

# DM4 MP2I : Enveloppe convexe et papier cadeau

## Thèmes

1. Traits fonctionnels en OCaml ;
2. Traitement des exceptions ;
3. Enveloppe convexe d'un nuage de points.

## Présentation

Comme d'habitude l'archive, à remettre sur cahier de prépa pour le mercredi 4 décembre avant 23h45, est nommée **jean\_dupont.zip** si vous vous appelez Jean Dupont. Même si Marie Durand a fait son devoir avec Jean, elle dépose également l'archive **marie\_durand.zip**. Les fichiers **\*.ml** à rendre doivent être OBLIGATOIREMENT complétés avec le nom (ou les deux noms) des auteurs au début des fichiers.

L'archive contient deux fichiers :

- **matrix.ml** qui contient des fonctions sur les matrices ;
- **envconv.ml** dont le but est de déterminer l'enveloppe convexe d'un nuage de points par l'algorithme « du papier cadeau ».

Le devoir est décomposé en deux parties indépendantes. Les seules fonctions de bibliothèque autorisées sont `List.rev` et `List.map`. La première inverse l'ordre des éléments d'une liste passée en argument, la seconde applique la fonction passée en premier paramètre à la liste passée en second paramètre.

Les fonctions `List.length` et plus généralement toute fonction qui calcule la longueur d'une liste sont interdites. Voir en appendice (ici 2.3) comment contourner cet apparent problème grâce à un filtrage.

Les fichiers **\*.ml** sont pré-remplis. Si vous n'arrivez pas à faire un exercice, laissez le code par défaut. Ainsi vos fichiers complétés par mes tests compileront.

## 1 Calculs de matrices

Les fonctions de cette partie sont écrites dans **matrix.ml**. Les étudiants peuvent laisser leurs tests. Les *matrices* sont implantées par des listes de listes d'entiers, toutes les lignes ayant le même nombre de coefficients.

Listing 1 – Un nuage de points

```
1 | (*une matrice 12 lignes de 2 colonnes*)
2 | let nuage = [[0;0];[1;4];[1;8];[4;1];[4;4];[5;9];
3 |               [5;6];[7;-1];[7;2];[8;5];[11;6];[13;11]];
```

**Q.1** Écrire la fonction `retire : int -> int list -> int list` qui retire un élément d'une liste et renvoie la liste sans modification si l'élément est absent.

```
1 | # retire [7;-1] nuage =
2 | [[0; 0]; [1; 4]; [1; 8]; [4; 1]; [4; 4];
3 |  [5; 9]; [5; 6]; [7; 2]; [8; 5];
4 |  [11; 6]; [13; 1]];;
5 | - : bool = true
```

**Q.2** Écrire la fonction `is_cloud2D (l: int list list) : bool` qui indique si une matrice est un *nuage de points*. Dans le contexte du devoir, cela signifie que toutes les lignes contiennent deux entiers. l'usage de la fonction `List.length` est interdit ainsi que toute fonction qui calcule la longueur d'une liste. Le code ne doit utiliser que le filtrage.

```
1 | # is_cloud2D nuage &&
2 | not(is_cloud2D [[1;2];[1;2;3]]);;
3 | - : bool = true
```

**Q.3** Écrire la fonction `deep_rev : int list list -> int list list` qui inverse l'ordre des lignes et des colonnes d'une matrice.

```
1 | # let m = [[1;2;3];[4;5;6]] in
2 |   deep_rev m = [[6; 5; 4]; [3; 2; 1]];;
3 | - : bool = true
```

**Q.4** Écrire la fonction `transpose : int list list -> int list list` qui transpose une matrice. Une exception est soulevée si l'objet passé en paramètre n'est pas une matrice correcte.

```
1 | # transpose [[1];[2;3]];;
2 | Exception: Failure "pb de dimension".
3 | # let m = [[1;2;3];[4;5;6]] in
4 |   transpose m = [[1; 4]; [2; 5]; [3; 6]];;
5 | - : bool = true
6 | # transpose @@ transpose nuage = nuage;;
7 | - : bool = true
```

Le code NE DOIT PAS utiliser la fonction `is_matrix` de la question suivante. Aucune fonction qui calcule la longueur d'une liste n'est autorisée.

