Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών

Υπολογιστική Γεωμετρία

Υποχρεωτική Εργασία

Ονοματεπώνυμο: Αντώνιος Ζήκας

Αριθμός Μητρώου: 1115202100038

---

1. Κυρτό Περίβλημα

Στο πρώτο ερώτημα της εργασίας ζητείται να υλοποιήσουμε ορισμένους αλγορίθμους για την εύρεση του **κυρτού περιβλήματος** (Convex Hull) ενός συνόλου δεδομένων σημείων στο επίπεδο. Συγκεκριμένα ζητείται να υλοποιήσουμε τους παρακάτω αλγορίθμους με τις αντίστοιχες πολυπλοκότητές τους:

Algorithm	**Complexity**
Graham Scan	$\mathcal{O}(n \log n)$
Gift Wrapping	$\mathcal{O}(n^2)$
Devide & Conquer	$\mathcal{O}(n \log n)$
Quick Hull	$\mathcal{O}(n \log n)$

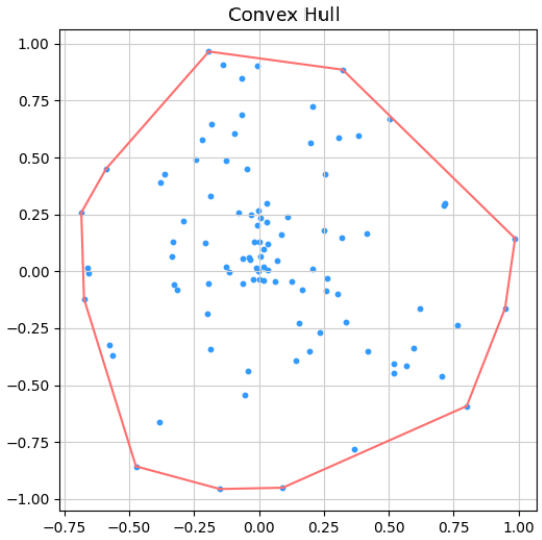
Οι παραπάνω αλγόριθμοι έχουν υλοποιηθεί με τη γλώσσα προγραμματισμού **Python** και συγκεκριμένα σε κατάλληλο **Jupyter Notebook** για την διευκόλυνση της ανάγνωσης και κατανόησης του κώδικα και της οπτικοποίησης των αποτελεσμάτων. Το συγκεκριμένο notebook μπορεί να βρεθεί στο:

`notebooks/convex_hull.ipynb`

Οι παραπάνω αλγόριθμοι δοκιμάστηκαν πάνω σε ένα σύνολο **100** σημείων της μορφής

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \ldots, \begin{bmatrix} x_{100} \\ y_{100} \end{bmatrix} \in \mathbb{R}^2$$

στο επίπεδο και τα αποτελέσματά τους ήταν ίδια. Παρακάτω φαίνεται το αποτέλεσμα μίας ενδεικτικής εκτέλεσης των αλγορίθμων πάνω σε 100 τυχαία σημεία στο επίπεδο, όπου με μπλε απεικονίζονται τα σημεία και με κόκκινο η περιφέρεια του κυρτού περιβλήματος.



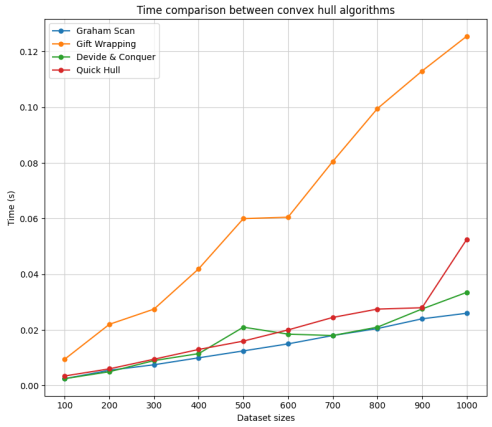
Πραγματοποιήθηκε επίσης εκτέλεση των αλγορίθμων σε διάφορα σύνολα δεδομένων με στόχο την ανάλυση στο χρόνο εκτέλεσής τους. Συγκεκριμένα εκτελέσθηκαν και η 4 αλγόριθμοι σε σύνολα δεδομένων με διαφορετικό πλήθος n σημείων. Πιο συγκεκριμένα

$$n \in \{100, 200, 300, \ldots, 1000\}$$

Η δοκιμή με 1000 σημεία επέφερε διαφορετικά αποτελέσματα στους χρόνους των αλγορίθμων τα οποία φαίνονται στον παρακάτω πίνακα:

