Θεοδώρα Αρχοντάκη – sdi2000014

ΥΛΟΠΟΙΗΣΗ - Α

A1

CONVEX HULL - 2D:

Έχουμε το αρχείο **"functions.py"** που περιέχει συναρτήσεις που θα χρησιμοποιηθούν στους αλγορίθμους:

- Η συνάρτηση "CCW " επιστρέφει μηδέν αν τα σημεία είναι συνευθειακά, επιστρέφει ένα αν ορίζουν δεξιά στροφή και δύο αν ορίζουν αριστερή στροφή.
- Η συνάρτηση "generate_point" επιστρέφει τις συντεταγμένες ενός τυχαίου σημείου στο επίπεδο.
- Η συνάρτηση "generate_points_general_position" επιστρέφει ένα σύνολο n σημείων σε γενική θέση. Φτιάχνει τυχαία σημεία και ελέγχει αν αυτά είναι σε γενική θέση.
- Η συνάρτηση "Plot_Convex_Hull" σχεδιάζει στο επίπεδο τα σημεία x,y και το κυρτό περίβλημά τους. Αρχικά σχεδιάζει τα σημεία. Στη συνέχεια, για όλα τα σημεία του κυρτού περιβλήματος ενώνει το τρέχον σημείο με το επόμενο σημείο του κυρτού περιβλήματος. Με το "convex_hull_points[(i+1) % len(convex_hull_points)][0]" διασφαλίζουμε ότι άμα φτάσουμε στο τελευταίο σημείο θα ξανά γυρίσουμε στο πρώτο σημείο ώστε να κλείσει το περίβλημα. Αποθηκεύει το plot στο αντίστοιχο αρχείο.
- Η συνάρτηση "generate_collinear_points" φτιάχνει n σημεία στο επίπεδο με μερικά από αυτά να είναι συνευθειακά. Αρχικά, βρίσκει μέσω της έτοιμης συνάρτησης "ConvexHull" το κυρτό περίβλημα των σημείων. Βρίσκει δύο τυχαίες κορυφές του κυρτού περιβλήματος και δημιουργεί μερικά συνευθειακά σημεία αυτών των κορυφών. Επιστρέφει το σύνολο των αρχικών σημείων εμπλουτισμένο με τα συνευθειακά σημεία.

Στο αρχείο **"Grahams_Scan.py"** έχουμε τις παρακάτω συναρτήσεις:

- Η συνάρτηση "Grahams_scan" δέχεται n σημεία και βρίσκει το κυρτό τους περίβλημα. Αρχικά, διατάσσει τα σημεία σε αύξουσα λεξικογραφική σειρά. Προσθέτει στη λίστα τα πρώτα δύο σημεία του συνόλου. Υπολογίζουμε το L_άνω, όσο η L έχει πάνω από δύο σημεία και η στροφή που ορίζουν τα δύο τελευταία σημεία του L με το p δεν είναι δεξιά αφαιρούμε το τελευταίο σημείο από το L. Προσθέτουμε το p στην λίστα L. Στη συνέχεια, πάμε να υπολογίσουμε το L_κατω, διατρέχουμε όλα τα σημεία από το n-2 μέχρι το 0 και κάνουμε την ίδια διαδικασία όπως και στο L_άνω. Δεν βάζουμε το σημείο n-1 μιας και υπάρχει ήδη στην λίστα L. Αφαιρούμε από το L το τελευταίο σημείο μιας και είναι ίδιο με το πρώτο στοιχείο του L. Τέλος, επιστρέφουμε την λίστα L που περιέχει τις κορυφές του κυρτού περιβλήματος.
- Η συνάρτηση "main" ρωτάει τον χρήση αν θέλει να χρησιμοποιήσει σαν σύνολο σημείων, σημεία σε γενική θέση ή να υπάρχουν και μερικά συνευθειακά. Στη συνέχεια, βρίσκει το κυρτό περίβλημά τους με την αντίστοιχη συνάρτηση και το σχεδιάζει στο επίπεδο.

Στο αρχείο **"Wrapping.py"** έχουμε τις παρακάτω συναρτήσεις:

- Η συνάρτηση "Wrapping" εκτελεί τον αλγόριθμο του περιτυλίγματος. Η συνάρτηση δέχεται n σημεία στο επίπεδο και επιστρέφει το κυρτό τους περίβλημα. Γίνεται αρχικοποίηση της τρέχουσας κορυφής με το αριστερότερο σημείο και βάζουμε αυτή την κορυφή στο κυρτό περίβλημα. Έχουμε ένα ατέρμονο while που θα σταματήσει όταν φτάσουμε στο αρχικό σημείο. Για όλα τα υπόλοιπα σημεία t του συνόλου που είναι διαφορετικά από το τρέχον u ελέγχουμε αν τα r,u,t ορίζουν δεξιά στροφή ή αν είναι συνευθειακά με το u να είναι εσωτερικό του rt αν ισχύει κάτι από αυτά τότε θέτουμε u = t βάζουμε το u στο κυρτό περίβλημα και το αφαιρούμε από το S. Τέλος, επιστρέφουμε το κυρτό περίβλημα.
- Η συνάρτηση "main" ρωτάει τον χρήστη αν θέλει να χρησιμοποιήσει σαν σύνολο σημείων, σημεία σε γενική θέση ή να υπάρχουν και μερικά συνευθειακά. Στη συνέχεια, βρίσκει το κυρτό περίβλημά τους με την αντίστοιχη συνάρτηση και το σχεδιάζει στο επίπεδο.

Στο αρχείο **“Divide_Conquer.py”** έχουμε τις παρακάτω συναρτήσεις:

- Η συνάρτηση “get_angle” δέχεται σαν όρισμα ένα σημείο και το κέντρο του πολυγώνου και υπολογίζει την γωνία σε rad που σχηματίζεται ανάμεσα στο σημείο και το κέντρο. Χρησιμοποιείται η συνάρτηση “math.atan2(y,x)” της βιβλιοθήκης math η οποία επιστρέφει την αντίστροφη της εφαπτομένης της γωνίας που δημιουργείται από το σημείο (x,y) στο επίπεδο.
- Η συνάρτηση “sort_points_clockwise” δέχεται σαν είσοδο μερικά σημεία στο επίπεδο και επιστρέφει τα σημεία αυτά διατεταγμένα σε ορολογική διάταξη. Πρώτα υπολογίζει το κέντρο των σημείων βρίσκοντας τον μέσο όρο των συντεταγμένων όλων των σημείων στους δύο άξονες. Στη συνέχεια, χρησιμοποιούμε την συνάρτηση για την εύρεση της γωνίας του κάθε σημείου με το κέντρο και ταξινομούμε τα σημεία. Αυτό γίνεται με την συνάρτηση “sort” της python χρησιμοποιώντας την συνάρτηση “get_angle” που έχουμε φτιάξει η πρόσθεση “2*π” και το modulo με το 2*π διασφαλίζει ότι η γωνία θα είναι σε διάστημα (0,2π).
- Αντίστοιχα είναι και η συνάρτηση “sort_points_counterclockwise” που επιστρέφει τα σημεία σε διάταξη αντίθετη με αυτή του ρολογιού.
- Η συνάρτηση “Divide_points” δέχεται ένα σύνολο n σημείων το χωρίζει σε δύο σύνολα και βρίσκει τα δύο κυρτά περιβλήματα. Αρχικά, χωρίζει τα σημεία σε δύο σύνολα το πρώτο σύνολο θα έχει τα σημεία με δείκτη από το μηδέν μέχρι το άνω ακέραιος του n/2 και το δεύτερο σύνολο θα είναι από το άνω ακέραιος του n/2+1 μέχρι το n-1. Στη συνέχεια, βρίσκει τα κυρτά περιβλήματα των δύο συνόλων A,B με την έτοιμη συνάρτηση της python “ConvexHull” και επιστρέφει τα δύο κυρτά περιβλήματα.
- Η συνάρτηση “upper_bridge” δέχεται δύο κυρτά περιβλήματα και επιστρέφει την άνω γέφυρα που τα ενώνει. Αρχικά ταξινομούμε τα σημεία του αριστερότερου κυρτού περιβλήματος A με ορολογική διάταξη και τα σημεία του B με διάταξη αντίθετη αυτής του ρολογιού. Βρίσκουμε το δεξιότερο σημείο του A και το αριστερότερο του B. Διατρέχουμε τα σημεία του A και του B και όταν η ευθεία που σχηματίζουν τα τρέχοντα σημεία δεν περνάει μέσα από τα δύο πολύγωνα τότε έχουμε βρει την άνω γέφυρα.
- Αντίστοιχα έχουμε και την συνάρτηση “lower_bridge” η οποία βρίσκει την κάτω γέφυρα των δύο κυρτών περιβλημάτων. Η μόνη διαφορά αυτής της συνάρτησης είναι ότι ταξινομούμε και τα δύο σύνολα σημείων με φορά αντίθετης αυτής του ρολογιού και αυξάνουμε τον δείκτη των σημείων του B ενώ μειώνουμε αυτόν του A. Και σταματάμε πάλι όταν η ευθεία που σχηματίζουν τα τρέχοντα σημεία δεν περνάει μέσα από τα δύο πολύγωνα.
- Η συνάρτηση “Convex_hull_Divide_Conquer” δέχεται ένα σύνολο σημείων και επιστρέφει το κυρτό τους περίβλημα. Αρχικά, χωρίζει το σύνολο των σημείων στα δύο και βρίσκει τα κυρτά περιβλήματα στα δύο σύνολα (συνάρτηση “divide_points”) και στη συνέχεια βρίσκει την άνω και κάτω γέφυρα με τις αντίστοιχες συναρτήσεις. Αποθηκεύει στο hull την ένωση όλων αυτών των σημείων και καλεί την συνάρτηση “ConvexHull” της python και βρίσκει το κυρτό περίβλημα των σημείων και το επιστρέφει.
- Η συνάρτηση “main” ρωτάει τον χρήση αν θέλει να χρησιμοποιήσει σαν σύνολο σημείων, σημεία σε γενική θέση ή να υπάρχουν και μερικά συνευθειακά σημεία. Στη συνέχεια, βρίσκει το κυρτό περίβλημά τους με την αντίστοιχη συνάρτηση και το σχεδιάζει στο επίπεδο.

