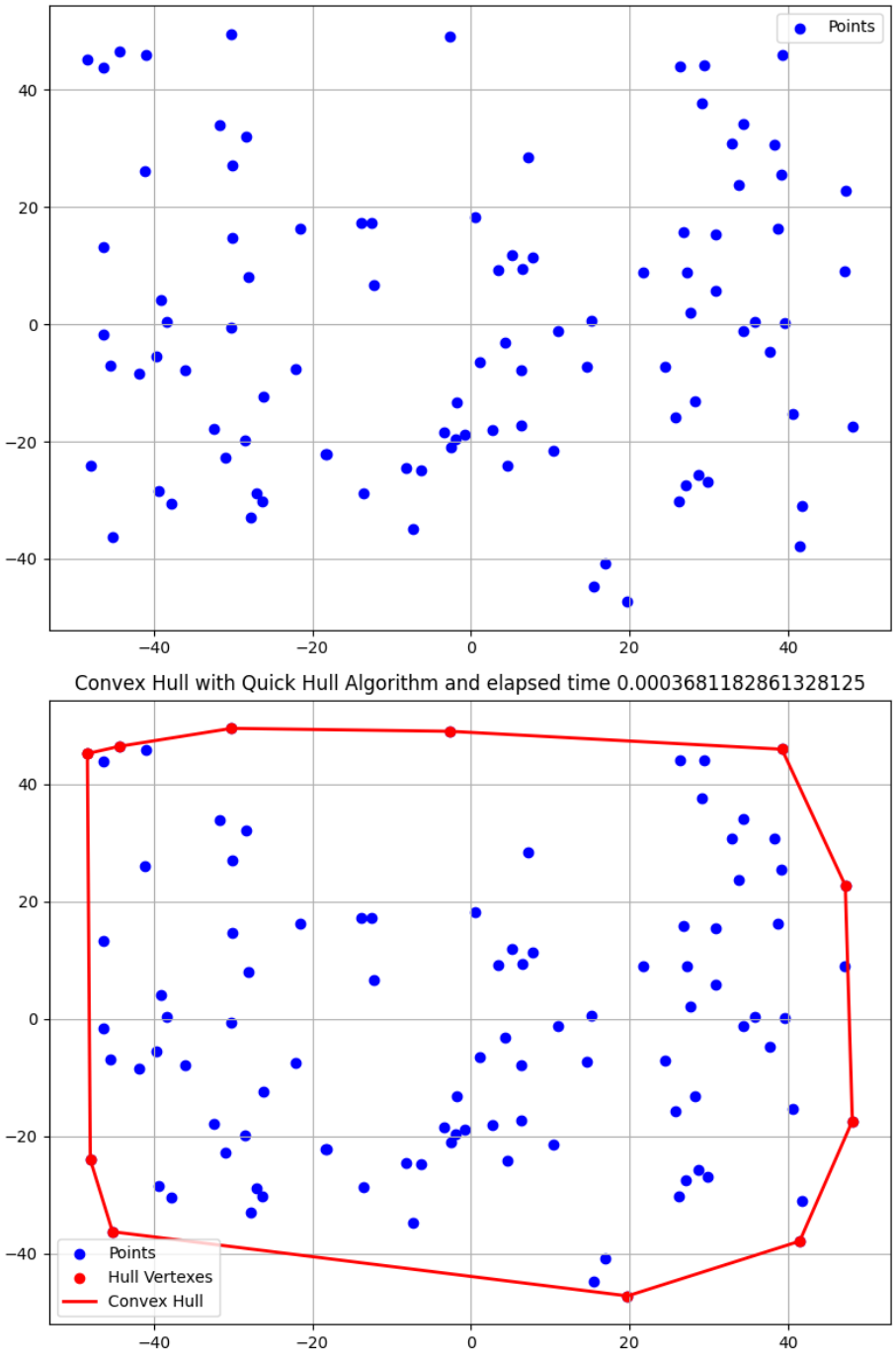
Jarvis March

1.3.1.3 Αλγόριθμος Διαίρε και Βασίλευε



Divide and Conquer

1.3.1.4 Αλγόριθμος Quick Hull



Quick Hull

Στα παραπάνω αποτελέσματα, οι γωνίες που συγκροτούν το ΚΠ2 είναι οι ίδιες. Αυτό συμβαίνει όπως αναλύσαμε παραπάνω, γιατί δεν έχουμε συνευθειακά σημεία. Αντιθέτως αν χρησιμοποιούσαμε την συνάρτηση `generator_random_2D_points()` θα βλέπαμε τα ίδια αποτελέσματα για τον αυξητικό αλγόριθμο με περισσότερες κορυφές (τα εσωτερικά σημεία των συνευθειακών τμημάτων) και διαφορετικά για τους αλγορίθμους περιτυλίγματος και Quick Hull.

