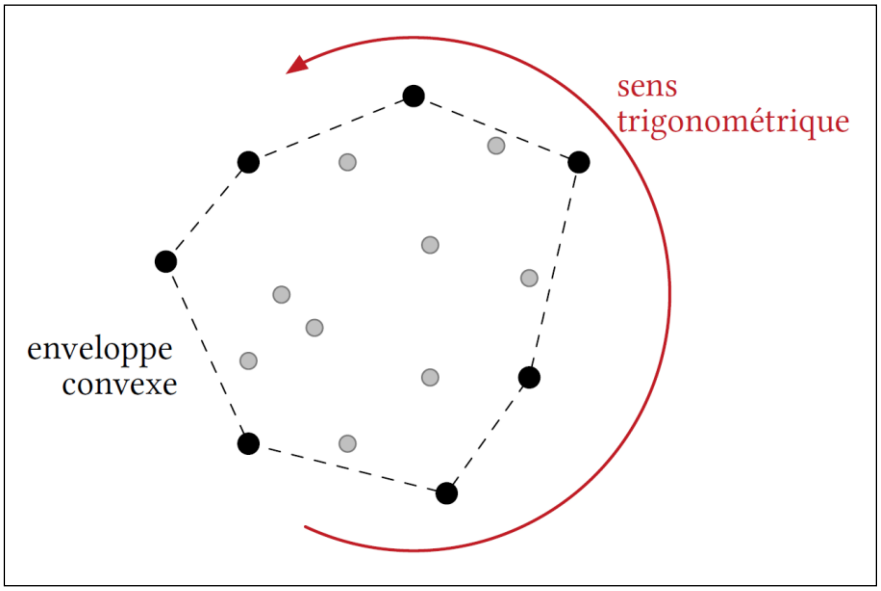
UNIVERSITÉ D'ORLÉANS

Rapport de Projet du bloc 2

Diviser-pour-Régner pour le calcul de l'enveloppe convexe

Grégoire BURNET

10/09/2020



Les points noirs et gris sont un exemple d'ensemble S.

Les points noirs forment l'enveloppe convexe E de S.

 BY-NC-SA

Table des matières

 (cc) BY-NC-SA

I. Introduction

Le problème du calcul de l'enveloppe convexe est un problème bien connu en informatique. Il peut se formaliser de plusieurs manières, mais ici il sera abordé grâce à un algorithme de type diviser pour régner. Cet algorithme déterminera l'enveloppe convexe de S, il sera noté EC(S) par la suite et sera également représentée par un tableau ; les points de coordonnée y appartenant seront donnés dans le sens antihoraire (aussi appelé sens trigonométrique par les mathématiciens).

II. Quelques questions algorithmiques

II. 1. Question 1

Une proposition d'algorithme pour la fonction $TangenteInf$ qui calcule la tangente inférieure. Les points des enveloppes $EC(S_G)$ et $EC(S_D)$ sont toujours dans le même ordre trigonométrique.

$Algorithme\ TangenteInf\ (EC(S_G) = [p_0;\ p_1;\ \dots;\ p_{r+1}];\ EC(S_D) = [q_0;\ q_1;\ \dots;\ q_{t+1})])$

$i = 0$

$j = 0$

$tant\ que\ y(D;\ p_{i+1}\ ;\ q_j) > y(D;\ p_i;\ q_j)\ ou\ y(D;\ p_i;\ q_{j-1}) > y(D;\ p_i;\ q_j)\ \ faire$

$si\ y(D;\ p_{i+1};\ q_j) > y(D;\ p_i;\ q_j)\ alors$

$i = i + 1$

$sinon$

$j = j - 1$

$retourner\ (i\ mod\ r, j\ mod\ t)$

II. 2. Question 2

Une propriété qui pourrait être utile pour démontrer que la fonction $TangenteSup$ se termine toujours et que son temps d'exécution au pire des cas est en $O(n)$ est la suivante :

« Le point de EC(S_G) ayant la plus petite abscisse x et le point d'EC(S_D) ayant la plus grande abscisse x font toujours partie d'EC(S) ».