Στην συγκεκριμένη συνάρτηση έχει χρησιμοποιηθεί η έτοιμη συνάρτηση “ConvexHull” της βιβλιοθήκης “scipy.spatial” η οποία βρίσκει το κυρτό περίβλημα ενός συνόλου σημείου σε τρεις διαστάσεις. Έχουμε την έτοιμη συνάρτηση “math.atan2(y,x)” της βιβλιοθήκης math η οποία επιστρέφει την αντίστροφη της εφαπτομένης της γωνίας που δημιουργείται από το σημείο (x,y) στο επίπεδο.

Στο αρχείο **“Quick_Hull.py”** έχουμε τις παρακάτω συναρτήσεις:

- Η συνάρτηση “find_distance” υπολογίζει την απόσταση ενός σημείου από μία ευθεία.
- Η συνάρτηση “find_furthest_point” βρίσκει το σημείο που έχει την μεγαλύτερη απόσταση από μία ευθεία.
- Η συνάρτηση “points_right_of_line” επιστρέφει τα σημεία που βρίσκονται δεξιά από μία ευθεία.
- Η συνάρτηση “find_min_max_y” δέχεται ένα σύνολο σημείων και επιστρέφει αυτό με την μεγαλύτερη και την μικρότερη τεταγμένη. Χρειαζόμαστε την συνάρτηση για να βρούμε τα ακραία σημεία.
- Η συνάρτηση “point_inside_quadrilateral” βρίσκει τα σημεία που δεν είναι εσωτερικά ενός τετράπλευρου. Χρειαζόμαστε την συνάρτηση για να αγνοήσουμε τα σημεία που είναι εσωτερικά του τετράπλευρου των τεσσάρων ακραίων σημείων. Για όλα τα σημεία προσθέτουμε στην λίστα τα σημεία που έχουν CCW <= 0 με τις κορυφές του τετράπλευρου. Επιστρέφουμε την λίστα των σημείων.
- Η συνάρτηση “QuickHull” εκτελεί τον αλγόριθμο Quick hull. Για πάνω από τρία σημεία ακολουθούμε την παρακάτω διαδικασία. Βρίσκουμε αρχικά το σημείο Γ που απέχει περισσότερο από τις κορυφές Α,Β. Βρίσκουμε τα σημεία που είναι δεξιά της ευθείας ΑΓ και της ευθείας ΓΒ και καλούμε αναδρομικά την συνάρτηση.
- Η συνάρτηση “Convex_hull_QuickHull” βρίσκει το κυρτό περίβλημα ενός συνόλου σημείων. Δέχεται σαν όρισμα το σύνολο των σημείων και τα τέσσερα ακραία σημεία του κυρτού περιβλήματος. Αρχικά, θα αναιρέσουμε από το σύνολο των σημείων τα σημεία που είναι εσωτερικά του τετράπλευρου των ακραίων κορυφών. Στη συνέχεια, εκτελούμε τον αλγόριθμο Quick hull στις τέσσερις περιοχές, προσθέτουμε τα σημεία στο κυρτό περίβλημα και στο τέλος αφαιρούμε τα διπλότυπα. Επιστρέφουμε το κυρτό περίβλημα.
- Η συνάρτηση “main” ρωτάει τον χρήση αν θέλει να χρησιμοποιήσει σαν σύνολο σημείων σημεία σε γενική θέση ή να υπάρχουν και μερικά συνευθειακά, ταξινομεί τα σημεία σε αύξουσα λεξικογραφική σειρά ως προς την τετμημένη. Άρα το πρώτο σημείο είναι αυτό με το μικρότερο x και το τελευταίο σημείο της λίστας είναι αυτό με το μεγαλύτερο x. Στη συνέχεια, με την αντίστοιχη συνάρτηση βρίσκουμε τα σημεία με την μικρότερη και την μεγαλύτερη τεταγμένη. Έτσι, έχουμε βρει τα τέσσερα ακραία σημεία του κυρτού περιβλήματος. Τέλος, καλούμε την συνάρτηση για την εύρεση του κυρτού περιβλήματος και το σχεδιάζουμε στο επίπεδο με την αντίστοιχη συνάρτηση.

A5

CONVEX HULL - 3D:

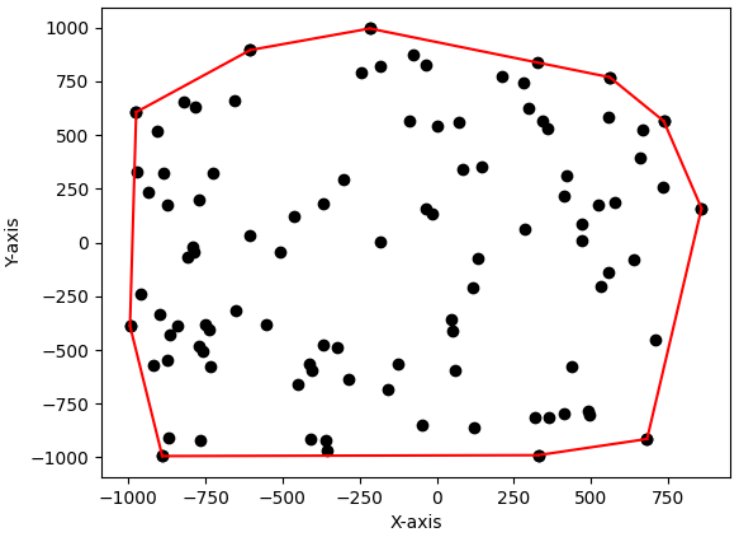
Στο αρχείο **“Convex_Hull_3D.py”** έχουμε τις παρακάτω συναρτήσεις:

- Η συνάρτηση “Convex_hull_3D” βρίσκει το κυρτό περίβλημα n σημείων σε τρεις διαστάσεις. Αρχικά, καλεί την έτοιμη συνάρτηση “ConvexHull” και βρίσκει το κυρτό περίβλημά τους. Στη συνέχεια, δημιουργείται ένα νέο σχήμα “fig” με ένα τρισδιάστατο άξονα. Καλούμε την συνάρτηση “scatter” η οποία απεικονίζει το σύνολο των σημείων στο σχήμα. Για κάθε τρίγωνο στο κυρτό περίβλημα σχεδιάζονται οι κορυφές του τριγώνου με μαύρες γραμμές χρησιμοποιώντας την συνάρτηση “plot”. Τέλος, αποθηκεύουμε το γράφημα σε ένα αρχείο.
- Έχουμε την συνάρτηση “main” που φτιάχνει n τυχαία σημεία στις τρεις διαστάσεις και μετά καλεί την συνάρτηση για να βρει το κυρτό περίβλημα των σημείων.

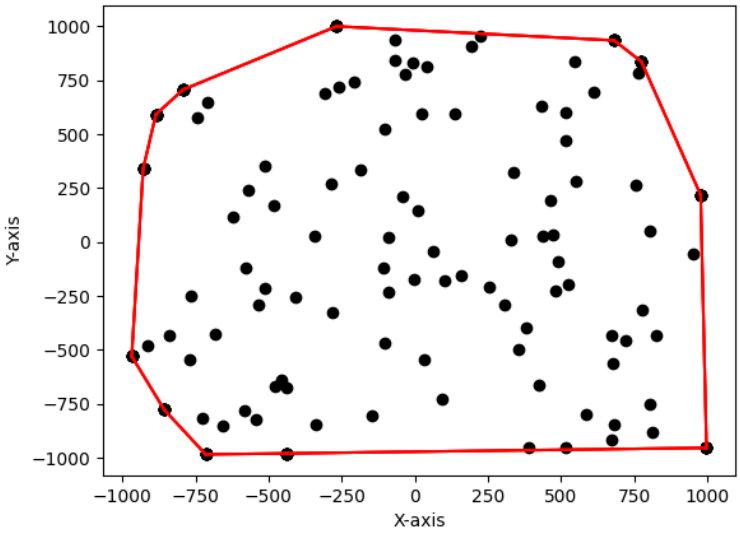
Στην συγκεκριμένη συνάρτηση έχει χρησιμοποιηθεί η έτοιμη συνάρτηση “ConvexHull” της βιβλιοθήκης “scipy.spatial” η οποία βρίσκει το κυρτό περίβλημα ενός συνόλου σημείου σε τρεις διαστάσεις. Η συνάρτηση δέχεται μία λίστα από σημεία ως είσοδο, όπου κάθε σημείο αναπαρίσταται από έναν πίνακα με τις συντεταγμένες του σημείου. Έπειτα, επιστρέφει το κυρτό περίβλημα των σημείων.

