ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ

Υπολογιστική εργασία | Εαρινό εξάμηνο 2023-2024

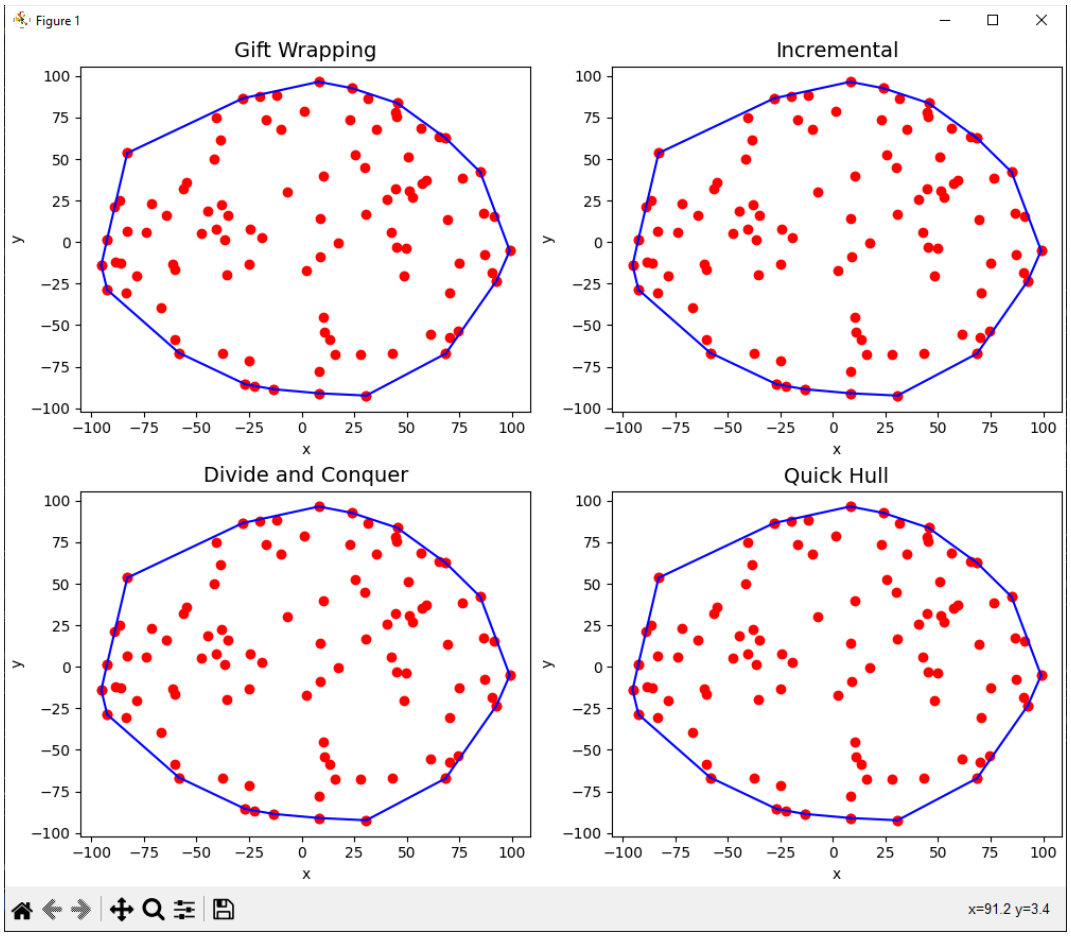
- **Κωνσταντίνος Πετράκης 1115202100154**

Μέρος Α: Κυρτό περίβλημα

Για το μέρος αυτό, όλες οι συναρτήσεις σχετιζόμενες με την εύρεση του κυρτού περιβλήματος περιέχονται στο αρχείο convex_hull_algs.py. Χρησιμοποιούνται αρκετές από τις συναρτήσεις που ορίζονται στο επιμέρους αρχείο helpFunctions.py οι οποίες, μεταξύ άλλων, επιτελούν βασικές λειτουργίες (παραδείγματος χάριν τον υπολογισμό ενός κατηγορήματος προσανατολισμού) που είναι απαραίτητες για την εκτέλεση των παρακάτω αλγορίθμων. Οι συναρτήσεις αυτού του αρχείου, εξηγούνται αργότερα.

