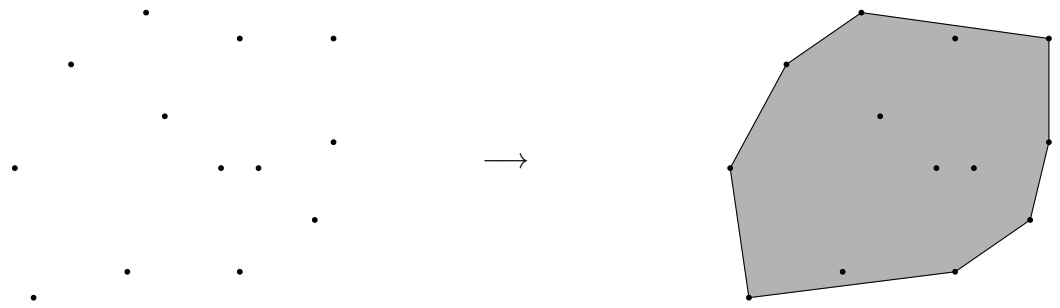
# Enveloppes convexes

Etant donnée une liste de points (ils seront à coordonnées entières dans ce qui suit, mais cela n'a pas d'importance), ou cherche un polygone convexe qui "contient" tous les points de la liste proposée :



La liste de points sera modélisée par une liste de tuples formés de deux entiers (il pourrait aussi s'agir d'une liste de deux entiers, cela n'a guère d'importance ici)

La démarche proposée est la suivante : on détermine le point $P$ situé le plus à gauche. S'il y en a plusieurs, on prendra celui qui est le plus bas, puis on réordonne tous les autres points selon l'angle $(\vec{i}, \overrightarrow{PM})$. Si plusieurs points sont alignés avec $P$, on ne garde que le plus éloigné de $P$ :

On a alors obtenu des points $P, M_0, \ldots, M_n$. Les deux premiers $P$ et $M_0$ font forcément partie du l'enveloppe convexe recherchée et on les ajoute à la liste qui construit notre polygone convexe. Pour chaque nouveau point, $M_i$, on regarde si la configuration formée avec les deux derniers points ajoutés tourne dans le sens positif ou non... Si oui, on ajoute $M_i$, sinon, on retire le dernier point ajouté et on ajoute $M_i$.

Dans l'exemple présenté : on part de $[P, M_0]$, puis on ajoute $M_1$, puis on "remplace" $M_1$ par $M_2$, on ajoute $M_3$, $M_4$. On remplace $M_4$ par $M_5$, on ajoute $M_6$ qu'on remplace aussitôt par $M_7$, on ajoute $M_8$ qu'on remplace par $M_9$, on ajoute $M_{10}$ est c'est fini...

1. Un point important est de déterminer si trois points $A, B, C$ "tournent" dans le sens direct. Pour en décider, on calculera le déterminant de $(\overrightarrow{AB}, \overrightarrow{AC})$ dont le signe donne la réponse à notre question (nul besoin de calculer un angle avec une arctangente par exemple)

   Ecrire alors une fonction `def orientation(a, b, c)` qui associe à trois points $A, B, C$ connus par leurs couples de coordonnées $a, b$ et $c$ la valeur du produit mixte $\mathrm{Det}(\overrightarrow{AB}, \overrightarrow{AC})$.

2. Ecrire encore une fonction `def pointGauche(L)` qui à une liste de points (une liste de couples, ou une liste de listes de deux valeurs) retourne l'indice $i$ du point $P$ d'abscisse minimale (et, le cas échéant, parmi les points de plus petite abscisse s'il y en a plusieurs, retourne celui de plus petite ordonnée)

3. Etant donnée une liste $L$ et une fonction `comp` qui satisfait aux conditions suivantes, pour tous $i, j$ et $k$ :
   - `comp(L[i], L[i]) = 0`
   - Si `comp(L[i], L[j]) >= 0` et `comp(L[j], L[k]) >= 0`, alors `comp(L[i], L[k]) >= 0`.
   - `comp(L[j], L[i]) = -comp(L[i], L[j])`

On souhaite réordonner les éléments de $L$ de telle sorte que pour tout $i$, `comp(L[i], L[i+1]) >= 0`.