A2

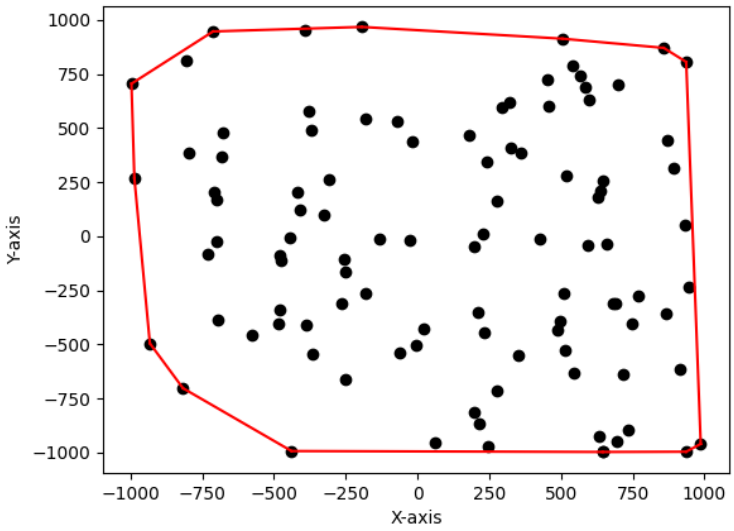
Για τον Graham's Scan:



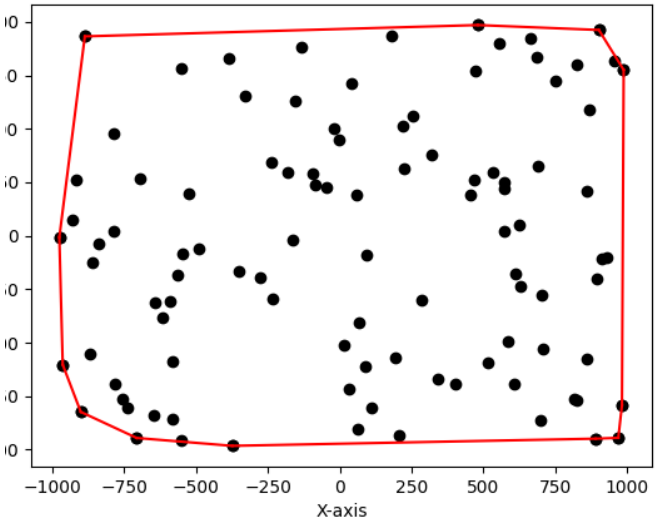
Για τον Quick Hull:



Για τον αλγόριθμο περιτυλίγματος:



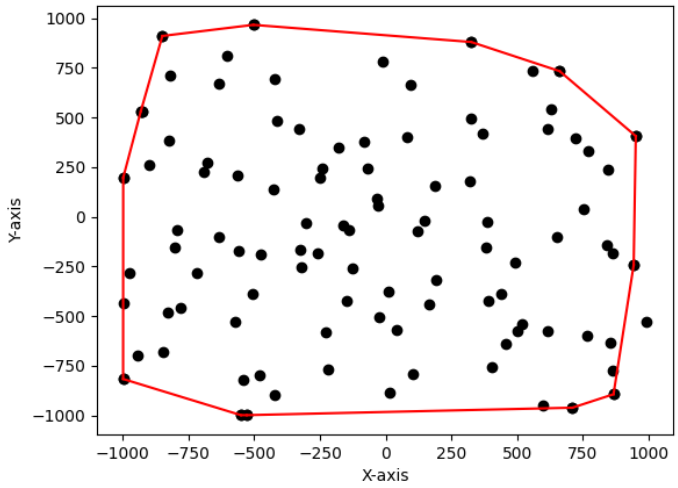
Για τον Divide and Conquer:



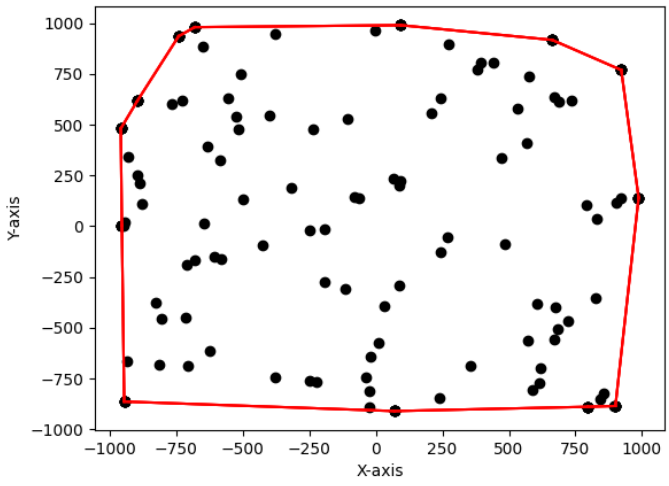
Δεν παρατηρούμε διαφορά στα αποτελέσματα των τεσσάρων αλγορίθμων. Έχουμε εκτελέσει και τους τέσσερις αλγορίθμους σε ένα σύνολο 100 τυχαίων σημείων σε γενική θέση.

Α3

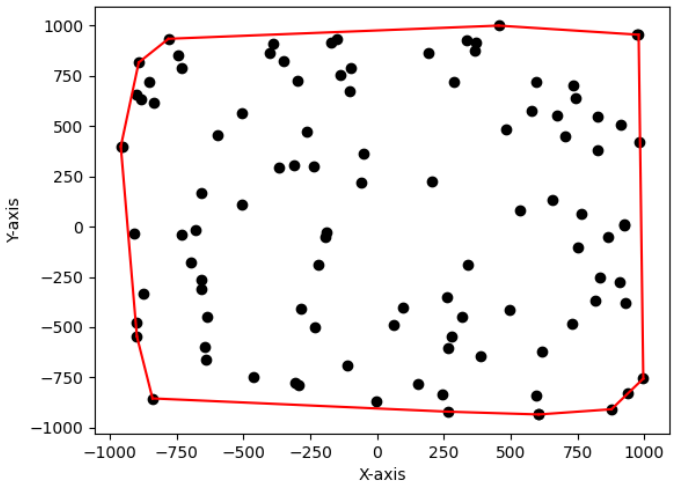
Για τον Graham's Scan:



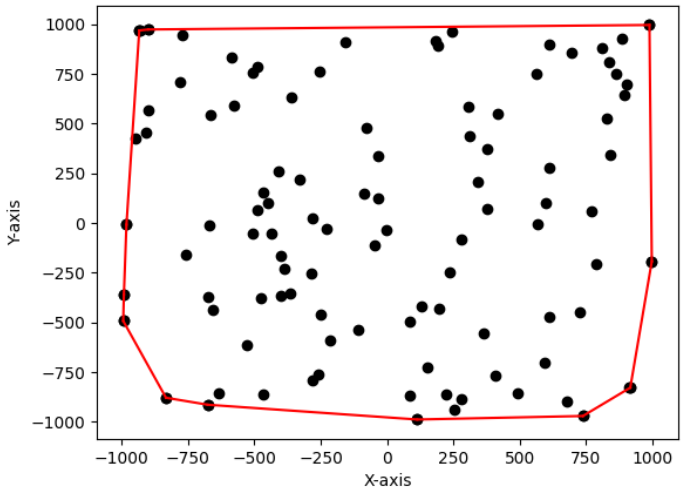
Για τον Quick Hull:



Για τον αλγόριθμο περιτυλίγματος:



Για τον Divide and Conquer:



Παρατηρούμε ότι όλοι αλγόριθμοι εκτός του Graham's Scan βρίσκουν σωστά το κυρτό περίβλημα των σημείων ακόμα και αν μερικά από αυτά είναι συνευθειακά. Ο αλγόριθμος Graham's Scan από την άλλη σε κάποιες περιπτώσεις αφήνει έξω από το κυρτό περίβλημα ένα σημείο το οποίο θα έπρεπε να είναι. Δεν παρατηρούμε σε κανέναν από τους αλγορίθμους να αφήνουν ανοικτό το κυρτό περίβλημα ή να προσθέτουν εσωτερικά σημεία στο κυρτό περίβλημα όπως θα έκανε ο εξαντλητικός αλγόριθμος σε αυτή την περίπτωση.