Pour rappel la fonction $TangenteSup$ est la suivante :

$$TangenteSup(EC(S_G) = [p_0;\ p_1;\ \dots;\ p_r];\ EC(S_D) = [q_0;\ q_1;\ \dots\ ;\ q_t)])$$

Cette propriété est vraie, car

Algorithme TangenteSup$(EC(S_G) = [p_0, p_1, \dots, p_r],\ EC(S_D) = [q_0, q_1, \dots, q_t)])$

$i \leftarrow 0$
$j \leftarrow 0$
tant que $y(D, p_i, q_{j+1}) > y(D, p_i, q_j)$ *ou* $y(D, p_{i-1}, q_j) > y(D, p_i, q_j)$ **faire**
  si $y(D, p_i, q_{j+1}) > y(D, p_i, q_j)$ **alors**
    $j \leftarrow j + 1$
  sinon
    $i \leftarrow i - 1$
retourner $(i \mod r, j \mod t)$

Figure 1 : Rappel de l'algorithme TangenteSup

Les affectations i et 1 ont un temps d'exécution de $O(1)$

La boucle $tant\ que$ à un temps d'exécution de $O(n)$

La condition si à un temps d'exécution de $O(1)$

La condition sinon à un temps d'exécution de $O(1)$

Donc le temps d'exécution de cet algorithme est de $O(n)$

BY-NC-SA

II. 3. Question 3

a) Algorithme Fusion

$Entrée : Un\ tableau\ EC(S)\ de\ taille\ n\ contenant\ les\ points\ de\ E$

$Sortie :$

$Fusion(EC(S))$

$si\ n \leq 2\ alors$

$retourner\ la\ liste\ circulaire\ associé\ à\ EC(S)$

$sinon$

$m \leftarrow Taille(S)//2$

$Trier(S)$

$Soit\ EC(S_G)\ le\ tableau\ contenant\ les \frac{n}{2}\ premiers\ elements\ de\ EC(S)$

$Soit\ EC(S_D)\ le\ tableau\ contenant\ les \frac{n}{2}\ premiers\ elements\ de\ EC(S)$

$listeG \leftarrow Fusion(EC(S_G))$

$listeD \leftarrow Fusion(EC(S_D))$

$(P, Q) \leftarrow TangenteSup(listeG;\ listeD)$

$(R, S) \leftarrow TangenteInf(listeG;\ listeD)$

$Copier\ dans\ une\ liste\ l\ les\ éléments\ de\ la\ liste\ l_1\ compris\ entre\ P\ à\ R\ inclus,$

$puis\ les\ éléments\ de\ la\ liste\ l_2\ de\ S\ à\ Q\ et\ boucler\ la\ liste\ en\ liant\ Q\ et\ P;$

$retourner\ l$

b) Algorithme Taille

$Taille(S)$

$t = len(S)$

$retourner\ t$

c) Algorithme Trier (S)

$Trier(S)$

$retourner\ S\ trier$

II. 4. Question 4

$T(n)$ est le temps d'exécution de $Fusion$ lorsqu'il est exécuté.

$$T(n) = \begin{cases} O(2) & si\ n \leq 2 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n) & sinon \end{cases} \qquad T(n) = \begin{cases} O(S) & si\ n \leq 2 \\ 2 * T\left(\frac{n}{2}\right) + O(n) & sinon \end{cases}$$

a) Proposition et preuve pour O (n²)

L'algorithme s'effectue avec une complexité en $O(n^2)$ dans le pire des cas.

S'il est supposé qu'à chaque partitionnement, l'un des deux sous-ensembles soit vide, la formule de récurrence vérifiée par la complexité est :

$$C(n) = C(n-1) + O(n)$$

$$C(n) = O(n^2)$$

BY-NC-SA

b) Proposition et preuve pour O (n log n)

L'algorithme s'effectue avec une complexité en $O(n \log n)$ dans le meilleur des cas.