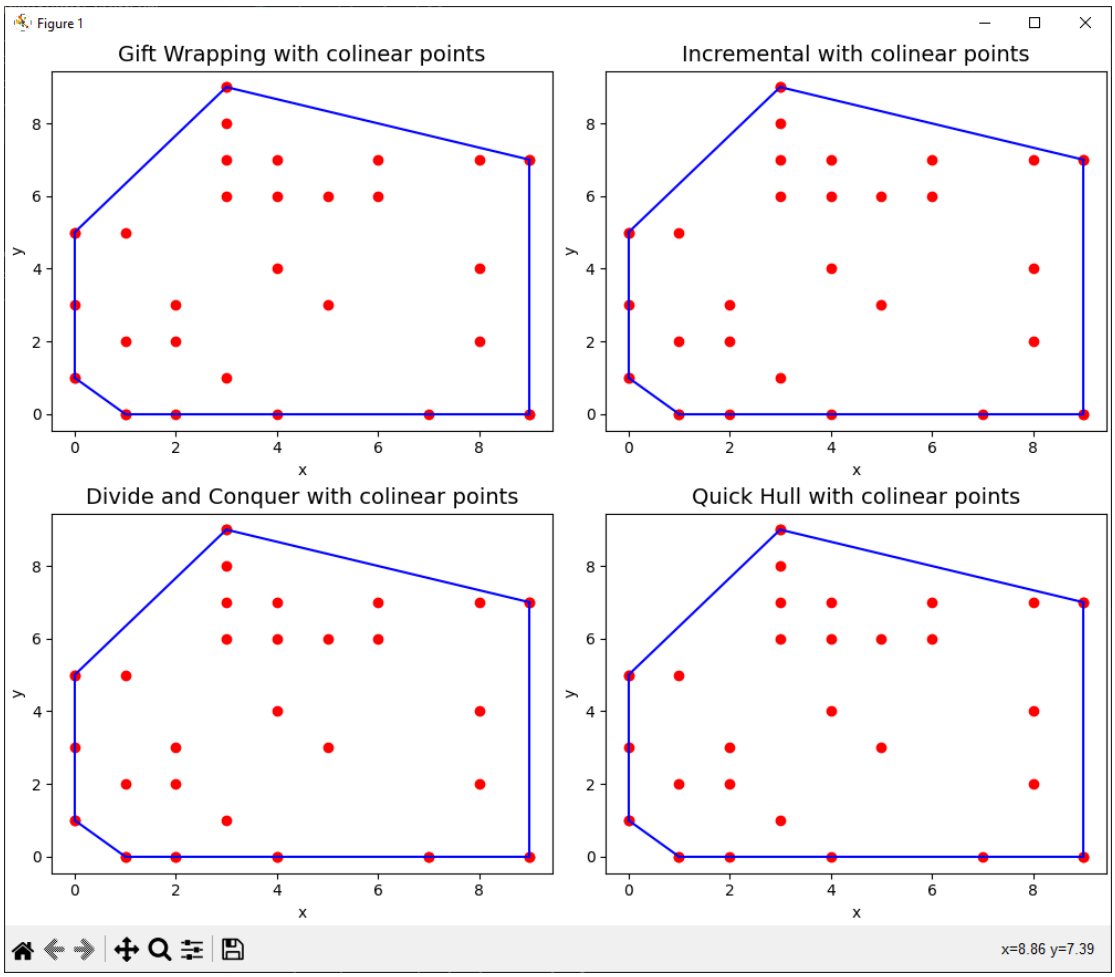
Για το μέρος αυτό, η κύρια συνάρτηση που εκτελεί τους παραπάνω αλγορίθμους και τις παραλλαγές τους έχει υλοποιηθεί στο αρχείο implementation_A.py.

Με την εκτέλεσή του, δημιουργείται ένα σύνολο 100 σημείων στο διάστημα ((-100, 100), (-100, 100)) και υπολογίζεται το ΚΠ2 των σημείων αυτών και με τους 4 αλγορίθμους που περιγράφονται στη συνέχεια. Δημιουργείται επίσης ένα plot με τα αποτελέσματα και των τεσσάρων αλγορίθμων για σύγκριση. Ένα ενδεικτικό αποτέλεσμα φαίνεται παρακάτω:



Παρατηρούμε ότι και οι 4 αλγόριθμοι είναι επιτυχείς στον υπολογισμό του ΚΠ2, δεδομένου ότι τα σημεία βρίσκονται σε γενική θέση, αφού οι συντεταγμένες τους δημιουργούνται μέσω της random.uniform() που επιστρέφει πραγματικούς αριθμούς και είναι μάλλον αδύνατον να προκύψει ο ίδιος περισσότερες από μια φορές. Το γεγονός επίσης πως δεν υπάρχουν ίδιες συντεταγμένες για διαφορετικά σημεία ικανοποιεί επίσης την υπόθεση του quickHull για διαφορετικά ακραία σημεία.