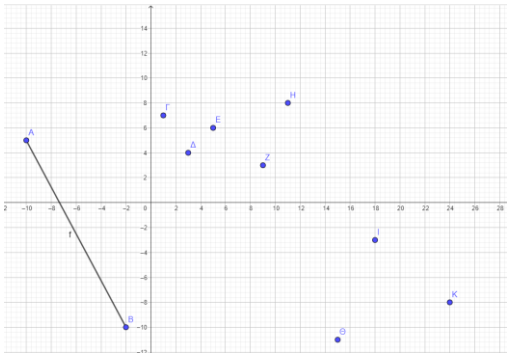
A4

Για τα σημεία p1 = (−10, 5), p2 = (−2, −10), p3 = (1, 7), p4 = (3, 4), p5 = (5, 6), p6 = (9, 3), p7 = (11, 8), p8 = (15, −11), p9 = (18, −3), p10 = (24, −8) και για τον αλγόριθμο Graham's Scan.

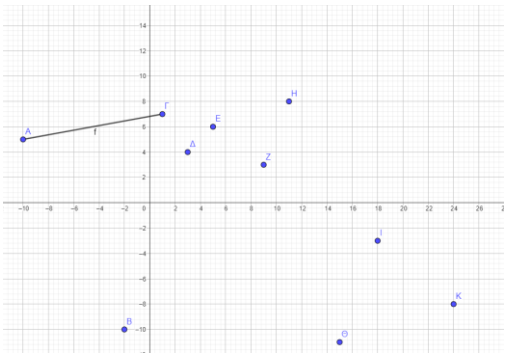
```
sdi2000014@LAPTOP-CVD600JI:~/c_programs/Ypologistikh_gewmetria/Convex_Hull$ python3 Grahams_Scan.py
L_anw
[(-10, 5), (-2, -10)]
[(-10, 5), (1, 7)]
[(-10, 5), (1, 7), (3, 4)]
[(-10, 5), (1, 7), (5, 6)]
[(-10, 5), (1, 7), (5, 6), (9, 3)]
[(-10, 5), (1, 7), (11, 8)]
[(-10, 5), (1, 7), (11, 8), (15, -11)]
[(-10, 5), (1, 7), (11, 8), (18, -3)]
[(-10, 5), (1, 7), (11, 8), (24, -8)]

L_katw
[(-10, 5), (1, 7), (11, 8), (24, -8), (18, -3)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (11, 8)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (9, 3)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (5, 6)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (3, 4)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (3, 4), (1, 7)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (-2, -10)]
[(-10, 5), (1, 7), (11, 8), (24, -8), (15, -11), (-2, -10), (-10, 5)]
```

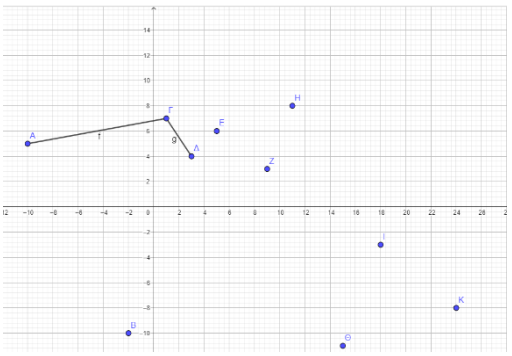
Αρχικά ταξινομεί τα σημεία σε αύξουσα λεξικογραφική σειρά. Στην εικόνα παρατηρούμε την λίστα L που περιέχει τα σημεία του κυρτού περιβλήματος από την αρχή μέχρι το τέλος του αλγορίθμου. Αφού πλέον το L έχει τουλάχιστον δύο σημεία πάμε και υπολογίζουμε το CCW των δύο τελευταίων σημείων του L με το επόμενο σημείο. Αν η στροφή που ορίζουν τα τρία σημεία δεν είναι δεξιά τότε αφαιρούμε το τελευταίο σημείο του κυρτού περιβλήματος. Στη συνέχεια, προσθέτουμε το νέο σημείο στο τέλος της λίστας και ακολουθούμε αυτή την διαδικασία για όλα τα σημεία από 0...n-1. Έτσι έχουμε βρει το άνω περίβλημα ακολουθούμε την ίδια διαδικασία και για το κάτω περίβλημα μόνο που διατρέχουμε τα σημεία από n-2...0. Για το κάτω περίβλημα η λίστα L έχει ήδη τα σημεία από το L_πάνω, για αυτό τον λόγο δεν χρειάζεται να προσθέσουμε το τελευταίο σημείο (n-1) αφού υπάρχει ήδη στη λίστα έτσι ξεκινάμε προσθέτοντας τα σημεία n-2 και n-3 και ακολουθούμε την ίδια διαδικασία όπως και στο άνω περίβλημα. Στο τέλος η λίστα L θα περιέχει τις κορυφές του κυρτού περιβλήματος των σημείων.



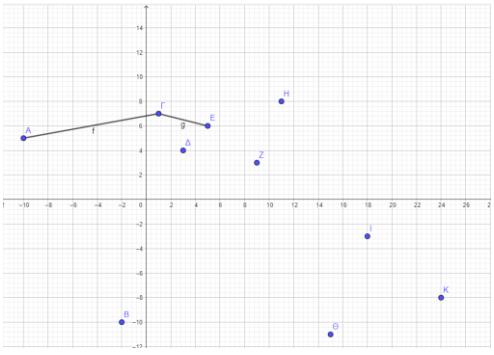
Βάζουμε τα πρώτα δύο σημεία Α, Β στη λίστα του κυρτού περιβλήματος.



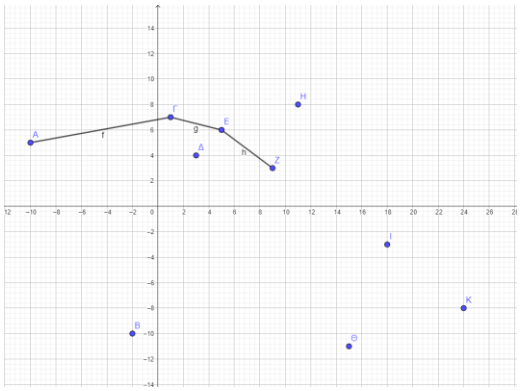
Ελέγχουμε το σημείο Γ και από τη στιγμή που η στροφή που ορίζει με το Α,Β δεν είναι δεξιά αφαιρούμε το Β και προσθέτουμε το Γ.



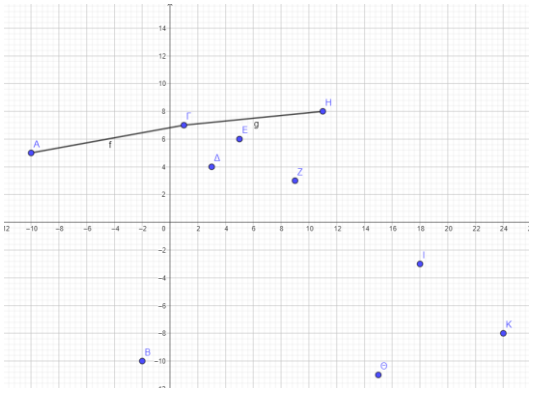
Ελέγχουμε το Δ η στροφή που ορίζει με το Α,Γ είναι δεξιά άρα δεν αφαιρούμε το Γ και απλά προσθέτουμε το Δ.



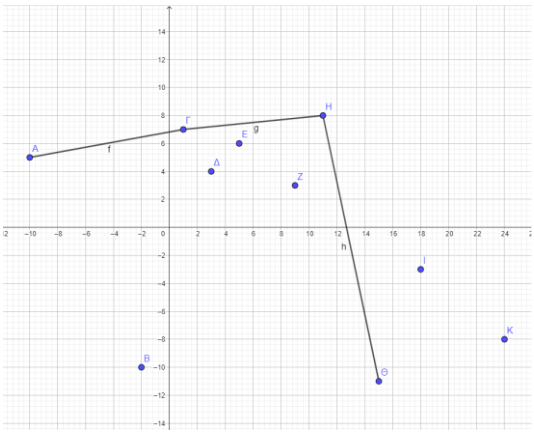
Ελέγχουμε την στροφή του Γ,Δ με το Ε και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Δ και προσθέτουμε το Ε.



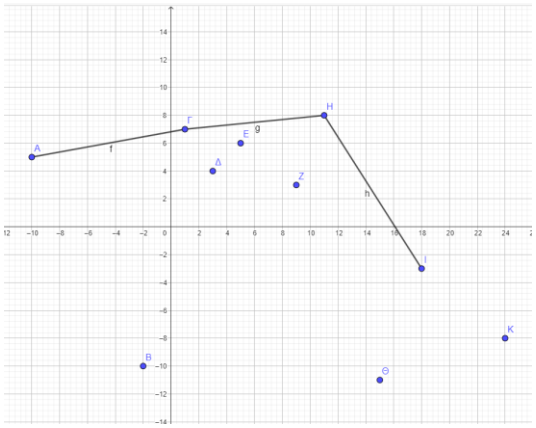
Ελέγχουμε την στροφή του Γ,Ε με το Z και από τη στιγμή που η στροφή που ορίζουν είναι δεξιά δεν αφαιρούμε το Ε και προσθέτουμε το Z.



