Enveloppe convexe

Algorithme de Graham

Marc Lorenzi

6 janvier 2019

In [1]:
```
import matplotlib.pyplot as plt
import random
from math import sqrt
```

In [2]:
```
plt.rcParams['figure.figsize'] = (8, 8)
```

1 Introduction

L'enveloppe convexe $\mathcal{C}(A)$ d'un ensemble A de points du plan est la plus petite partie convexe du plan contenant A. Lorsque l'ensemble A est fini l'ensemble $\mathcal{C}(A)$ peut être "calculé" (nous verrons ce qu'il faut entendre par là) par un certain nombre d'algorithmes. Nous allons commencer par un peu de théorie sur les parties convexes du plan. Le lecteur pressé peut se contenter de lire les définitions et les énoncés des propriétés ..

1.1 Ensembles convexes

Définition : Un ensemble $A \subset \mathbb{R}^2$ est convexe lorsque pour tous points $p, q \in A$, le segment $[p, q]$ est inclus dans A.

Exemples :

- $\mathbb{R}^2$ et $\varnothing$ sont convexes.
- Un segment est convexe.
- Un triangle, un disque (avec leur intérieur cela va de soi) sont convexes.
- etc.

Propriété : Toute intersection d'ensembles convexes est encore convexe.

Démonstration : Soit $(A_i)_{i\in I}$ une famille d'ensembles convexes. Soit $A = \bigcap_{i\in I} A_i$. Soient $p, q \in A$. p et q appartiennent à tous les A_i, donc $\forall i \in I$, $[p, q] \subset A_i$. Ainsi, $[p, q] \subset A$.

Remarque : Soient $p, q, m \in \mathbb{R}^2$. Soit Ω un point quelconque du plan. On a $m \in [p, q]$ si et seulement si il existe $\lambda \in [0, 1]$ tel que $\overrightarrow{pm} = \lambda\overrightarrow{pq}$, c'est à dire $\overrightarrow{\Omega m} - \overrightarrow{\Omega p} = \lambda(\overrightarrow{\Omega q} - \overrightarrow{\Omega p})$. Ceci peut encore s'écrire

$$\overrightarrow{\Omega m} = (1 - \lambda)\overrightarrow{\Omega p} + \lambda\overrightarrow{\Omega q}$$

Généralisons cette égalité ...

1.2 Combinaisons convexes

Proposition : Soient $p_1, \dots, p_n$ n points du plan ($n \geq 1$). Soient $\lambda_1, \dots, \lambda_n$ n réels tels que $\sum_{i=1}^{n} \lambda_i = 1$. Soit $\Omega \in \mathbb{R}^2$. Soit $p \in \mathbb{R}^2$ le point du plan défini par

$$\overrightarrow{\Omega p} = \sum_{i=1}^{n} \lambda_i \overrightarrow{\Omega p_i}$$

Alors le point p ne dépend pas de Ω.

Notation : Nous noterons plus simplement $p = \sum_{i=1}^{n} \lambda_i p_i$.

Démonstration : Laissée au lecteur. Prendre deux points Ω et Ω' et faire intervenir le vecteur $\overrightarrow{\Omega\Omega'}$.

Avec les notations ci-dessus, lorsque pour tout i $\lambda_i \geq 0$ on dit que p est une **combinaison convexe** des p_i.

Remarque : Un cas important est le cas où $n = 2$. Une combinaison convexe des points p et q est un point de la forme $(1 - \lambda)p + \lambda q$, où $0 \leq \lambda \leq 1$. L'ensemble de toutes les combinaisons convexes de p et q est donc le **segment** $[p, q]$.

1.3 Enveloppe convexe

Proposition : Soit $A \subset \mathbb{R}^2$. Il existe un ensemble convexe contenant A minimum au sens de l'inclusion. Cet ensemble s'appelle **l'enveloppe convexe** de A.

Notation : Nous noterons $\mathcal{C}(A)$ l'enveloppe convexe de A.

Démonstration : L'ensemble A est inclus dans au moins un ensemble convexe, à savoir $\mathbb{R}^2$. Considérons tous les ensembles convexes contenant A et formons leur intersection $\mathcal{C}$. C'est encore un ensemble convexe qui contient A, et c'est clairement le plus petit au sens de l'inclusion.

Proposition : Soit $A \subset \mathbb{R}^2$. $\mathcal{C}(A)$ est l'ensemble des combinaisons convexes de points de A.

Démonstration : Soit B l'ensemble des combinaisons convexes de points de A. Montrons que $B = \mathcal{C}(A)$.

Montrons tout d'abord que B est convexe. Soient $x, y \in B$. En numérotant judicieusement, on peut écrire $x = \sum_{i=1}^{m} \lambda_i p_i$ et $y = \sum_{i=m+1}^{n} \lambda_i p_i$ où les $p_i \in A$ et les λ_i sont des réels positifs tels que $\sum_{i=1}^{m} \lambda_i = \sum_{i=m+1}^{n} \lambda_i = 1$. Soit $z \in [x, y]$. On a $z = (1 - \lambda)x + \lambda y$ où $0 \leq \lambda \leq 1$. Petit calcul :

$$z = \sum_{i=1}^{m} (1 - \lambda)\lambda_i p_i + \sum_{i=m+1}^{n} \lambda\lambda_i p_i = \sum_{i=1}^{n} \mu_i p_i$$

où $\mu_i = (1 - \lambda)\lambda_i$ si $1 \leq i \leq m$ et $\mu_i = \lambda\lambda_i$ si $m + 1 \leq i \leq n$. On vérifie aisément que pour tout i, $\mu_i \geq 0$, et $\sum_{i=1}^{n} \mu_i = 1$. Ainsi, z est une combinaison convexe de points de A, donc $z \in B$.

Montrons maintenant que $B \subset \mathcal{C}(A)$. Précisément, montrons par récurrence sur $n \geq 1$ que toute combinaison convexe $\sum_{i=1}^{n} \lambda_i p_i$ de points de A est un point de $\mathcal{C}(A)$. C'est évident pour $n = 1$. Soit maintenant $n \geq 1$. Supposons la propriété vérifiée pour n. Soit $x = \sum_{i=1}^{n+1} \lambda_i p_i$.

- Si $\lambda_{n+1} = 1$, alors tous les autres λ_i sont nuls puisque leur somme vaut 1. Donc $x = p_{n+1} \in A \subset \mathcal{C}(A)$.
- Sinon, posons $y = \sum_{i=1}^{n} \mu_i p_i$ où $\mu_i = \frac{\lambda_i}{1 - \lambda_{n+1}}$. On vérifie que $\sum_{i=1}^{n} \mu_i = 1$ et que les μ_i sont positifs. Ainsi, y est une combinaisons convexe de n points de A. Par l'hypothèse de récurrence, $y \in \mathcal{C}(A)$. Pour finir, remarquons que $x = (1 - \lambda_{n+1})y + \lambda_{n+1} p_{n+1} \in [y, p_{n+1}]$, donc $x \in \mathcal{C}(A)$.

Pour terminer il est clair que $A \subset B$. B est donc un convexe contenant A et inclus dans $\mathcal{C}(A)$. Par minimalité de $\mathcal{C}(A)$, on a $B = \mathcal{C}(A)$.

1.4 Points extrémaux

Définition : Soit $C \subset \mathbb{R}^2$ un ensemble convexe. Un point **extrémal** (ou **sommet**) de C est un point $p \in C$ vérifiant :

$$\forall a, b \in C, p \in [a, b] \Rightarrow a = b \ \texttt{ou} \ b = p$$

Proposition : Soit C un ensemble convexe. Soit $p \in C$. Alors p est un point extrémal de C si et seulement si $C \setminus \{p\}$ est convexe.

Démonstration : p n'est pas extrémal si et seulement si il existe $a, b \in C$ tels que $a \neq p$, $b \neq p$ et $p \in [a, b]$. Ou encore il existe $a, b \in C \setminus \{p\}$ tels que $p \in [a, b]$. On a donc immédiatement que si p n'est pas extrémal alors $C \setminus \{p\}$ n'est pas convexe. Inversement, supposons p extrémal. Soient $a, b \in C \setminus \{p\}$. On a $[a, b] \subset C$ puisque C est convexe. Et $p \notin [a, b]$ puisque p est extrémal. Donc, $[a, b] \subset C \setminus \{p\}$. Donc $C \setminus \{p\}$ est convexe.

