T.P. 10 : Enveloppes convexes dans le plan

Ce T.P. a pour objectif de calculer des enveloppes convexes de nuages de points dans le plan affine, un grand classique de la géométrie algorithmique. On rappelle qu'un ensemble $C \subset \mathbb{R}^2$ est convexe si et seulement si pour toute paire de points $p, q \in C$ le segment de droite $[p, q]$ est inclus dans C. L'enveloppe convexe d'un ensemble $P \subset \mathbb{R}^2$, notée $\text{Conv}(P)$, est le plus petit convexe contenant P.[1] Dans le cas où P est un ensemble fini (appelé *nuage de points*), le bord de $\text{Conv}(P)$ est un polygone convexe dont les sommets appartiennent à P, comme illustré dans la figure 1.

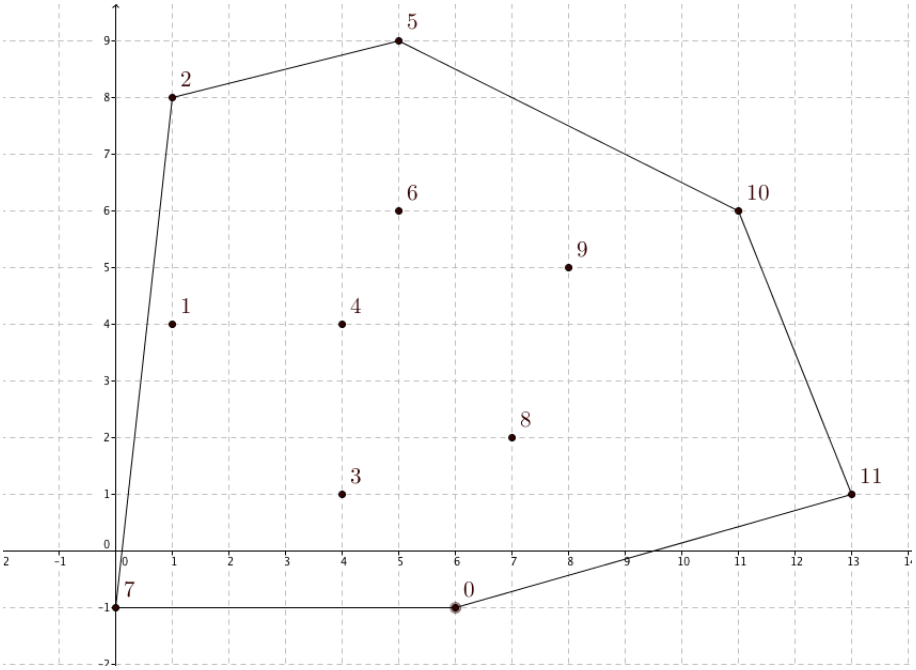

FIGURE 1– Un nuage de points, numérotés de 0 à 11 et le bord de son enveloppe convexe.

Dans ce sujet, nous allons écrire un algorithme de calcul du bord de l'enveloppe convexe d'un nuage de points P dans le plan affine. Cette algorithme, dit *algorithme du paquet cadeau*, consiste à envelopper le nuage de points progressivement en faisant pivoter une droite tout autour.

On verra que le temps d'exécution de cet algorithme est majoré par une constante fois nm, où n désigne le nombre total de points de P et m le nombre de points de P appartenant au bord de $\text{Conv}(P)$. Rappelons que le temps d'exécution d'un programme A (fonction ou procédure) est le nombre d'opérations élémentaires (comparaisons, additions, soustractions, multiplications, divisions, affectations, etc) nécessaires à l'exécution de A. Sauf mention contraire dans l'énoncé du sujet, le candidat n'aura pas à justifier des temps de calculs de ses programmes. Toutefois, il devra veiller à ce que ces derniers ne dépassent pas les bornes prescrites.

Dans toute la suite on supposera que le nuage de points P est de taille $n \geq 3$ et en position générale, c'est-à-dire qu'il ne contient pas 3 points distincts alignés.