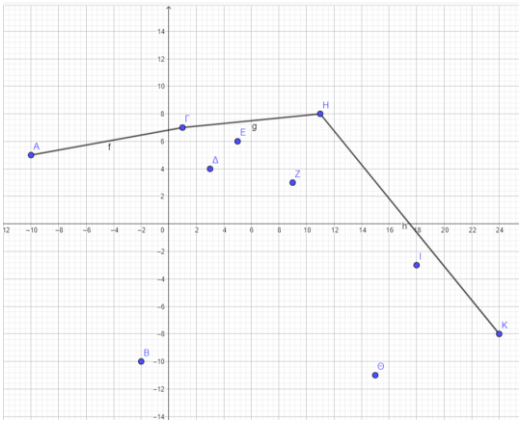
Ελέγχουμε την στροφή του Ε,Z με το Η και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Ε. Μετά ελέγχουμε την στροφή του Γ,Z με το Η η στροφή δεν είναι δεξιά άρα αφαιρούμε το Z και προσθέτουμε το Η.



Ελέγχουμε την στροφή του Γ,Η με το Θ και από τη στιγμή που η στροφή που ορίζουν είναι δεξιά δεν αφαιρούμε το Η και προσθέτουμε το Θ.



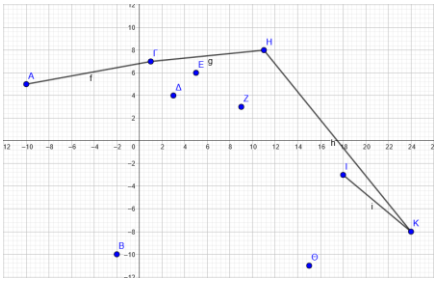
Ελέγχουμε την στροφή του Η,Θ με το Ι και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Θ. Προσθέτουμε το Ι



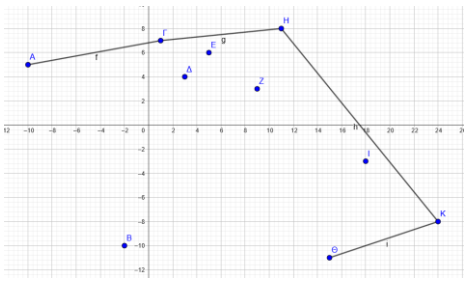
Ελέγχουμε την στροφή του Η,Ι με το Κ και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Ι και προσθέτουμε το Κ.

Με αυτόν τον τρόπο έχουμε βρει το άνω κυρτό περίβλημα. Με την ίδια διαδικασία θα βρούμε και το κάτω με την μόνη διαφορά ότι θα ξεκινήσουμε τη λούπα μας από το προτελευταίο σημείο του συνόλου μιας και το τελευταίο υπάρχει ήδη στην λίστα του κυρτού περιβλήματος.

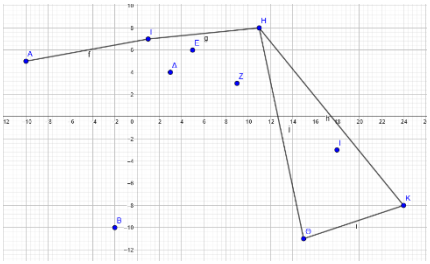
Για την εύρεση του κάτω κυρτού περιβλήματος:



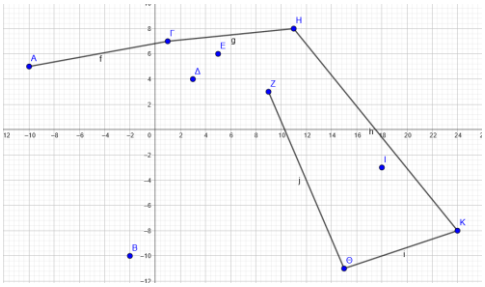
Ελέγχουμε την στροφή του Η,Κ με το Ι και από τη στιγμή που η στροφή που ορίζουν είναι δεξιά δεν αφαιρούμε το Κ και προσθέτουμε το Ι.



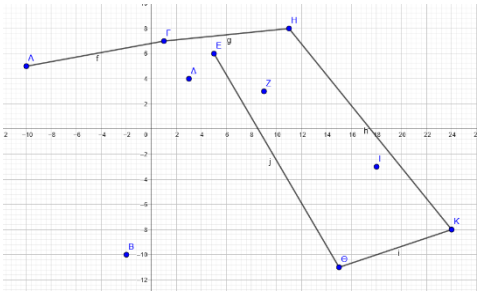
Ελέγχουμε την στροφή του Κ,Ι με το Θ και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Ι και προσθέτουμε το Θ.



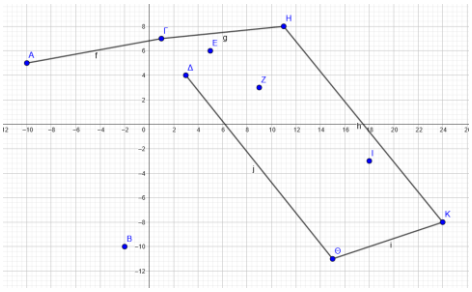
Ελέγχουμε την στροφή του Κ,Θ με το Η και από τη στιγμή που η στροφή που ορίζουν είναι δεξιά δεν αφαιρούμε το Θ και προσθέτουμε το Η.



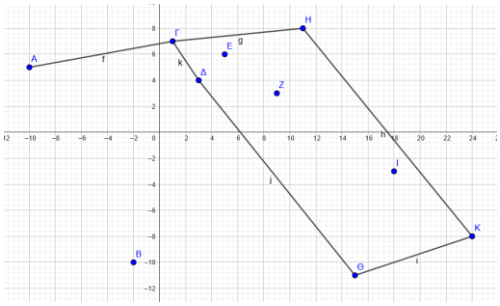
Ελέγχουμε την στροφή του Θ,Η με το Ζ και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Η και προσθέτουμε το Ζ.



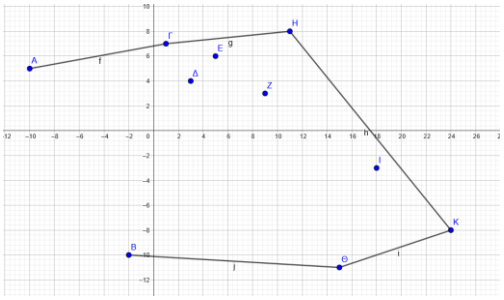
Ελέγχουμε την στροφή του Θ,Ζ με το Ε και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Ζ και προσθέτουμε το Ε.



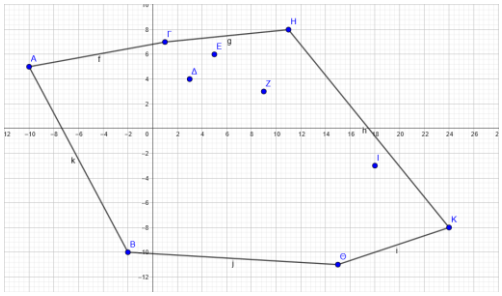
Ελέγχουμε την στροφή του Ε,Θ με το Δ και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Ε και προσθέτουμε το Δ.



Ελέγχουμε την στροφή του Θ, Δ με το Γ και από τη στιγμή που η στροφή που ορίζουν είναι δεξιά δεν αφαιρούμε το Δ και προσθέτουμε το Γ.



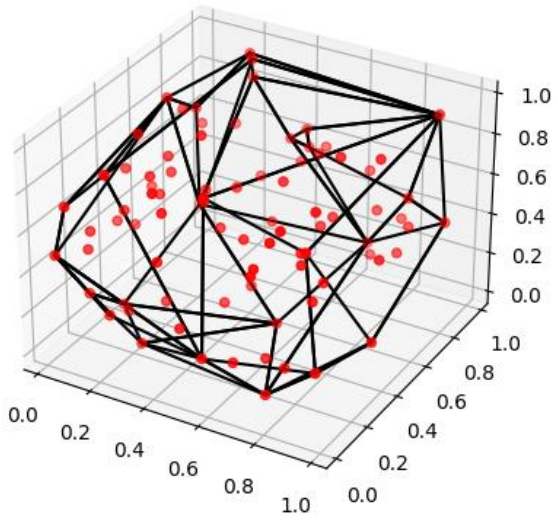
Ελέγχουμε την στροφή του Δ, Γ με το Β και από τη στιγμή που η στροφή που ορίζουν δεν είναι δεξιά αφαιρούμε το Γ. Η στροφή του Θ, Δ με το Β δεν είναι δεξιά άρα αφαιρούμε το Δ και προσθέτουμε το Β.



Ελέγχουμε την στροφή του Θ, Β με το Α και από τη στιγμή που η στροφή που ορίζουν είναι δεξιά δεν αφαιρούμε το Β και προσθέτουμε το Α.

Έτσι στο τέλος η λίστα L θα περιέχει τις κορυφές του κυρτού περιβλήματος. Θα αφαιρέσουμε από το L το τελευταίο σημείο μιας και υπάρχει δύο φορές, αφού ο αλγόριθμος το πρόσθεσε και στη λίστα και στην εύρεση του L_άνω αλλά και του L_κάτω.

