X2015-IPT

January 3, 2017

1 Sujet X info B (IPT) 2h - 2015

1.0.1 (corrigé Marc REZZOUK mrezzouk@free.fr)

on choisit son langage de programmation mais ensuite on veut du python… Petite coquille.

1.1 Partie I

Question 1

```
In [1]: def plusBas(tab, n):
            jmin = 0
            ordmin = tab[1][jmin]
            abscis = tab[0][jmin]

            for j in range(1, n):
                if tab[1][j] < ordmin or (tab[1][j] == ordmin and tab[0][j] < absci
                    jmin = j
                    ordmin = tab[1][jmin]
                    abscis = tab[0][jmin]

            return jmin
```

Question 2
Petit intermède numpy non demandé…

```
In [2]: import numpy as np
        tab = np.array([[0, 1, 1, 4, 4, 5, 5, 7, 7, 8, 11, 13],
                        [0, 4, 8, 1, 4, 9, 6, -1, 2, 5, 6, 1]])

        def produitmixte(a, b, c):
            print(np.vstack((b - a, c - a)))
            return np.linalg.det(np.vstack((b - a, c - a)))

        print(produitmixte(tab[:, 0], tab[:, 3], tab[:, 4]))

        print(produitmixte(tab[:, 8], tab[:, 9], tab[:, 10]))
```

```
[[4 1]
 [4 4]]
12.0
[[1 3]
 [4 4]]
-8.0
```

Le premier triangle est direct, le second indirect

Question 3

```
In [3]: def produitmixte(a, b, c):
            return np.linalg.det(np.vstack((b - a, c - a)))

        def orient(tab, i, j, k):
            a = produitmixte(tab[:, i], tab[:, j], tab[:, k])  # à écrire de façon
            if a > 0:
                return 1
            elif a < 0:
                return -1
            else:
                return 0
```

1.2 Partie II

Question 4

- réflexivité: orient(tab, i, j, j) = 0 est bien <= 0
- antisymétrie: i, j, k sont alors alignés donc avec la **restriction de l'énoncé** (position générale des points), p_j et p_k étant différents de p_i, ils sont égaux
- transitivité: assez *subtile*. Le point p_i est **supposé sur l'enveloppe convexe** donc tous les autres points sont dans un même demi-plan (c'est essentiel). On prouve alors le résultat par exemple des condérations d'angles principaux: $arg(ij/ik) \in]-\pi, 0]$ et en s'appuyant sur une droite frontière (faire une figure).
- totalité: orient est soit posivite soit négative

Question 5

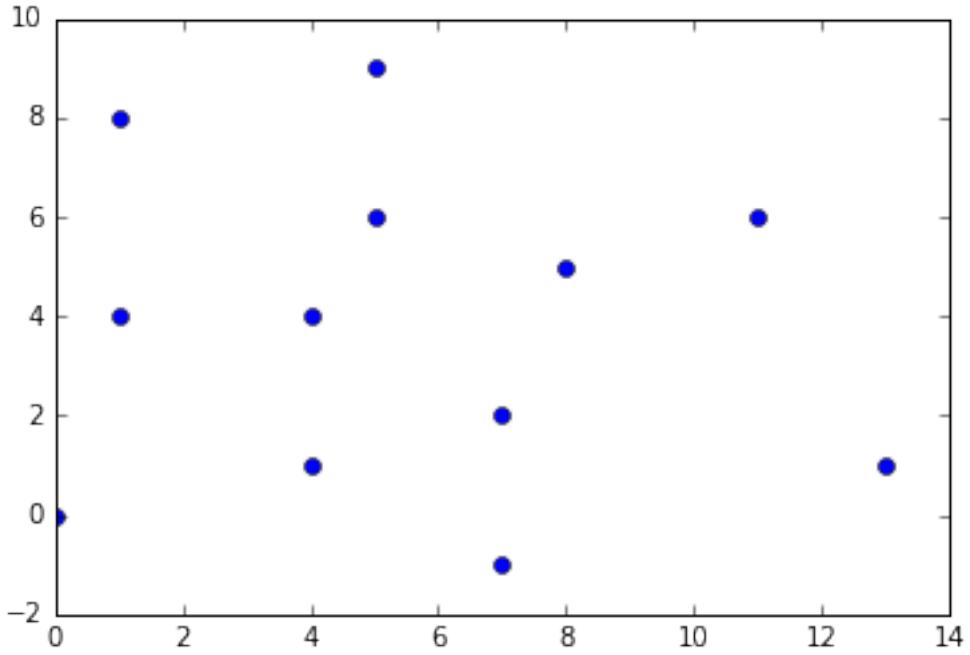
```
In [4]: def prochainPoint(tab, n, i):
            L = [j for j in range(n) if j != i]
            imax = L[0]
            for l in L[1:]:
                if orient(tab, i, imax, l) == -1:
                    imax = l

            return imax
```