Έπειτα, δημιουργείται ένα ιδιαίτερο σύνολο 30 σημείων στο επίπεδο, ορισμένα εκ των οποίων είναι συνευθειακά, και μέσω παράλληλων plots συγκρίνονται τα αποτελέσματά των αλγορίθμων, τα οποία είναι τα εξής:



Ξεκινώντας από τον gift wrapping, διαπιστώνουμε ότι δεν επηρεάζεται από την ύπαρξη συνευθειακών σημείων. Αυτό είναι αναμενόμενο, διότι ο gift wrapping υπολογίζει κατά την εκτέλεσή του, για κάθε τριάδα σημείων r,u,t που εξετάζει, ένα κατηγόρημα προσανατολισμού. Στην περίπτωση που αυτά είναι συνευθειακά, τότε το ενδιάμεσο σημείο δεν συμπεριλαμβάνεται στο ΚΠ2 αλλά αντικαθίσταται από το t, και ο έλεγχος επαναλαμβάνεται για κάθε νέο t. Στο παραπάνω παράδειγμα, r=(0,1), u=(0,3), t=(0,5) συνευθειακά. Το u όμως είναι στο μέσο του τμήματος rt. Άρα, δεν θα συμπεριληφθεί στο ΚΠ2. Μετά, θα γίνει ο ίδιος έλεγχος με u=(0,5), t=(0,8) και το (0,5) επίσης θα απορριφθεί.

Ο αυξητικός αλγόριθμος, επίσης φαίνεται να μην επηρεάζεται από την ύπαρξη συνευθειακών σημείων. Κατά τον υπολογισμό του L_άνω (και του L_κάτω αντίστοιχα) και καθώς προσθέτουμε νέα σημεία, αφαιρούμε το μεσαίο από τα 3 τελευταία όσο η στροφή που ορίζουν δεν είναι δεξιά (συνεπώς, είναι είτε αριστερή ή τα σημεία είναι

συνευθειακά). Άρα, αν τα σημεία είναι συνευθειακά, και πάλι αφαιρείται το μεσαίο από αυτά γιατί δεν ανήκει στο ΚΠ2. Για λόγο παρόμοιο με τον gift wrapping λοιπόν, ο αυξητικός λειτουργεί καλά με συνευθειακά σημεία.

Στον divide and conquer, επίσης φαίνεται να μην υπάρχει θέμα από την ύπαρξη συνευθειακών σημείων, ως προς τη μορφή του αποτελέσματος. Ωστόσο, το ΚΠ2 μπορεί να περιέχει και συνευθειακά σημεία, τη στιγμή που οι υπόλοιποι αλγόριθμοι δεν τα περιλαμβάνουν στο ΚΠ2. Στο παραπάνω παράδειγμα, η αλυσίδα των κορυφών που επιστρέφει ο divide and conquer είναι η παρακάτω:

[array([0, 1]), array([0, 5]), array([3, 9]), array([9, 7]), array([9, 0]), array([7, 0]), array([4, 0]), array([2, 0]), array([1, 0])]

Τη στιγμή που οι υπόλοιποι αλγόριθμοι επιστρέφουν:

[array([0, 1]), array([0, 5]), array([3, 9]), array([9, 7]), array([9, 0]), array([1, 0])]

Που είναι το σωστό ΚΠ2. Παρατηρούμε, ότι τα σημεία (7,0), (4,0), (2,0) συμπεριλαμβάνονται στο ΚΠ2 ενώ δεν θα έπρεπε . Αυτό μπορεί να ερμηνευτεί αν σκεφτούμε τη λειτουργία του divide and conquer: στα μικρότερα υποσύνολα σημείων στα οποία υπολογίζει το ΚΠ2, μπορεί όντως τα επιπλέον σημεία (7,0), (4,0) ή το (2,0) να περιέχονται στο ΚΠ2. Καθώς τα δύο κυρτά περιβλήματα συγχωνεύονται, τίποτα στον divide and conquer δεν προβλέπει πως τα συνευθειακά θα πρέπει να εξεταστούν ξεχωριστά και να προστεθούν μόνο τα ακραία μιας ακμής που περιέχει συνευθειακά σημεία.

Τελικά, ο quickhull επίσης συμπεριφέρεται σωστά παρά την ύπαρξη συνευθειακών σημείων. (Εξακολουθεί να ισχύει η υπόθεση των 4 διαφορετικών ακραίων σημείων φυσικά). Ο quickhull, αφού δημιουργήσει το τετράπλευρο των 4 σημείων πάνω-αριστερά, πάνω-δεξιά, κάτω-αριστερά, κάτω-δεξιά, υπολογίζει το πιο απομακρυσμένο σημείο από κάθε πλευρά και δεξιά της. Έστω ότι σε κάποιο βήμα, η πλευρά AC περιέχει ένα σημείο p. Τότε, υπολογίζεται το εξής κατηγόρημα για να αποφασιστεί αν το p είναι δεξιά της κατευθυντής ευθείας: if CCW(A, C, p) > 0: M.append(p). Συνεπώς, τα συνευθειακά σημεία όπου το κατηγόρημα αποτιμάται σε 0, δεν περιλαμβάνονται στο σύνολο (εδώ M) των σημείων δεξιά της AΓ, άρα ούτε και στο κυρτό περίβλημα, το οποίο περιέχει μόνο τα 2 ακραία κάθε τέτοιου τμήματος.