A6



Υλοποιήσαμε τον αλγόριθμο Graham's Scan σε τρεις διαστάσεις για ένα σύνολο 85 σημείων σε γενική θέση.

ΥΛΟΠΟΙΗΣΗ - Β

B1

LINEAR PROGRAMMING:

Στο αρχείο **"seidel_algorithm.py"** έχουμε τις παρακάτω συναρτήσεις:

- Η συνάρτηση "concatenation_A" υπολογίζει για κάποιον περιορισμό j τον νέο περιορισμό H_j' που είναι η σύζευξη του περιορισμού H_j με $H_i = 0$. Θα έχουμε πλέον μία μεταβλητή. Η συνάρτηση λοιπόν θα επιστρέφει έναν ακέραιο που θα είναι ο συντελεστής της μεταβλητής x_1 στον περιορισμό H_j'.
- Η συνάρτηση "new_b" υπολογίζει το νέο διάνυσμα b' των περιορισμών μετά την σύζευξη του περιορισμού H_j με $H_i = 0$ όπου j = 0,1,…,i-1. Η συνάρτηση επιστρέφει το νέο διάνυσμα b' μεγέθους i.

Οι τύποι με τους οποίους υπολογίζουμε τους συντελεστές των νέων περιορισμών τους έχω υπολογίσει με τον παρακάτω τρόπο:

Για το αρχικό πρόβλημα γραμμικού προγραμματισμού [με $d = 2$, $n = \#$ περιορισμών] έχουμε:

$$A = \begin{bmatrix} a[0][0] & a[0][1] \\ a[1][0] & a[1][1] \\ \vdots & \\ a[n-1][0] & a[n-1][1] \end{bmatrix} \quad b^T = \begin{bmatrix} b[0] \\ \vdots \\ b[n-1] \end{bmatrix}$$

$$f^T = \begin{bmatrix} f[0] \\ f[1] \end{bmatrix}$$

Για κάποιους περιορισμούς H_i, H_j $i \neq j$ πάμε να υπολογίσουμε του H_j' που είναι η σύζευξη του $H_i = 0$ με του H_j.

$H_i = 0 \Rightarrow A[i][0] \cdot x_0 + A[i][1] \cdot x_1 = b[i]$

$\Leftrightarrow$ $\boxed{x_0 = \frac{b[i]}{A[i][0]} - \frac{A[i][1]}{A[i][0]} \cdot x_1}$ (1)

$H_j \Rightarrow A[j][0] \cdot x_0 + A[j][1] \cdot x_1 \leq b[j]$

$\overset{(1)}{\Leftarrow} A[j][0] \cdot \left[\frac{b[i]}{A[i][0]} - \frac{A[i][1]}{A[i][0]} \cdot x_1 \right] + A[j][1] \cdot x_1 \leq b[j]$

$\Leftrightarrow \left[A[j][1] - \frac{A[j][0] \cdot A[i][1]}{A[i][0]} \right] \cdot x_1 \leq b[j] - A[j][0] \cdot \frac{b[i]}{A[i][0]}$

$\hookrightarrow H'_j$ με $d-1 \ (=1)$ μεταβλητές.

Για την f: $f[0] \cdot x_0 + f[1] \cdot x_1$

$\overset{(1)}{=} f[0] \cdot \left[\frac{b[i]}{A[i][0]} - \frac{A[i][1]}{A[i][0]} \cdot x_1 \right] + f[1] \cdot x_1$

$= \left[f[1] - \frac{A[i][1]}{A[i][0]} \cdot f[0] \right] \cdot x_1 + \frac{f[0] \cdot b[i]}{A[i][0]}$

Άρα: $A'[j] = A[j][1] - \frac{A[j][0] \cdot A[i][1]}{A[i][0]}$

$b'[j] = b[j] - A[j][0] \cdot \frac{b[i]}{A[i][0]}$

$f' = f'[0] = f[1] - \frac{A[i][1]}{A[i][0]} \cdot f[0]$

με $j = 0, \dots, i-1, i = d+1, \dots, n-1$.

Οπότε πλέον λύνουμε το πρόβλημα γραμμικού προγραμματισμού με τους νέους περιορισμούς A', b' και την νέα αντικειμενική συνάρτηση f'.

- Η συνάρτηση “seidel_algorithm” εκτελεί τον αλγόριθμο για την επίλυση ενός προβλήματος γραμμικού προγραμματισμού. Αρχικά, κρατάμε τους συντελεστές των πρώτων d+1 περιορισμών (0,1..,d). Και λύνουμε αυτό το πρόβλημα γραμμικού προγραμματισμού με την συνάρτηση “linprog”. Αν το πρόβλημα είναι maximization τότε θα το μετατρέψουμε σε πρόβλημα ελαχιστοποίησης (αλλάζοντας τα πρόσημα των συντελεστών της αντικειμενικής συνάρτησης) γιατί η συνάρτηση linprog λύνει προβλήματα ελαχιστοποίησης. Αφού θα έχουμε βρει το x_{d+1}^*, έχουμε αρχικοποιήσει και το διάνυσμα x, πάμε και αποθηκεύουμε το x_{d+1}^* στη θέση i-1=d. Για όλους τους υπόλοιπους περιορισμούς i = d+1,d+2,…,n-1 ακολουθούμε την παρακάτω διαδικασία. Ελέγχουμε πρώτα αν η λύση x[i-1] είναι “None” αν αυτό δεν ισχύει: ελέγχουμε αν η λύση x[i-1] ικανοποιεί τον περιορισμό H_i. Αν τον ικανοποιεί τότε θέτουμε x[i] = x[i-1]. Αν η x[i-1] δεν ικανοποιεί τον H_i ή x[i-1] = None τότε: καλούμε τις προηγούμενες συναρτήσεις και υπολογίζουμε τους νέους συντελεστές των περιορισμών, το b’ και τους νέους συντελεστές της αντικειμενικής συνάρτησεις με μία μεταβλητή πλέον. Στη συνέχεια, λύνουμε το νέο πρόβλημα γραμμικού προγραμματισμού με την συνάρτηση “linprog”.
  Η συνάρτηση “linprog” θα επιστρέψει την τιμή του x_1 την τιμή του x_0 την βρίσκουμε από τον τύπο (1) που αναφέρεται πιο πάνω. Τέλος, αποθηκεύουμε στο x[i] το αποτέλεσμα και το επιστρέφουμε.

Στην συγκεκριμένη συνάρτηση έχει χρησιμοποιηθεί η έτοιμη συνάρτηση “linprog” της βιβλιοθήκης “scipy.optimize” η οποία βρίσκει την λύση ενός προβλήματος ελαχιστοποίησης γραμμικού προγραμματισμού. Για ορίσματα χρειάζεται:

i. Τους συντελεστές της αντικειμενικής συνάρτησης. Πρέπει να είναι πρόβλημα ελαχιστοποίησης αλλιώς απλά αντιστρέφουμε τα πρόσημα των συντελεστών των x_i.
ii. Τους συντελεστές των περιορισμών
iii. Το διάνυσμα b
iv. Και τις τιμές που μπορούν να πάρουν οι μεταβλητές x_1, x_2 στην συγκεκριμένη περίπτωση έχουμε: $x_1, x_2 \geq 0$.

- Η συνάρτηση “main” αρχικοποιεί τους συντελεστές των περιορισμών στον πίνακα A, το διάνυσμα b, τους συντελεστές της αντικειμενικής συνάρτησης στον πίνακα f και έχουμε και μία λίστα L στην οποία αποθηκεύουμε την φορά της κάθε ανίσωσης των περιορισμών (<= ή >=). Στην περίπτωση που υπάρχει περιορισμός με φορά “>=” πρέπει να μετατρέψουμε το πρόβλημα στην τυπική του μορφή. Οπότε απλά αντιστρέφουμε τα πρόσημα των συντελεστών του συγκεκριμένου περιορισμού και του αντίστοιχου b. Στη συνέχεια καλούμε την συνάρτηση για να λύσει το πρόβλημα με την μέθοδο Seidel. Η συνάρτηση “seidel_algorithm” θα επιστρέψει την δομή της βέλτιστης λύσης του προβλήματος. Έπειτα, υπολογίζουμε την βέλτιστη τιμή της αντικειμενικής συνάρτησης. Τέλος, τυπώνουμε την δομή της βέλτιστης λύσης και την βέλτιστη τιμής της f. Αν το πρόβλημα είναι μεγιστοποίησης τυπώνουμε την αντίθετη τιμή της βέλτιστης τιμής της f μιας και η συνάρτηση “seidel_algorithm” έχει επιστρέψει λύση προβλήματος ελαχιστοποίησης (αφού χρησιμοποιούμε την συνάρτηση “linprog”).

B2

Η δομή της βέλτιστης λύσης που βρίσκει ο αλγόριθμος στο συγκεκριμένο παράδειγμα είναι (x,y) = (12,4) και η μέγιστη τιμή της αντικειμενικής συνάρτησης κάτω από τους συγκεκριμένους περιορισμούς είναι: z = 12.

