Université Bordeaux 1 — Année 2012-2013
Cycle Préparatoire — UE J1CP402F (Info 3 MPC)

Projet: calculs d'enveloppes convexes

L'objectif du projet est de répondre à un certain nombre de questions *mathématiques*, *algorithmiques* et *pratiques* sur un problème classique de géométrie algorithmique: le calcul d'une enveloppe convexe en deux dimensions.

1 Problème posé et objectifs

Étant donné un ensemble E de n points dans le plan, on souhaite "calculer" leur *enveloppe convexe*, c'est-à-dire le plus petit polygone convexe du plan contenant (sur son bord, ou dans son intérieur) tous les points de E. Une image intuitive est celle de la ficelle: si l'on plante des clous dans une planche, et que l'on cherche à "entourer" d'une ficelle l'ensemble des clous de la manière la plus "serrée" possible, celle-ci décrira exactement l'enveloppe convexe.

Par *calculer l'enveloppe convexe*, on entend plus précisément que, supposant que les n points sont donnés par leurs n paires de coordonnées, on souhaite obtenir la liste des points situés sur le polygône enveloppe convexe. Pour que la question soit bien définie, et la réponse correcte unique, on précise les choses:

- les points doivent être donnés dans un ordre correspondant à un parcours dans le sens trigonométrique (antihoraire) du polygône;
- le premier point sera le point "le plus à gauche" (d'abscisse minimale); en cas d'égalité, "le plus en bas parmi les plus à gauche" (s'il existe plusieurs points d'abscisse minimale, celui qui a la plus petite ordonnée);
- s'il y a plusieurs points alignés sur le polygône, on ne retiendra que les deux points extrêmaux sur le segment qu'ils forment;
- au cas très particulier où tous les points de E seraient alignés, l'enveloppe convexe est un segment; on retiendra alors seulement les deux points extrêmaux du segment; si l'ensemble E ne comporte qu'un unique point, on ne retiendra que cet unique point.

Les ensembles de points E et les enveloppes convexes seront représentées dans les programmes de la même manière: sous la forme d'une liste dont chaque élément sera une paire de coordonnées. Pour les enveloppes convexes, on ne répètera pas le premier point en fin de liste (de sorte que, pour tracer le polygône complet, il faudra penser à relier également le dernier point au premier).

On peut dans un premier temps simplifier le problème en supposant que les points sont en *position générique*, c'est-à-dire que E ne contient pas 3 points situés sur une même droite. Il est ensuite bon de s'assurer que l'algorithme fonctionne correctement si la position n'est pas générique (en particulier, choisir un

grand nombre de points à coordonnées entières, aléatoirement dans un rectangle même assez grand, donne facilement des points en position non générique - donc ce type de méthode ne doit être utilisé qu'avec parcimonie quand il s'agit de construire des jeux de tests des algorithmes, à moins que les algorithmes ne soient effectivement conçus pour traiter les cas de points alignés).

1.1 Présentation des algorithmes et des programmes

Dans le cours du sujet, on demande un certain nombre d'algorithmes (et les programmes correspondants). Les fonctions `Python` seront systématiquement écrites en prenant, parmi leurs entrées, une *liste* L qui décrira l'ensemble de points; chaque élément de la liste sera lui-même une liste (ou un tuple) de deux nombres, respectivement l'abscisse et l'ordonnée d'un des points de la liste. Ainsi le nombre de points à traiter est simplement la longueur de la liste L (`len(L)`), le point d'indice j est accessible par `L[j]`, et `L[j][0]` désigne l'abscisse du j-ème point.

Les enveloppes convexes calculées seront renvoyées sous un format similaire: une liste de paires de coordonnées, avec la double contrainte que les points seront présentés dans l'ordre convenable et que seuls les points de l'enveloppe convexe apparaîtront dans cette liste.

2 Préliminaires mathématiques

Cette section présente quelques notions qui devraient s'avérer utiles pour la production d'algorithmes de calcul d'enveloppes convexes.

Dans la suite, pour un ensemble E de points, on note $C(E)$ l'enveloppe convexe. Notre objectif est de "calculer" $C(E)$ sous la forme de la liste de ses *sommets*. Un point $P \in E$ est un sommet de $C(E)$, ou un *point extrêmal* de E, si le polygône $C(E)$ a un angle (non plat) au point P.