S'il est supposé qu'à chaque partitionnement, les deux tableaux sont de même taille, alors la formule de récurrence devient

$$C(n) = 2.C(n^2) + O(n)$$
$$C(n) = O(n \log n)$$

c) Proposition Admise

L'algorithme s'effectue avec une complexité $O(n \log n)$ en moyenne.

II. 5. Question 5

La solution de la récurrence est $T(n)$:

$$T(n) = 2 * T\left(\frac{n}{2}\right) + O(n)$$

Après identification grâce au théorème Maître, on a bien une récurrence de type :

$$T(n) = a * T\left(\frac{n}{b}\right) + f(n)$$

Avec $a = 2$; $b = 2$ et $f(n) = n$

Calcul de $\log_b(a) = \log_2(2) = 1$

Selon le théorème maitre, c'est le deuxième cas qui s'applique ici, car $f(n) = n = 1$

Donc la solution $T(n)$ est $O(n \log n)$.

III. Implémentation en Python

III. 1. Question 6

Voici une partie de l'implémentation du code en python :

a) Taille de l'ensemble convexe

```
#Pour le calcul de la taille max de l'ensemble convexe
def taille_ensembles(L):
    t=len(L)
    return t
```

b) Affichage des coordonnées des points de l'ensemble convexe

```
#Pour l'affichage de l'ensemble des couples X et Y des points de l'ensemble convexe, avec L la liste des coordonnées et G la taille du tableau de l'ensemble convexe
def affichage_couple_enveloppe(L,G):
    t=int(taille_ensembles(G))
    for i in range (t):
        position=G[i]
        X=L[position][0]
        Y=L[position][1]
        print(f"Position dans le tableau : {position} ;avec pour coordonnée X={X} et Y={Y}")
```

a) Trie des points en nuage

```
#Renvoie la liste des points du nuage, triée par abscisse puis ordonnée croissante.
def tri_nuage(L):
    return tri_fusion_points(L)
```


(cc) BY-NC-SA

b) Trie de la liste de point

```
def tri_fusion_points(L):
    n = len(L)
    if n <= 1:
        return L
    else:
        m = n // 2
        return fusion_points(tri_fusion_points(L[:m]), tri_fusion_points(L[m:]))
```

c) Fonction qui crée l'ensemble convexe

```
def fonction_enveloppe_convexe(L_origin):
    L = tri_nuage(L_origin)
    association = {i: L_origin.index(L[i]) for i in range(len(L))}
    EnvSup = []
    EnvInf = []
    for i in range(len(L)):
        while len(EnvSup) >= 2 and orientation(L[i], L[EnvSup[-1]], L[EnvSup[-2]]) <= 0:
            EnvSup.pop()
        EnvSup.append(i)
        while len(EnvInf) >= 2 and orientation(L[EnvInf[-2]], L[EnvInf[-1]], L[i]) <= 0:
            EnvInf.pop()
        EnvInf.append(i)
    sommets = EnvInf[:-1] + EnvSup[::-1]
    # on convertit les indices
    return [association[i] for i in sommets]
```

III. 2. Question 7

Le nombre de points de l'enveloppe convexe pour les jeux de données suivants sont :

n	points
1	1
2	2
3	3

Jeux	n	points
data0.txt	**8**	**5**
data1.txt	**32**	**8**
data2.txt	**64**	**9**

Jeux	n	points
data_a.txt	5	5
data_b.txt	10	5
data_c.txt	20	7

Jeux	n	points
data_d.txt	50	11
data_e.txt	250	13
data_f.txt	1000	21

Pour n>=1000 jupyter nootebook et python3 mettent une erreur de type *maximum recursion depth exceeded in comparison*

IV. En bonus, pour les plus téméraires

IV. 1. Question 8

a) Explication

Le format SVG (Scalable Vector Graphics) est un format basé sur le langage XML. Il décrit des images vectorielles bidimensionnelles.

