Enveloppe convexe 2D, emballage cadeau (« gift wrapping »)

Question.

Le principe de l'algorithme a été donné en cours. Tentez de préciser l'algorithme pour pouvoir le programmer. Ci dessous, vous trouverez une formalisation possible, et un programme en ocaml. Cet exercice ne vous sera profitable que si vous réfléchissez d'abord, avant de consulter la solution ci-dessous. Il y a bien sûr de nombreuses solutions possibles.

Solution .

Pour calculer l'enveloppe convexe d'un ensemble E de n points 2D, cette méthode procède ainsi.

Calculer le point p0, de x minimal (et d'ordonnée minimale s'il y en a plusieurs). C'est le premier point de l'enveloppe convexe.

Le point q, suivant un point p donné de l'enveloppe convexe, est calculé ainsi :

suivant(p, E) :
soit E' l'ensemble E auquel les points égaux à p sont enlevés. Le point q est l'élément minimum de E', avec cette comparaison entre deux points a et b :
si p a b tourne à gauche (le déterminant |p a b| est positif) alors a est meilleur que b.
si p a b tourne à droite (le déterminant |p a b| est négatif) alors b est meilleur que a.
si p a b sont alignés (le déterminant |p a b| est nul), alors
    si les points p a b sont alignés dans cet ordre p a b (le produit scalaire pa . ab est positif) ,
    alors b est meilleur
    sinon a est meilleur

La méthode calcule le premier point p0, le suivant de p0, le suivant du suivant de p0,, etc jusqu'à retomber sur p0.

Cette méthode est en O(nk) où k est le nombre de sommets de l'enveloppe convexe. En effet le calcul du suivant est en O(n). Elle est donc en $O(n^2)$ dans le pire des cas.

Si les calculs sont exacts, la méthode fonctionne, même en présence de cas particuliers (points alignés, points confondus). C'est le cas ici, car les points ont de petites coordonnées entières.

Voici le source en ocaml du programme.

```
(* programme Gift Wrapping (calcul de l'enveloppe convexe) en ocaml *)
module L=List;;
open Graphics;;
open_graph " 512x512";;
let square x = x * x ;;
let rec genere n l =   (* generer une liste de points dans un cercle *)
        if n=0 then l
        else let x,y=Random.int 512, Random.int 512 in
        if square (x-256) + square (y-256) <= square 220
        then genere (n-1) ((x,y)::l)
        else genere n l;;
let determinant (x1, y1) ( x2, y2) ( x3, y3) =   x1 * (y2 - y3) + x2 * (y3 – y1) + x3 * (y1 - y2) ;;
(* comparaison de 2 points; le + petit est le plus à gauche (le plus bas si égalité des x ) :  *)
let cmp (x1, y1) (x2, y2) = let dx=x1-x2 in if dx <> 0 then dx else y1-y2 ;;
let adroite p q r = determinant p q r < 0;;
let agauche p q r = determinant p q r > 0;;
let pscal (u1,u2) (v1,v2) = u1*v1 + v1*v2;;
let vecteur (x0, y0) (x1, y1) = x1-x0, y1-y0 ;;
(* quel est le point suivant p dans la liste l des points ?
   eliminer les p de l  ;
   a est meilleur que b ssi p a b tourne à gauche (= determinant p a b > 0 )
   b est meilleur que a ssi p a b tourne à droite (= determinant p a b < 0 )
   si a b c alignés (= determinant p a b = 0 )
   alors  si pa . ab > 0 alors b est meilleur que a (les points sont alignés dans l'ordre : p, a, b)
        sinon a est meilleur que b
        remarque : si a=b, pas d'importance qu'on prenne a ou b *)
let suivant p l =
        let l' = L.filter (function pt-> p <> pt) l in
        let q= L.fold_left (fun a b-> let det=determinant p a b in
                if det > 0 then a else if det < 0 then b
                else let pa= vecteur p a in let ab= vecteur a b in
                        if pscal pa ab > 0 then b else a) (L.hd l') (L.tl l') in q;;
let premier l = L.fold_left (fun a b-> if cmp a b <= 0 then a else b) (L.hd l) (L.tl l);;
let gift l = let p0= premier l in
        let rec marche p hull = let q= suivant p l in if q=p0 then q::hull else marche q (q::hull) in
        marche p0 [p0];;
let affiche l=   clear_graph();
        L.iter (function x,y -> fill_circle x y 2) l;
        let e=gift l in
        match e with   |   [] -> ()
        |   (x1,y1)::q -> ( moveto x1 y1;  L.iter (function x,y -> lineto x y) q );;
let rec  genere_alignes n l = if n=0 then l else genere_alignes (n-1) ((Random.int 500, 256)::l) ;;
let demo n = affiche (genere n [] );;
let demop n = affiche (genere_alignes n  [] );;
```