# 1 Enveloppe convexe d'un nuage de points

## 1.1 Nuage de points et définitions

Un point est défini par la donnée de ses deux coordonnées :

```
type point = int * int;;
```

Dans tout ce problème nous allons appuyer nos exemples sur la séquence de points $P_0, \dots, P_{20}$ définie de façon suivante :

— On définit par récurrence une séquence d'entiers $(x_n)_{0\leqslant n\leqslant 20}$ par :

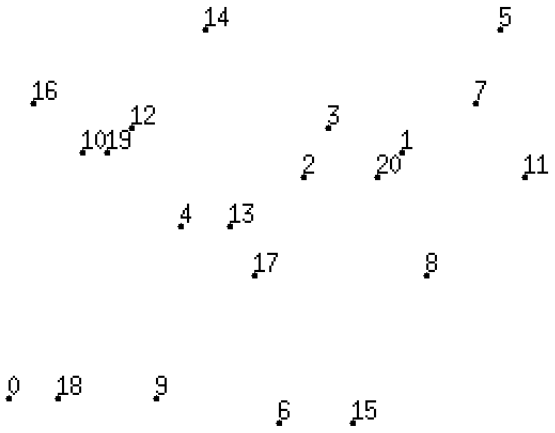
$$x_0 = 1 \text{ et } \forall n \in [\![1, 20]\!], x_n = (17x_{n-1}) \mod 23$$

(où mod désigne le reste de la division euclidienne).

— On définit par récurrence une séquence d'entiers $(y_n)_{0\leqslant n\leqslant 20}$ par :

$$y_0 = 3 \text{ et } \forall n \in [\![1, 20]\!], y_n = (17y_{n-1}) \mod 19$$

— Pour tout $i \in [\![0, 20]\!]$, $P_i$ est le point de coordonnées $(x_i, y_i)$.




1) Créer une variable globale `p : point array` dont la valeur est $[|(x_0, y_0); \dots; (x_{20}, y_{20})|]$. Test : $P_{20} = (16, 12)$.

Soient trois points $A, B, C$, avec $A \neq B$ et $A \neq C$. Soit $\theta \in ]-\pi, \pi]$ la mesure principale de l'angle $\left(\overrightarrow{AB}, \overrightarrow{AC}\right)$. On dit que le point $C$ est *à gauche* de $(A, B)$ si $\theta \geqslant 0$ ; on dit que le point $C$ est *à droite* de $(A, B)$ si $\theta \leqslant 0$ ou si $\theta = \pi$.
On définit alors la quantité :

$$\det\left(\overrightarrow{AB}, \overrightarrow{AC}\right) = (x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A) = \begin{vmatrix} x_B - x_A & x_C - x_A \\ y_B - y_A & y_C - y_A \end{vmatrix}$$

Par la suite on posera $\alpha(A, B, C) = \det\left(\overrightarrow{AB}, \overrightarrow{AC}\right)$.

On admettra (mais vous pouvez chercher à le démontrer s'il vous reste du temps) que

$$\sin\left(\overrightarrow{AB},\overrightarrow{AC}\right) = \frac{\det\left(\overrightarrow{AB},\overrightarrow{AC}\right)}{||\overrightarrow{AB}||.||\overrightarrow{AC}||} = \frac{\alpha(A,B,C)}{AB.AC}$$

Ainsi $C$ est à gauche de $(A,B)$ si et seulement si $\alpha(A,B,C) \geqslant 0$ ; $C$ est à droite de $(A,B)$ si et seulement si $\alpha(A,B,C) \leqslant 0$ ; et $A,B,C$ sont alignés si et seulement si $\alpha(A,B,C) = 0$.