Ce format est aux images ce que le HTML est au texte. Il est explicitement conçu pour fonctionner avec le standard du W3C.

Les images vectorielles peuvent être redimensionnées sans perte de qualité, tandis que ce n'est pas possible avec des images matricielles

(cc) BY-NC-SA

b) Module Python existant

Il est possible de créer une image SVG grâce à un module à importer dans Python.

Comme avec celui de svgwrite, de drawSvg, ou bien pysvg.

Mais celui utilisé ici est le module matplotlib, que l'on retrouve de bases dans jupyter notebook.

c) Images de test

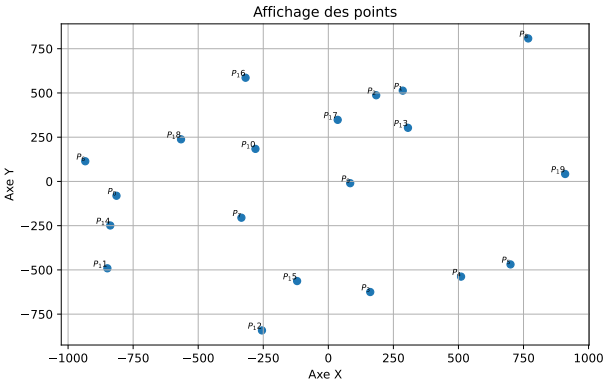


Figure 2 : Nuage de point pour le jeu de test data_c

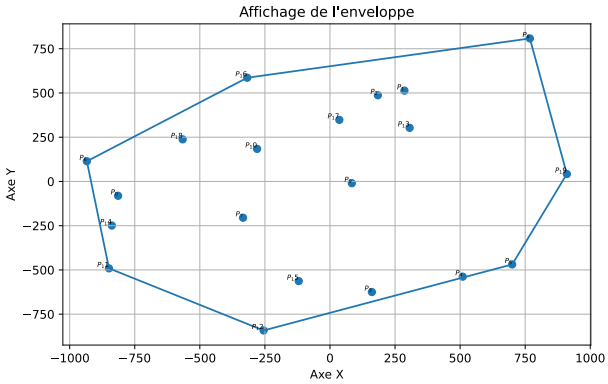


Figure 3 : enveloppe pour le jeu de test data_c

V. Explication de l'exécution du code python

V. 1. Fichier Prog.py

Le fichier Prog.py est un programme qui se lance grâce à Python 3 et permet l'exécution de toutes les entrées qui sont des fichiers de jeu de données.

ATTENTION : Il est important que le module matplotlib soit installé sinon le programme fonctionnera mal.

V. 2. Fichier Prog.ipynb

Le fichier Prog.ipynb s'ouvre avec Jupyter Notebook, il suffit de lancer les lignes de code les unes à la suite des autres pour que tous fonctionnent correctement.

Il n'y a pas besoin d'installer matplotlib qui est déjà implémenté de base dans le logiciel.

V. 3. Commandes d'exécution :

Pour exécuter le code Python, il faut entrer les commandes suivantes pour les définitions suivantes en fonctions des besoins de ce que l'on souhaite tester ou faire.

Commandes	Paramètres + Module	Explications
points_aleatoire(n)	n : le nombre de nombres aléatoires qui sera généré	Sers à créer une liste de coordonnée aléatoire comprise entre -1000 et 1000
Sauvegarde_liste_point_aleatoire (nomfichier,points)	nomfichier : le nom du fichier à sauvegarder avec extension .txt à la fin point : la liste des coordonnées	Sers à sauvegarder la liste des coordonnées dans un fichier TXT
`lirePoints(nomfichier)`	nomfichier : les fichiers data.txt fournis	Sers à créer la liste de coordonnée pour le nuage de point