Ecrire pour ce faire une fonction `insertSort(L, comp)` qui effectue ce travail en réalisant un tri par insertion.

4. Quelle est la complexité de la fonction précédente ?
5. Ecrire une fonction `prepare(L)` qui, étant donnée une liste $L$ de points, détermine le point $P$ d'abscisse minimale, retire celui-ci de $L$ et réordonne les éléments restants de $L$ de telle sorte que les angles entre $\vec{i}$ et $\overrightarrow{PL[i]}$ (admettant une mesure entre $-\frac{\pi}{2}$ et $\frac{\pi}{2}$) soient en progression croissante.)

   On pourra, au choix, ne retourner que le point $P$ et trier en place ce qui reste de $L$ (on modifie donc la liste fournie en argument) ou bien retourner le couple $(P, L')$ où $L'$ est la liste obtenue à partir de $L$ en retirant le point $P$ et en triant ses éléments.

   On fera bien sûr appel aux fonctions précédentes.
6. Ecrire encore une fonction `faitMenage(p, L)` qui prend en argument un point $p$ et une liste de points $L$ triée (autrement dit les valeurs de retour de `prepare(L)`) et qui retourne une liste de points extraite de $L$, toujours triée, mais où deux points consécutifs ne sauraient être alignés avec $p$. (Lorsque deux points consécutifs de $L$ sont alignés avec $p$, on rappelle qu'on ne garde que celui qui est le plus éloigné de $p$)

   Une fonction qui agit en temps linéaire est attendue (plutôt que de modifier $L$ en en retirant des éléments, il vaut mieux repartir d'une liste vide et y incorporer des éléments de $L$...)
7. Ecrire enfin une fonction `enveloppe(L)` qui d'une liste de points $L$ retourne le polygone convexe recherché. Son premier point sera celui le plus à gauche, et il devra tourner dans le sens positif. Bien sûr, on fera appel aux fonctions précédentes.
8. Quelle est la complexité de la fonction précédente ?
9. Pour en améliorer la complexité, ou souhaite remplacer le tri par insertion par un tri fusion. Ecrire pour ce faire deux fonctions : `merge(L1, L2, comp)` et `mergeSort(L, comp)`.

   Sans surprise, la première suppose triées les deux listes $L1$ et $L2$, et les fusionne en une liste triée $L$, tandis que la seconde fait appel à la première et retourne les éléments de $L$ en une liste triée.
10. Quelle devient la complexité de la fonction `enveloppe(L)` en ayant remplacé la fonction `insertSort(L, comp)` par `mergeSort(L, comp)` ?
11. Compléter la fonction `enveloppe(L)` pour tracer à la fois le nuage de points $L$ (on pourra utiliser `scatter` de matplotlib), le polygone convexe obtenu (`plot` en rajoutant le point initial pour boucler le polygone...) et également l'intérieur du polygone, avec `fill` de matplotlib (on pourra utiliser l'option `alpha` afin que les points à l'intérieur du polygone restent visibles). Les illustrations sur ce sujet ont été réalisés avec matplotlib...
12. Variante : remplacer le tri fusion par l'algorithme de tri rapide, en écrivant deux fonctions :
    `partition(L, a, b, p, comp)` qui modifie en place $L$ pour ses indices de $a$ (compris) à $b$ (non compris) et autour du pivot $p$ (où $a \leqslant p < b$), puis :
    `quickSort(L, comp)` qui bien sûr fait appel à la fonction précédente.

**Annexe :** Si vous voulez travailler sur la liste de points qui est représentée ici, vous pouvez définir la liste L ainsi :

```
L = [(1, 0), (0, 5), (6, 1), (8, 7), (3, 9), (7, 11), (11, 5), (13, 5), \
     (12, 1), (16, 3), (17, 6), (17, 10), (12, 10)]
```

Voici pour référence, le résultat de `prepare(L)` :

```
((0, 5), [(1, 0), (6, 1), (12, 1), (16, 3), (13, 5), (11, 5), (17, 6), (8, 7), (17, 10),
(12, 10), (7, 11), (3, 9)])
```

Quand on fait "le ménage", seul `(11,5)` disparaît, puis l'enveloppe convexe obtenue est :

```
 [(0, 5), (1, 0), (12, 1), (16, 3), (17, 6), (17, 10), (7, 11), (3, 9)]
```