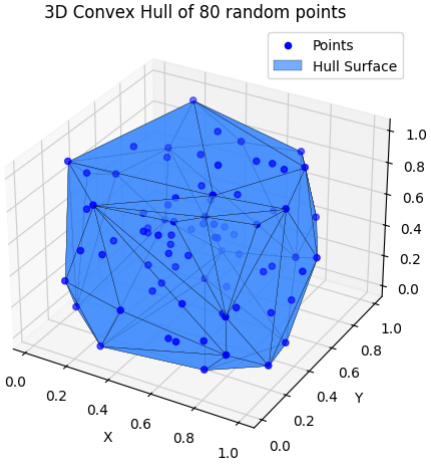
Algorithm	**Time (s)**
Graham Scan	≈ 0.023
Gift Wrapping	≈ 0.14
Devide & Conquer	≈ 0.032
Quick Hull	≈ 0.056

Από τις παραπάνω τιμές των αποτελεσμάτων μπορούμε να θεωρήσουμε πως ο **Αλγόριθμος Περιτυλίγματος** (Gift Wrapping) είναι σίγουρα χειρότερος από τους άλλους 3 για μεγάλο πλήθος δεδομένων και αυτό εξηγεί και την πολυπλοκότητά του. Αναλυτικά τα αποτελέσματα που λήφθηκαν για όλα τα πλήθη δεδομένων απεικονίζονται παρακάτω.



Στο παραπάνω διάγραμμα παρατηρούμε πως όσο η τιμή του n αυξάνεται τόσο αυξάνεται και η διαφορά μεταξύ του χρόνου εκτέλεσης του Gift Wrapping και των χρόνων των υπόλοιπων αλγορίθμων, πράγμα που τον καθιστά τον χειρότερο αλγόριθμο από τους 4.

Για την εύρεση του κυρτού περιβλήματος σε 3 διαστάσεις επιλέχθηκε να υλοποιηθεί ο **Αυξητικός Αλγόριθμος Χρωματισμού** (Beneath Beyond), κυρίως λόγω της απλότητάς του και της εύκολης γενίκευσης στις 3 διαστάσεις. Ο κώδικας του αλγορίθμου βρίσκεται στο ίδιο Jupyter Notebook . Παρακάτω φαίνεται το αποτέλεσμα μίας ενδεικτικής εκτέλεσης του αλγορίθμου.



Στο παραπάνω διάγραμμα απεικονίζεται το αποτέλεσμα του αλγορίθμου Beneath Beyond σε ένα σύνολο 80 σημείων της μορφής

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}, \ldots, \begin{bmatrix} x_{80} \\ y_{80} \\ z_{80} \end{bmatrix} \in \mathbb{R}^3$$

στον τρισδιάστατο χώρο, όπου με σκούρο μπλε απεικονίζονται τα σημεία στον τρισδιάστατο χώρο και με μπλε ανοιχτό παρουσιάζονται οι **έδρες** του κυρτού περιβλήματος.

2. Γραμμικός Προγραμματισμός

Στο δεύτερο μέρος της εργασίας ζητείται να υλοποιήσουμε τον **Αυξητικό Αλγόριθμο** για την επίλυση ενός προβλήματος **Γραμμικού Προγραμματισμού** στο επίπεδο. Ο ζητούμενος αλγόριθμος έχει υλοποιηθεί και βρίσκεται στο ακόλουθο αρχείο

`notebooks/linear_programming.ipynb`

Ο συγκεκριμένος αλγόριθμος δοκιμάστηκε με το εξής πρόβλημα γραμμικού προγραμματισμού:

Πρόβλημα

Έστω η αντικειμενική συνάρτηση

$$\max\{3x_1 - 10x_2\}$$

και οι εξής περιορισμοί:

1. $-2x_1 + x_2 \leq 12$
2. $x_1 - 3x_2 \geq -3$
3. $6x_1 + 7x_2 \leq 18$
4. $-3x_1 + 12x_2 \geq 8$
5. $2x_1 - 7x_2 \leq 35$
6. $-x_1 + 8x_2 \leq 29$
7. $-2x_1 + 6x_2 \geq -9$
8. $x_1, x_2 \geq 0$

Να βρείτε τη βέλτιστη λύση της παραπάνω αντικειμενικής συνάρτησης η οποία ικανοποιεί τους παραπάνω περιορισμούς, χρησιμοποιώντας γραμμικό προγραμματισμό.