(cc) BY-NC-SA

Commandes	Paramètres + Module	Explications
`affichage(L, nomfichier)`	L : la liste de coordonnée nomfichier : le nom du fichier sans à sauvegarder extension **ATTENTION : module matplotlib**	Sers à afficher le nuage de point sans l'enveloppe convexe et à sauvegarder ce nuage de point au format svg
`fonction_enveloppe_convexe(L)`	L : la liste de coordonnée	Sers pour afficher les indices de la liste de coordonnée pour le nuage de points et son enveloppe convexe
`taille_ensembles(G)`	G : taille de l'ensemble convexe G = fonction_enveloppe_convexe(L) L : la liste de coordonnée	Sers à définir le nombre de points que l'ensemble convexe a.
`affichage_couple_enveloppe(L,G)`	G : taille de l'ensemble convexe G = fonction_enveloppe_convexe(L) L : la liste de coordonnée	Sert à afficher l'ensemble des couples x et y de l'enveloppe convexe ainsi sue la position dans le tableau L
`affiche_enveloppe(L, algorithme, nomfichier)`	L : la liste de coordonnée Algorithme : fonction en paramètre fonction_enveloppe_convexe nomfichier : le nom du fichier sans à sauvegarder extension **ATTENTION : module matplotlib**	Sers à afficher le nuage de points et son enveloppe convexe puis à faire une sauvegarde de l'enveloppe au format svg
`AffichagePropreFichier (nomfichier)`	nomfichier : les fichiers data.txt fournis **ATTENTION : module matplotlib**	Exécution de toutes les commandes précédentes
`tempsExecution (L,algo)`	L : la liste de coordonnée Algo : fonction en paramètre fonction_enveloppe_convexe	Sers à calculer le temps d'exécution de la fonction fonction_enveloppe_convexe
`complexite_temporelle(debug)` `complexite_temporelle_alea (m,debug)`	(optionnel) debug : À True active le mode graphique pour étude **ATTENTION : module matplotlib actif lorsque debug=True**	Sers à Calcul la complexité temporelle de fonction_enveloppe_convexe pour n allant jusqu'à 1000 ou 2000

VI. Explications, difficultés et solutions

VI. 1. Explications

Pour trouver une solution au problème posé, il a fallu lire un jeu de donnée fourni grâce à la fonction fournie ou bien en l'absence de celui-ci en créer une grâce aux fonctions point_aleatoire et Sauvegarde_liste_point_aleatoire.

Il a été choisi de faire un tri fusion des deux sous-ensembles de coordonnée de points et faire, ainsi de suite par récurrence jusqu'aux conditions limites. Pour ce faire plusieurs implémentations de code ont été nécessaires avec les fonctions tri_nuage, tri_fusion_points.

Afin de fusionner les sous-ensembles triés, il a été nécessaire de faire appel à une méthode récursive grâce à ces deux fonctions fusion_points et fusion_bis.

(cc) BY-NC-SA

L'implémentation pour les divers affichages est faite :

Soit sous la forme de graphe se fait grâce aux fonctions affichage et affiche_enveloppe qui utilise le module matplotlib permettant entre autres de sauvegarder les graphes au format SVG et PNG.

Soit sous la forme de texte comme avec la fonction affichage_couple_enveloppe, qui affiche les coordonnées x et y de chaque point de l'enveloppe.

Une dernière implémentation a été faite pour voir quelle était la complexité de notre algorithme fonction_enveloppe_convexe garce à la fonction tempsExecution (L,algo).

VI. 2. Difficultés

Il a été dur de déterminer une proposition de preuves pour la récursivité de Fusion et de même pour l'identification de la solution grâce au Théorème Maitre. L'implantation en python a été complexe à mettre en œuvre surtout au niveau du tri et du tri fusion. Le module matplotlib m'a beaucoup aidé pour la visualisation du problème et de la solution à apporter.