1.5 Ensembles finis

Définition : Un **polygone convexe** est l'enveloppe convexe d'un ensemble fini non vide.

Exemples :

- Un segment est l'enveloppe convexe de deux points distincts.
- Un triangle est l'enveloppe convexe de trois points non alignés.
- Un segment est l'enveloppe convexe de 15 points alignés dont deux au moins sont distincts.

Proposition : Soit A un ensemble fini non vide. Alors

- Les sommets de $\mathcal{C}(A)$ sont des points de A.
- $\mathcal{C}(A)$ est l'enveloppe convexe de ses sommets.

Démonstration :

- Soit $p \in \mathcal{C}(A) \setminus A$. Alors $p = \sum_{i \in I} \lambda_i p_i$ où l'ensemble I est fini, les $p_i \in A$ et les λ_i sont positifs de somme 1. Choisissons de plus I de cardinal minimal. Commme $p \notin A$, deux au moins des λ_i, disons λ_{i_1} et λ_{i_2}, sont non nuls. Partitionnons I en $I = I_1 \cup I_2$ des sorte que $\lambda_1 \in I_1$ et $\lambda_2 \in I_2$. Soient $\mu = \sum_{i \in I_1} \lambda_i$ et $\nu = \sum_{i \in I_2} \lambda_i$. Par notre choix de I_1 et I_2, μ et ν sont non nuls. De plus $\mu + \nu = 1$. Posons $q = \sum_{i \in I_1} \frac{\lambda_i}{\mu} p_i$ et $r = \sum_{i \in I_2} \frac{\lambda_i}{\nu} p_i$. On vérifie aisément que $q, r \in \mathcal{C}(A)$ et $p = \mu q + \nu r \in [q, r]$. De plus $p \neq q$ car I_1 a strictement moins d'éléments que I et on a choisi I de cardinal minimal. De même, $p \neq r$. Ainsi, p n'est pas un point extrémal de $\mathcal{C}(A)$.
- Montrons par récurrence sur $n = Card\, A$ que $\mathcal{C}(A)$ est l'enveloppe convexe de ses points extrémaux.
  - Pour $n = 1$ c'est évident : $\mathcal{C}(A) = A$ a un unique point extrémal.
  - Soit $n \geq 2$. Supposons la propriété vraie pour $n - 1$. Soit $A = \{p_1, \dots, p_n\}$. Si tous les p_i sont des points extrémaux de $\mathcal{C}(A)$, il n'y a rien à prouver. Sinon, l'un des points, par exemple p_n, n'est pas un point extrémal de $\mathcal{C}(A)$. p_n s'écrit donc comme combinaison convexe des autres points de A : $p_n = \sum_{i=1}^{n-1} \lambda_i p_i$. On vérifie aisément que tout point de $\mathcal{C}(A)$ est alors une combinaison convexe de $p_1, \dots, p_{n-1}$. Ainsi, $\mathcal{C}(A) = \mathcal{C}(A')$ où $A' = \{p_1, \dots, p_{n-1}\}$. Par l'hypothèse de récurrence, $\mathcal{C}(A')$ est l'enveloppe convexe de ses sommets. Il reste à montrer que les sommets de $\mathcal{C}(A')$ sont les sommets de $\mathcal{C}(A)$. Cela se fait en revenant à la définition de sommet (laissé au lecteur).

Proposition : Soit $A \subset \mathbb{R}^2$ un ensemble fini. On suppose que A contient trois points non alignés. Alors $\mathcal{C}(A)$ est la réunion des triangles dont les sommets sont des points de A.

Démonstration : Soient p, q, r trois points de A non alignés. Les points du triangle (pqr) sont les combinaisons convexes de p, q et r. Ce sont donc des points de $\mathcal{C}(A)$. Ainsi, $(pqr) \subset \mathcal{C}(A)$. On a une inclusion.

Pour l'inclusion inverse, histoire de ne pas rallonger inconsidérément ce notebook, je vais vous demander de faire un petit dessin.

- Dessinez un polygone convexe $\mathcal{C}$. Il représente $\mathcal{C}(A)$ et ses sommets sont des points de A.
- Choisissez un sommet de $\mathcal{C}$, celui que vous voulez.
- Tracez tous les segments reliant ce sommet aux autres sommets.
- Maintenant choisissez un point dans le polygone et constatez qu'il est dans un triangle :-).

1.5 Qu'allons-nous faire ?

L'enveloppe convexe d'un ensemble fini est donc caractérisée par l'ensemble de ses points extrémaux. Il existe un certain nombre d'algorithmes permettant le calcul des sommets en question. Nous allons étudier l'un d'entre-eux : l'algorithme de **Graham**, publié par Ronald Graham en 1972.

Dorénavant, nous ne considérerons que des points à coordonnées entières, ce qui permettra à nos algorithmes de calculer des valeurs exactes. Lorsque les coordonnées des points sont connues de façon approchée, des problèmes se posent, comme par exemple celui de savoir si trois points sont alignés.

Avant de nous attaquer à l'algorithme de Graham, préparons le terrain.

2 Quelques fonctions utiles

2.1 Liste de points aléatoires

Pour tester nos fonctions il sera judicieux de posséder des listes de points aléatoires. La fonction `random_list` prend en paramètre un entier n et renvoie une liste de n points à coordonnées entières choisies aléatoirement. Pour avoir de jolis graphiques, (x, y) suit une distribution gaussienne de moyenne $(0, 0)$ et d'écart-type $(1000, 1000)$ avec une corrélation égale à 0.3.

La fonction `bivariate` simule une distribution gaussienne à deux dimensions. Je n'entrerai pas dans les détails.

In [3]:
```
def bivariate(mu1, mu2, s1, s2, r):
    z1 = random.gauss(0, 1)
    z2 = random.gauss(0, 1)
    x1 = mu1 + s1 * z1
    x2 = mu2 + s2 * (z1 * r + z2 * sqrt(1 - r ** 2))
    return (x1, x2)
```

In [4]:
```
def random_list(n):
    M = 1000
    s = []
    cnt = 0
    for k in range(n):
        x, y = bivariate(0, 0, 1000, 1000, 0.3)
        x, y = int(x), int(y)
        if not (x, y) in s: s.append((x, y))
    return s
```