Γ1

ΥΛΟΠΟΙΗΣΗ - Γ

Triangulation Delaunay:

Έχουμε το αρχείο **"triangulation_delaunay.py"** που περιέχει τις συναρτήσεις για την τριγωνοποίηση Delaunay:

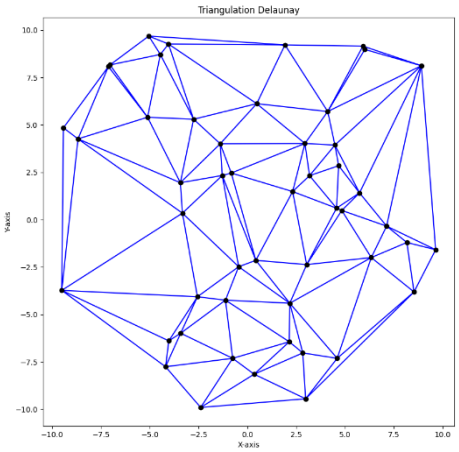
- Η συνάρτηση "InCircle" υπολογίζει το κατηγόρημα InCircle. Η συνάρτηση δέχεται σαν όρισμα τις συντεταγμένες ενός σημείου q, και μία λίστα με τρία σημεία p_1, p_2, p_3 που είναι αυτά που ορίζουν τον κύκλο. Επιστρέφει ένα αν το σημείο q είναι εξωτερικό του κύκλου που ορίζουν τα σημεία p_1, p_2, p_3 αλλιώς, επιστρέφει μηδέν.
- Η συνάρτηση "Create_Bigger_Triangle" δέχεται σαν όρισμα ένα σύνολο N σημείων, δημιουργεί ένα μεγαλύτερο τρίγωνο που περιέχει όλα τα σημεία και επιστρέφει το σύνολο των αρχικών σημείων εμπλουτισμένο με τις τρεις κορυφές του μεγαλύτερου τριγώνου. Αρχικά, βρίσκουμε την ελάχιστη και μέγιστη τιμή από την x και y συντεταγμένη και υπολογίζουμε το 'delta' που είναι το μέγιστο εύρος των παραπάνω συντεταγμένων με αυτόν τον τρόπο θα εξασφαλίσουμε ότι το τρίγωνο θα είναι αρκετά μεγάλο ώστε να περιέχει όλα τα σημεία του αρχικού συνόλου. Στη συνέχεια, φτιάχνουμε τις τρεις κορυφές του τριγώνου και τέλος επιστρέφουμε ένα σύνολο σημείων που περιέχει τα αρχικά σημεία μαζί με τις τρεις κορυφές του μεγάλου τριγώνου.
- Η συνάρτηση "Triangulation_Delaunay" δέχεται σαν όρισμα ένα σύνολο σημείων και εκτελεί τον αλγόριθμο για την τριγωνοποίηση Delaunay. Αρχικά, καλούμε την παραπάνω συνάρτηση για να διευρύνουμε το σύνολο των αρχικών σημείων και υπολογίζουμε μία αρχική τριγωνοποίηση με την έτοιμη συνάρτηση της python 'Delaunay'. Έχουμε την μεταβλητή 'simplices' στην οποία έχουμε αποθηκεύσει τα αρχικά τρίγωνα που έχουν σχεδιαστεί. Έπειτα ελέγχουμε όλα τα σημεία αν το τρέχον σημείο δεν ανήκει σε κάποιο τρίγωνο πηγαίνουμε και ελέγχουμε το επόμενο σημείο. Αν το τρέχον σημείο i είναι μέρος ενός τριγώνου, βρίσκουμε το index του τριγώνου στο οποίο ανήκει το i και στη συνέχεια βρίσκουμε το ίδιο το τρίγωνο. Έπειτα έχουμε ένα for-loop που ελέγχει όλες τις πλευρές του τρέχοντος τριγώνου. Βρίσκουμε το index του γειτονικού τριγώνου, αν δεν υπάρχει γειτονικό τρίγωνο προχωράμε στην επόμενη ακμή του τριγώνου. Βρίσκουμε την κοινή ακμή του γειτονικού και του τρέχοντος τριγώνου. Αν δεν υπάρχει κάποια γειτονική ακμή εξετάζουμε τον επόμενο γείτονα. Αν το σημείο i είναι μέσα στον κύκλο του γειτονικού τριγώνου τότε η συγκεκριμένη τριγωνοποίηση δεν είναι Delaunay οπότε πρέπει να αλλάξουμε τα ήδη σχηματισμένα τρίγωνα. Για να αλλάξουμε τα τρίγωνα στην ουσία ρολάρουμε τις κορυφές τους ώστε να αλλάξουμε την κοινή ακμή. Τέλος, αφαιρούμε όλα τα τρίγωνα που έχουν να κάνουν με το μεγάλο τρίγωνο που δημιουργήθηκε στην αρχή της συνάρτησης και επιστρέφουμε τα τρίγωνα.
- Η συνάρτηση "Plot_Triangulation" σχεδιάζει την τριγωνοποίηση Delaunay στο επίπεδο. Σχεδιάζουμε τα τρίγωνα με την συνάρτηση "triplot" της βιβλιοθήκης "matplotlib", στη συνέχεια σχεδιάζουμε τα σημεία και αποθηκεύουμε την γραφική παράσταση σε αρχείο με όνομα 'Delaunay.png'.
- Η "main" αρχικά δημιουργεί N τυχαία σημεία στο επίπεδο όπου οι τετμημένες και τεταγμένες των σημείων παίρνουν τιμές στο διάστημα (min, max), όπου οι μεταβλητές min, max έχουν οριστεί στην αρχή της main. Στη συνέχεια, καλούμε την συνάρτηση για να γίνει η τριγωνοποίηση και υπολογίζουμε και τον χρόνο που χρειάζεται για να εκτελεστεί ο αλγόριθμος. Σχεδιάζουμε την τριγωνοποίηση και εκτυπώνουμε τον χρόνο που χρειάστηκε η τριγωνοποίηση.

Στη συνάρτηση "Triangulation_Delaunay" χρησιμοποιείται η έτοιμη συνάρτηση της βιβλιοθήκης scipy "Delaunay" η οποία δέχεται σαν όρισμα ένα σύνολο σημείων και επιστρέφει την τριγωνοποίηση Delaunay.

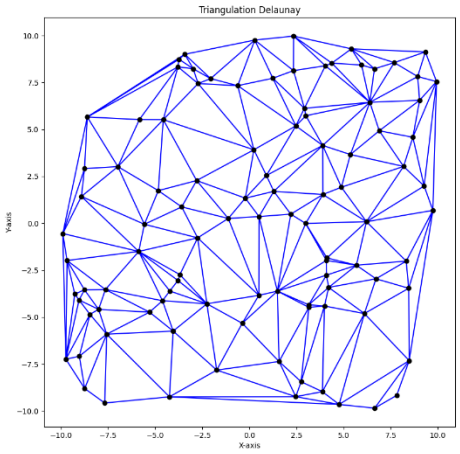
Γ2

# of points	Time (seconds)
50	0.00987
100	0.03192
500	0.71449
1000	2.43732

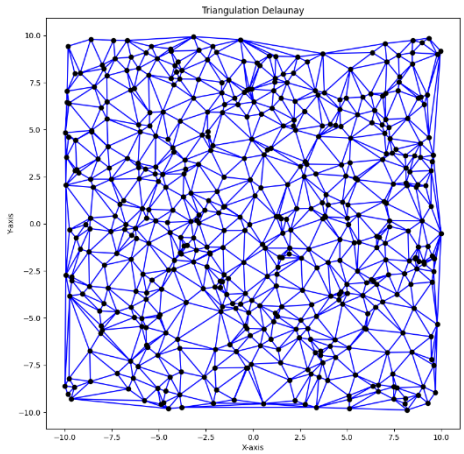
Παρατηρούμε ότι όσο αυξάνονται τα σημεία ο χρόνος εκτέλεσης του αλγορίθμου αυξάνεται εκθετικά.