VI. 3. Solutions

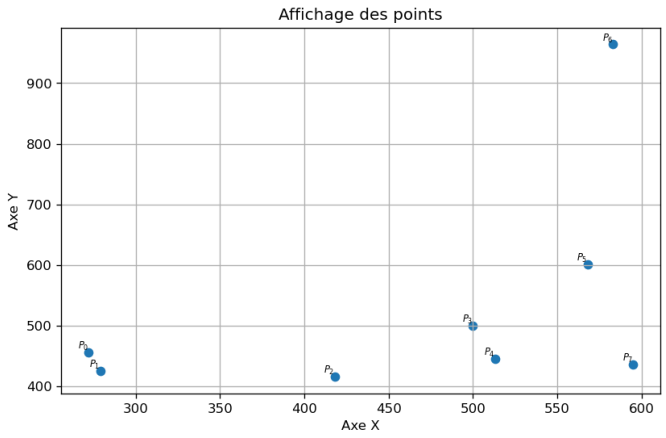
Illustrations d'une des solutions avec le fichier data0.txt. Pour trouver l'enveloppe convexe de ce fichier, l'utilisation de la fonction AffichagePropreFichier (nomfichier) a été utilisée. En effet cette dernière regroupe toutes les fonctions utiles pour le calcul de l'enveloppe ainsi que de son affichage et la sauvegarde au format SVG.

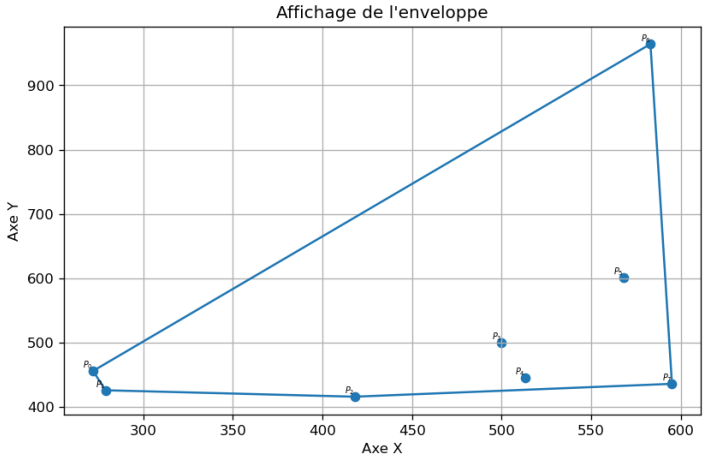
a) Entrée pour data0.txt

```
# Affichage pour data0
AffichagePropreFichier('data0.txt')
```

b) Sortie pour data0.txt

```
La liste des points est la suivante pour le fichier data0.txt :
[(272, 456),(279, 426),(418, 416),(500, 500),(513, 446),(568, 601),(583, 964),(595, 436)]
Emplacement des points de l'enveloppe convexe dans le Tableau est
[0, 1, 2, 7, 6, 0]
Les coordonnées de chacun des points de l'enveloppe convexe sont
Position dans le tableau : 0 ; avec pour coordonnée X=272 et Y=456
Position dans le tableau : 1 ; avec pour coordonnée X=279 et Y=426
Position dans le tableau : 2 ; avec pour coordonnée X=418 et Y=416
Position dans le tableau : 7 ; avec pour coordonnée X=595 et Y=436
Position dans le tableau : 6 ; avec pour coordonnée X=583 et Y=964
Position dans le tableau : 0 ; avec pour coordonnée X=272 et Y=456
Le nombre de points de l'enveloppe convexe est de 6
```