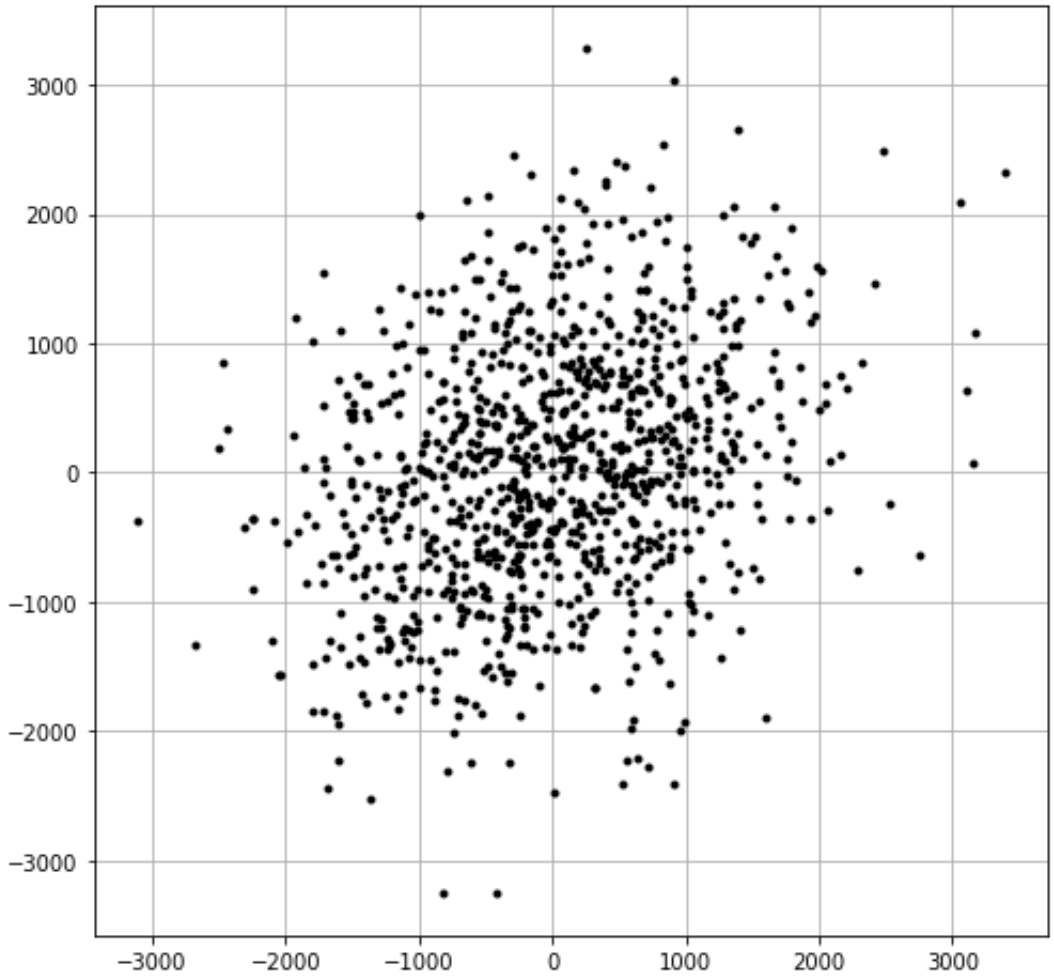
In [5]:
```
s = random_list(100)
print(s)
```

```
[(1483, 1857), (-184, 85), (-682, -427), (629, 1), (1586, -115), (-1
369, 220), (982, 1459), (-1423, 607), (-1259, -1588), (135, 566), (6
89, -691), (224, 705), (-779, -1161), (942, -303), (-460, -310), (-5
2, -35), (1183, -840), (1892, 510), (-727, -402), (-1396, -346), (17
04, 1005), (-587, -378), (-35, -215), (-1035, -279), (-24, 540), (17
3, 887), (2041, 978), (635, 110), (572, 418), (-2256, -176), (331, 9
92), (-448, -760), (2743, -1100), (-805, -1212), (303, 218), (-598,
77), (185, 568), (-414, 731), (1258, -344), (337, -292), (1126, 180)
, (752, -358), (-5, -282), (-776, -856), (754, 779), (-53, 1546), (-
1118, 499), (-222, 563), (43, 2383), (562, 546), (2170, 299), (-994,
-812), (-51, 944), (103, -249), (449, -583), (763, -109), (-1343, -2
13), (1582, -2050), (-87, -1341), (-422, -316), (811, -878), (-278,
618), (-31, 1805), (-3089, -1978), (1296, 425), (172, 727), (-533, -
2674), (-273, -833), (948, 586), (1851, 645), (176, -963), (677, 166
5), (2421, 402), (1449, 58), (304, 806), (184, -681), (-120, -491),
(-1498, 12), (-264, 1024), (172, -251), (-1308, 1578), (414, 827), (
-306, 895), (-323, -1346), (-1669, -394), (-313, -1307), (-123, 728)
, (752, 1708), (119, 1183), (-2183, -2038), (-131, -1547), (-353, -2
52), (960, -612), (891, 326), (956, -164), (-981, -2682), (-467, -10
43), (617, 859), (-347, 269), (-65, 2525)]
```

Un petit dessin ?

In [6]:
```
def tracer(A):
    xs = [P[0] for P in A]
    ys = [P[1] for P in A]
    plt.plot(xs, ys, 'ok', markersize=3)
    plt.grid()
```

In [7]:
```
s = random_list(1000)
tracer(s)
```



2.2 Points et vecteurs

La fonction `determinant` calcule le déterminant de deux vecteurs u et v du plan, représentés par des couples. Rappelons que

- $\det(u, v) > 0$ si et seulement si (u, v) est une base directe.
- $\det(u, v) < 0$ si et seulement si (u, v) est une base indirecte.
- $\det(u, v) = 0$ si et seulement si (u, v) est une famille liée.

In [8]:
```
def determinant(u, v):
    return u[0] * v[1] - u[1] * v[0]
```

In [9]:
```
determinant((1, 2), (3, 4))
```

Out[9]:
```
-2
```

La fonction `vecteur` prend en paramètres deux points A et B, représentés par des couples, et renvoie le vecteur $\overrightarrow{AB}$, lui aussi représenté par un couple.

In [10]:
```
def vecteur(A, B):
    return (B[0] - A[0], B[1] - A[1])
```

In [11]:
```
vecteur((1, 2), (3, 5))
```

Out[11]:
```
(2, 3)
```

2.3 Angles et distances

Étant donnés trois points A, B, C où B et C sont distincts, nous aurons bientôt besoin de savoir si la famille $(\overrightarrow{AB}, \overrightarrow{AC})$ est libre ou liée, et, si elle est libre, si c'est une base directe ou indirecte.

La fonction `pseudo_angle` prend les points A, B, C en paramètres. Elle renvoie :

- 0 si la famille $(\overrightarrow{AB}, \overrightarrow{AC})$ est liée.
- Un réel strictement positif si $(\overrightarrow{AB}, \overrightarrow{AC})$ est une base directe.
- Un réel strictement négatif si $(\overrightarrow{AB}, \overrightarrow{AC})$ est une base indirecte.

In [12]:
```
def pseudo_angle(A, B, C):
    u = vecteur(A, B)
    v = vecteur(A, C)
    return determinant(u, v)
```

In [13]:
```
pseudo_angle((0, 0), (2, 0), (3, 3))
```

Out[13]:
```
6
```

Nous aurons également besoin de comparer des distances entre points. La valeur exacte des distances ne nous intéressera pas et une distance au carré sera suffisante pour nos objectifs.

In [14]:
```
def distance(A, B):
    x, y = vecteur(A, B)
    return x ** 2 + y ** 2
```

In [15]:
```
distance((1, 2), (3, 5))
```

Out[15]:
```
13
```

2.4 L'odre l'exicographique

Nous définissons une relation $\leq_L$ sur $\mathbb{R}^2$ en posant :

$$(x, y) \leq_L (x', y') \Leftrightarrow y < y' \vee (y = y' \wedge x \leq x')$$

Proposition : La relation $\leq_L$ est un ordre total sur $\mathbb{R}^2$.

Démonstration : sans difficulté.

Cet ordre s'appelle **l'ordre lexicographique**. Remarquons tout de même que nous commençons par comparer les **ordonnées** des points, et en cas d'égalité nous comparons leurs **abscisses**. On fait souvent le contraire lorsqu'on définit l'ordre lexicographique.

La fonction `ordre_lexico` renvoie `True` si $A \leq_L B$ et `False` sinon.

In [16]:
```
def ordre_lexico(A, B):
    return (A[1] < B[1]) or (A[1] == B[1] and A[0] <= B[0])
```

In [17]:
```
ordre_lexico((2, 2), (3, 1))
```

Out[17]:
```
False
```

La fonction `min_lexico` renvoie le plus petit élément de la liste de points s pour l'ordre lexicographique $\leq_L$.