**Q.5** Écrire la fonction `is_matrix : int list list -> bool` qui indique si l'objet passé en argument est une matrice correcte. Cette fonction utilise `transpose` et est donc à écrire APRÈS la fonction précédente.

```
1 | # let m1 = [[1;2;3];[4;5;6]] and
2 |   m2 = [[1;2;3];[4;5;6];[7;9]]
3 | in not (is_matrix m1 && is_matrix m2);;
4 | - : bool = true
```

**Q.6** Écrire la fonction `add : int list list -> int list list -> int list list` qui réalise la somme de deux matrices. Une exception est soulevée si les matrices ne sont pas de dimensions compatibles.

```
1 | # let m = [[1;2;3];[4;5;6]] in add m m;;
2 | - : int list list = [[2; 4; 6]; [8; 10; 12]]
3 | # let m1 = [[1;2;3];[4;5;6]] and
4 |   m2 = [[1;2;3];[4;5;6];[7;8;9]] in add m1 m2;;
5 | Exception: Failure "pb de dimension".
```

## 2 Calcul d'enveloppe convexe

Un *nuage de points* est ici une matrice dont toutes les lignes sont des listes d'entiers (voir 1).

Ce sujet a pour objectif de calculer des enveloppes convexes de nuages de points dans le plan affine, un grand classique en géométrie algorithmique. On rappelle qu'un ensemble  $C \subset \mathbb{R}^2$  est convexe si et seulement si pour toute paire de points  $p, q \in C$ , le segment de droite  $[p, q]$  est inclus dans  $C$ . L'enveloppe convexe d'un ensemble  $P \subset \mathbb{R}^2$ , notée  $\text{Conv}(P)$ , est le plus petit convexe contenant  $P$ . Dans le cas où  $P$  est un ensemble fini (appelé nuage de points), le bord de  $\text{Conv}(P)$  est un polygône convexe dont les sommets appartiennent à  $P$ .

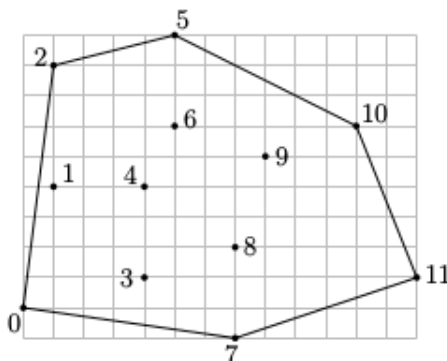


FIGURE 1 – Un nuage de points et son enveloppe convexe (les points sont numérotés ici de 0 à 11 par abscisse croissante)

Le calcul de l'enveloppe convexe d'un nuage de points est un problème fondamental en informatique, qui trouve des applications dans de nombreux domaines comme :

- la robotique, par exemple pour l'accélération de la détection de collisions dans le cadre de la planification de trajectoire,
- le traitement d'images et la vision, par exemple pour la détection d'objets convexes (comme des plaques minéralogiques de voiture) dans des scènes 2d,
- l'informatique graphique, par exemple pour l'accélération du rendu de scènes 3d par lancer de rayons,
- la théorie des jeux, par exemple pour déterminer l'existence d'équilibres de Nash,
- la vérification formelle, par exemple pour déterminer si une variable risque de dépasser sa capacité de stockage ou d'atteindre un ensemble de valeurs interdites lors de l'exécution d'une boucle dans un programme, et bien d'autres encore.

Dans ce sujet nous allons écrire un algorithme de calcul du bord de l'enveloppe convexe d'un nuage de points  $P$  dans le plan affine. L'algorithme du paquet cadeau, consiste à envelopper le nuage de points  $P$  progressivement en faisant pivoter une droite tout autour.