Η επίλυση του παραπάνω προβλήματος έχει επιτευχθεί χρησιμοποιώντας και τον υλοποιημένο αυξητικό αλγόριθμο στο notebook αλλά και χρησιμοποιώντας έτοιμο πακέτο της Python. Το πακέτο που χρησιμοποιήθηκε ονομάζεται **Pulp**.

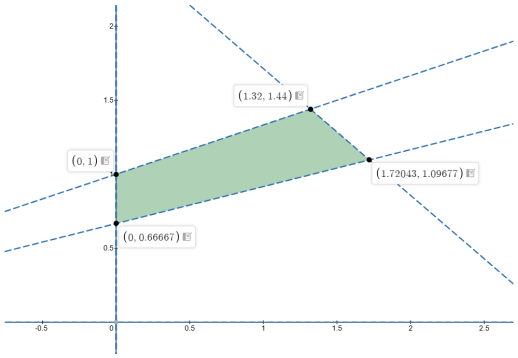
Αρχικά για να μπορέσει να επιλυθεί το παραπάνω πρόβλημα γραμμικού προγραμματισμού, χρειάστηκε να μετατραπεί σε κατάλληλη μορφή, που επιτρέπει ο υλοποιημένος αλγόριθμος. Συγκεκριμένα το πρόβλημα θα πρέπει να έχει τη μορφή:

$$\min\{f(x)\},\ and\ H_i(x) \leq 0,\ i \in \{1, 2, \ldots, n\}$$

όπου $H_i(x)$ ο $i-$οστός περιορισμός του προβλήματος και $f(x)$ η αντικειμενική συνάρτηση. Με βάση τη προηγούμενη μορφή το πρόβλημα μετατρέπεται σε

- Αντικειμενική Συνάρτηση: $\min\{-3x_1 + 10x_2\}$
- Περιορισμοί
  1. $-2x_1 + x_2 - 12 \leq 0$
  2. $-x_1 + 3x_2 - 3 \leq 0$
  3. $6x_1 + 7x_2 - 18 \leq 0$
  4. $3x_1 - 12x_2 + 8 \leq 0$
  5. $2x_1 - 7x_2 - 35 \leq 0$
  6. $-x_1 + 8x_2 - 29 \leq 0$
  7. $2x_1 - 6x_2 - 9 \leq 0$
  8. $-x_1 \leq 0$
  9. $-x_2 \leq 0$

Η εφικτή περιοχή του προβλήματος φαίνεται στο παρακάτω διάγραμμα.



όπου με γαλάζιο χρώμα απεικονίζονται οι περιορισμοί ως ευθείες στο επίπεδο, με μαύρο χρώμα τα υποψήφια βέλτιστα σημεία και με πράσινο χρώμα η εφικτή περιοχή.

Η βέλτιστη λύση που επέφερε ο υλοποιημένος αλγόριθμος και το πακέτο Pulp της Python είναι το σημείο