1.3.2 Χρόνος Εκτέλεσης–Πολυπλοκότητας

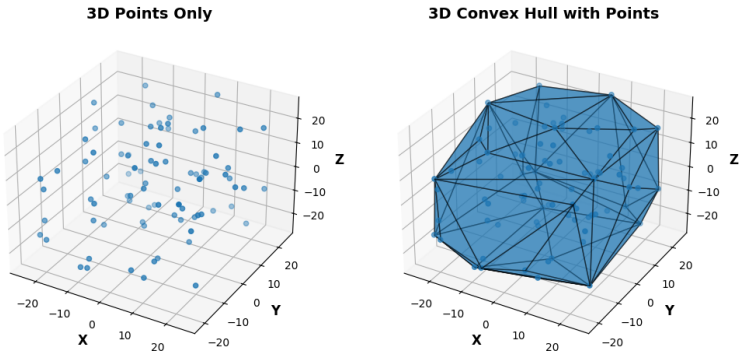
Όσον αφορά τους χρόνους εκτέλεσεις, αν και τα μεγέθη είναι μικρά και δεν μπορούμε να αποφανθούμε εύκολα, παρατηρούμε πως ο αυξητικός ήταν ο πιο γρήγορος ενώ ο αλγόριθμος περιτυλίγματος ο πιο αργός. Οι δύο αυτοί αλγόριθμοι έχουν πολυπλοκότητες $O(nlogn)$ και $O(n^2)$ αντίστοιχα. Ο αλγόριθμος διαίρε και βασίλευαι αν κι έχει πολυπλοκότητα $O(nlogn)$ φαίνεται να έχει τον δεύτερο πιο αργό χρόνο, πράγμα που είναι λογικό δεδομένου πως αν και στην χειρότερη περίπτωση η πολυπλοκότητα του αλγορίθμου Quick Hull είναι $O(n^2)$, συνήθως η πολυπλοκότητα που έχει είναι $O(nlogn)$.

1.4 Οπτικοποίηση

Στον κατάλογο-ρίζα της εργασίας βρίσκεται ένα αρχείο που ονομάζεται `incremental.gif`. Το gif αυτό, δείχνει βήμα προς βήμα τον αυξητικό αλγόριθμο που εκτελείται για την εύρεση του ΚΠ2 με 100 τυχαία σημεία. Αρχικά υπολογίζει το άνω περίβλημα, έπειτα το κάτω ενώ στο τέλος τα ενώνει.

1.5 Εύρεση Κυρτού Περιβλήματος σε 3 διαστάσεις – ΚΠ3

Στο ερώτημα αυτό κληθήκαμε να παρουσιάσουμε έναν αλγόριθμο για την εύρεση κουρτού περιβλήματος στον χώρο (ΚΠ3). Προκειμένου λοιπόν να προσδιορίσουμε το ΚΠ3 85 τυχαίων σημείων στον χώρο όπως και ζητήθηκε, χρησιμοποιήσαμε την βιβλιοθήκη `SciPy.spacial` καθώς και την μέθοδο `ConvexHull()`. Η μέθοδος αυτή υλοποιεί τον αλγόριθμο που βρίσκει τόσο το ΚΠ2 όσο και το ΚΠ3. Εν προκειμένω, εργαστήκαμε για ΚΠ3, δημιουργώντας 85 τυχαία σημεία στον χώρο με την βοήθεια της ομοιόμορφης κατανομής (uniform), και καλώντας την συνάρτηση. Ο κώδικας που υπάρχει μετά την κλήση έχει να κάνει καθαρά με την παρακάτω απεικόνιση του αποτελέσματος :



Quick Hull 3D

Σημείωση: Η γραφική παράσταση που προκύπτει από τον κώδικα σε python επιτρέπει στον χρήστη να πλοηγηθεί στον χώρό προκειμένου να δει από όποια οπτική γωνία θέλει το ΚΠ3-αποτέλεσμα που προκύπτει.

2 Γραμμικός Προγραμματισμός

Δοσμένου του προβλήματος της εκφώνησης, καλούμαστε λύσουμε το πρόβλημα γραμμικού προγραμματισμού. Πρόκειται για ένα πρόβλημα μεγιστοποίησης της τιμής μιας συνάρτησης δεδομένων κάποιων περιορισμών :

- $max\{-3x_1 + 12x_2\}$
- Περιορισμοί:

$$x_1 - 2x_2 \geq 1$$
$$2x_1 - 3x_2 \geq 6$$
$$-x_1 + 3x_2 \leq 0$$
$$-x_1 + 6x_2 \leq 12$$
$$4x_1 - 9x_2 \leq 27$$
$$x_1, x_2 \geq 0$$

Προκειμένου να λύσουμε το παραπάνω πρόβλημα, θα προσπαθήσουμε να το φέρουμε στη μορφή ελαχιστοποίησης προκειμένου να συμπίπτει με την μέθοδο που δημιουργήσαμε, η οποία λύνει πρόβλήματα ελαχιστοποίησης.