Το πρόγραμμα implementation_A.py, επίσης καλεί τη συνάρτηση gift_wrapping_show_steps(), η οποία οπτικοποιεί τα βήματα του αλγορίθμου gift wrapping, χρησιμοποιώντας ένα plot μέσω της matplotlib και αλλάζοντάς το κατάλληλα σε ζωντανό χρόνο για να δείξει με ποια σειρά εξετάζονται οι κόμβοι. Ο αλγόριθμος ξεκινά από το αριστερότερο σημείο (r=r_0) (μπλε είναι τα σημεία που αποτελούν μέρος του ΚΠ2) και επιλέγοντας το επόμενο υποψήφιο (u), υπολογίζει με το κατηγόρημα προσανατολισμού τη φορά της γωνίας που σχηματίζουν τα παραπάνω με κάθε σημείο t. Σταδιακά, τα σημεία του ΚΠ2 χρωματίζονται μπλε και ενώνονται με ακμές μέχρις ότου συναντηθεί το r_0. Σημείωση: η οπτικοποίηση με matplotlib είναι λίγο χρονοβόρα (~2 λεπτά για 100 σημεία εντός κύκλου).

Έχει υλοποιηθεί επίσης ένας αλγόριθμος για τον υπολογισμό του ΚΠ3. Συγκεκριμένα, χρησιμοποιείται ο gift wrapping, γενικευμένος για 3 διαστάσεις: gift_wrapping_3d().

Επιπλέον, γίνεται οπτικοποίηση του αλγορίθμου με ένα plot σε 3D.

Μια σύντομη περίληψη της υλοποίησης των αλγορίθμων για ΚΠ2, ΚΠ3 είναι η παρακάτω:

- **Αυξητικός αλγόριθμος ΚΠ2**
  Ο απλούστερος αλγόριθμος για τον υπολογισμό του ΚΠ2. Υλοποιείται με την συνάρτηση incremental() στο αρχείο convex_hull_algs.py. Ακολουθεί πιστά τον ψευδοκώδικα του αυξητικού αλγορίθμου, κατασκευάζοντας πρώτα το L_άνω και έπειτα το L_κάτω περίβλημα. Χρησιμοποιεί λίστα ως τη δομή αναπαράστασης της αλυσίδας κορυφών.
- **Gift Wrapping ΚΠ2**
  Επίσης ακολουθεί την κατεύθυνση του ψευδοκώδικα. Το κύριο μέρος του αποτελείται από ένα loop (για κάθε σημείο u) που ελέγχει σε ποιον ημιχώρο βρίσκονται όλα τα σημεία t που δεν έχουν μπει ακόμα στο ΚΠ2. Τερματίζει όταν ως u συναντήσει πάλι το πρώτο σημείο (αριστερότερο).
- **Divide and Conquer ΚΠ2**
  Το κύριο μέρος του αλγόριθμου υλοποιείται στη συνάρτηση divide_and_conquer(). Αφού ταξινομήσει τα σημεία, τα διαιρέσει σε δύο υποσύνολα, και καλέσει αναδρομικά τον εαυτό της (απλούστερη περίπτωση να της δοθούν 1, 2 ή κανένα σημείο), συνδυάζει τις λύσεις που έλαβε. Ο συνδυασμός γίνεται με τη συνάρτηση merge(), στο ίδιο αρχείο. Η συνάρτηση αυτή υπολογίζει την άνω και την κάτω γέφυρα των δύο περιβλημάτων (έστω Α και Β). Έπειτα, διατρέχει τα σημεία του ΚΠ2(Α),και τα προσθέτει στο ΚΠ ξεκινώντας από το αριστερότερο, μέχρι να βρει την αριστερή ακμή της άνω γέφυρας. Όταν τη βρει, προσθέτει αυτή και την δεξιά ακμή της γέφυρας στο ΚΠ. Έπειτα, πρέπει να βρει από ποιο σημείο του ΚΠ(Β) θα συνεχίσει, διατρέχει τα σημεία του (χωρίς να τα προσθέτει στο ΚΠ) μέχρις ότου βρει το δεξί σημείο της άνω γέφυρας. Μόλις το βρει, αρχίζει να προσθέτει τα σημεία του ΚΠ(Β) με τη σειρά που τα συναντά μέχρι να βρει το δεξί σημείο της κάτω γέφυρας. Όταν το βρει, το προσθέτει και μεταπηδά πάλι στο ΚΠ(Α) όπου με παρόμοιο τρόπο βρίσκει από πού θα συνεχίσει μέχρι να βρει και πάλι το αρχικό σημείο. Το αποτέλεσμα θα είναι το ενιαίο ΚΠ2.
- **QuickHull ΚΠ2**
  Ο quickhull υλοποιείται σε δύο στάδια. Αρχικά, με τη συνάρτηση quick_hull(), βρίσκει τα 4 ακραία σημεία του συνόλου σημείων, και δεδομένου ότι είναι διαφορετικά, καλεί αναδρομικά τη συνάρτηση hull_rec(), η οποία επίσης αναδρομικά εφαρμόζει τον αλγόριθμο του quickhull για καθεμία από τις πλευρές του τετράπλευρου και τις νέες που θα οριστούν. Οι λύσεις τελικά συνδυάζονται στο τελικό, ενιαίο ΚΠ.
- **Gift Wrapping ΚΠ3**
  Ο gift wrapping για τις 3 διαστάσεις υλοποιείται από τις συναρτήσεις: gift_wrapping_3d(), getThirdPoint(), choose_initial_edge(). Για την αρχικοποίησή του, ο αλγόριθμος πρέπει να επιλέξει την πρώτη ακμή του ΚΠ3. Η συνάρτηση choose_initial_edge() βρίσκει το πρώτο σημείο της εύκολα ως το σημείο με το ελάχιστο x, το οποίο θα ανήκει σίγουρα στο κυρτό περίβλημα. Το επόμενο σημείο μπορεί να βρεθεί αν προβληθούν όλα τα σημεία στο επίπεδο και εκτελεστεί gift