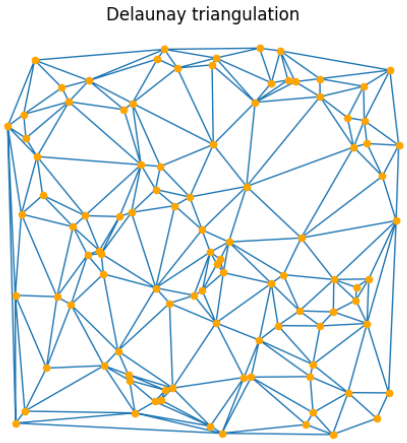
$$(1.72, 1.09)$$

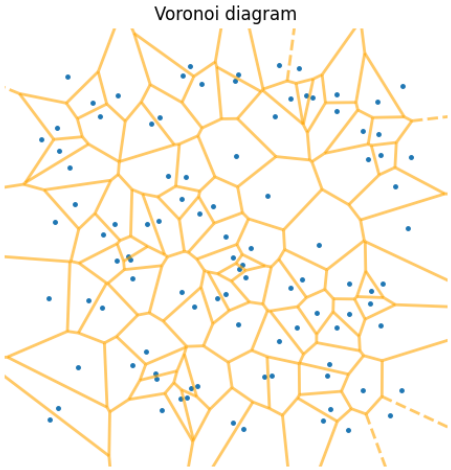
το οποίο ανήκει στα υποψήφια σημεία στο παραπάνω διάγραμμα, και τιμή της αντικειμενική συνάρτησης σε αυτό το σημείο είναι περίπου **-5.8**.

3. Voronoi - Delaunay

Στην τρίτη υλοποίηση της εργασίας έχουμε ένα σύνολο από σημεία στο επίπεδο και ζητείται να βρούμε το **διάγραμμα Voronoi** (Voronoi Diagram) και την **τριγωνοποίηση Delaunay** (Delaunay Triangulation).

Για την εύρεση των παραπάνω χρησιμοποιήθηκαν έτοιμες συναρτήσεις της Python και συγκεκριμένα του πακέτου **SciPy**. Μία ενδεικτική εκτέλεση των αλγορίθμων του SciPy έφερε τα παρακάτω αποτελέσματα:



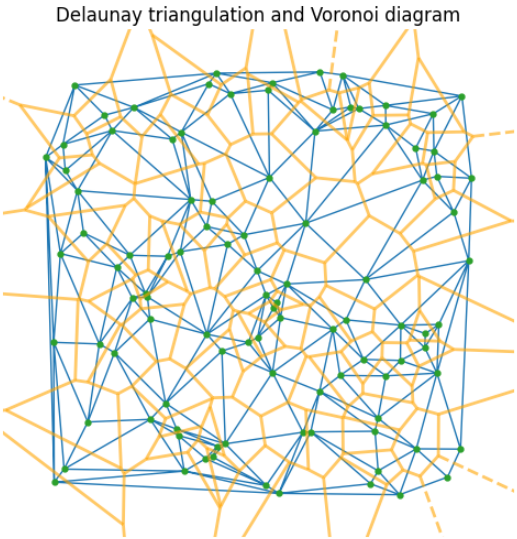


Το πρώτο διάγραμμα αναπαριστά την τριγωνοποίηση Delaunay του συνόλου σημείων, ενώ το δεύτερο διάγραμμα απεικονίζει το διάγραμμα Voronoi.

Η **Τριγωνοποίηση Delaunay** του συνόλου σημείων στο επίπεδο αποτελεί το **δυϊκό γράφο** του **διαγράμματος Voronoi** των σημείων αυτών (εστίες). Συγκεκριμένα:

1. Τα κελιά Voronoi αντιστοιχούν σε **εστίες**, δηλαδή **κορυφές τριγώνων Delaunay**.
2. Κάθε ζεύγος γειτονικών κελιών, δηλαδή **ακμή Voronoi**, αντιστοιχεί σε μία **ακμή Delaunay** που ορίζεται από τις δύο εστίες. Οι ευθείες των δύο ακμών είναι κάθετες μεταξύ τους.
3. Κάθε **κορυφή Voronoi** αντιστοιχεί σε ένα **τρίγωνο Delaunay** (ή κυρτό πολύγωνο).

Τα παραπάνω μπορούν να γίνουν πιο κατανοητά οπτικοποιώντας το διάγραμμα Voronoi και την τριγωνοποίηση Delaunay των σημείων, στο ίδιο διάγραμμα.



Το **διάγραμμα Voronoi** έχει το πολύ $\mathcal{O}(n)$ περιοχές. Ο συνολικός αριθμός ακμών και κορυφών του είναι της τάξης $\mathcal{O}(n)$. Η κατασκευή του μπορεί να γίνει σε χρόνο $\mathcal{O}(n \log n)$.

