TP: Calcul d'enveloppes convexes

M1 MEEF maths

4 décembre 2023

Le but de ce TP est d'étudier un algorithme pour calculer l'enveloppe convexe d'un nuage de points. Fixons le vocabulaire:

- Un *nuage de points* est juste un ensemble fini de points dans le plan.
- On appelle *enveloppe convexe* d'un ensemble de points le plus petit polygone convexe qui contienne tous ces points, sur le bord ou à l'intérieur. On admettra que ce polygone est toujours bien défini et qu'il est unique.

On fixe aussi la représentation de ces objets en Python, pour les besoins de ce TP:

- On représente un nuage de n points par deux listes `Xp` et `Yp` de longueur n contenant les coordonnées des points.
- On représente un polygone à n sommets par deux listes `Xe` et `Ye` de longueur n contenant les coordonnées des sommets, ordonnés dans le sens trigonométrique (peu importe le point de départ).

Cette représentation permet de faire facilement des affichages avec `matplotlib`. Pour pouvoir faire des tests, on fournit les fonctions suivantes:

```
import random
def nuage_aleatoire(n):
    """Produit un nuage de n points aléatoires à coordonnées entre -1 et 1."""
    X = [random.triangular(-1, 1) for _ in range(n)]
    Y = [random.triangular(-1, 1) for _ in range(n)]
    return (X, Y)

from matplotlib import pyplot
def trace_nuage_et_polygone(Xp, Yp, Xe, Ye):
    """Trace le nuage de points (Xp,Yp) et le polygone (Xe, Ye)."""
    pyplot.figure()
    pyplot.fill(Xe, Ye, color=(.8, .8, .8))
    pyplot.plot(Xp, Yp, 'bo')
    pyplot.show()
```

Voici un exemple de tracé d'un nuage de points aléatoire et d'un polygone qui n'a rien avoir avec, pour illustrer l'emploi de ces fonctions:

```
Xp, Yp = nuage_aleatoire(100)
Xe = [-0.5, 0.5, 0.5, 0]
Ye = [0, -0.5, 0, 1]
trace_nuage_et_polygone(Xp, Yp, Xe, Ye)
```

Enveloppe rectangulaire

On commence par calculer une enveloppe rectangulaire. Ce n'est pas l'enveloppe convexe mais c'est le plus petit rectangle à bords parallèles aux axes qui contient tous les points.

Définir une fonction `min_max` qui prend en entrée une liste de nombres non vide et renvoie un couple `(m,M)` composé du minimum et du maximum des valeurs de la liste.

*Note: Python fournit des fonctions `min` et `max` qui calculent le minimum et le maximum d'une liste, on demande de **ne pas** s'en servir dans la fonction `min_max`. On peut par contre les utiliser pour tester.*

```
def min_max(L):
    ...
```

Le test suivant doit renvoyer `True`, sinon il y a un problème:

```
L = [random.random() for _ in range(50)]
min_max(L) == (min(L), max(L))
```

Définir une fonction `enveloppe_rectangulaire` qui prend en entrée un nuage de points et renvoie son enveloppe rectangulaire, sous forme d'un polygone selon la représentation donnée au début.

```
def enveloppe_rectangulaire(Xp, Yp):
    ...
```

Le code suivant permet de tester le fonction. Tous les points doivent se trouver dans le rectangle gris ou sur le bord.

```
Xp, Yp = nuage_aleatoire(100)
Xe, Ye = enveloppe_rectangulaire(Xp, Yp)
trace_nuage_et_polygone(Xp, Yp, Xe, Ye)
```

Vérification d'enveloppe

On va maintenant écrire des fonction pour vérifier qu'une liste de points est bien un polygone convexe qui contient tous les éléments du nuage de points.

Pour deux vecteurs non nuls $\vec{u}$ et $\vec{v}$ de coordonnées (x_u, y_u) et (x_v, y_v), on admet que $\vec{v}$ pointe *à gauche* de $\vec{u}$ si le produit en croix $x_u y_v - x_v y_u$ est positif.

Définir une fonction `a_gauche` qui prend en entrée les coordonnées de trois points A, B, C et renvoie un booléen qui indique si le point C se situe à gauche de la droite (AB) si on l'oriente de A vers B. La fonction doit renvoyer `True` aussi si le point C est sur la droite (AB).

```
def a_gauche(xA, yA, xB, yB, xC, yC):
    ...
```

Le test suivant permet d'afficher quels points C sont considérés *à gauche* par la fonction précédente pour deux points A et B donnés, parmi un nuage de points aléatoire. Les points à gauche sont en rouge, ceux à droite sont en vert.

```
xA, yA = (-0.5, -0.5)
xB, yB = (1, 0.5)
X, Y = nuage_aleatoire(100)
Xg, Yg = [], []
Xd, Yd = [], []
for i in range(100):
    if a_gauche(xA, yA, xB, yB, X[i], Y[i]):
        Xg.append(X[i])
        Yg.append(Y[i])
    else:
        Xd.append(X[i])
        Yd.append(Y[i])
pyplot.figure()
pyplot.plot([xA, xB], [yA, yB], 'bo')
pyplot.plot(Xg, Yg, 'ro')
pyplot.plot(Xd, Yd, 'go')
pyplot.show()
```