Le temps d'exécution de cet algorithme est majoré par une constante fois  $nm$ , où  $n$  désigne le nombre total de points de  $P$  et  $m$  le nombre de points de  $P$  appartenant au bord de  $\text{Conv}(P)$ .

Rappelons que le temps d'exécution d'un programme  $A$  (fonction ou procédure) est le nombre d'opérations élémentaires (comparaisons, additions, soustractions, multiplications, divisions, affectations, etc.) nécessaires à l'exécution de  $A$ . Sauf mention contraire dans l'énoncé du sujet, l'a.e étudiant.e n'aura pas à justifier des temps de calcul de ses programmes. Toutefois, iel devra veiller à ce que ces derniers ne dépassent pas les bornes prescrites.

Dans toute la suite on supposera que le nuage de points  $P$  est de taille  $n \geq 3$  et *en position générale*, c'est-à-dire qu'il ne contient pas 3 points distincts alignés.

Ces hypothèses vont permettre de simplifier les calculs en ignorant les cas pathologiques, comme par exemple la présence de 3 points alignés sur le bord de l'enveloppe convexe. Nos programmes prendront en entrée un nuage de points  $P$  dont les coordonnées sont stockées dans une matrice à deux colonnes.

Dans cette partie, les nuages passés en argument sont des listes de listes de deux entiers et, chaque liste de deux entiers représentant un point. Il n'y a pas 3 points alignés dans le nuage (en particulier le nuage est sans doublon). **On ne DEMANDE PAS de vérifier ces propriétés** dans les fonctions qui suivent.

## 2.1 Fonctions outils

**Q.1** Écrire la fonction `unpack : int list -> int * int` qui prend en paramètre un point 2D (une liste de 2 entiers) et renvoie ses deux coordonnées.

```
1 | # unpack [1;2];;
2 | - : int * int = (1, 2)
3 | # unpack [1;2;3];;
4 | Exception: Failure "pas une liste de 2 valeurs".
```

**Q.2** Écrire la fonction `plusbas : int list list -> int list` qui prend en paramètre un nuage de points et renvoie celui qui a la plus petite ordonnée. Si plusieurs points sont possibles, la fonction renvoie le premier d'entre eux.

```
1 | # plusbas nuage;;
2 | - : int list = [7; -1]
```

Nous aurons besoin dans la suite d'effectuer un seul type de test géométrique : celui de l'*orientation*.

**Définition 1.** Étant donnés trois points  $p_i, p_j, p_k$  du nuage  $P$ , distincts ou non, le test d'orientation renvoie  $+1$  si la séquence  $(p_i, p_j, p_k)$  est orientée positivement,  $-1$  si elle est orientée négativement, et  $0$  si les trois points sont alignés

*Remarque.* Si le nuage vérifie bien l'hypothèse de position générale et si l'orientation du triplet de points est nulle cela signifie que deux des 3 points sont égaux.

Voir l'exemple figure 2.

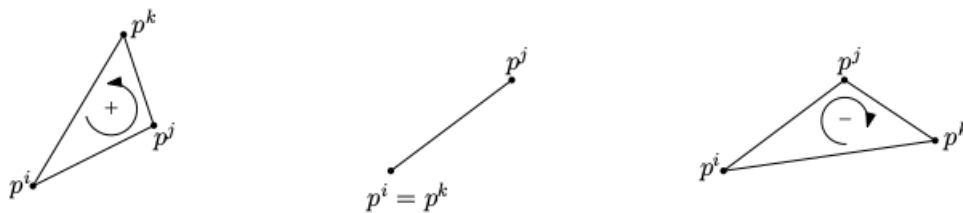


FIGURE 2 – Test d'orientation sur la séquence  $(p_i, p_j, p_k)$  : positif à gauche, nul au centre, négatif à droite.