Ces hypothèses vont permettre de simplifier les calculs en ignorant les cas pathologiques comme par exemple la présence de 3 points alignés sur le bord de l'enveloppe convexe. Nos programmes prendront en entrée un nuage de points P dont les coordonnées sont stockées dans un tableau *tab* à 2 dimensions, comme dans l'exemple ci-dessous qui contient les coordonnées du nuage de points de la figure 1.

i \j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	6	1	1	4	4	5	5	0	7	8	11	13
1	-1	4	8	1	4	9	6	-1	2	5	6	1

1. Question de maths : pourquoi cela existe-t-il ?

Précisons que les coordonnées, supposées entières, sont données dans une base orthonormée du plan, orientée dans le sens direct. La première ligne du tableau contient les abscisses, tandis que la deuxième contient les ordonnées. Ainsi, la colonne d'indice j contient les deux coordonnées du point d'indice j. Ce dernier sera nommé p_j dans la suite.

N.B. Le tableau *tab* prend en Python la forme :

```
# d'une liste de listes :
tab=[[6,1,1,4,4,5,5,0,7,8,11,13],
[-1,4,8,1,4,9,6,-1,2,5, 6, 1]]
```

```
# ou d'un tableau numpy.
tab=np.array([[6,1,1,4,4,5,5,0,7,8,11,13],
[-1,4,8,1,4,9,6,-1,2,5, 6, 1]])
```

Partie I. Préliminaires

Question 1. Ecrire une fonction `plusBas`(*tab*) qui prend en paramètre un tableau *tab* de taille $2 \times n$ et qui renvoie l'indice j du point le plus bas (c'est-à-dire de plus petite ordonnée) parmi les points du nuage P. En cas d'égalité, votre fonction devra renvoyer l'indice du point de plus petite abscisse parmi les points les plus bas.

Sur le tableau exemple précédent, le résultat de la fonction doit être l'indice 7.

Dans la suite, nous aurons besoin d'effectuer un seul type de test géométrique : celui de l'orientation.

Définition 1 *Etant donnés trois points p_i,p_j, p_k du nuage P, distincts ou non, le test d'orientation renvoie* +1 *si la séquence (p_i, p_j, p_k) est orientée positivement,* −1 *si elle est orientée négativement, et* 0 *si les trois points sont alignés (c'est-à-dire si deux au moins sont égaux d'après l'hypothèse de position générale).*

Pour déterminer l'orientation de (p_i, p_j, p_k), il suffit de calculer l'aire signée du triangle, comme illustré sur la figure ci-dessous. Cette aire est la moitié du déterminant de la matrice 2×2 formée par les coordonnées des vecteurs $\overrightarrow{p_i p_j}$ et $\overrightarrow{p_i p_k}$.

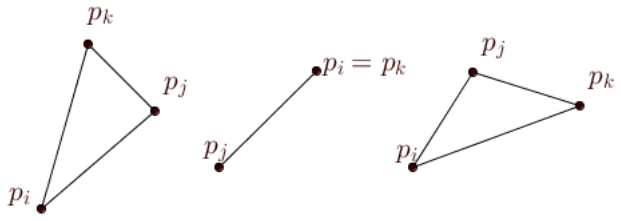

FIGURE 2 – *Test d'orientation sur (p_i, p_j, p_k) : positif à gauche, nul au centre, négatif à droite*

Question 2 Sur le tableau précédent, donner le résultat du test d'orientation pour les choix d'indices suivantes :

— $i = 7$, $j = 3$, $k = 4$
— $i = 8$, $j = 9$, $k = 10$.

Question 3 Ecrire une fonction `orient`(*tab,i,j,k*) qui prend en paramètres le tableau *tab* et trois indices de colonnes, potentiellement égaux, et qui renvoie le résultat (-1,0, ou +1) du test d'orientation sur la séquence (p_i, p_j, p_k) de points de P.

Partie II. L'algorithme du paquet cadeau

Cet algorithme a été proposé par R. Jarvis en 1973. Il consiste à envelopper peu à peu le nuage de points P dans une sorte de paquet cadeau, qui à la fin du processus est exactement le bord de Conv(P). On commence par insérer le point de plus petite ordonnée (et parmi ceux-ci le plus à gauche, celui d'indice 7 dans l'exemple précédent) dans le paquet cadeau, puis à chaque étape de la procédure on sélectionne le prochain point du nuage P à insérer.

La procédure de sélection fonctionne comme suit. Soit p_i le dernier point inséré dans le paquet cadeau à cet instant. Par exemple $i = 10$ dans l'exemple de la figure 3.