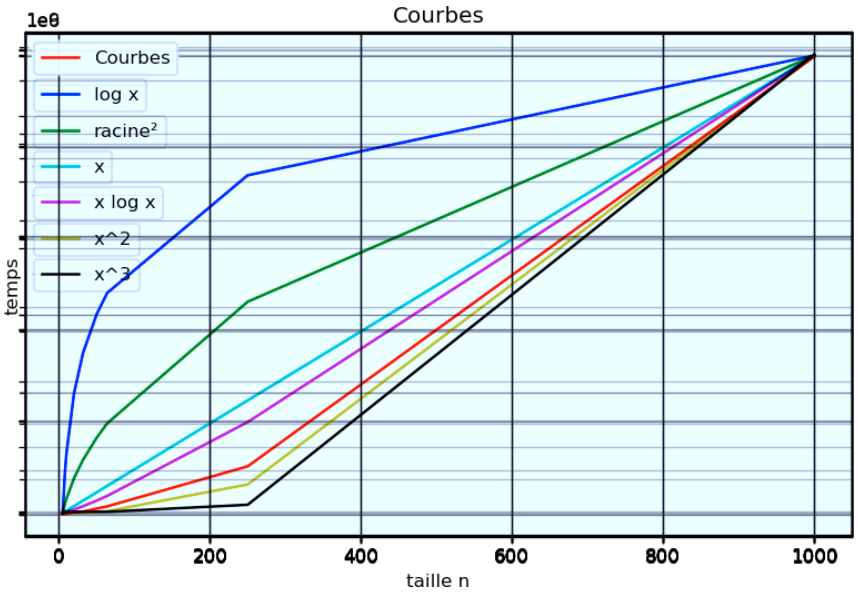


c) Complexité temporelle pour n=1000

```
Taille de la liste pour data_a.txt : 5
Le temps d'exécution pour data_a.txt :
17.3 µs ± 885 ns per loop (mean ± std. dev. of 3 runs, 10 loops each)
1.826666660538952e-05
Taille de la liste pour data0.txt : 8
Le temps d'exécution pour data0.txt :
38.6 µs ± 9.79 µs per loop (mean ± std. dev. of 3 runs, 10 loops each)
```


(cc) BY-NC-SA

```
3.040000001419685e-05
Taille de la liste pour data_b.txt : 10
Le temps d'exécution pour data_b.txt :
44.6 µs ± 3.64 µs per loop (mean ± std. dev. of 3 runs, 10 loops each)
4.470000006525273e-05
Taille de la liste pour data_c.txt : 20
Le temps d'exécution pour data_c.txt :
108 µs ± 2.71 µs per loop (mean ± std. dev. of 3 runs, 10 loops each)
0.00012876666672430778
Taille de la liste pour data1.txt : 32
Le temps d'exécution pour data1.txt :
163 µs ± 767 ns per loop (mean ± std. dev. of 3 runs, 10 loops each)
0.00016596666667586155
Taille de la liste pour data_d.txt : 50
Le temps d'exécution pour data_d.txt :
329 µs ± 19.9 µs per loop (mean ± std. dev. of 3 runs, 10 loops each)
0.0003202000001086465
Taille de la liste pour data2.txt : 64
Le temps d'exécution pour data2.txt :
365 µs ± 2.66 µs per loop (mean ± std. dev. of 3 runs, 10 loops each)
0.00036680000008952146
Taille de la liste pour data_e.txt : 250
Le temps d'exécution pour data_e.txt :
2.63 ms ± 71.5 µs per loop (mean ± std. dev. of 3 runs, 10 loops each)
0.0025508333333164046
Taille de la liste pour data_f.txt : 1000
Le temps d'exécution pour data_f.txt :
25.8 ms ± 3.48 ms per loop (mean ± std. dev. of 3 runs, 10 loops each)
0.024017066666660175
```



Grâce au jeu de fichier data0 à data2 et data_a à data_f, la complexité temporelle obtenue d'après le graphe et les calculs est comprise entre $O(n^2)$et $O(n \log n)$

VII. Conclusion

Ce projet est intéressant, car il a permis de comprendre comment fonctionner le calcul de l'enveloppe convexe et a permis de faire des recherches sur ce problème algorithmique et de trouver qu'il en existait d'autre et qu'il y avait plusieurs méthodes algorithmiques pour résoudre ce problème.

Grâce à la recherche sur le SVG, il y a été possible de sauvegarder le nuage de point et l'enveloppe convexe dans deux fichiers séparés.

Il a été aussi possible d'étudier la complexité temporelle théoriquement et de faire une interprétation des résultats de la complexité temporelle lors de l'implémentation en python grâce aux courbes obtenues avec le programme.

(cc) BY-NC-SA