wrapping για την εύρεση του ΚΠ2. Το σημείο που θα επιλεγεί αμέσως μετά από εκείνο με το ελάχιστο x, είναι το δεύτερο σημείο που ψάχνουμε. Το σημείο αυτό (η 3D έκδοσή του) επιλέγεται ως η άλλη άκρη της πρώτης ακμής. Να σημειωθεί πως δεν υπολογίζεται ολόκληρο το ΚΠ2 αλλά μόνο μια επανάληψή του, καθώς η υπόλοιπη πληροφορία είναι άχρηστη.
Με γνώση της πρώτης ακμής, ο gift_wrapping_3d(), την τοποθετεί στο ΚΠ3, βρίσκει το επόμενο σημείο του ΚΠ και προσθέτει στη λίστα προς εξέταση τις νέες ακμές που προέκυψαν από το νέο σημείο. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να μην υπάρχουν άλλες ακμές προς εξέταση.
Σε κάθε επανάληψη, όπως αναφέρθηκε, είναι απαραίτητος ο προσδιορισμός ενός νέου σημείο που θα ορίσει καινούργιες ακμές. Για αυτόν το σκοπό, χρησιμοποιείται η συνάρτηση getThirdPoint(). Η συνάρτηση αυτή, δεδομένης μιας ακμής, εξετάζει όλα τα πιθανά σημεία και επιλέγει εκείνο με τον μέγιστο προσημασμένο όγκο ως το νέο σημείο. Τελικά, το σημείο αυτό επιστρέφεται και χρησιμοποιείται για τον ορισμό νέων εδρών.

Μέρος Β: Γραμμικός Προγραμματισμός

Δεν υλοποιήθηκε, γίνεται χρήση της συνάρτησης optimize.linprog() από τη βιβλιοθήκη scipy. Το πρόβλημα είναι μεγιστοποίησης, άρα για το ενδεικτικό πρόβλημα που εκτελείται, η αντικειμενική συνάρτηση μετατρέπεται από $\max\{-3x_1 + 12x_2\}$ σε

$-\min\{3x_1 - 12x_2\}$. Οι περιορισμοί επίσης έρχονται στη μορφή $H_i(x) <= b_i$

Τα στοιχεία αυτά δίνονται στη συνάρτηση ως πίνακες.

Στο τέλος, τυπώνεται το αντίθετο αποτέλεσμα της λύσης του προβλήματος, ως μέρος του μετασχηματισμού του πίσω σε πρόβλημα μεγιστοποίησης. Τυπώνονται επίσης οι τιμές των μεταβλητών x_i που δίνουν αυτό το μέγιστο.