Question 6

```
In [5]: %matplotlib inline
        import matplotlib.pyplot as plt

        plt.plot(tab[0, :], tab[1, :], 'bo')
        plt.show()
```



On est censé le décrire à la main... Voici une version python

```
In [6]: def prochainPointdetail(tab, n, i):
            L = [j for j in range(n) if j != i]
            imax = L[0]
            print('on démarre arbitrairement par le point', imax)
            for l in L[1:]:
                if orient(tab, i, imax, l) == -1:
                    imax = l
                    print('on choisit le point intermédiaire', imax)
                else:
                    print('on ne choisit pas', l, 'on garde', imax)  # sauf pour 0.

            print('prochain point', imax)
            return imax

        prochainPointdetail(tab, tab.shape[1], 10)
```

```
on démarre arbitrairement par le point 0
on choisit le point intermédiaire 1
on choisit le point intermédiaire 2
on ne choisit pas 3 on garde 2
on ne choisit pas 4 on garde 2
on choisit le point intermédiaire 5
on ne choisit pas 6 on garde 5
on ne choisit pas 7 on garde 5
on ne choisit pas 8 on garde 5
on ne choisit pas 9 on garde 5
on ne choisit pas 11 on garde 5
prochain point 5
```

```
Out[6]: 5
```

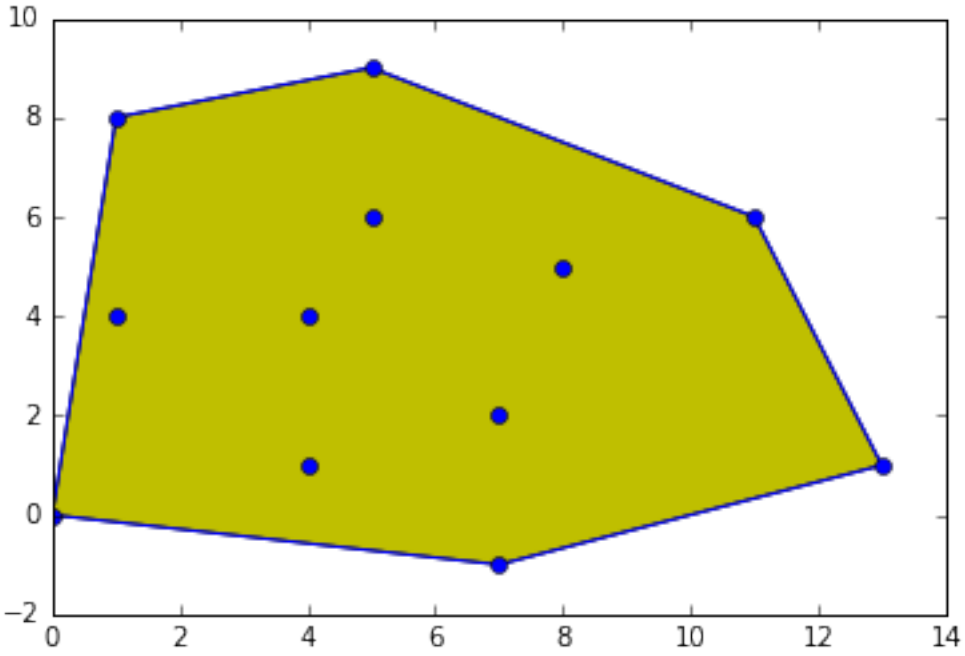
Question 7

```
In [7]: def convJarvis(tab, n):
            premierpoint = plusBas(tab, n)
            L = [premierpoint]
            prochain = prochainPoint(tab, n, premierpoint)
            while prochain != premierpoint:
                L.append(prochain)
                prochain = prochainPoint(tab, n, prochain)
            return L

        L = convJarvis(tab, tab.shape[1])
        print(L)
```

```
[7, 11, 10, 5, 2, 0]
```

```
In [8]: L.append(L[0])
        plt.plot(tab[0, L], tab[1, L], 'b')
        plt.fill(tab[0, L], tab[1, L], 'y')  # pour faire plus joli...
        plt.plot(tab[0, :], tab[1, :], 'bo')
        plt.show()
```