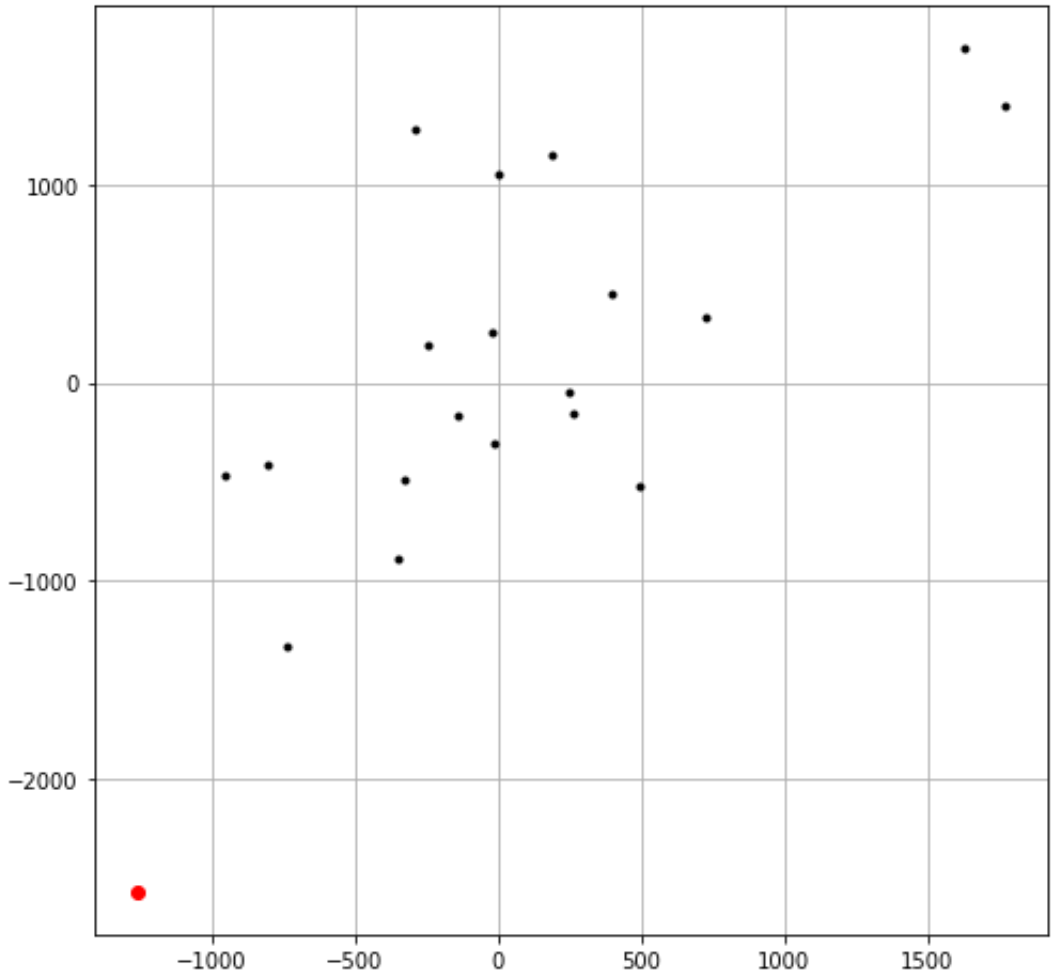
In [18]:
```
def min_lexico(s):
    m = s[0]
    for x in s:
        if ordre_lexico(x, m): m = x
    return m
```

In [19]:
```
s = random_list(10)
print(s)
print(min_lexico(s))
```

```
[(-181, 1543), (-800, -1390), (-1257, 196), (-97, -1075), (-123, 190
3), (-1571, -353), (-409, -43), (745, -707), (2791, 88), (196, -585)
]
(-800, -1390)
```

In [20]:
```
s = random_list(20)
x, y = min_lexico(s)
tracer(s)
plt.plot([x], [y], 'ro', markersize=6)
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x115a09908>]
```



2.5 Une autre relation d'ordre

Soit Ω un point fixé du plan. Nous définissons une relation sur les points du plan de la façon suivante : pour tous $A, B \in \mathbb{R}^2$, $A \preceq B$ lorsque :

- La famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est une base directe du plan

ou bien

- La famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est liée et $d(\Omega, A) \leq d(\Omega, B)$.

Proposition : Soit $\mathcal{E} = \{A \in \mathbb{R}^2, \Omega \preceq_L A\}$. La relation $\preceq$ est une relation d'ordre total sur $\mathcal{E}$.

Démonstration :

- Soit $A \in \mathcal{E}$. La famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega A})$ est liée et $d(\Omega, A) = d(\Omega, A)$, donc $A \preceq A$. La relation $\preceq$ est réflexive.
- Soient $A, B \in \mathcal{E}$. Supposons que $A \preceq B$ et $B \preceq A$. Si $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est une base directe, alors $(\overrightarrow{\Omega B}, \overrightarrow{\Omega A})$ est une base indirecte et on ne peut pas avoir $B \preceq A$. Comme $A \preceq B$, la famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est donc liée, et $d(\Omega, A) \leq d(\Omega, B)$. Comme $B \preceq A$, on a aussi $d(\Omega, B) \leq d(\Omega, A)$. Ainsi, $d(\Omega, A) = d(\Omega, B)$. Les points A, B, Ω sont alignés et A et B sont à la même distance de Ω. Donc, $\overrightarrow{\Omega B} = \pm\overrightarrow{\Omega A}$. On laisse au lecteur la fin de la démonstration (on est dans $\mathcal{E}$!).
- Soient $A, B, C \in \mathcal{E}$. Supposons que $A \preceq B$ et $B \preceq C$. Il y a 4 cas à considérer selon que des familles sont liées ou pas. Je ne détaille pas : si l'une des familles est liée c'est facile. Si les deux familles $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ et $(\overrightarrow{\Omega B}, \overrightarrow{\Omega C})$ sont des bases directes alors la famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega C})$ est aussi une base directe parce que $A, B, C \in \mathcal{E}$.
- Montrons enfin que l'ordre est total. Soient $A, B \in \mathcal{E}$. Si la famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est une base directe, alors $A \preceq B$. Si la famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est une base indirecte, alors $B \preceq A$. Sinon, la famille $(\overrightarrow{\Omega A}, \overrightarrow{\Omega B})$ est liée. Si $d(\Omega, A) \leq d(\Omega, B)$ alors $A \preceq B$, sinon $B \preceq A$. Dans tous les cas, les points A et B sont comparables par la relation $\preceq$.

Proposition : L'ensemble $\mathcal{E}$ a un plus petit élément pour l'ordre $\preceq$: le point Ω.

Démonstration : Tout d'abord, $\Omega \in \mathcal{E}$ puisque $\Omega \preceq_L \Omega$. Soit $A \in \mathcal{E}$. La famille $(\overrightarrow{\Omega\Omega}, \overrightarrow{\Omega A}) = (\vec{0}, \overrightarrow{\Omega A})$ est évidemment liée, et $d(\Omega, \Omega) = 0 \leq d(\Omega, A)$. Donc $\Omega \preceq A$.

La fonction `comp` ci-dessous prend en paramètres trois points Ω, A, B. Elle renvoie `True` si $A \preceq B$ et `False` sinon.

In [21]:
```
def comp(Omega, A, B):
    t = pseudo_angle(Omega, A, B)
    if t > 0: return True
    elif t < 0: return False
    else: return distance(Omega, A) <= distance(Omega, B)
```

2.6 Trier par angles et distances

Nous aurons besoin dans l'algorithme de Graham d'effectuer des tris. Je ne décris pas la fonction `quicksort` ci-dessous. Pour plus de détails référez vous au notebook sur le **tri rapide**.

La fonction `quicksort` prend en paramètres une liste d'objets `s` et une fonction `ordre` censée représenter une relation. Elle trie sur place la liste s pour la relation `ordre`. Évidemment, pour pouvoir vraiment parler de "tri" la relation `ordre` doit être une ... relation d'ordre.

In [22]:
```
def partition(s, l, r, ordre):
    k = random.randint(l, r)
    s[k], s[r] = s[r], s[k]
    i = l - 1
    for j in range(l, r):
        if ordre(s[j], s[r]):
            i = i + 1
            s[i], s[j] = s[j], s[i]
    s[i + 1], s[r] = s[r], s[i + 1]
    return i + 1

def tri_aux(s, l, r, ordre):
    if l <= r:
        q = partition(s, l, r, ordre)
        tri_aux(s, l, q - 1, ordre)
        tri_aux(s, q + 1, r, ordre)

def quicksort(s, ordre):
    tri_aux(s, 0, len(s) - 1, ordre)
```

Quels types d'ordres allons nous utiliser dans l'algorithme de Graham ? Eh bien l'ordre "angles-distances" sur l'ensemble $\mathcal{E}$ dont nous avons parlé plus haut. La fonction `trier_angles_distances` fait le travail.