Τα παραπάνω γίνονται με την εκτέλεση του προγράμματος implementation_B_scipy.py

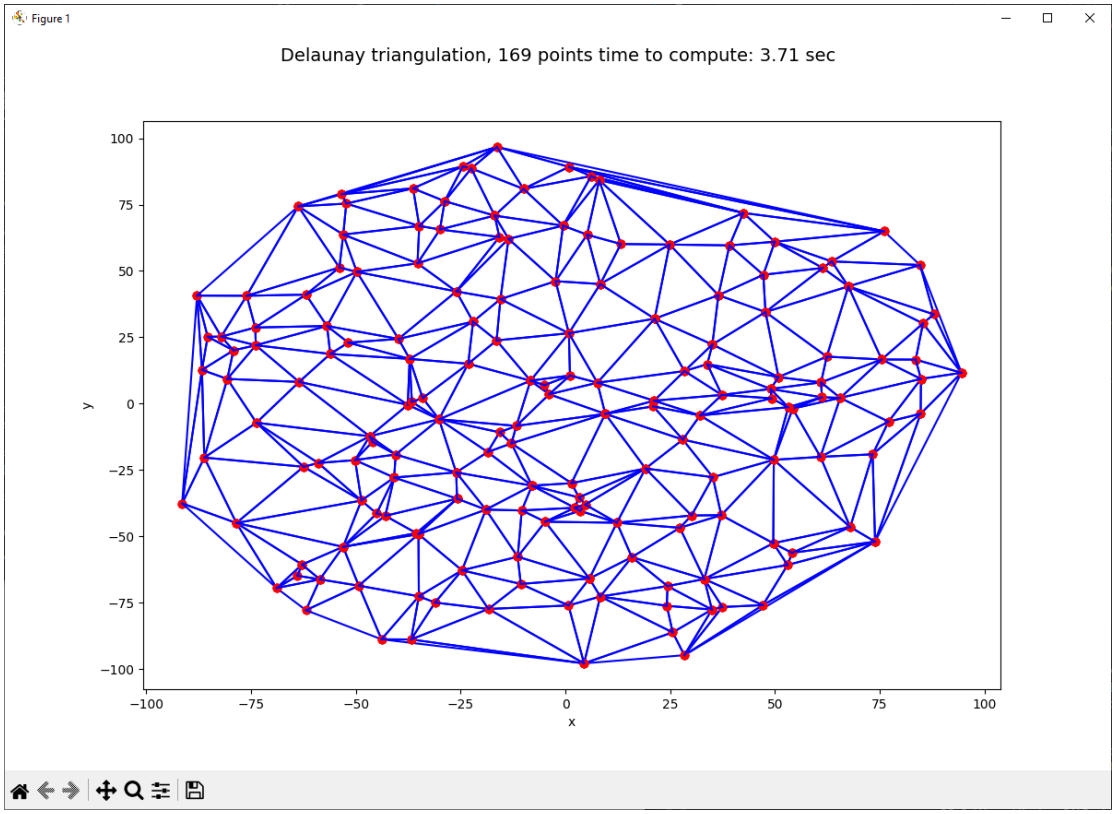
---

Μέρος Γ: Τριγωνοποίηση Delaunay

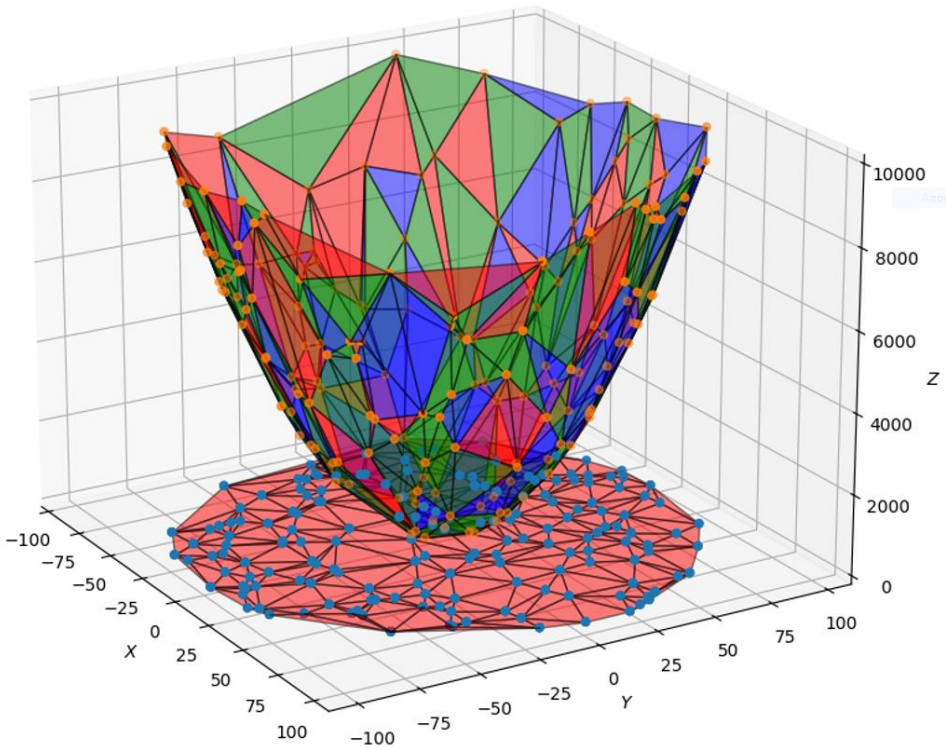
Ο αλγόριθμος που ακολουθήθηκε είναι ο εξής: πρώτα, ανυψώνουμε τα σημεία του επιπέδου παραβολικά στις 3 διαστάσεις. Έπειτα, υπολογίζεται το κυρτό περίβλημα των 3D σημείων μέσω του αλγορίθμου gift_wrapping_3d() που υλοποιήθηκε στο Α μέρος.