Γνωρίζοντας πως ισχύει $maxf(x) = -min-f(x) \Rightarrow minf(x) = -max-f(x)$, μπορούμε να μετατρέψουμε το παραπάνω πρόβλημα μεγιστοποίησης στο ισοδύναμο πρόβλημα ελαχιστοποίησης της τιμής της συνάρτησης:

- $-min\{-(-3x_1 + 12x_2)\} = -min3x_1 - 12x_2$
- Περιορισμοί:

$$-x_1 + 2x_2 \geq 1$$
$$-2x_1 + 3x_2 \geq 6$$
$$x_1 - 3x_2 \leq 0$$
$$x_1 - 6x_2 \leq 12$$
$$-4x_1 + 9x_2 \leq 27$$
$$-x_1, -x_2 \geq 0$$

Στο σημείο αυτό όμως θέλουμε όλοι οι περιορισμοί να είναι της μορφής :

$$ax + by \leq d$$

Έτσι πολλαπλασιάζουμε όπου έχουμε $\geq$ με -1 κι έχουμε το τελικό ισοδύναμο πρόβλημα :

- $-min\{-(-3x_1 + 12x_2)\} = -min3x_1 - 12x_2$
- Περιορισμοί:

$$-x_1 + 2x_2 \leq -1$$
$$-2x_1 + 3x_2 \leq -6$$
$$x_1 - 3x_2 \leq 0$$
$$x_1 - 6x_2 \leq 12$$
$$-4x_1 + 9x_2 \leq 27$$
$$x_1, x_2 \leq 0$$

Τυπικά θα έπρεπε να φέρουμε το πρόβλημα στην εξής κανονική μορφή :

- $-min\{-(-3x_1 + 12x_2)\} = -min3x_1 - 12x_2$
- Περιορισμοί:

$$-x_1 + 2x_2 + s_1 \leq 0$$
$$-2x_1 + 3x_2 + s_2 \leq 0$$
$$x_1 - 3x_2 + s_3 \leq 0$$
$$x_1 - 6x_2 - s_4 \leq 0$$
$$-4x_1 + 9x_2 - s_5 \leq 0$$
$$x_1, x_2 \leq 0$$

$$s_1 = 1$$
$$s_2 = 6$$
$$s_3 = 0$$
$$s_4 = 12$$
$$s_5 = 27$$

Ωστόσο το τελευταίο βήμα δεν είναι ανάγκη, καθώς οι μέθοδοι που χρησιμοποιήσαμε λύνουν το πρόβλημα και στη προηγούμενή τους (ισοδύναμη) μορφή. Για την αυξητική προσέγγιση, δημιουργήθηκε τη μέθοδος `incremental_lp_solver()`, η οποία είναι βασισμένη στον αλγόριθμο των διαφανειών. Ωστόσο χρειάζεται να γίνει μια πολύ σημαντική σημείωση. Στο πρώτο βήμα του αλγορίθμου, καλούμαστε να παράγουμε την πρώτη μερική βέλτιστη λύση του προβλήματος χρησιμοποιώντας τους πρώτους $d + 1$ περιορισμούς. Αυτό πρόκειται για ένα λεπτό σημείο καθώς το πρόβλημα που προκύπτει **μπορεί να είναι μη-φραγμένο** κι έτσι να μη καταφέρουμε να βρούμε λύση. Για παράδειγμα, στο πρόβλημα που καλούμαστε να επιλύσουμε. οι πρώτοι 3 περιορισμοί απομονωμένοι δημιουργούν ένα μη-φραγμένο πρόβλημα. Προκειμένου να αποφύγουμε στην προκειμένη το εν λόγω πρόβλημα, αλλάξαμε τη σειρά των περιορισμών, όπου προέκυψε :

- $-min\{-(-3x_1 + 12x_2)\} = -min3x_1 - 12x_2$
- Περιορισμοί:

$$x_1 - 3x_2 \leq 0$$
$$x_1 - 6x_2 \leq 12$$
$$-4x_1 + 9x_2 \leq 27$$
$$-x_1 + 2x_2 \leq -1$$
$$-2x_1 + 3x_2 \leq -6$$
$$x_1, x_2 \leq 0$$

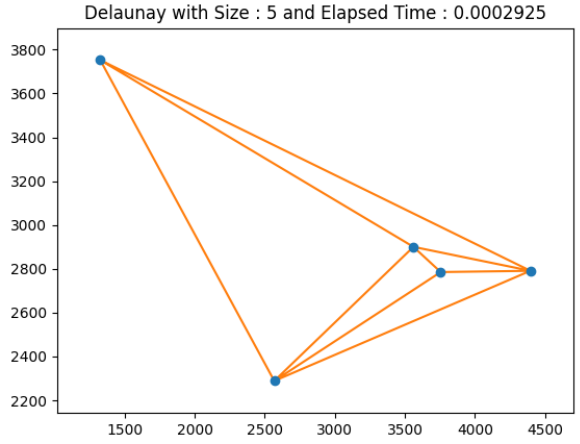
Για μια ολοκληρωμένη προσέγγιση εύρεσης της πρώτης μερικής λύσης, θα έπρεπε να διαλέγαμε $d + 1$ περιορισμούς από τους n κι έπειτα να βλέπαμε αν είναι φραγμένο το πρόβλημα. Στην χειρότερη, θα ελέγχαμε $\binom{n}{d+1}$ περιπτώσεις. Για την εύρεση λοιπόν της αρχικής λύσης χρησιμοποιήθηκε η συνάρτηση `linprog()` της βιβλιοθήκης `SciPy`. Το αποτέλεσμα που προέκυψε για το πρόβλημά μας είναι :