In [23]:
```
def trier_angles_distances(Omega, s):
    ordre = lambda A, B: comp(Omega, A, B)
    quicksort(s, ordre)
```

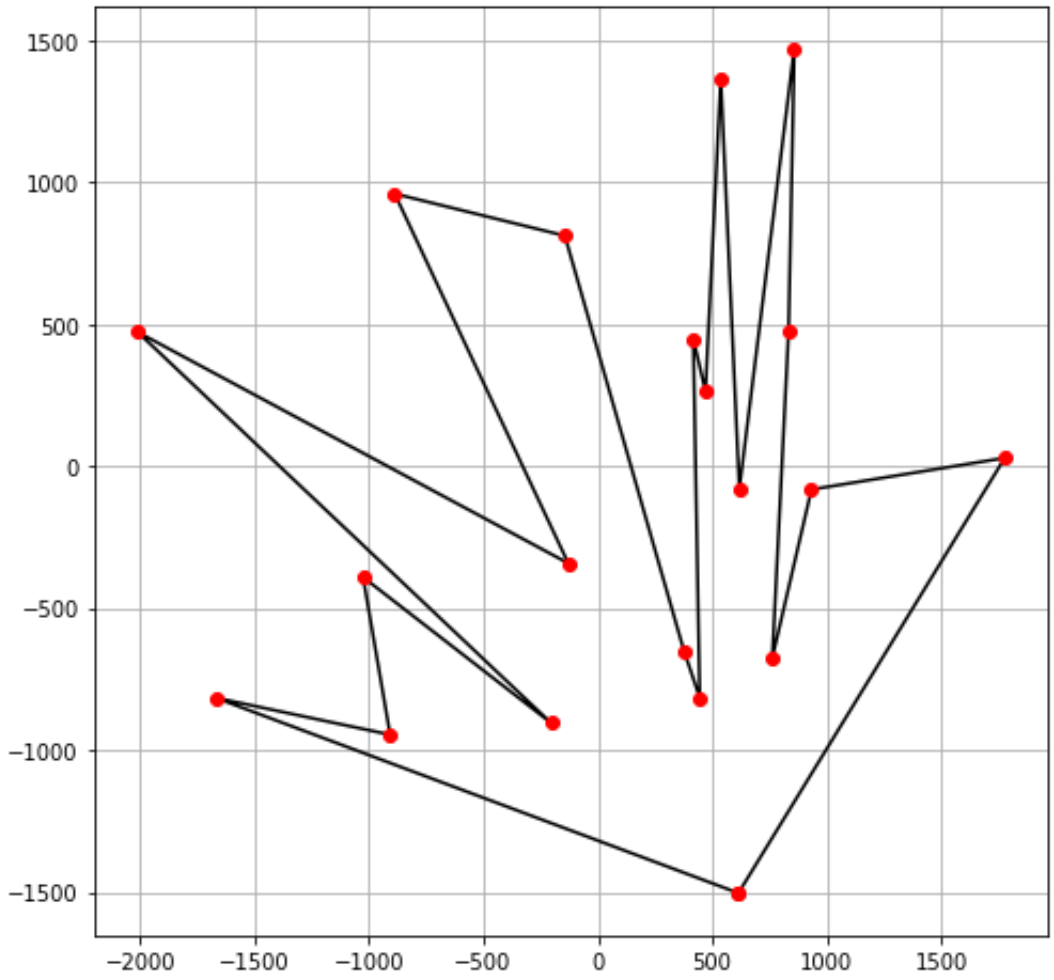
In [24]:
```
s = random_list(10)
print(s)
Omega = min_lexico(s)
print(Omega)
trier_angles_distances(Omega, s)
print(s)
```

```
[(-1233, -131), (-2105, 240), (-110, 913), (-583, 124), (-784, -1163
), (-13, -1530), (1153, 1175), (576, -547), (1395, 274), (664, -801)
]
(-13, -1530)
[(-13, -1530), (664, -801), (1395, 274), (576, -547), (1153, 1175),
(-110, 913), (-583, 124), (-1233, -131), (-2105, 240), (-784, -1163)
]
```

La fonction `tracer2` prend en paramètres une liste de points A. Elle trace les points de A en rouge et relie les sommets de A.

In [25]:
```
def tracer2(A):
    xs = [P[0] for P in A]
    ys = [P[1] for P in A]
    xs.append(A[0][0])
    ys.append(A[0][1])
    plt.plot(xs, ys, 'k')
    plt.plot(xs, ys, 'ro', markersize=6)
    plt.grid()
```

In [26]:
```
s = random_list(20)
Omega = min_lexico(s)
trier_angles_distances(Omega, s)
tracer2(s)
```



Bon, tout a l'air de fonctionner.

3 L'algorithme de Graham

3.1 L'algorithme

Regardez la figure ci-dessus. Pour obtenir l'enveloppe convexe de l'ensemble A, l'idée est de partir du point de plus petite ordonnée et de sélectionner certains des sommets de A en tournant dans le sens trigonométrique. Quels sommets ? C'est ce que nous allons voir maintenant.

Voici l'algorithme de Graham. Soit A un ensemble fini de points du plan, représenté par une liste de couples distincts d'entiers. On suppose que A a au moins 3 éléments. Dans le cas contraire, le calcul de $\mathcal{C}(A)$ est trivial.

On dispose d'une **pile** S. L'algorithme fonctionne comme suit :

- Soit Ω le plus petit point de A pour l'ordre lexicographique. On a alors $A \subset \mathcal{E}$ où $\mathcal{E}$ a été défini plus haut.
- Trier A pour la relation $\preceq$. On a maintenant $A = [p_0, \dots, p_{n-1}]$ où $p_0 = \Omega \preceq p_1 \preceq \dots \preceq p_{n-1}$, et $n \geq 3$ est le cardinal de A.
- Empiler p_0, p_1 et p_2.
- Pour i allant de 3 à $n-1$:
  - Soient p le sommet de S, q le sous-sommet de S. Tant que la famille $(\overrightarrow{qp}, \overrightarrow{qp_i})$ est une base indirecte, dépiler S (remarquer qu'à chaque dépilement, p et q changent !).
  - Empiler p_i.
- Renvoyer S : la pile contient les sommets de $\mathcal{C}(A)$.

Le fichier `Graham.py` est une animation de l'algorithme de Graham. Vous pouvez l'exécuter à part, dans une ligne de commande, ou bien évaluer la cellule ci-dessous. Si vous voulez plus ou moins de points, ou une animation plus ou moins rapide, ouvrez le fichier dans un éditeur et modifiez la valeur des variables globales au début du fichier.

Lors de l'animation, un trait **rouge** est tracé à la fin de chaque itération de la boucle `pour i ...`. Il représente un succès (provisoire) dans la recherche de l'enveloppe convexe. Ce succès peut être invalidé à l'itération suivante : lors de la boucle `Tant que la famille ...`, certains traits **verts** sont tracés. Des triangles comportant un côté rouge et deux côtés verts apparaissent. Selon la position relative des traits verts et du trait rouge, on sort ou pas de la boucle `Tant que`.

In [28]:
```
%run Graham.py
```

Nous allons pour l'instant faire aveuglément confiance à Ronald Graham, coder cet algorithme et l'exécuter. À la fin de ce notebook nous prouverons sa correction et nous étudierons sa complexité.

Voici la fonction `graham`.

In [29]:
```
def graham(A):
    S = pile_vide()
    Omega = min_lexico(A)
    trier_angles_distances(Omega, A)
    empiler(S, A[0])
    empiler(S, A[1])
    empiler(S, A[2])
    for i in range(3, len(A)):
        while pseudo_angle(sous_sommet(S), sommet(S), A[i]) < 0:
            depiler(S)
        empiler(S, A[i])
    return S
```

Que nous reste-t-il à écrire ? Eh bien les opérations de pile. Mais c'est facile, on peut modéliser une pile par une liste.

3.2 Piles

In [30]:
```
def pile_vide(): return []
def empiler(S, A): S.append(A)
def depiler(S): S.pop()

def sommet(S): return S[-1]
def sous_sommet(S): return S[-2]
```

In [31]:
```
S = pile_vide()
empiler(S, 1)
empiler(S, 3)
empiler(S, 7)
print(S)
depiler(S)
depiler(S)
print(S)
```

```
[1, 3, 7]
[1]
```

3.3 Premier test

Nous pouvons tester notre algorithme.