2) Écrire la fonction $\alpha$ :

```
alpha : point -> point -> point -> int
```

Afficher (par exemple avec les fonctions `print_int`, `print_string` et `print_newline`) les triplets $(i,j,k) \in [\![0,20]\!]^3$ tels que $i<j<k$ et $P_i, P_j, P_k$ sont alignés. Résultat :

```
alignement 0 2 5
alignement 6 8 11
alignement 4 6 12
alignement 10 15 16
alignement 2 3 17
alignement 0 9 18
alignement 3 13 18
alignement 1 10 19
alignement 1 9 20
alignement 2 11 20
```

3) Point le plus bas ; le plus haut

Soit $A_0,\dots,A_n$ une séquence de points. Le *point le plus bas* est le point $y_k$ avec $k \in [\![0,n]\!]$ tel que $\forall i \in [\![0,n]\!], y_i \geqslant y_k$. En cas d'égalité on choisira le point le plus à gauche.

On définit de façon similaire le *point le plus haut* (en cas d'égalité on choisit le point le plus à droite).

Écrire deux fonctions :

```
point_bas : point array -> point
point_haut : point array -> point
```

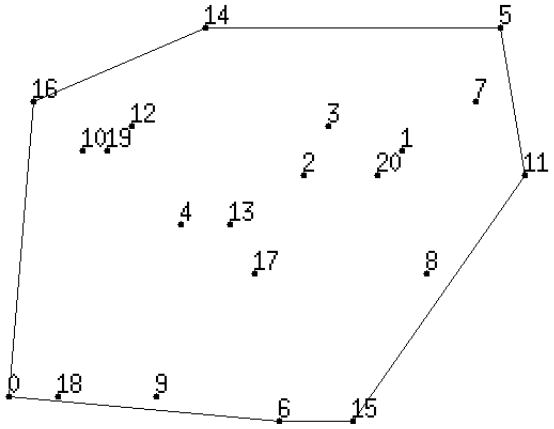
Tests : avec le tableau de points `p` défini précédemment, le point le plus bas est le point $(12,2)$ d'indice 6 ; et le point le plus haut est le point $(21,18)$ d'indice 5.

## 1.2 Une approche "diviser pour régner" (QuickHull)

Soit $A_0,\dots,A_n$ une séquence de points du plan. Une *enveloppe convexe* est une séquence d'indices $i_0,i_1,\dots,i_{p-1} \in [\![0,n]\!]$, 2 à 2 distincts, tels que, en posant $i_p = i_0$ :

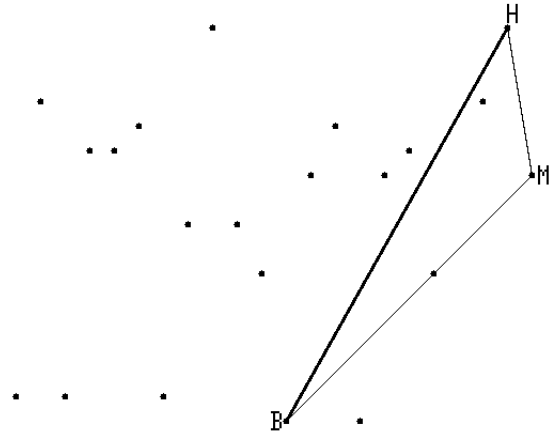
$$\forall k \in [\![1,p]\!], \forall j \in [\![0,n]\!] \setminus \{i_{k-1},i_k\}, A_j \text{ est à gauche de } (A_{i_{k-1}},A_{i_k})$$

On peut visualiser l'enveloppe convexe comme un élastique tendu qui enserrerait tous les points de nuage. Cette notion d'enveloppe convexe a de nombreuses applications, notamment en reconnaissance de forme.




Dans la suite, nous allons désigner respectivement par $B$ le point point le plus bas et par $H$ le point le plus haut (au sens de la question 3). $B$ et $H$ font partie de l'enveloppe convexe.
On calcule alors récursivement l'*enveloppe convexe à droite* de $(B, H)$ : c'est la liste des points de l'enveloppe convexe qui sont à droite de $(B, H)$. Pour cela :