2.1 Sens direct, sens indirect, croisements

Trois points P, Q, R du plan, non alignés, sont dits *en sens direct* si, lorsque l'on oriente la droite (PQ) de P vers Q, le point R se trouve dans le demi-plan situé à gauche de la droite. Dans le cas contraire, ils sont dits *en sens indirect*.

Une autre façon de formuler cette définition est la suivante: si, vu "depuis P", parcourir le segment $[QR]$ de Q vers R revient à aller de la droite vers la gauche, alors (P, Q, R) sont en sens direct; sinon, en sens indirect.

Il s'agit d'une propriété cyclique: si (P, Q, R) sont en sens direct, il en est de même de (Q, R, P) et de (R, P, Q), et inversement, (P, R, Q), (R, Q, P) et (Q, P, R) sont en sens indirect.

Trouver une caractérisation analytique, sur les coordonnées des trois points, du fait que (P, Q, R) sont en sens direct. ***Indication:*** *Chercher une caractérisation sous forme du signe d'une expression formée à partir des coordonnées.*

Écrire une fonction `EstSensDirect` *qui prend en entrée trois points, et retourne* 1 *si les trois points sont en sens direct,* -1 *s'ils sont en sens indirect, et* 0 *s'ils sont alignés.*

Le fait que deux segments $[AB]$ et $[CD]$ se croisent peut également (au moins dans le cas où les points sont en position générique) s'exprimer assez simplement en fonction de cette notion de sens direct ou indirect.

Écrire une fonction `Croisement` *qui prend en entrée quatre points* A*,* B*,* C*,* D*, et qui retourne* `True` *si les segments géométriques* $[AB]$ *et* $[CD]$ *ont un point commun, et* `False` *sinon.*

2.2 Trouver un point et un segment de l'enveloppe convexe

Décrire un algorithme, de complexité au plus linéaire, qui trouve un point dont on peut garantir qu'il fait partie de l'enveloppe convexe. Écrire une fonction `TrouvePointEnveloppe` *qui prend en entrée une liste de points, et retourne l'un des points faisant partie de l'enveloppe convexe.*

Considérons deux points P et Q de E. On se demande si le segment $[PQ]$ fait ou non partie du bord de l'enveloppe convexe de E. La règle est relativement simple: $\overline{PQ}$ fait partie du bord de $C(E)$, si et seulement si tous les points de E sont "du même côté" de la droite (PQ). À quelle condition le segment $[PQ]$, *dans ce sens de* P *vers* Q, fait-il partie du "tour en sens direct" de l'enveloppe convexe?

Écrire un algorithme qui décide si oui ou non un segment orienté fait partie de l'enveloppe convexe. Quelle est sa complexité? Écrire une fonction `EstSegmentEnveloppe` *qui prend en entrée une liste de points et deux points de la liste, et retourne* `True` *ou* `False` *selon que le segment fait ou non partie du bord de l'enveloppe convexe.*

2.3 Ordre autour d'un point extrêmal

La notion d'"ordre autour d'un point" correspond à l'image d'un rayon issu d'un point fixe, qui "balaie" le plan en tournant autour - dans notre cas, la convention est de tourner dans le sens trigonométrique. Ranger des points de E dans un ordre autour d'un point A, c'est les mettre dans l'ordre où le rayon tournant les rencontrera pour la première fois.

Si l'on cherche à définir cet ordre autour d'un point A, pour un point qui n'est pas un point extrêmal (un point qui forme un angle non plat de l'enveloppe convexe), on se heurte à une difficulté: il n'y a pas de "premier point", celui-ci dépend de la position initiale du "rayon". Fort heureusement, nous allons principalement nous intéresser au cas où P est un point extrêmal.