In [32]:
```
A = random_list(10)
print('A =', A)
print('C(A) =', graham(A))
```

```
A = [(821, 663), (-358, 339), (818, 847), (-500, 733), (-757, -871),
(-2665, 148), (679, 1071), (-102, 2043), (-844, -39), (-711, 537)]
C(A) = [(-757, -871), (821, 663), (818, 847), (679, 1071), (-102, 20
43), (-2665, 148)]
```

Euh, oui, bon, peut-être, il vaudrait mieux dessiner tout cela.

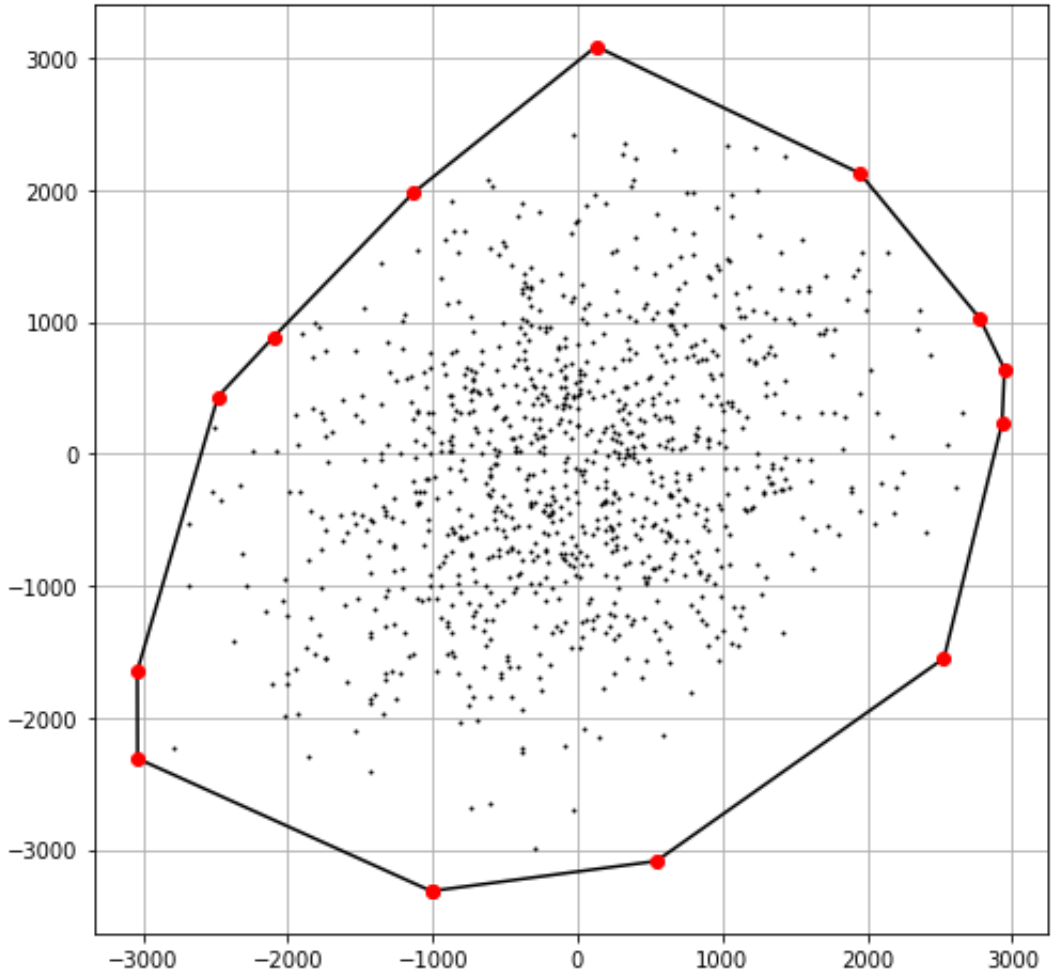
3.3 Dessiner le tout

La fonction `tracer_graham` prend en paramètres une liste de points A et une seconde liste S censée contenir les sommets de A. Elle trace les points de A en noir, les points de S en rouge, et relie les sommets de S.

In [33]:
```
def tracer_graham(A, S, ms=1):
    xs = [P[0] for P in A]
    ys = [P[1] for P in A]
    plt.plot(xs, ys, 'ok', markersize=ms)
    xs = [P[0] for P in S]
    ys = [P[1] for P in S]
    xs.append(S[0][0])
    ys.append(S[0][1])
    plt.plot(xs, ys, 'k')
    plt.plot(xs, ys, 'ro', markersize=6)
    plt.grid()
```

In [34]:
```
A = random_list(1000)
S = graham(A)
print('nombre de sommets :', len(S))
tracer_graham(A, S)
```

```
nombre de sommets : 13
```



4 Étude de l'algorithme

4.1 Non "plantage" de l'algorithme de Graham

Rappelons l'algorithme :

1. Soit Ω le plus petit point de A pour l'ordre lexicographique. On a alors $A \subset \mathcal{E} = \{p \in \mathbb{R}^2, \Omega \preceq_L p\}$.
2. Trier A pour la relation $\preceq$. On a maintenant $A = [p_0, \dots, p_{n-1}]$ où $p_0 = \Omega \preceq p_1 \preceq \dots \preceq p_{n-1}$, et $n \geq 3$ est le cardinal de A.
3. Empiler p_0, p_1 et p_2.
4. Pour i allant de 3 à $n-1$:
    - Soient p le sommet de S, q le sous-sommet de S. Tant que la famille $(\overrightarrow{pq}, \overrightarrow{pp_i})$ est une base directe, dépiler S.
    - Empiler p_i.
5. Renvoyer S, la pile contient les sommets de $\mathcal{C}(A)$.

Pourquoi la fonction `graham` ne plante-t-elle pas ? Dit autrement, n'est-il pas possible qu'à un moment donné nous tentions de dépiler une pile vide ? La réponse est non.

Propriété : Les points p_0 et p_1 ne sont jamais dépilés.

Démonstration : Rappelons-nous que la liste $[p_0, p_1, \dots, p_{n-1}]$ est triée selon l'ordre $\preceq$, et que $p_0 = \Omega$, le plus petit point de l'ensemble A pour l'ordre lexicographique. On a donc pour tout $i \geq 2$, $p_1 \prec p_i$. Dit autrement,

- Les points p_0, p_1 et p_i sont alignés et $d(p_0, p_1) < d(p_0, p_i)$

ou bien

- La famille $(\overrightarrow{p_0p_1}, \overrightarrow{p_0p_i})$ est une base directe.

Soit $i \geq 3$. Supposons qu'à la i-ème itération de la boucle `for`, la pile S ne contient plus à un moment que p_0 et p_1. Le sommet de S est donc p_1 et son sous-sommet est p_0.

- Si les points p_0, p_1 et p_i sont alignés, `pseudo_angle` prend la valeur 0 et on sort donc de la boucle `while`.
- Si la famille $(\overrightarrow{p_0p_1}, \overrightarrow{p_0p_i})$ est une base directe, alors `pseudo_angle(p_0, p_1, p_i)` prend une valeur strictement positive. On sort donc également de la boucle `while`.

4.2 Terminaison et complexité de l'algorithme de Graham

Qu'allons nous compter pour mesurer la complexité de l'algorithme de Graham ? Un peu tout et n'importe quoi : comparaisons, opérations de piles. Nous allons évaluer le nombre d'opérations effectuées par l'algorithme lorsqu'il est exécuté sur un ensemble de n points.

- L'étape 1 (obtention du point Ω) s'effectue en complexité $O(n)$.
- L'étape 2 (tri) est effectuée (sauf accident) en complexité $O(n \log n)$ (voir le notebook sur le tri rapide).
- Les étapes 3 et 5 s'effectuent en temps constant.
- Reste à évaluer l'étape 4, la complexité de la boucle `for`. Le point inquiétant est la boucle `while` à l'intérieur de la boucle `for`, mais restons zen. En effet, dans une pile on ne peut pas dépiler plus d'objets qu'on n'en a empilé. Or on sait exactement quel est le nombre d'empilements : c'est n. Regardez le code ! Il y a donc au plus n ($n-2$ en fait car p_0 et p_1 ne sont jamais dépilés) dépilements et la boucle `for` a donc une complexité en $O(n)$.