Considérons la relation binaire $\preceq$ définie sur l'ensemble $P \setminus \{p_i\}$ par :

$$p_j \preceq p_k \Leftrightarrow \texttt{orient}\ (tab,i,j,k) \leq 0.$$

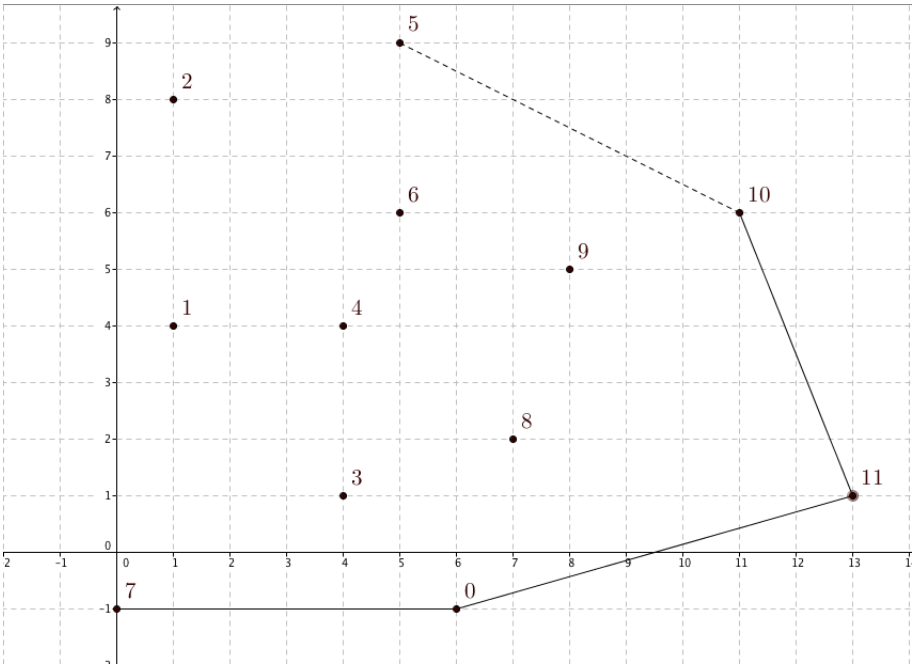

FIGURE 3 – Mise à jour du paquet cadeau après insertion du point p_{10}.

Question 4 (de maths, admise pour le T.P.). Justifier brièvement le fait que $\preceq$ est une relation d'ordre total sur l'ensemble $P \setminus \{p_i\}$ c'est-à-dire :

- (réflexivité) pour tout $j \neq i$, $p_j \preceq p_j$,
- (antisymétrie) pour tous $j, k \neq i$, $p_j \preceq p_k$ et $p_k \preceq p_j$ implique $p_j = p_k$
- (transitivité) pour tous $j, k, l \neq i$, $p_j \preceq p_k$ et $p_k \preceq p_l$ implique $p_j \preceq p_l$.
- (totalité) pour tous $j, k \neq i$, $p_j \preceq p_k$ ou $p_k \preceq p_j$.

N.B. Pour ceux et celles qui veulent y réfléchir à la maison, la transitivité est un peu subtile, et repose de manière essentielle sur les hypothèses définissant p_i à chaque étape.

Ainsi, le prochain point à insérer (le point d'indice 5 dans la figure ci-dessus) est l'élément maximum de $P \setminus \{p_{10}\}$ pour la relation d'ordre $\preceq$. Il peut se calculer en temps linéaire (c'est-à-dire majoré par une constante fois n) par une simple itération sur les points de $P \setminus \{p_i\}$.

Question 5. Ecrire une réalisation en Python de la procédure. Elle prendra la forme d'une fonction `prochainPoint`(*tab*,*i*) qui prend en paramètre le tableau *tab* de taille $2 \times n$ ainsi que l'indice i du point inséré en dernier dans le paquet cadeau, et qui renvoie l'indice du prochain point à insérer. Le temps d'exécution de votre fonction doit être majoré par une constante fois n, pour tous n et i. La constante doit être indépendante de n et i et on ne demande pas de la préciser.