Question 8

On boucle sur m itérations et à l'intérieur de chaque itération on fait appel à la fonction prochainPoint qui a un temps d'exécution en O(n).

1.3 Intermède: piles d'entiers

```
In [9]: ''' on simule avec des listes python '''

        def newStack():
            return []

        def isEmpty(s):
            return len(s) == 0

        def push(i, s):
            s.append(i)

        def top(s):
            return s[-1]

        def pop(s):
            return s.pop()
```

1.4 Partie III Algorithme de balayage

Question 9

Le tri fusion a un temps d'exécution en n log n. (le tri rapide a un temps d'exécution *moyen* en n log n

Question 10

```
In [10]: def majES(tab, es, i):
             ''' es supposée non vide '''
             p1 = pop(es)
             if isEmpty(es):   ### on a qu'un élément dans la pile
                 push(p1, es)
                 push(i, es)
                 return  # on s'arrête là

             p2 = top(es)
             while not isEmpty(es) and orient(tab, i, p1, p2) == -1:
                 p1 = pop(es)   # on dépile p1
                 if not isEmpty(es):
                     p2 = top(es)  # on met à jour p2

             push(p1, es)  # que es soit vide ou que l'orientation soit bonne
             # il faut remettre p1

             push(i, es)
```

Question 11

```
In [11]: def majEI(tab, ei, i):
             ''' ei supposée non vide '''
             p1 = pop(ei)
             if isEmpty(ei):
                 push(p1, ei)
                 push(i, ei)
                 return

             p2 = top(ei)
             while not isEmpty(ei) and orient(tab, i, p1, p2) == 1:
                 p1 = pop(ei)
                 if not isEmpty(ei):
                     p2 = top(ei)

             push(p1, ei)

             push(i, ei)
```

Question 12

```
In [12]: def convGraham(tab, n):
             es = newStack()
             ei = newStack()
             push(0, es)
```

```
            push(0, ei)
            for i in range(1, n):
                majES(tab, es, i)
                print(i, es)
                majEI(tab, ei, i)

            pop(es)    # on élimine le point le plus à droite redondant dans les de
            while not isEmpty(es):
                push(pop(es), ei)

            pop(ei)
            # on ne rajoute pas le premier point de l'enveloppe
            # (merci Judicaël)
            return ei
```

```
In [13]: convGraham(tab, tab.shape[1])
```

```
1 [0, 1]
2 [0, 2]
3 [0, 2, 3]
4 [0, 2, 4]
5 [0, 2, 5]
6 [0, 2, 5, 6]
7 [0, 2, 5, 7]
8 [0, 2, 5, 8]
9 [0, 2, 5, 9]
10 [0, 2, 5, 10]
11 [0, 2, 5, 10, 11]


Out[13]: [0, 7, 11, 10, 5, 2]
```

Question 13

Rappel: les points sont *déjà* triés par abscisses croissantes (coût n log n)

On fait n itérations et dans chaque fonction de mise à jour des piles **on ne rajoute qu'une seule fois (modulo les jongleries de pile) un nouveau point et les points enlevés ne reviennent plus** donc on a une complexité en O(n).

En étant plus précis, à chaque étape i, on fait au plus k1 $\times$ (points retirés étape i) + k2 opérations or somme sur i des (points retirés étape i) $\leq n$ car ils ne reviennent plus…