- Soit $\mathcal{D}$ l'ensemble des points du nuage strictement à droite de $(B, H)$, on calcule le point $M \in \mathcal{D}$ tel que $\alpha(B, M, H)$ soit maximum (on pourrait montrer que dans un repère orthonormé direct dont l'axe des ordonnées serait $(BH)$ orienté de $B$ vers $H$, l'abscisse de $M$ serait $\dfrac{\alpha(B, M, H)}{AB}$ : il s'agit donc de chercher le point d'abscisse maximum parmi ceux qui sont à droite de $(B, H)$).
  Un tel point $M$ n'existe pas toujours (si tous les points sont à gauche de $(B, H)$, c'est–à–dire si $\mathcal{D}$ est vide). S'il existe, on l'appelle le *point le plus à droite de* $(B, H)$. En cas d'ex–æquo, on n'impose pas de choix sur $M$.
- Si le point $M$ n'existe pas (aucun point strictement à droite de $(B, H)$) alors on s'arrête là : l'enveloppe convexe à droite $(B, H)$ est réduite au segment $[BH]$.
- Sinon on relance récursivement la recherche de l'enveloppe convexe à droite de $(B, M)$, et l'enveloppe à droite de $(M, H)$, en se restreignant au nuage des points de $\mathcal{D}$ (les points à gauche de $(B, H)$ sont écartés).




4) Écrire une fonction qui calcule le point le plus à droite :

`plus_a_droite : point -> point -> point list -> point * point list`

Si l'ensemble $\mathcal{D}$ des points strictement à droite de (`b`, `h`) (parmi ceux de `l`) est non vide, l'appel `plus_a_droite b h l` renvoie `(m, d)` où `m` est le point de `l` tel que $\alpha$(`b`, `m`, `h`) soit strictement positif et maximum, et `d` est la liste des points de $\mathcal{D}$.

Si l'ensemble $\mathcal{D}$ est vide, l'appel `plus_a_droite b h l` renvoie `(m,[])` où `m` est un point quelconque (par exemple `b`).

5) Écrire une fonction (récursive) qui calcule l'enveloppe convexe à droite :

`quickHull_droite : point -> point -> point list -> point list`

L'appel `quickHull_droite b h l` renvoie l'enveloppe convexe (parmi les points de `l`) à droite de (`b`, `h`) sous forme d'une liste de points.

On s'arrangera pour qu'un même point n'apparaisse pas deux fois.

6) L'enveloppe convexe à gauche de $(B, H)$ est l'enveloppe convexe à droite de $(H, B)$ (car en changeant l'orientation de la droite $(B, H)$ on intervertit la gauche et la droite). Il n'y a ensuite plus qu'à recoller les deux morceaux d'enveloppe conxexe de part–et–d'autre de la droite $(BH)$.

Écrire une fonction qui calcule l'enveloppe convexe totale :

`quickHull : point array -> point list`

On pourra si on veut utiliser la fonction `Array.to_list : 'a array -> 'a list`.

Résultat : la liste des points de l'enveloppe convexe est :

$$[(21, 18); (9, 18); (2, 15); (1, 3); (12, 2); (15, 2); (22, 12)]$$

(mais on peut éventuellement trouver une liste obtenue par permutation circulaire de cette liste). La liste des indices correspondants est $[5; 14; 16; 0; 6; 5; 11]$.

La complexité de cet algorithme est dans le pire des cas $O(n^2)$ mais il est possible de montrer que sa complexité en moyenne est $O(n \log n)$.

# 2 Nombre d'inversions dans une liste

Dans cette section, on s'intéresse au nombre d'inversions dans une liste `l` à `n` éléments, c'est-à-dire au nombre de couples $(i, j) \in [\![1; n]\!]$ tels que $i < j$ et le $i$-ième élément de `l` est strictement plus grand que son $j$-ième élément.

### Une solution naïve

7) Implémenter une fonction `nb_inv_tete : int list -> int` qui calcule le nombre d'éléments d'une liste qui sont strictement inférieurs à la tête de cette liste.

8) En déduire l'implémentation d'une fonction `nb_inv_naive : int list -> int` qui calcule le nombre d'inversions dans une liste.