$$Optimal\_solution : [12., 4.]$$
$$Optimal\_value : 12.0$$

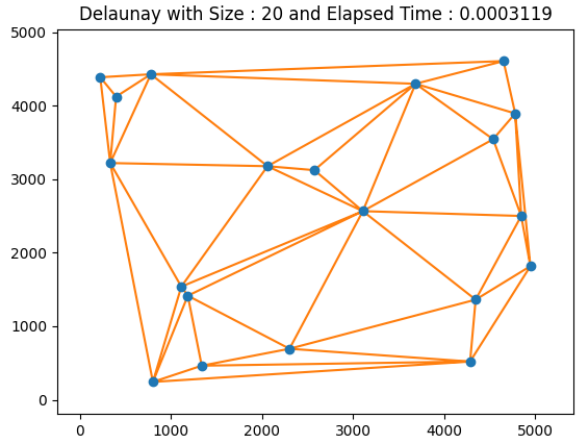
3 Τριγωνιοποίηση Delaunay

Προκειμένου να παρουσιάσουμε τα αποτελέσματα μας για την τριγωνιοποίηση Delaunay, ορίσαμε μια κλάση `IncrementalDelaunay`, η οποία επί της ουσίας προσθέτει ένα προς ένα τα σημεία στο επίπεδο και χρησιμοποιώντας την κλάση `Delaunay` της βιβλιοθήκης της python, `scipy.spatial`, υπολογίζει το τελικό γράφημα τριγωνοποιήσης κατά Delaunay ενώ παράλληλα δημιουργεί και την οπτικοποίηση του αλγορίθμου σε μορφή `.gif`.

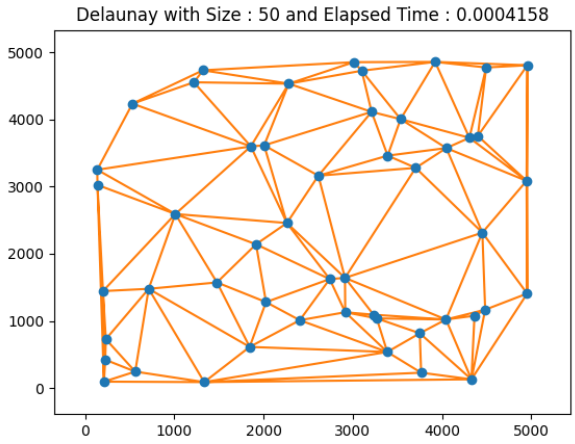
Στο σημείο αυτό, είναι σημαντικό να σημειώσουμε πως η υλοποίηση του αλγορίθμου τριγωνιοποίησης Delaunay της βιβλιοθήκης, είναι και πάλι ο αλγόριθμος `Quick Hull`. Έτσι λοιπόν, αρχικά υπολογίζει το κυρτό περίβλημα στο σύνολο των σημείων και το τελευταίο το χρησιμοποιεί για να παράγει την τελική τριγωνιοποίηση. Στον κατάλογο-ρίζα της εργασίας βρίσκεται το αρχείο `.gif` που οπτικοποιεί τον αλγόριθμο. Στη συνέχεια, παραθέτουμε κάποιες εκτελέσεις του αλγορίθμου με σκοπό τη σύγκριση αυτών.