Dire que A est un point extrêmal de E, revient à dire qu'il existe, parmi les droites du plan passant par A, au moins une droite telle que *tous les points de* E *autres que* A *sont (strictement) du même côté de la droite.* En prenant la direction d'une telle droite comme direction de référence, on peut alors définir de manière unique l'ordre des autres points de E autour de A, et le caractériser par

la propriété suivante: *dans cet ordre, un point B est "avant" C si et seulement si le triplet (A, B, C) est dans le sens direct.*

*Montrer que cette relation est **transitive**: si B est avant C autour de A, et que C est avant D autour de A, alors B est avant D autour de A. Cette propriété n'est **pas vraie** si A n'est pas extrêmal (le vérifier sur un exemple au moins).*

En déduire un algorithme, et programmer la fonction correspondante, qui prend en entrée un ensemble E de points, et un point extrêmal A de E, et qui trouve le premier point de E dans l'ordre autour de A. Votre algorithme devrait avoir une complexité linéaire. Expliquer pourquoi ce point est lui aussi un point extrêmal.

La remarque importante, pour le calcul d'enveloppes convexes, est la suivante: si P est un point extrêmal de E, alors le point qui suit P dans le "tour" de l'enveloppe convexe, est le *premier* point de l'ordre des points de $E - \{P\}$ autour de P. C'est cette propriété qui sert de base à l'algorithme de la "marche de Jarvis".

2.4 Les cas simples d'enveloppes convexes

L'enveloppe convexe d'un point unique, ou de deux points, est triviale: il s'agit des points eux-mêmes, que l'ont peut laisser dans n'importe quel ordre.

L'enveloppe convexe de trois points est presque aussi simple: *a priori* les trois points font partie du polygone, il s'agit seulement de les mettre dans le bon ordre.

2.5 Recherche d'algorithmes de calcul d'enveloppe convexe

À partir des différentes propriétés précédentes, vous pouvez d'ores et déjà chercher à proposer et à programmer des algorithmes de calcul d'enveloppes convexes. Il est important de se soucier de la complexité des algorithmes obtenus!

Une solution possible est de commencer par chercher à identifier, segment par segment, ceux qui forment le polygone recherché, puis à les mettre dans le bon ordre.

Une seconde solution possible est de calculer successivement les enveloppes convexes des 3 premiers points, puis des 4 premiers, etc. À chaque étape, on a donc besoin, disposant d'une précédente enveloppe convexe et d'un nouveau point, de déterminer si le point se situe ou non à l'intérieur du polygone; et, si ce n'est pas le cas, de déterminer comment doit évoluer ce polygone. Pour ce dernier cas, on remarquera que les segments qui doivent "disparaître" de l'enveloppe sont exactement ceux qui forment un triangle indirect avec le nouveau point.

3 Quelques algorithmes de calcul d'enveloppe convexe

Dans cette section, on présente quelques pistes classiques pour élaborer des algorithmes de calcul d'enveloppes convexes. Il est conseillé de s'être d'abord attaqué au problème par ses propres moyens avant de les aborder!

3.1 La "marche" de Jarvis

L'algorithme de Jarvis correspond à l'image de l'emballage. Partant d'un point dont on sait qu'il appartient au polygone recherché, on cherche successivement à calculer, dans l'ordre, les autres points de l'enveloppe convexe. Le principe est le suivant: à tout moment, on maintient un point "courant", dont on sait qu'il s'agit d'un point extrêmal. Le point suivant est le premier dans l'ordre cyclique autour du point courant, et devient le nouveau point courant. L'algorithme se termine quand le nouveau point courant devient égal à celui dont on est parti. Cela correspond à l'image de la ficelle qui, attachée au premier clou, tourne autour des autres.

Quelle est la complexité de la marche de Jarvis? Elle peut s'exprimer de façon plus favorable (toujours dans le cas le pire) en fonction de deux paramètres: le nombre n de points, et le nombre h de points qui composent le polygone (h peut valoir entre 3 et n).

3.2 Convexification d'un polygône non croisant

Un autre algorithme demande de partir d'un polygône dont les sommets sont tous les points de E, et qui doit être *non croisant* - les segments qui composent ce polygône ne doivent pas se croiser. La façon d'obtenir (efficacement!) un tel polygône est à inventer. On suppose également que l'ordre dans lequel sont donnés les sommets de ce polygône est l'ordre trigonométrique (antihoraire).