### Une approche "diviser pour régner"

9) Notons $l_1$ et $l_2$ deux listes telles que `l` $= l_1.l_2$.

   a) **Cas où $l_1$ et $l_2$ sont triées.**
   Exprimer le nombre d'inversions de `l` en fonction du nombre de fois où un élément de $l_2$ est placé devant des éléments de $l_1$ dans la fusion de $l_1$ et $l_2$.

   b) **Cas où $l_1$ et $l_2$ ne sont pas triées.**
   Notons $nb_1$ et $nb_2$ les nombres respectifs d'inversions dans $l_1$ et $l_2$ et notons $lt_1$ et $lt_2$ les listes $l_1$ et $l_2$ triées. Exprimer le nombre d'inversions de `l` en fonction de $nb_1$, $nb_2$ et du nombre d'inversions dans $lt_1.lt_2$.

10) Implémenter une fonction `scinde_stable : 'a list -> 'a list * 'a list` qui découpe une liste `l` en deux listes $l_1$ et $l_2$ de même longueur (à un élément prêt). Les éléments doivent rester dans le même ordre, c'est-à-dire que `l` doit être égale à la concaténation de $l_1$ et $l_2$.

11) Adapter l'implémentation du tri fusion en utilisant la fonction `scinde_stable` et une référence `cpt` qui servira à compter les inversions lors de l'opération de fusion, pour obtenir une fonction `nb_inv_fus : int list -> int` qui calcule le nombre d'inversions dans une liste avec une complexité en $O(n \log n)$.

# 3 Un peu de graphisme sous OCaml

La bibliothèque standard d'OCaml possède une module `Graphics` qui affre quelques fonctionnalités sommaires pour tracer des graphiques.

12) Charger le module et ouvrir la fenêtre graphique :

```
#load "graphics.cma";;
open Graphics;;
open_graph "";;
```

Vous devez voir s'ouvrir une fenêtre graphique, dans laquelle apparaîtront les tracés au fur–et–à–mesure que vous exécuterez les commandes qui les créent. Les coordonnées sont nécessairement entières ; le point de coordonnées $(0, 0)$ étant le coin inférieur gauche.

13) Tracer le nuage de points (sans l'enveloppe convexe). Vous pouvez utiliser les fonctions suivantes :
   — `clear_graph : unit -> unit`
   Efface le contenu de la fenêtre graphique.
   — `size_x : unit -> int` et `size_y : unit -> int` : ces fonctions renvoient respectivement la taille en abscisse et en ordonnée de la fenêtre graphique. Ainsi le coin supérieur droit de la fenêtre graphique a pour coordonnées `(size_x () - 1, size_y () - 1)`.
   Remarque : les points $P_i$ ayant leur abscisse (strictement) comprise entre 0 et 23, et leur ordonnée (strictement) comprise entre 0 et 19, il pourra être nécessaire d'effectuer un changement d'échelle avant de représenter les points dans la fenêtre graphique.
   — `plot : int -> int -> unit`
   Trace un point aux coordonnées passées en paramètre. Le point se limite à un pixel.
   — `plots : (int * int) array -> unit`
   Trace plusieurs points.

— `fill_circle : int -> int -> int -> unit`
Trace un disque (plein). Les paramètres sont dans l'ordre : l'abscisse du centre, l'ordonnée du centre et le rayon.
Remarque : sur la figure de l'énoncé, les points $P_i$ sont représentés par des disques de rayon 3.

Si vous souhaitez peaufiner en rajoutant le numéro de chaque point, vous pouvez aussi utiliser le fonctions :

— `moveto : int -> int -> unit`
Positionne le curseur graphique au point dont les coordonnées sont passées en paramètre (cette fonction ne trace rien).
— `draw_string : string -> unit`
Écrit la chaîne passée en paramètre. Le coin inférieur gauche du texte correspond à la position actuelle du curseur graphique.
— `string_of_int : int -> string`
Convertit un entier en chaîne de caractères.

Il existe de nombreuses autres fonctions proposées par le module `Graphics` d'OCaml. Vous pouvez consulter l'aide sur internet à leur sujet.

14) Tracer l'enveloppe convexe calculée.

On pourra utiliser la fonction `draw_poly : (int * int) array -> unit` qui trace un polygone dont on passe les sommets en paramètre.

On pourra aussi utiliser la fonction `Array.of_list : 'a list -> 'a array`