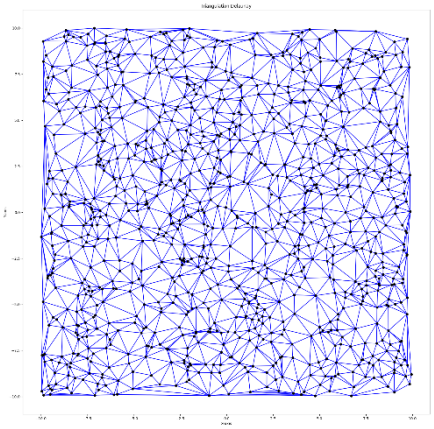
N = 50



N = 100



N = 500



N = 1000

Δ1

ΥΛΟΠΟΙΗΣΗ - Δ

KD-TRESS:

Έχουμε το αρχείο **"KD_tree_construction.py"** που περιέχει τις συναρτήσεις για τη δημιουργία ενός kd-tree:

- Έχουμε την κλάση "Node" που αντιπροσωπεύει έναν κόμβο. Η παράμετρος 'split_dim' αντιπροσωπεύει τη διάσταση κατά μήκος της οποίας χωρίζονται τα δεδομένα. Η 'split_value' αντιπροσωπεύει την διχαστική τιμή, 'left', 'right' αντιπροσωπεύουν το αριστερό και το δεξί παιδί του κόμβου αντίστοιχα. Το 'points' είναι μία παράμετρος στην οποία αποθηκεύονται οι συντεταγμένες του σημείου.
- Η συνάρτηση "KD_tree_construction" φτιάχνει το kd-tree. Δέχεται σαν όρισμα το σύνολο των σημείων και το βάθος του δένδρου. Αν το σύνολο είναι κενό δεν επιστρέφουμε κάτι. Αν το σύνολο έχει ένα σημείο επιστρέφει ένα φύλλο που θα περιέχει το μοναδικό σημείο. Υπολογίζουμε την διάσταση που θα γίνει το split 'dim = depth%2' μιας και τα σημεία είναι δυσδιάστατα η διάσταση θα παίρνει τις τιμές μηδέν ή ένα. Στη συνέχεια, ταξινομούμε τα σημεία με βάση την διάσταση ώστε να βρούμε πιο εύκολα το ενδιάμεσο σημείο. Οπότε μετά την ταξινόμηση το μεσαίο σημείο θα είναι το σημείο του οποίου ο δείκτης είναι στη μέση της ταξινομημένης λίστας. Χωρίζουμε μετά τα σημεία σε δύο σύνολά P1, P2 το πρώτο έχει τα σημεία αριστερά του μεσαίου και το δεύτερο τα σημεία δεξιά του μεσαίου. Έπειτα, υπολογίζουμε το αριστερό και το δεξιό υποδένδρο καλώντας αναδρομικά την συνάρτηση με ορίσματα τα δύο νέα σύνολα P1, P2 αντίστοιχα και έχουμε αυξήσει το βάθος κατά ένα. Τέλος, δημιουργούμε έναν νέο κόμβο με παραμέτρους την τρέχουσα διάσταση, την διχαστική τιμή, το αριστερό υπόδενδρο, το δεξιό υπόδενδρο και το μεσαίο σημείο. Και επιστρέφουμε τον κόμβο.
- Η συνάρτηση "Plot_splitting_lines" σχεδιάζει τις διχαστικές ευθείες στο επίπεδο. Αρχικά βρίσκουμε την διάσταση που γίνεται ο διαχωρισμός και την διχαστική τιμή. Αν η διάσταση είναι μηδέν θα σχεδιαστεί κάθετη ευθεία αλλιώς θα σχεδιαστεί οριζόντια. Όταν θέλουμε να σχεδιάσουμε μία κάθετη ευθεία: την σχεδιάζουμε με x σταθερό και ίσο με την διχαστική τιμή και y που να κυμαίνεται από την ελάχιστη μέχρι την μέγιστη τιμή. Στη συνέχεια, καλούμε αναδρομικά την συνάρτηση να κάνει το ίδιο για το δεξιό και το αριστερό υπόδενδρο. Αντίστοιχη διαδικασία ακολουθούμε και για τις οριζόντιες ευθείες με την διαφορά ότι τώρα το x κυμαίνεται από την ελάχιστη στη μέγιστη τιμή ενώ το y μένει σταθερό στην διχαστική τιμή.
- Η συνάρτηση "Plot_KD_tree" σχεδιάζει το kd-tree. Αρχικά, σχεδιάζει τα σημεία στη γραφική παράσταση και στη συνέχεια με την παραπάνω συνάρτηση σχεδιάζει τις διχαστικές γραμμές. Τέλος, αποθηκεύει την γραφική παράσταση στο αντίστοιχο αρχείο με όνομα "KD-tree.png".
- Η συνάρτηση "main" φτιάχνει N τυχαία σημεία στο επίπεδο όπου οι τεταγμένες και οι τετμημένες των σημείων είναι ανάμεσα στο διάστημα (min, max), όπου min, max μεταβλητές που ορίζονται στην αρχή της main. Στη συνέχεια, καλεί την συνάρτηση για την δημιουργία του kd-tree και στο τέλος με την αντίστοιχη συνάρτηση το σχεδιάζει στο επίπεδο.

Δ2

ORTHOGONAL SEARCH:

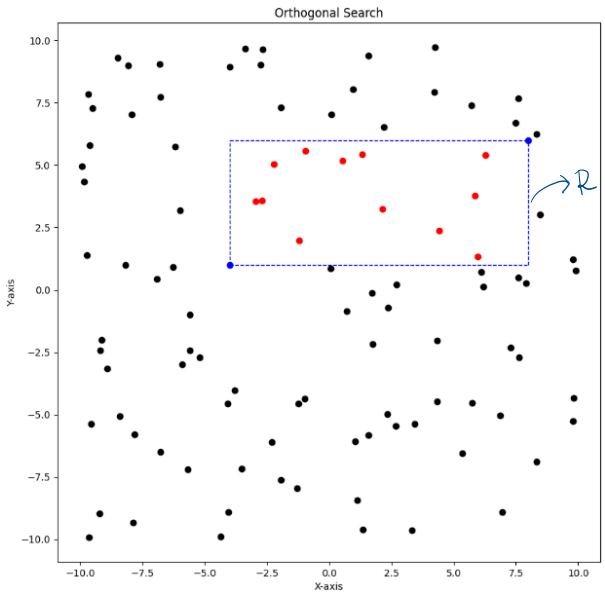
Έχουμε το αρχείο **"orthogonal_search.py"** που περιέχει τις συναρτήσεις για την αναζήτηση σε ένα kd-tree:

- Η συνάρτηση "search_kd_tree" εκτελεί ορθογώνια αναζήτηση σε ένα kd-tree. Δέχεται σαν όρισμα το σύνολο των σημείων, την περιοχή R και το βάθος του δένδρου. Αρχικά, έχουμε μία κενή λίστα στην οποία θα αποθηκεύουμε τα σημεία που θα είναι μέσα στην περιοχή R. Θα ελέγξουμε αν ο κόμβος είναι φύλλο. Αν ισχύει αυτό τσεκάρουμε αν είναι μέσα στην περιοχή R και αν ισχύει το προσθέτουμε στην λίστα με τα σημεία και επιστρέφουμε τα σημεία. Ελέγχουμε

το αριστερό και το δεξιό υπόδενδρο αντίστοιχα και καλούμε σε κάθε περίπτωση αναδρομικά την συνάρτηση και ανανεώνουμε την λίστα με τα σημεία. Στη συνέχεια, ελέγχουμε αν η ρίζα είναι μέσα στην περιοχή R και αν ισχύει αυτό την προσθέτουμε στην λίστα των σημείων. Τέλος, επιστρέφουμε τα σημεία.

- Η συνάρτηση “Plotting” σχεδιάζει τα σημεία στο επίπεδο, την περιοχή R και τα σημεία που είναι μέσα σε αυτή. Δέχεται σαν όρισμα τα σημεία του P, τα σημεία που είναι μέσα στο R και την περιοχή R. Αρχικά, σχεδιάζει τα σημεία που είναι στο P με μαύρο και όσα είναι μέσα στην περιοχή R με κόκκινο. Στην συνέχεια, σχεδιάζει τις τέσσερις ευθείες που σχηματίζουν την περιοχή R. Τέλος, αποθηκεύει την γραφική παράσταση στο αρχείο με όνομα “Orthogonal Search.png”.
- Η συνάρτηση “main” φτιάχνει N τυχαία σημεία στο επίπεδο όπου οι τεταγμένες και οι τετμημένες των σημείων είναι ανάμεσα στο διάστημα (min, max), όπου min, max μεταβλητές που ορίζονται στην αρχή της main. Στην αρχή της main ορίζεται και η περιοχή R. Στη συνέχεια, καλεί την συνάρτηση για την δημιουργία του kd-tree και την συνάρτηση για την ορθογώνια αναζήτηση σε μία περιοχή R. Τέλος, τυπώνει τα σημεία που είναι μέσα στην περιοχή R και καλεί την παραπάνω συνάρτηση που σχεδιάζει τα σημεία στο επίπεδο και αυτά που είναι μέσα στην περιοχή R.

Δ3



Έχουμε δημιουργήσει 100 τυχαία σημεία όπου οι τετμημένες και τεταγμένες τους παίρνουν τιμές στο διάστημα (-10, 10). Η περιοχή R είναι η: [-4, 1] x [8, 6], στο σχήμα είναι αυτή που έχει σχεδιαστεί με τις μπλε διακεκομμένες γραμμές και τα δύο μπλε σημεία είναι τα δύο σημεία της R. Τα κόκκινα σημεία είναι αυτά που έχει επιστρέψει η ορθογώνια αναζήτηση στην R. Οι συντεταγμένες των κόκκινων σημείων φαίνονται στην παρακάτω εικόνα.

```
Points within region R = [-4, 1] x [8, 6] :

[-2.964, 3.539]
[-1.208, 1.971]
[-2.232, 5.035]
[-2.687, 3.584]
[1.325, 5.42]
[0.546, 5.188]
[4.411, 2.359]
[2.131, 3.244]
[6.276, 5.394]
[5.964, 1.341]
[5.863, 3.77]
[-0.952, 5.577]

We have found 12 points within region R = [-4, 1] x [8, 6]
```