Un polygone convexe doit être représenté par une liste de points qui fait un parcours en sens trigonométrique, c'est-à-dire qu'en parcourant les points dans l'ordre donné on doit toujours tourner à gauche. Écrire une fonction `polygone_convexe` qui vérifie cette propriété.

```
def polygone_convexe(X, Y):
    ...
```

Écrire trois tests pour vérifier cette fonction, avec au moins un cas où elle doit renvoyer `True` et un cas où elle doit renvoyer `False`.

```
...
```

Si un polygone est bien convexe et parcouru dans le sens trigonométrique (ce que la fonction `polygone_convexe` est censée vérifier), les points à l'intérieur sont ceux qui sont à gauche de chaque côté. Écrire une fonciton `interieur` qui prend en entrée un polygone supposé convexe et un point et qui détermine si ce point est à l'intérieur du polygone.

```
def interieur(Xe, Ye, x, y):
    ...
```

Le test suivant doit marquer en vert les points qui sont dans le polygone et en rouge ceux qui n'y sont pas.

```
X, Y = nuage_aleatoire(100)
Xe = [-0.5, 0.5, 0.5, 0]
Ye = [0, -0.5, 0, 1]
Xi, Yi = [], []
Xo, Yo = [], []
for i in range(len(X)):
    if interieur(Xe, Ye, X[i], Y[i]):
        Xi.append(X[i])
        Yi.append(Y[i])
    else:
        Xo.append(X[i])
        Yo.append(Y[i])
pyplot.figure()
pyplot.fill(Xe, Ye, color=(0.8, 0.8, 0.8))
pyplot.plot(Xo, Yo, 'ro')
pyplot.plot(Xi, Yi, 'go')
pyplot.show()
```

Écrire une fonction `est_enveloppe` qui prend en entrée un polygone et un nuage de points et qui vérifie que le polygone est convexe et que tous les points du nuage sont à l'intérieur.

```
def est_enveloppe(Xe, Ye, Xp, Yp):
    ...
```

L'enveloppe rectangulaire calculée au début doit être une enveloppe, donc le test suivant doit renvoyer `True`.

```
Xp, Yp = nuage_aleatoire(100)
Xe, Ye = enveloppe_rectangulaire(Xp, Yp)
est_enveloppe(Xe, Ye, Xp, Yp)
```

Calcul de l'enveloppe convexe

On va maintenant calculer précisément l'enveloppe convexe d'un nuage de points, en partant d'un point au bord et en suivant le bord de proche en proche en tournat toujours vers la gauche.

Écrire une fonction `depart` qui prend en entrée un nuage de points et renvoie le point de l'ensemble dont l'abscisse est maximale (et dont l'ordonnée est minimale s'il y a plusieurs points d'abscisse maximale).

```
def depart(X, Y):
    ...
```

Le test suivant met en évidence le point choisi.

```
X, Y = nuage_aleatoire(100)
x0, y0 = depart(X, Y)
pyplot.figure()
pyplot.plot(X, Y, 'bo')
pyplot.plot(x0, y0, 'kx', markersize=12)
pyplot.show()
```

Par construction on sait que ce point est nécessairement sur le bord et que tous les points du nuage sont à sa gauche si l'on regarde vers le haut (dans le sens des ordonnées croissantes).

Écrire une fonction `suivant` qui pend en entrée les coordonnées de deux points A et B et un nuage de points, supposé entièrement contenu à gauche de la droite (AB) orientée de A vers B, et qui renvoie les coordonnées d'un point M du nuage qui est distinct de A et B et tel qu'aucun point du nuage ne soit à droite de la droite (BM) orientée de B vers M. *On a le droit de faire un dessin au brouillon pour comprendre la situation!*

```
def suivant(xA, yY, xB, yB, X, Y):
    ...
```

Utiliser `depart` et `suivant` pour écrire une fonction `enveloppe` qui prend en entrée un nuage de points et renvoie l'enveloppe convexe de ce nuage.

```
def enveloppe(X, Y):
    ...
```

Le test suivant doit afficher `True` (puisque le polygone renvoyé doit bien être une enveloppe du nuage de points) et le fait que le polygone soit bien l'enveloppe convexe doit se voir sur la figure.

```
Xp, Yp = nuage_aleatoire(100)
Xe, Ye = enveloppe(Xp, Yp)
print(est_enveloppe(Xe, Ye, Xp, Yp))
trace_nuage_et_polygone(Xp, Yp, Xe, Ye)
```

Évaluer la complexité de l'algorithme implémenté dans la question précédente, en ordre de grandeur du nombre d'appels à la fonction `a_gauche` en fonction du nombre de points.

Pour aller plus loin

L'algorithme de Graham permet de faire ce calcul avec un nombre d'opérations de l'ordre de $n \log n$. Vous pouvez implémenter cet algorithme avec les ingrédients définis dans les questions précédentes (sauf la fonction `suivant` qui ne sert plus).