Η **τριγωνοποίηση Delaunay** έχει το πολύ $\mathcal{O}(n)$ τρίγωνα. Ο συνολικός αριθμός των ακμών της είναι της τάξης $\mathcal{O}(n)$ και η κατασκευή της μπορεί να γίνει σε χρόνο $\mathcal{O}(n \log n)$ με τη χρήση του αλγορίθμου **Διαίρει και Βασίλευε** (Devide and Conquer) ή του αλγορίθμου **Παραμετρικής Σάρωσης** (Fortune's Sweep). Σε ειδικές περιπτώσεις ο χρόνος κατασκευής μπορεί να αυξηθεί και είναι της τάξης $\mathcal{O}(n^2)$. Αυτή είναι η χείριστη περίπτωση κατασκευής.

Η πολυπλοκότητες στους αλγορίθμους κατασκευής φαίνονται αναλυτικά στον παρακάτω πίνακα:

Algorithm	Complexity
Lower Hull	$\mathcal{O}(n \log n)$
Increasing Algorithm	$\mathcal{O}(n \log n)$, $\mathcal{O}(n^2)$
Fortune's Sweep	$\mathcal{O}(n \log n)$
Devide & Conquer	$\mathcal{O}(n \log n)$

Το πλήθος n έχει διαφορετική επίδραση στην πολυπλοκότητα των αλγορίθμων ανάλογα με την τιμή του. Για μικρές τιμές του n η διαφορά στους χρόνους κατασκευής δεν είναι τόσο αισθητή. Για μεγάλες τιμές του n η πολυπλοκότητα $\mathcal{O}(n \log n)$ είναι αποδοτική αλλά σε χείριστη περίπτωση η πολυπλοκότητα $\mathcal{O}(n^2)$ καθιστά την κατασκευή πολύ αργή και η διαφορά στους χρόνους γίνεται αρκετά αισθητή.

Η εκτέλεση των παραπάνω έγιναν στο παρακάτω Python notebook:

`notebooks/voronoi_delaunay.ipynb`

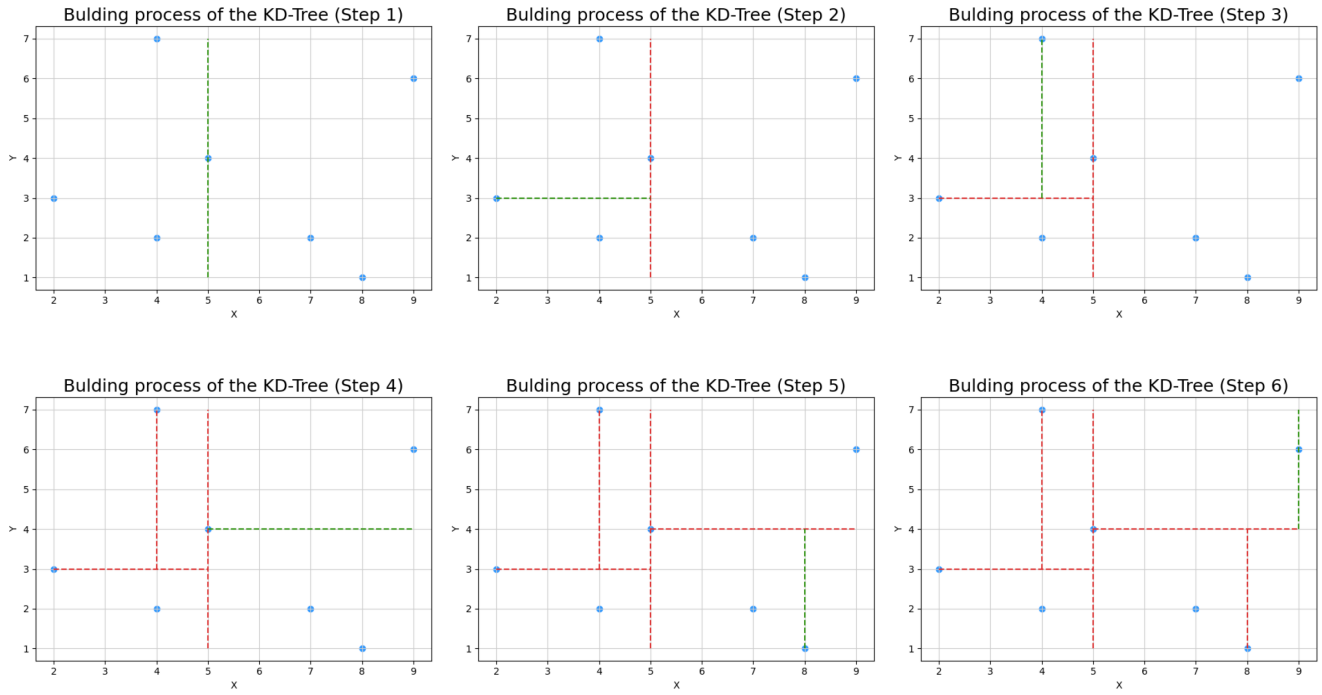
4. Γεωμετρική Αναζήτηση

Το τέταρτο και τελευταίο ερώτημα της εργασίας έχει σχέση με **Γεωμετρική Αναζήτηση** (Geometrical Search) και συγκεκριμένα ζητάει να υλοποιήσουμε έναν αλγόριθμο κατασκευής ενός **kd-Tree** , για ένα σύνολο δεδομένων P με n σημεία στο επίπεδο.

Όπως και στα προηγούμενα 3 ερωτήματα, το αντίστοιχο Python Notebook γι΄ αυτο το ερώτημα είναι το ακόλουθο:

`notebooks/geometrical_search.ipynb`

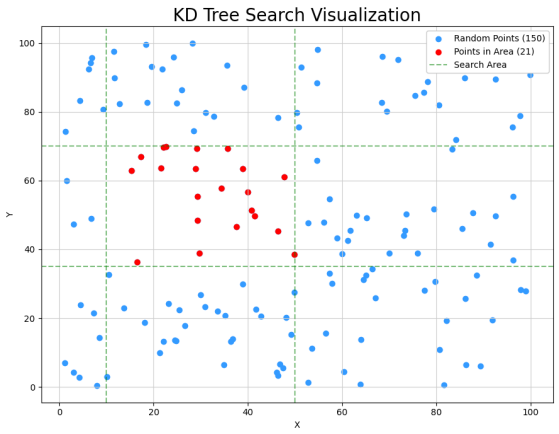
Ο αλγόριθμος κατασκευής του kd-Tree έχει υλοποιηθεί σε αυτό το notebook και έχει δοκιμαστεί για ένα μικρό σύνολο δεδομένων σημείων στο επίπεδο. Συγκεκριμένα για 7 τυχαία σημεία, η διαδικασία κατασκευής του δένδρου φαίνεται στο Σχήμα 1.



Σχήμα 1: Διαδικασία Κατασκευής kd-Tree

Στο Σχήμα 1 με πράσινο χρώμα αναπαριστάται η καινούρια διάμεσος στα σημεία, ενώ με κόκκινο χρώμα αναπαρίστανται εκείνες που έχουν ήδη καταχωρηθεί.

Στο ίδιο Python notebook έχει υλοποιηθεί επίσης μία συνάρτηση για την γεωμετρική αναζήτηση σε ένα σύνολο από σημεία στο επίπεδο. Έγινε δοκιμή της συγκεκριμένης συνάρτησης δημιουργώντας ένα σύνολο από **150 σημεία** στο επίπεδο και παίρνοντας μία δεδομένη **ορθογώνια έκταση** πάνω στην οποία θα γίνει η αναζήτηση των σημείων στο δένδρο.



Το αποτέλεσμα που επέφερε η αναζήτηση ήταν μία λίστα των σημείων που βρίσκονται στην δοσμένη ορθογώνια έκταση, η οποία βρίσκεται στο Python notebook καθώς και μία οπτικοποίηση των αποτελεσμάτων η οποία φαίνεται στο προηγούμενο διάγραμμα. Με μπλε χρώμα απεικονίζονται τα δεδομένα σημεία στο επίπεδο, με πράσινο χρώμα οι ευθείες που ορίζουν την ορθογώνια έκταση και με κόκκινο τα σημεία τα οποία βρίσκονται εντός της ορθογώνιας έκτασης, δηλαδή τα ζητούμενα σημεία.