Από το ΚΠ3 των σημείων, μας ενδιαφέρει μόνο το κάτω μέρος του κυρτού περιβλήματος. Για να απομονωθεί, εξετάζεται κάθε τριγωνική περιοχή που ορίζει το ΚΠ3 και κρατούνται μόνο εκείνες οι περιοχές που έχουν κατεύθυνση προς τα κάτω. Τελικά, από κάθε τρίγωνο που έμεινε, εξάγονται οι ακμές του, η διάσταση z μηδενίζεται, και έτσι ορίζονται όλες οι ακμές της τριγωνοποίησης delaunay στις 2 διαστάσεις.

Έχει υλοποιηθεί επίσης η συνάρτηση delaunay_plot_2d() που σχεδιάζει ένα plot της τριγωνοποίησης στις 2 διαστάσεις. Το αποτέλεσμα μιας ενδεικτικής εκτέλεσης φαίνεται παρακάτω:



Υλοποιήθηκε επίσης μια ακόμα συνάρτηση, delaunay_steps() η οποία δείχνει τη φιλοσοφία του αλγορίθμου και πώς η προβολή ενός ΚΠ3 μπορεί να δώσει μια τριγωνοποίηση στο επίπεδο:



Τα παραπάνω βρίσκονται στο αρχείο κώδικα: implementation_C.py. Η συνάρτηση delaunay_triangulation() καλείται για έναν αυξανόμενο αριθμό σημείων, 4 φορές. Παρατηρούμε ότι ο χρόνος αυξάνεται αρκετά με την αύξηση του πλήθους των σημείων.

Κάτι τέτοιο είναι αναμενόμενο άμα σκεφτούμε ότι ο gift wrapping σε 3d έχει πολυωνυμική πολυπλοκότητα και στην πραγματικότητα είναι το πιο χρονοβόρο κομμάτι του αλγορίθμου. Από την άλλη, τόσο ο υπολογισμός του κάτω περιβλήματος, όσο και η προβολή των τριγώνων στις 2 διαστάσεις, έχουν γραμμική πολυπλοκότητα χρόνου.

Μέρος Δ: Γεωμετρική Αναζήτηση

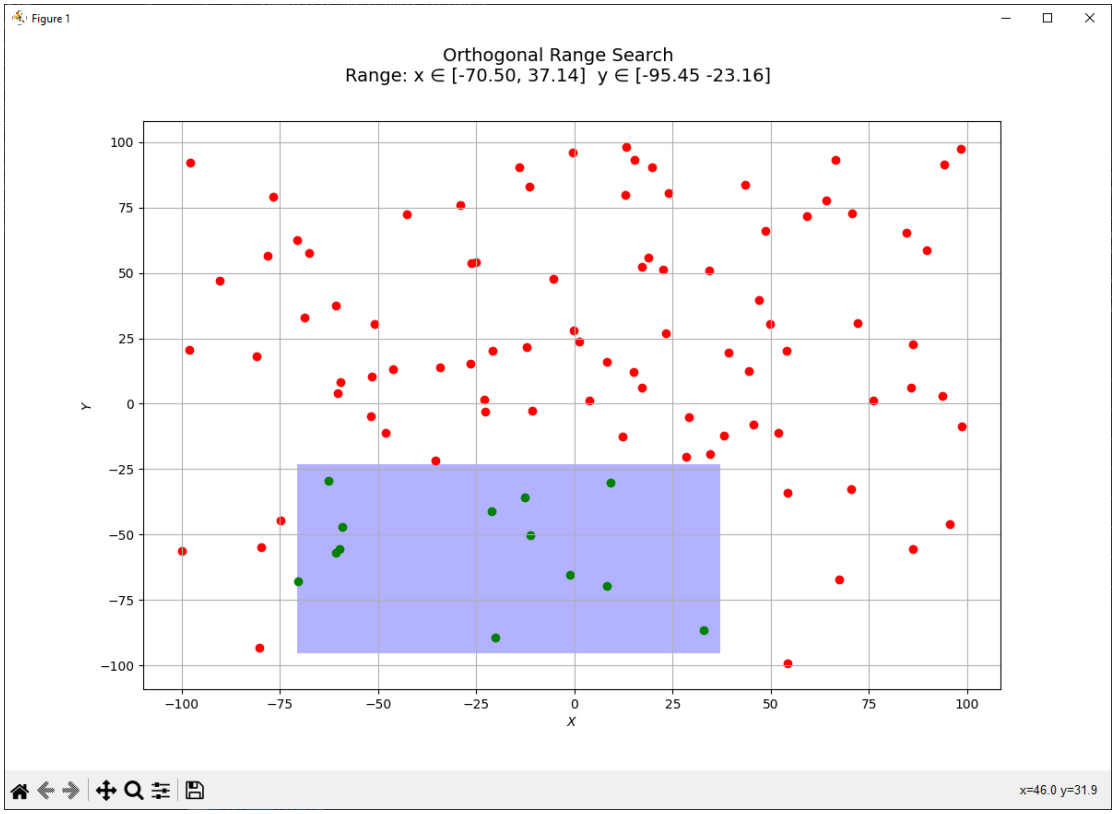
Οι αλγόριθμοι του τελευταίου μέρους βρίσκονται στο αρχείο kd_trees.py. Για να προσεγγιστεί η δεντρική δομή που απαιτείται, δημιουργήθηκε μια κλάση αντικειμένων τύπου node, που αναπαριστούν έναν κόμβο του kd tree. Υπάρχουν 2 τύποι κόμβων: τα φύλλα, που αποθηκεύουν ένα σημείο, και οι ενδιάμεσοι που περιέχουν εξισώσεις ευθειών. Στην πραγματικότητα, αποθηκεύεται απλώς η τιμή που χωρίζει το επίπεδο οριζόντια ή κάθετα.