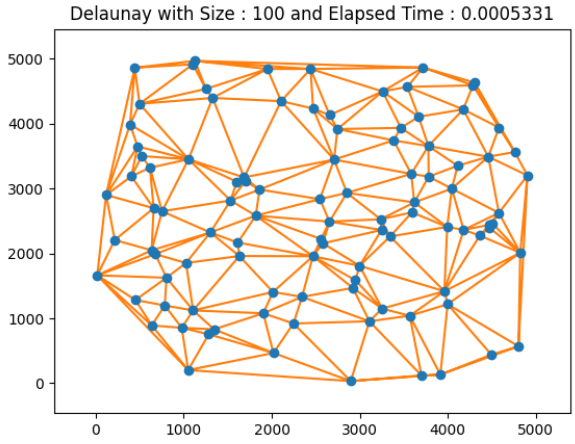
Delaunay – 5 Points



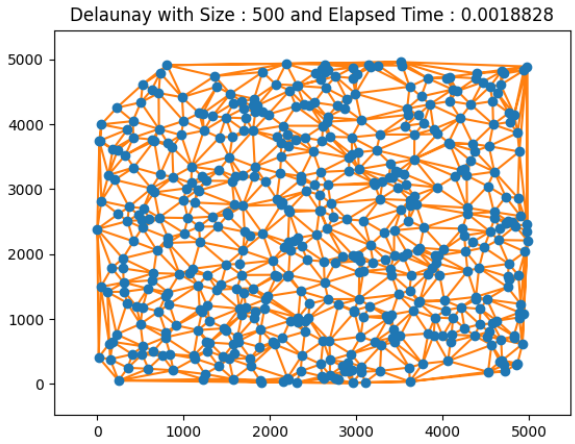
Delaunay – 20 Points



Delaunay – 50 Points



Delaunay – 100 Points



Delaunay – 500 Points

Αρχικά παρατηρούμε πως όσο μεγαλώνει ο αριθμός των σημείων, όλο και πιο δυσνόητο γίνεται το αποτέλεσμα οπτικά. Παρατηρούμε δηλαδή πως για 500 σημεία, νούμερο όχι πάρα πολύ μεγάλο, ήδη είναι δύσκολο να το παρατηρήσουμε και να ξεχωρίζουμε τις πλευρές και τις κορυφές των τριγώνων. Επίσης, όσο μεγαλώνει ο αριθμός των σημείων, θα έχουμε όλο και πιο συχνά σενυθειακές κορυφές τριγώνων που χρήζουν ειδικής μεταχείρισης από τους αλγορίθμους μας. Όσον αφορά την πολυπλοκότητα του αλγορίθμου, γνωρίζουμε πως ο αλγόριθμος Quick Hull είναι μια μορφή διαίρε και βασίλευε. Αυτό σημαίνει πως η πολυπλοκότητα είναι $O(nlogn)$. Πράγματι στα παραδείγματά μας φαίνεται μικρή αύξηση του χρόνου εκτέλεσης, αλλά υπαρκτή, κυρίως όταν πάμε από τα 100 στα 500 σημεία. Τέλος, παραθέτουμε έναν πίνακα με χρόνους εκτελέσεις για συγκεκριμένα πιο μεγάλα τυχαία σύνολα σημείων στο επίπεδο.

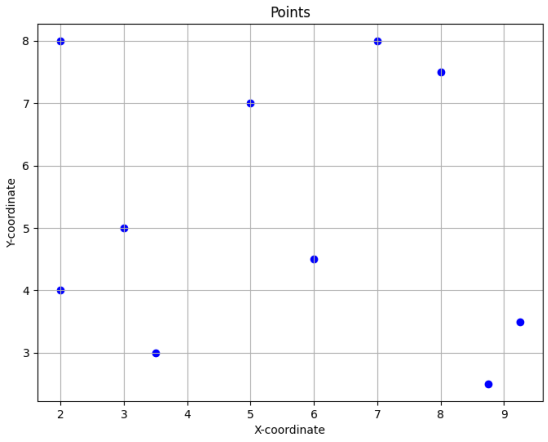
Size	Elapsed Time [seconds]
1000	0.00681
5000	0.0395
20000	0.18087
50000	0.49868
100000	1.13066
500000	6.41506

4 Γεωμετρική Αναζήτηση

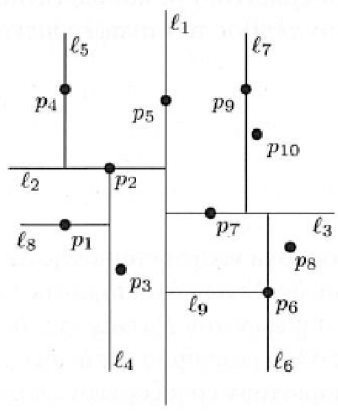
4.1 Κατασκευή KD-Tree

Για την κατασκευή KD-Tree χρησιμοποιήσαμε τον αλγόριθμο των διαφανειών. Ορίσαμε λοιπόν δύο κλάσεις : την `KDTreeNode` και την `KDTree`. Με βοηθητικές συναρτήσεις όπως η `median_point` (η οποία βρίσκει την $\lceil \frac{n}{2} \rceil$ μικρότερη τιμή από ένα σύνολο σημείων, δοθέντος του άξονα αναζήτησης) και `separate_to_subsets` (διαχωρίζει τα σημεία σε σύνολα αριστερά/κάτω – δεξιά/πάνω από την διαχστική ευθεία), υλοποιήσαμε τον αλγόριθμο που περιγράφεται στα πλαίσια του μαθήματος για την κατασκευή ενός KD-Tree δεδομένου ενός συνόλου σημείων στο επίπεδο. Για την προσωμείωση του αποτελέσματος, χρησιμοποιήθηκε ένα σύνολο σημείων παρεμφερές αυτού των διαφανειών. Στην κλάση `KDTreeNode`, η συνάρτηση εκτύπωσης δημιουργεί μια βασική οπτικοποίηση του αποτελέσματος της δομής του δέντρου που παράχθηκε.