Pour déterminer l'orientation de  $(p_i, p_j, p_k)$ , il suffit de calculer l'aire signée du triangle, comme illustré sur la figure ci-dessous. Cette aire est la moitié du déterminant de la matrice  $2 \times 2$  formée par les coordonnées des vecteurs  $\overrightarrow{p_i p_j}, \overrightarrow{p_i p_k}$ .

*Rappel.* Le *déterminant* de deux vecteurs du plan  $\vec{u} = (x, y), \vec{v} = (x', y')$  est la quantité notée  $\begin{vmatrix} x & x' \\ y & y' \end{vmatrix}$  et valant  $xy' - yx'$ .

**Q.3** Écrire la fonction `oriente : int list -> int list -> int list -> int` qui prend en paramètres 3 points  $p_0, p_1, p_2$  et renvoie le résultat (-1, 0 ou +1) du test d'orientation sur la séquence  $(p_i, p_j, p_k)$ .

## 2.2 Algorithme du papier cadeau

Cet algorithme a été proposé par R. Jarvis en 1973. Il consiste à envelopper peu à peu le nuage de points  $P$  dans une sorte de paquet cadeau, qui à la fin du processus est exactement le bord de  $\text{Conv}(P)$ . On commence par insérer le point de plus petite ordonnée (le point (7, -1) dans l'exemple de la figure 1) dans le paquet cadeau, puis à chaque étape de la procédure on sélectionne le prochain point du nuage  $P$  à insérer.

La procédure de sélection fonctionne comme suit. Soit  $p_i$  le dernier point inséré dans le paquet cadeau à cet instant. Par exemple, (11, 6) dans l'exemple de la figure 3. Considérons la relation binaire  $\preccurlyeq$  définie sur l'ensemble  $P \setminus \{p_i\}$  par :  $p_j \preccurlyeq p_k \iff \text{oriente}(p_i, p_j, p_k) \leq 0$  (l'angle de vecteur  $\widehat{p_i p_j, p_i p_k}$  a une mesure dans  $[-\pi, 0]$ ).

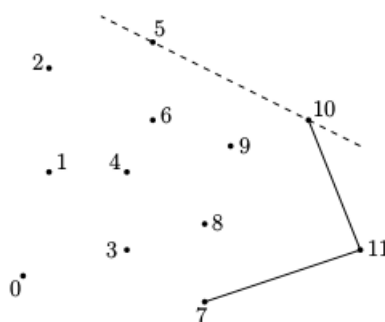


FIGURE 3 – Prochain point sélectionné si le point courant est (11, 6) (10ème point du nuage)

La relation  $\preccurlyeq$  est une relation d'ordre total. Le prochain point à insérer (le point d'indice 5 dans la figure 3) est l'élément maximum pour la relation d'ordre  $\preccurlyeq$ . Il peut se calculer en temps linéaire en fonction de la taille du nuage en effectuant une simple itération sur les points de  $P \setminus \{p_i\}$ .

**Q.4** Sur une copie papier, justifier brièvement le fait que  $\preccurlyeq$  est une relation d'ordre sur l'ensemble  $P \setminus \{p_i\}$ , c'est-à-dire :

- (réflexivité) pour tout  $j \neq i, p_j \preccurlyeq p_j$  ;
- (antisymétrie) pour tous  $j, k \neq i, p_j \preccurlyeq p_k$  et  $p_k \preccurlyeq p_j$  implique  $j = k$  ;
- (transitivité) pour tous  $j, k, l \neq i, p_j \preccurlyeq p_k$  et  $p_k \preccurlyeq p_l$  implique  $p_j \preccurlyeq p_l$  ;
- (totalité) pour tous  $j, k \neq i, p_j \preccurlyeq p_k$  ou  $p_k \preccurlyeq p_j$ .