La conclusion surprenante est que la partie la plus coûteuse de l'algorithme se situe à l'extérieur des boucles, lors du tri selon les angles-distances croissants. Tout le reste est négligeable devant ce tri. Bilan : l'algorithme de Graham s'exécute en complexité $O(n \log n)$ où n est le nombre de points de l'ensemble A.

4.3 Correction de l'algorithme

NB : En réalité je vous mens depuis un moment. Tel quel, notre algorithme est incorrect. Certaines conditions d'alignement peuvent ajouter à la pile S des points qui ne sont pas des sommets de $\mathcal{C}(A)$. Nous supposerons dans la preuve ci-dessous que de tels alignements ne se produisent pas. Nous reviendrons sur ces problèmes d'alignement au paragraphe suivant.

Proposition : Soit $2 \leq i < n$. À l'issue de la i-ème itération de la boucle `for` , la pile S contient les sommets de l'enveloppe convexe de l'ensemble $\{p_0, \ldots, p_i\}$ triés dans l'ordre croissant pour la relation $\preceq$.

Démonstration : Euh, les itérations commencent à $i = 3$, non ? Pour $i = 2$, reformulons : que contient s juste avant la boucle `for` ? Les points p_0, p_1 et p_2 , qui sont exactement les sommets de l'enveloppe convexe de l'ensemble $\{p_0, p_1, p_2\}$ triés dans l'ordre croissant pour la relation $\preceq$. Soit maintenant $2 < i < n$. Supposons la propriété vérifiée pour $i - 1$. Montrons qu'elle l'est encore pour i. Je ne détaillerai pas complètement la preuve parce qu'il y a beaucoup de choses sur les ensembles convexes que nous n'avons pas montrées. Vous êtes invités à faire des petits dessins pour comprendre ce que je raconte.

Soient p le sommet de S et q le sous-sommet de S.

- Si la famille $(\overrightarrow{qp}, \overrightarrow{qp_i})$ est une base indirecte, alors p est à l'intérieur du triangle $(p_0 q p_i)$. Rappelez-vous lorsque vous faites le dessin (vous le faites, n'est-ce pas ?) qu les points p_i sont triés pour la relation $\preceq$! Le point p n'est donc pas un sommet de $C(\{p_0, \ldots, p_i\})$.
- Si, au contraire, la famille $(\overrightarrow{qp}, \overrightarrow{qp_i})$ est une base directe, alors le point p n'est à l'intérieur d'aucun triangle dont les sommets sont parmi les points de $S \cup \{p_i\}$, hormis p lui même : c'est un un sommet de $C(\{p_0, \ldots, p_i\})$ (on utilise ici l'hypothèse de récurrence).

Les points éliminés par dépilement dans la boucle `while` ne sont pas des sommets de $C(\{p_0, \ldots, p_i\})$. Il resterait à prouver que les éléments de S à l'issue de la $i$ème itération de la boucle `for` sont **exactement** les sommets de $C(\{p_0, \ldots, p_i\})$. Je l'admettrai.

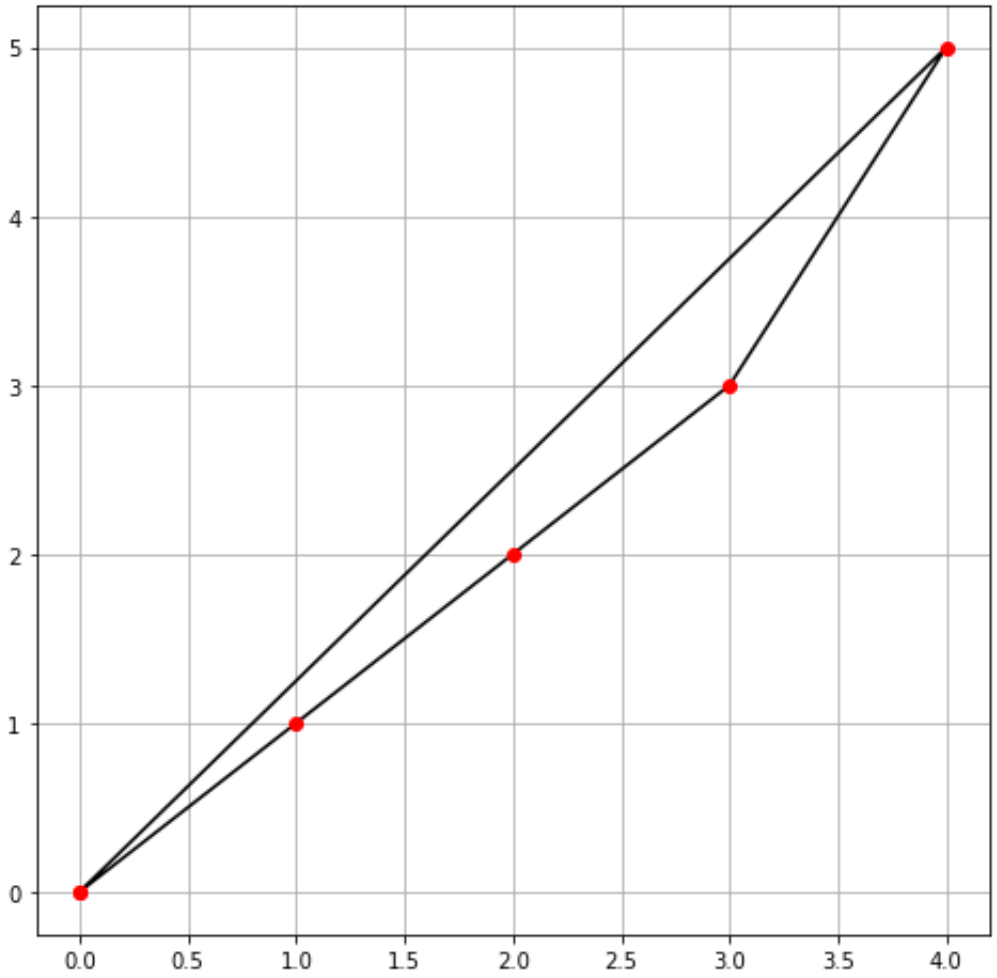
4.4 Les problèmes d'alignement

Je me contente ici de soulever les problèmes et de voir comment on peut les résoudre.

Voici un ensemble de points un peu particulier. Son enveloppe convexe est un triangle, mais certains points de A sont sur le bord du triangle.

In [35]:
```
A = [(0, 0), (2, 2), (1, 1), (3, 3),(4, 5)]
S = graham(A)
print('nombre de sommets :', len(S))
tracer_graham(A, S)
```

```
nombre de sommets : 5
```



Nous avons 4 points alignés. "Notre" algorithme de Graham considère que ces 4 points sont des sommets de $\mathcal{C}(A)$ alors que, clairement, les points intermédiaires n'en sont pas. Comment remédier à cela ? Eh bien en supprimant les points alignés !

La fonction `supprimer_alignes` prend en paramètres un ensemble A supposé trié pour l'ordre $\preceq$ et un point Ω, minimal pour cet ordre. Elle renvoie une liste de points dans laquelle les points alignés "intermédiaires" ont été supprimés.

In [36]:
```
def supprimer_alignes(s, Omega):
    s1 = [s[0]]
    n = len(s)
    k = 1
    while k < n:
        k = suppr_aux(s, k, Omega)
        s1.append(s[k - 1])
    return s1
```

In [37]:
```
def suppr_aux(s, k, Omega):
    p = s[k]
    j = k + 1
    while j < len(s) and pseudo_angle(Omega, p, s[j]) == 0: j = j + 1
    return j
```

Si nous reprenons l'exemple ci-dessus, les points $(1, 1)$ et $(2, 2)$ sont éliminés.