Στο σημείο αυτό θα δούμε δύο γραφήματα, τα οποία δείχνουν ποιοτικά τις διχαστικές ευθείες που ορίστηκαν για την δημιουργία KD-Tree βασισμένο σε αυτό των διαφανειών του μαθήματος

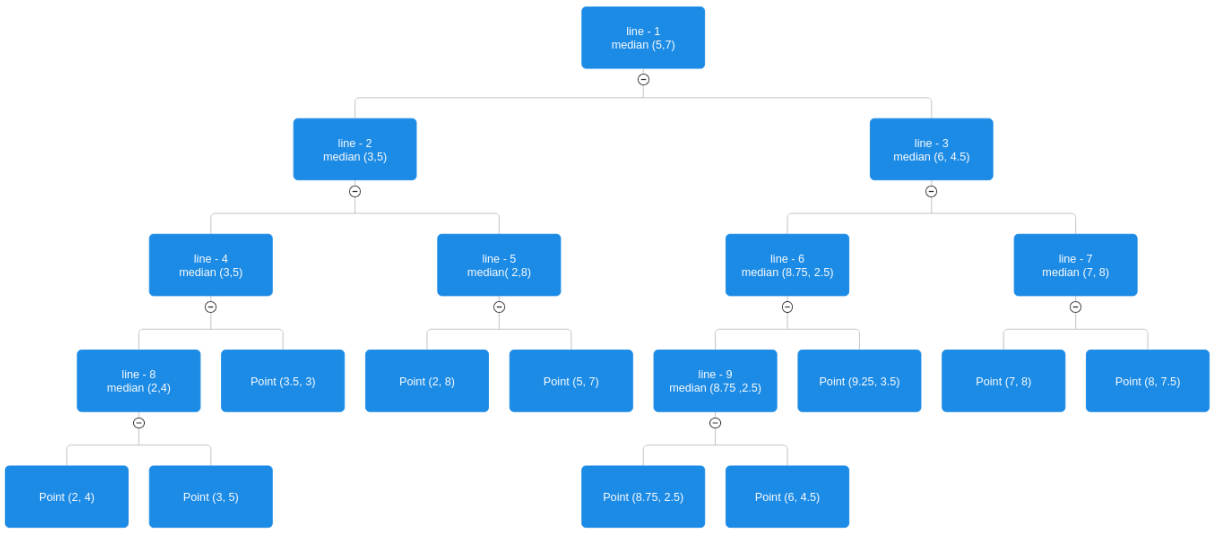


(α′) Σημεία στο Επίπεδο



(β′) Διχαστικές Ευθείες

Για τα παραπάνω λοιπόν σημεία, δημιουργείται το παρακάτω KD-Tree :



Στο σημείο αυτό να σημειώσουμε πως η δημιουργία ενός KD-Tree έχει χωρική και χρονική πολυπλοκότητα $O(nlogn)$.

4.2 Διερεύνηση Περιοχής με χρήση KD-Tree

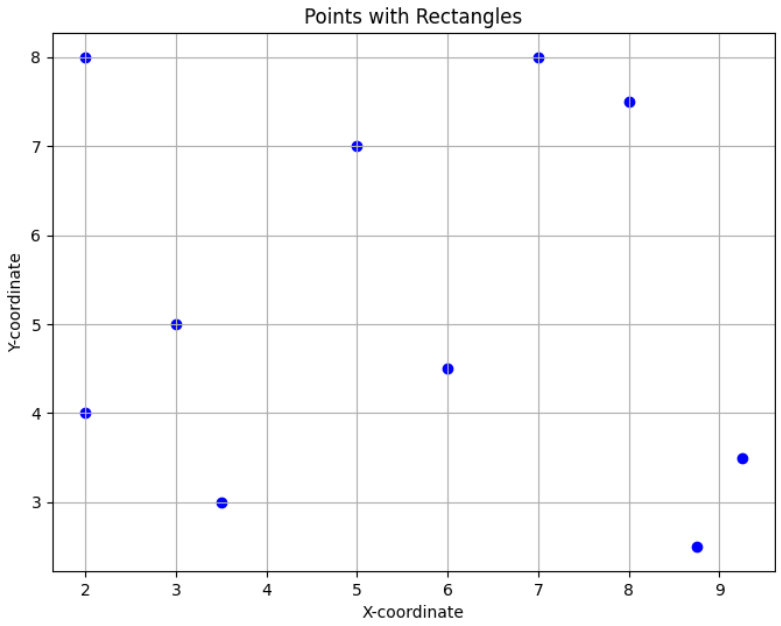
Τέλος, μας ζητήθηκε να υλοποιήσουμε την μέθοδο διερεύνησης μιας περιοχής (region) χρησιμοποιώντας KD-Tree. Ο αλγόριθμος που υλοποιήθηκε βρίσκεται στην μέθοδο `investigate` της κλάσης `KDTree` και βασίστηκε για άλλη μια φορά πλήρως σε αυτόν των διαφανειών του μαθήματος. Επί της ουσίας, χωρίζεται σε 2 μέρη : (α) την περίπτωση βάσης, στην οποία καθώς διατρέχουμε το δέντρο βρισκόμαστε πλέον σε κόμβο-φύλλο κι έτσι απλά ελέγχουμε αν το σημείο του κόμβου εμπεριέχεται στην δοθείσα περιοχή προκειμένου να προσθέσουμε τον κόμβο στο αποτέλεσμα και (β) την περίπτωση που δεν ισχύει η παραπάνω συνθήκη και ελέγχουμε τόσο το αριστερό υποδέντρο όσο και το δεξί και βρίσκουμε το σύνολο των κόμβων που ζητούνται.