Τη δημιουργία του kd tree αναλαμβάνει η συνάρτηση create_KDtree(). Ξεκινώντας από τη ρίζα, και συνεχίζοντας αναδρομικά στους απογόνους, όποτε πρέπει να διατάξει περισσότερα από ένα σημεία δημιουργεί ενδιάμεσο κόμβο, διαιρεί τα σημεία που ανήκουν στο εύρος του σε 2 ίσα υποσύνολα και καλεί τον εαυτό της αναδρομικά. Αν της δοθεί ένα σημείο, δημιουργεί κόμβο φύλλο και το αναθέτει σε αυτόν.

Ο αλγόριθμος της ορθογώνιας γεωμετρικής αναζήτησης υλοποιείται από τη συνάρτηση search_KDtree(). Η συνάρτηση αυτή, διασχίζει το δέντρο από τη ρίζα, μέχρι να εντοπίσει τον κόμβο φύλλο που πιθανόν να υπάρχει το σημείο που ψάχνει. Ακολουθείται ο ψευδοκώδικας.

Τέλος, η συνάρτηση print_rect_problem(), τυπώνει τα αποτελέσματα της γεωμετρικής αναζήτησης σε ένα αυθαίρετο εύρος. Σχεδιάζονται, τόσο το εύρος των σημείων, όσο και τα σημεία που γίνονται δεκτά (πράσινα) και αυτά που απορρίπτονται (κόκκινα).

Το πρόγραμμα implementation_D.py δημιουργεί ένα στιγμιότυπο του παραπάνω προβλήματος:



Βοηθητικές Συναρτήσεις

Σε όλα τα παραπάνω τμήματα της άσκησης, χρησιμοποιούνται συναρτήσεις του αρχείου helpFuctions.py. Μερικές από τις πιο σημαντικές είναι:

CCW(), CCW_3d(): για τον υπολογισμό κατηγορημάτων προσανατολισμού σε 2 και 3 διαστάσεις αντίστοιχα.

vecInList(), remVec(): η πρώτη επιστρέφει αν ένα σημείο (np.array object) είναι παρόν σε μια λίστα, και η δεύτερη την αφαιρεί.

getNext(), getPrev(): χρησιμοποιούνται ώστε να προσομοιάσουν πως μια λίστα στοιχείων είναι κυκλική, αναθέτοντας το index ξανά στην αρχή αν φτάσουμε στο τέλος της ή αντίστροφα.

printHull(): χρησιμοποιείται για την παράλληλη δημιουργία και παρουσίαση 4 κυρτών περιβλημάτων σε 2 διαστάσεις και σύγκρισή τους.

hull_3d_print(): δημιουργεί ένα plot ενός ΚΠ3 σε ζωντανό χρόνο.

genRandompoints(), genColinearpoints(), getRandompoints_3d(): παράγουν τυχαία σημεία στο χώρο. Η genRandompoints και η getRandompoints_3d προσφέρουν επιλογές για την κατανομή των σημείων στο χώρο, το σχήμα τους κοκ.

Η κλάση edge και οι συναρτήσεις edgeInList(), equalEdges(): το αντικείμενο edge χρησιμοποιείται από τον αλγόριθμο περιτυλίγματος για το ΚΠ3, για την αναπαράσταση των ακμών του κυρτού περιβλήματος.

graphicalTriangle(), graphicalTriangle_2d(): δημιουργία αντικειμένων τριγώνων σε 3 και 2 διαστάσεις αντίστοιχα . Χρησιμοποιούνται από την matplotlib στα plots.

Εκτέλεση του κώδικα:

Τα παραπάνω προγράμματα αναπτύχθηκαν και δοκιμάστηκαν σε python 3.11.3 και περιβάλλον windows. Για την εκτέλεσή τους είναι απαραίτητες οι βιβλιοθήκες που υπάρχουν στο αρχείο requirements.txt. Για την εγκατάστασή τους:

```
pip install -r requirements.txt
```

Στη συνέχεια, το κάθε main πρόγραμμα μπορεί να εκτελεστεί όπως παρακάτω:

```
python implementation_A.py
```

```
python implementation_B_scipy.py
```

```
python implementation_C.py
```

```
python implementation_D.py
```