In [38]:
```
A = [(0, 0), (1, 1), (2, 2), (3, 3), (4, 5)]
supprimer_alignes(A, (0, 0))
```

Out[38]:
```
[(0, 0), (3, 3), (4, 5)]
```

L'algorithme de Graham devient :

In [39]:
```
def graham2(A):
    S = pile_vide()
    Omega = min_lexico(A)
    trier_angles_distances(Omega, A)
    A1 = supprimer_alignes(A, Omega)
    empiler(S, A1[0])
    empiler(S, A1[1])
    empiler(S, A1[2])
    for i in range(3, len(A1)):
        while pseudo_angle(sous_sommet(S), sommet(S), A1[i]) <= 0:
            depiler(S)
            #print(S)
        empiler(S, A1[i])
    return S
```

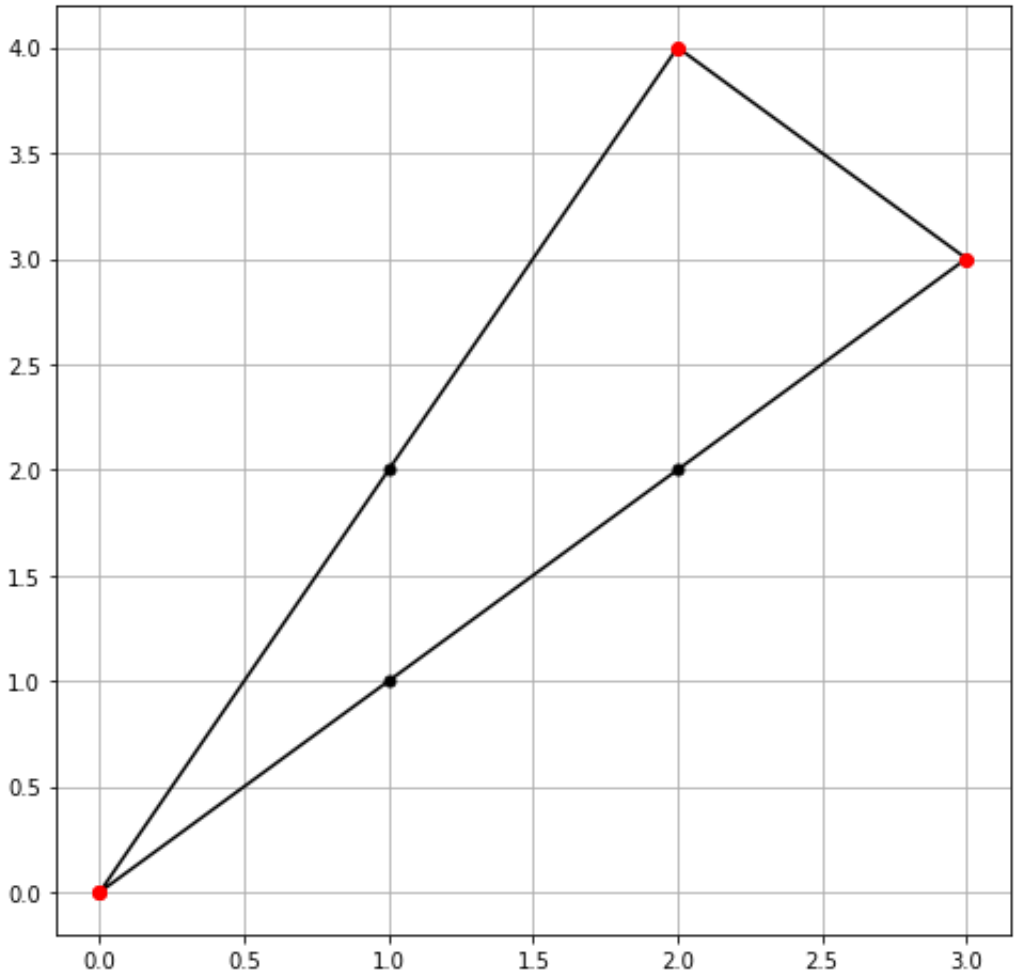
Remarquez qu'il y a **deux** différences entre les fonctions `graham` et `graham2` :

- L'appel à `supprimer_alignes`
- Un signe $\leq$ à la place de $<$ dans le test de la boucle `while`

Je ne prouverai pas que `graham2` est correct à condition que A possède 3 points non alignés. Contentons-nous de le tester sur un petit exemple.

In [40]:
```
A = [(0, 0), (1, 1), (2, 2), (3, 3), (1, 2), (2, 4)]
S = graham2(A)
print('nombre de sommets :', len(S))
tracer_graham(A, S, ms=5)
```

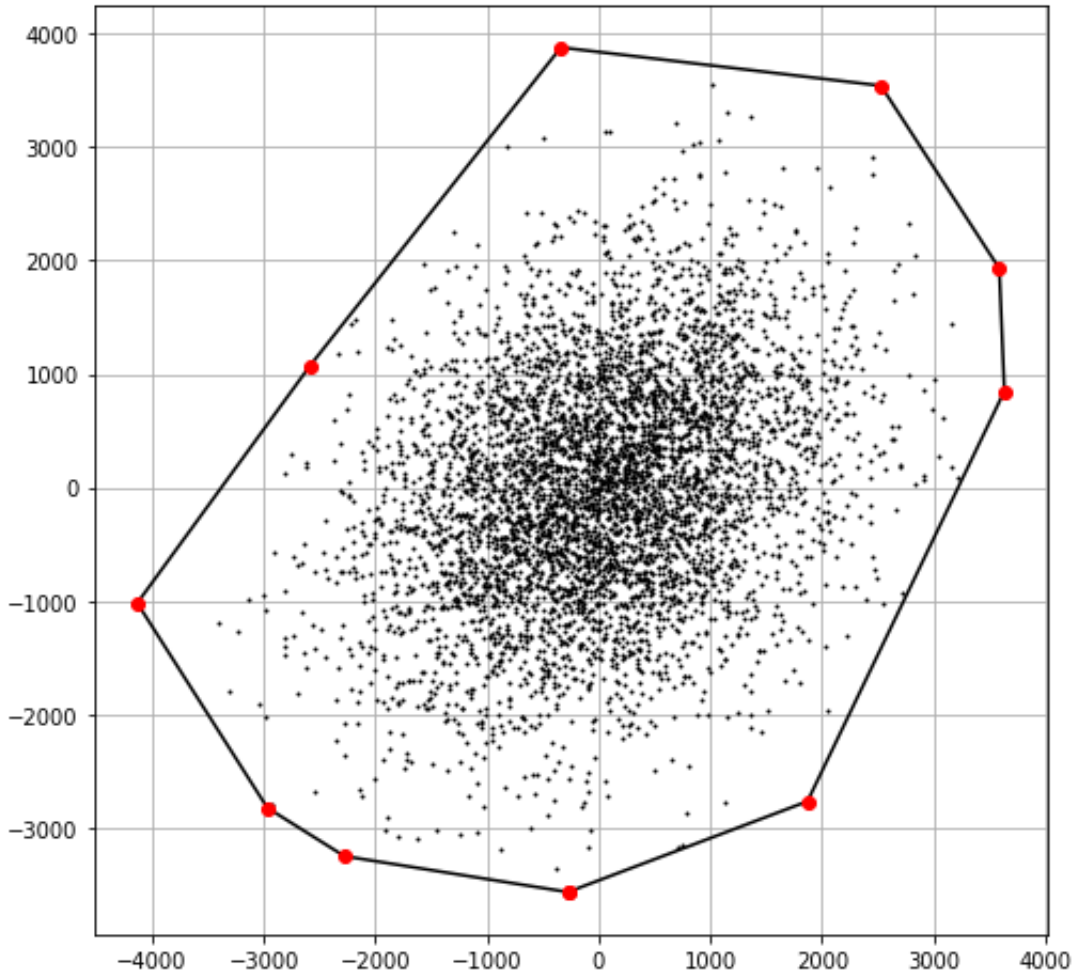
```
nombre de sommets : 3
```



Et sur un gros exemple.

In [41]:
```
A = random_list(5000)
S = graham2(A)
print('nombre de sommets :', len(S))
tracer_graham(A, S)
```

```
nombre de sommets : 10
```



Nous voilà à peu près satisfaits :-).