Στην προσπάθειά μας να μπορούμε να κάνουμε τους κατάλληλους ελέγχους όσον αφορά τις περιοχές που ορίζει ένα υποδέντρο σε σχέση με την δοθείσα περιοχή που ερευνούμε, χρειάστηκαν κάποιες βοηθητικές μέθοδοι :

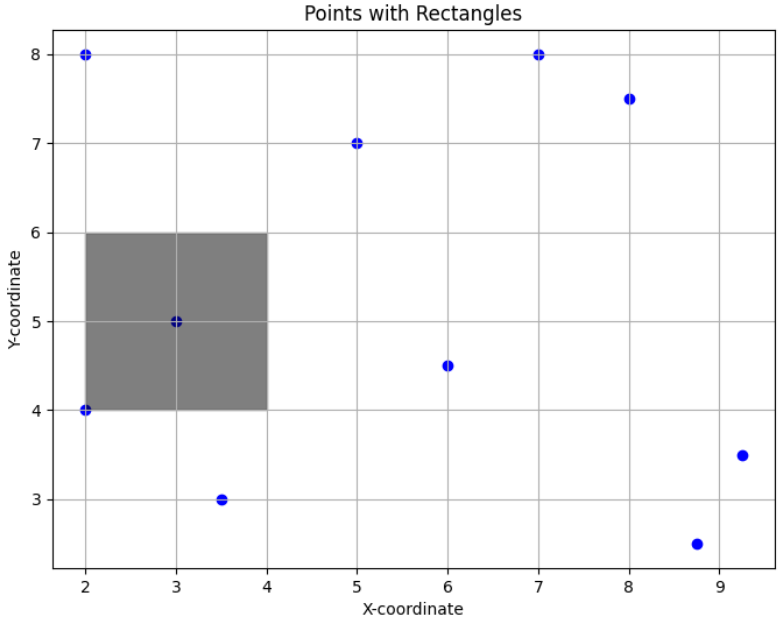
- `is_point_in_rectangle(point, rec)`: ελέγχει αν το σημείο `point` βρίσκεται εσωτερικά του ορθογωνίου `rec`
- `is_sub_rectangle(rec, target_rec)`: ελέγχει αν η περιοχή που ορίζεται από το ορθογώνιο `rec` είναι εξ΄ ολοκλήρου εσωτερικά της περιοχής που ορίζεται από το ορθογώνιο `target_rec`.
- `is_rectangles_intersection_non_empty(rec, target_rec)`: ελέγχει αν η τομή δύο ορθογώνιων περιοχών είναι μη-κενή.

Να σημειωθεί επίσης, πως για λόγους ευκολίας χρησιμοποιήθηκαν τέσσερα επιπλέον χαρακτηριστικά (`leftmost_x`, `rightmost_x`, `lower_y`, `upper_y`), τα οποία κρατάνε τα αντίστοιχα ακραία σημεία (αριστερά-δεξιά-κάτω-πάνω), προκειμένου να οριοθετηθεί κατάλληλα η περιοχή που ορίζει συνολικά το KD-Tree, δηλαδή η ρίζα του δέντρου.

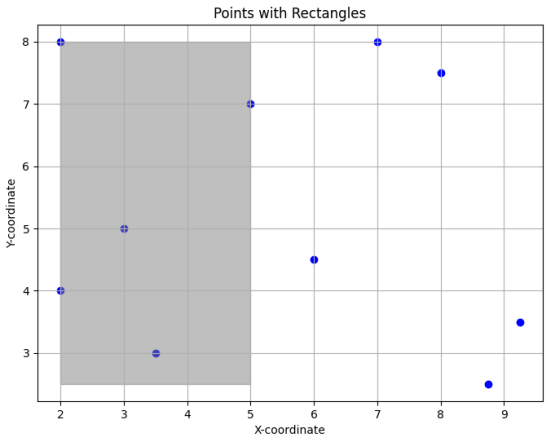
Προς επίδειξην της λειτουργικότητας της μεθόδου διερεύνησεις, υπάρχει το αντίστοιχο παράδειγμα στον κώδικα της συνάρτησης `main`. Στις παρακάτω εικόνες φαίνεται βήμα προς βήμα η διερεύνση του KD-Tree που δημιουργήσαμε με τα σημεία που αναφέρθηκαν στην ενότητα που περιγράφει την κατασκευή των KD-Tree και δοθείσας έκτασης [2,4,4,6]:



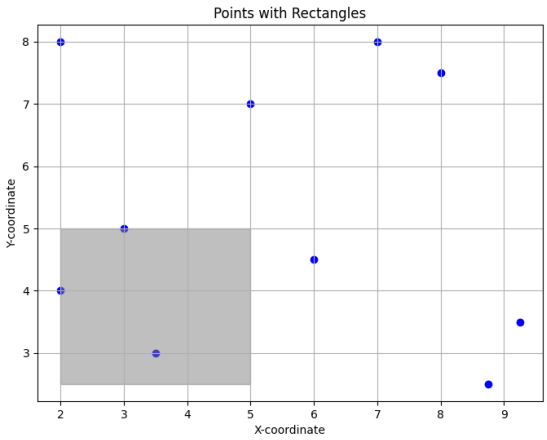
Points



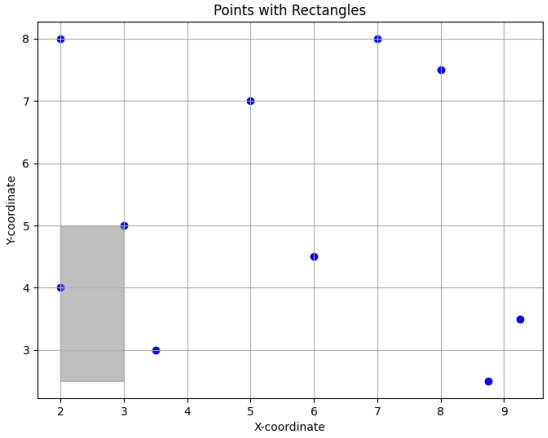
Given Region [2,4,4,6]



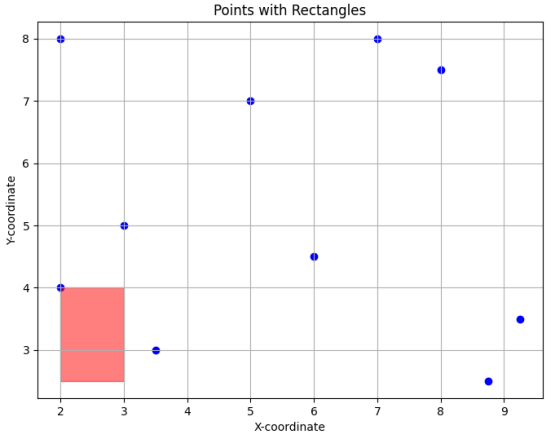
Depth 1



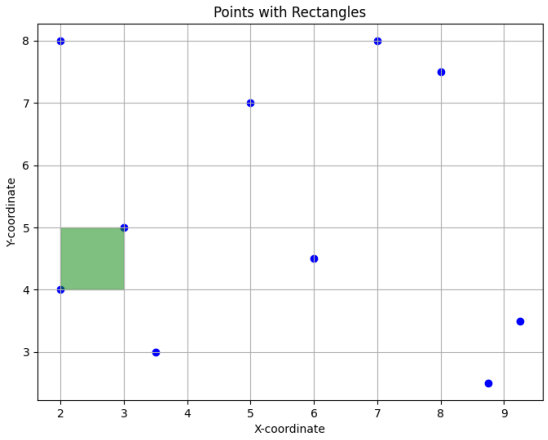
Depth 2



Depth 3



Left Child Region (lc(v))



Right Child Region (rc(v))

Να σημειωθεί σε αυτό το σημείο πως στις παραπάνω εικόνες δεν φαίνονται οι συμμετρικές αναζητήσεις στα δεξιά υποδέντρα εκτός της τελευταίας εικόνας. Προφανώς ο αλγόριθμος διερεύνησης διατρέχει και τα δεξιά υποδέντρα απλά στο συγκεκριμένο παράδειγμα δεν βρίσκει αποτελέσματα, δηλαδή το σύνολο των κόμβων που περιέχουν σημεία που είναι εντός της δοθείσας έκτασης είναι κενό. Έτσι, η μέθοδος τελικά επιστρέφει το άθροισμα των κόμβων του αριστερού και του δεξιού υποδέντρου που επισημάνθηκαν. Εν προκειμένου τους κόμβους φύλλα (πράσινη και κόκκινη περιοχή). Χρησιμοποιώντας μια βοηθητική μέθοδο `KDTreeNode.to_list()` η οποία επιστρέφει όλα τα στοιχεία ενός υποδέντρου δεδομένης μιας ρίζας, έχουμε τελικά τα αποτελέσματα [(2, 4), (3, 5)].