Question 6. Décrire à la main le déroulement de la procédure `prochainPoint` sur l'exemple de la figure ci-dessus. Plus précisément, indiquer la séquence de points de $P \setminus \{p_{10}\}$ considérés et la valeur de l'indice du maximum à chaque itération.

On peut maintenant combiner la fonction `prochainPoint` avec la fonction `plusBas` de la question 1 pour calculer le bord de l'enveloppe convexe de P. On commence par insérer le point p_i d'ordonnée la plus basse, puis on itère le processus de mise à jour du paquet cadeau jusqu'à ce que le prochain point à insérer soit de nouveau p_i. A ce moment-là, on renvoie le paquet cadeau comme résultat sans insérer p_i une seconde fois.

Un détail technique : comme la taille du paquet cadeau augmente peu à peu lors du processus, et qu'à la fin elle peut être petite par rapport au nombre n de points de P, nous stockerons les indices des points du paquet cadeau dans une liste. Par exemple sur le nuage de la figure 1, le résultat sera la liste [7,0,11,10,5,2].

Question 7. Ecrire une fonction `convJarvis`(*tab*) qui prend en paramètre le tableau *tab* de taille $2\times n$ représentant le nuage P, et qui renvoie une liste contenant les indices des sommets du bord de l'enveloppe convexe de P, sans doublon. Le temps d'exécution de votre fonction doit être majoré par une constante fois nm, où m est le nombre de points de P situés sur le bord de $\mathrm{Conv}(P)$.

Question 8. Justifier brièvement le temps d'exécution de l'algorithme du paquet cadeau.

Question 9. La fonction suivante `paquetCadeau` suivante, due à Mathieu Vermeil, (MPSI 1 2016/2017), est disponible dans `/home/profs/bondil/public/TP-MPSI` dans le fichier `afficheNuage.py`.

Elle s'applique à un tableau `tab` dont les entrées ne sont pas nécessairement des entiers, ce qui sera utile à la question suivante.

Vérifiez le bon fonctionnement de votre fonction `convJarvis` de la question 7 à l'aide de cette fonction d'affichage, appliquée au tableau `tab` de l'énoncé.

```
def arrondi(x):
    """Arrondit un nombre au plus loin de 0."""
    from math import ceil
    return(sgn(x)*ceil(abs(x)))

def paquetCadeau(tab):
    """Affiche le nuage de points correspondant à tab
    et son paquet cadeau obtenu par l'algorithme de Jarvis."""
    fig, ax = plt.subplots()
    plt.grid(True)
    plt.axis("equal")
    plt.axhline(color="black")
    plt.axvline(color="black")
    xMin,xMax=min(tab[0])-1,max(tab[0])+1
    yMin,yMax=min(tab[1])-1,max(tab[1])+1
    xMin,xMax=arrondi(xMin),arrondi(xMax)
    yMin,yMax=arrondi(yMin),arrondi(yMax)
    plt.xlim((xMin,xMax))
    plt.ylim((yMin,yMax))
    ax.xaxis.set_ticks(np.arange(xMin,xMax+1,1))
    ax.yaxis.set_ticks(np.arange(yMin,yMax+1,1))
    plt.scatter(tab[0],tab[1],s=60)
    for i in range(len(tab[0])):
        plt.annotate(str(i),(tab[0][i]+0.1,tab[1][i]+0.1),size="large")
    iPaquet=convJarvis(tab)
    iPaquet.append(iPaquet[0])
    xPaquet=[tab[0][i] for i in iPaquet]
    yPaquet=[tab[1][i] for i in iPaquet]
    plt.plot(xPaquet,yPaquet,color="red",linewidth=2)
    plt.show()
```

Question 10 Mieux, écrire une fonction : `jarvisAleatoire(n=10,fenetre=[[-5,5],[-5,5]])` qui prend en argument un entier `n` et une fenêtre `[[xMin,xMax],[yMin,yMax]]` avec les valeurs par défaut ci-dessous, et affiche un nuage de n points pris aléatoirement dans cette fenêtre et son paquet cadeau obtenu par l'algorithme de Jarvis.

Pour éviter d'avoir des risques d'alignement de points (i.e. vérifier les conditions de positions générales données plus haut), on pourra utiliser la fonction `uniform` du module `random` : `uniform(a,b)` renvoie un flottant aléatoire suivant une loi uniforme, entre a et b.