L'algorithme procède alors comme suit: on examine successivement les triplets de trois sommets consécutifs du polygône, P_{i-1}, P_i, P_{i+1}. Si le triplet est dans le sens direct, rien ne se passe et on passe au triplet suivant; si le triplet est dans le sens indirect, alors le point P_i doit être supprimé, et le prochain triplet à examiner doit être choisi soigneusement. Si l'on s'y prend bien, on peut arriver à passer, en temps *linéaire*, du polygône non croisant à l'enveloppe convexe - et, ce qui n'est pas forcément évident, à prouver cette complexité.

(Dans l'analyse de la complexité de l'algorithme, ne pas oublier le calcul du polygône non croisant! Si la complexité finale est linéaire, il y a forcément une erreur.)

3.3 Diviser pour régner

On peut parfaitement appliquer le paradigme *diviser pour régner* au problème du calcul de l'enveloppe convexe. On cherchera à séparer l'ensemble de points en deux parties, de tailles équivalentes, en assurant que *les enveloppes convexes des*

deux parties ne s'intersectent pas (comment? on peut, par exemple, commencer par trier l'ensemble des points par abscisses croissantes, avant même de lancer l'algorithme diviser pour régner). Ayant calculé les enveloppes convexes des deux parties, il faudra ensuite en déduire l'enveloppe convexe de l'ensemble; si cette opération peut être accomplie avec une complexité $O(n)$, on sait que l'algorithme complet aura une complexité $O(n \log n)$.

4 Travail demandé

4.1 Travail avec `Python`

Vous produirez deux fichiers écrits en `Python`. Le premier contiendra un ensemble de fonctions (autant que nécessaire) permettant de résoudre le problème du calcul d'une enveloppe convexe, si possible avec tous les algorithmes évoqués ci-dessus. Le deuxième fichier contiendra des tests de toutes les fonctions du premier fichier. Chaque fonction sera testée en la faisant s'exécuter sur différents jeux de données, choisis autant que possible pour vérifier que le maximum de cas particuliers sont correctement traités.

Autant que possible, ces tests seront indépendants les uns des autres (si le test d'une fonction utilise, par exemple, un ensemble de points, la liste correspondante sera créée et initialisée spécialement pour ce test, sans supposer que les tableaux déjà créés pour des tests antérieurs sont encore disponibles).

4.2 Tracés graphiques

Vous utiliserez les fonctions de la bibliothèque `pyplot` pour tracer, sur le même graphique, un ensemble de points et l'enveloppe convexe que vous aurez calculée.

4.3 Expérimentation

Certains théorèmes (qu'il n'est pas question de redémontrer!) prédisent des comportements *a priori* surprenants pour l'enveloppe convexe d'un (grand) ensemble de points pris aléatoirement dans une zone donnée du plan.

- Si l'on prend n points aléatoires à l'intérieur d'un *disque* (par exemple: le disque D centré en l'origine, de rayon 1), le nombre moyen de points situés sur l'enveloppe convexe est $\Theta(n^{1/3})$.
- Si l'on prend n points aléatoires à l'intérieur d'un *carré* ou d'un rectangle (par exemple: le carré $C = [-1, 1] \times [-1, 1]$ des points dont abscisse et ordonnée sont comprises entre -1 et 1; noter que D est un disque inscrit dans C), le nombre moyen de points situés sur l'enveloppe convexe est $\Theta(\log(n))$.

Vous exploiterez vos algorithmes de calcul d'enveloppe convexe en cherchant à mettre en évidence ce phénomène. En particulier, vous chercherez des valeurs probables pour les constantes cachées dans la notation Θ.

4.4 Rapport écrit

En plus de votre travail sous `Python`, vous produirez un rapport écrit (qu'il soit manuscrit ou rédigé au moyen d'un logiciel de traitement de texte est peu important; une dizaine de pages maximum, hors d'éventuelles figures) qui détaillera vos réponses aux questions posées dans le sujet, ainsi que celles résultant de votre réflexion personnelle. Les algorithmes y seront détaillés, et leur complexité dans le cas le pire sera étudiée. Dans la mesure du possible, vous chercherez à produire des exemples qui correspondent aux cas les pires.