**Q.5** Écrire la fonction `next : int list list -> int list -> int list` qui prend en paramètre un nuage de points et un point  $A$  de ce nuage supposé sur l'enveloppe convexe. La fonction renvoie le prochain point de l'enveloppe convexe tel que calculé dans la description ci-dessus.

```

1 | # next nuage [11;6] = [5;9];;
2 | - : bool = true

```

L'algorithme du papier cadeau consiste à partir d'une liste contenant uniquement le point le plus bas du nuage. Puis à insérer de proche en proche les points successifs qui réalisent le maximum de la relation d'ordre  $\preceq$  correspondant au point courant. L'algorithme s'arrête lorsqu'on retombe sur le point de départ.

La terminaison et la correction de cet algorithme sont admises.

**Q.6** Écrire la fonction `convJarvis : int list list -> int list list` qui réalise l'algorithme du papier cadeau et renvoie l'enveloppe convexe (sous forme de liste de points) du nuage passé en argument.

```

1 | # convJarvis nuage=
2 | [[7; -1]; [13; 1]; [11; 6]; [5; 9];
3 | [1; 8]; [0; 0]; [7; -1]];;
4 | - : bool = true

```

### 2.3 Question subsidiaire

Dans la section précédente, on a supposé que les points du nuage vérifiaient l'hypothèse de position générale et on ne demandait pas de tester si c'était bien le cas.

**Q.7** Écrire la fonction `alignes (nuage: int list list) : bool` qui prend en paramètre un nuage de points (supposé sans doublon) et renvoie le booléen vrai si et seulement si le nuage contient 3 points distincts alignés.

On ne demande pas de vérifier que le nuage ne contient pas de doublon.

```

1 | # not @@ alignes nuage;;
2 | - : bool = true
3 | # let nuage2 = [[0;0];[1;4];[1;8];[4;1];[4;4];
4 |                [5;9];[10;-13];[5;6];[7;-1];[7;2];[8;5];
5 |                [11;6];[13;1]]
6 | in alignes nuage2;;
7 | - : bool = true

```

## Appendice

### Traitement des exceptions de type `Failure`

Nous connaissons déjà les exceptions de type `Failure`. Nous les utilisons pour signaler qu'un paramètre passé en argument d'une fonction est non conforme à la sémantique.

Par exemple, la fonction suivante renvoie le premier élément d'une liste s'il existe et soulève une exception accompagnée d'un message d'erreur sinon :

```
1 | let first l = match l with
2 |   [] -> failwith "liste vide"
3 |   x::_ -> x;;
```

```
1 | # first [];;
2 | Exception: Failure "liste vide".
3 | # first [1;2;3];;
4 | - : int = 1
```

Une exception est *bloquante* : sans traitement, le processus ne peut plus continuer son exécution. Avec la construction `try ... with ... ->`, on peut traiter l'exception et poursuivre l'exécution. La fonction suivante utilise le déclenchement de l'exception dans `first` pour indiquer si la liste passée en argument est vide ou non :

```
1 | let empty l =
2 |   try
3 |     let _ = first l in false
4 |   with Failure _ -> true;;
```

```
1 | # empty [], empty [1;2;3];;
2 | - : bool * bool = (true, false)
```

### Filtrage de deux listes simultanément

Plusieurs fonctions de ce devoir prennent deux listes en argument et retournent un résultat uniquement si les deux opérandes sont de même longueur. Pour éviter d'utiliser au préalable une fonction calculant la longueur des listes avant d'utiliser la combinaison voulue, voici un squelette de code qui filtre les deux paramètres en même temps :

```
1 | let rec fonction l1 l2 =
2 |   match l1, l2 with
3 |   | [], [] -> faire quelque chose
4 |   | [], _ -> soulever une exception
5 |   | _ , [] -> soulever une exception
6 |   | _ , _ -> faire quelque chose avec des listes moins longues
```

Comme on le voit, une exception est soulevée si, après consommation des données d'une liste, l'autre liste n'est pas vide.