TP D'OPTION INFORMATIQUE 2
Manipulation de polynômes en Caml

Dans toute la suite, on représente le polynôme réel $a_nX^n + \ldots + a_1X + a_0$ par la liste de flottants $[a_0; \ldots; a_n]$ (on notera bien que les coefficients sont listés selon les puissances croissantes de X).

1 Opérations élémentaires sur les polynômes

1. Écrire une fonction `produit_externe` prenant en argument un flottant et un polynôme et renvoyant leur multiplication.
2. Écrire une fonction `somme` calculant la somme de deux polynômes.
3. En déduire une fonction `difference` calculant la différence de deux polynômes.
4. Lorsqu'on considère la division euclidienne d'un polynôme p par X, comment s'expriment le quotient et le reste, et à quoi correspondent-ils du point de vue de la représentation par liste ?
5. En déduire une fonction `evaluation` prenant en argument un polynôme p et un flottant v et renvoyant l'évaluation de p en v (ie la valeur $p(v)$).
6. Écrire une fonction `produit` calculant le produit de deux polynômes $p1$ et $p2$. On raisonnera de nouveau à partir d'une division euclidienne par X.
7. Écrire une fonction `afficher` prenant en argument un polynôme et affichant à l'écran son écriture mathématique. Par exemple, `afficher [1. ; 0. ; 3. ; -2.];;` pourrait afficher à l'écran `1. + 3.X^2 - 2.X^3`.

   On utilisera une fonction récursive auxiliaire prenant en argument la liste des coefficients restant à afficher, ainsi que la puissance de X associée au premier élément de cette liste.
8. Écrire une fonction `derivation` calculant le polynôme dérivé d'un polynôme donné.

2 Normalisation

On dit que la représentation d'un polynôme est normale si son dernier terme, s'il existe, n'est pas nul. Ainsi, $X^2 - 2$ a pour représentation normale `[-2.; 0.; 1.]`, mais peut aussi être représenté par `[-2.; 0.; 1.; 0.; 0.]`. Le polynôme nul a pour représentation normale `[]`.

1. Pour éviter les tests d'égalité entre flottants, qui seraient faussés par la moindre erreur d'arrondi, nous allons convenir qu'un coefficient est nul si sa valeur absolue est inférieure à 10^{-9}. Écrire une fonction `est_nul` testant si un coefficient est nul pour cette convention. Dans la suite, on utilisera systématiquement cette fonction pour déterminer si un coefficient est nul.
2. Écrire une fonction `est_normal : float list -> bool` testant si une liste est une représentation normale.
3. Écrire une fonction `normalisation : float list -> float list` prenant un polynôme en argument et renvoyant sa représentation normale.

3 Algorithme de Karatsuba

Soient A, B deux polynômes de $\mathbb{R}_{2^m-1}[X]$, représentés par des listes de longueur $n = 2^m$, dont on veut calculer le produit, de manière plus efficace que par la fonction `produit` déjà implémentée.

Pour utiliser la méthode *diviser pour régner*, on veut se ramener aux produits de polynômes de taille deux fois moindre. Pour cela, on utilise la division euclidienne par $X^{n/2} = X^{2^{m-1}}$:

$$A = A_1X^{n/2} + A_0 \qquad B = B_1X^{n/2} + B_0$$

$$AB = A_1B_1X^n + (A_1B_0 + A_0B_1)X^{n/2} + A_0B_0 \qquad (1)$$

1. (a) Déterminer la relation de récurrence que vérifierait la complexité d'un algorithme récursif utilisant directement l'égalité (1), et faisant donc 4 appels récursifs.
   (b) En déduire la complexité asymptotique d'un tel algorithme. Est-elle meilleure que la complexité de la fonction `produit` ?

   On pourra utiliser le résultat suivant, pour la complexité d'une fonction suivant une relation de récurrence de la forme

$$C(n) = a\,C\left(\frac{n}{b}\right) + O(n^c)$$

   - Si $\log_b(a) > c$, alors $C(n) = O(n^{\log_b(a)})$
   - Si $\log_b(a) < c$, alors $C(n) = O(n^c)$
   - Si $\log_b(a) = c$, alors $C(n) = O(n^c \log(n))$
2. Descendre le nombre d'appels récursifs à 3 en utilisant le produit $(A_0 + A_1)(B_0 + B_1)$.
3. Déterminer la complexité de cet algorithme se limitant à trois appels récursifs.
4. Écrire une fonction `karatsuba : float list -> float list -> float list` implémentant cet algorithme.
5. Lorsque la longueur de la liste représentant un polynôme n'est pas une puissance de 2, on peut la compléter avec des zéros pour obtenir une longueur puissance de 2 pour pouvoir appliquer la fonction `karatsuba`. Écrire une fonction

   ```
   produit_karatsuba : float list -> float list -> float list
   ```

   prenant en argument deux polynômes de longueur quelconque, et renvoyant leur produit par l'algorithme de karatsuba. On renverra un résultat normalisé.

4 Interpolation de Lagrange

Écrire une fonction `lagrange` prenant en argument deux **tableaux** de flottants `x` et `y` de même longueur n, et calculant le polynôme d'interpolation de Lagrange valant `y.(i)` en `x.(i)` pour chaque indice i. On rappelle que ce polynôme est donné par la formule suivante :

$$P = \sum_{j=0}^{n-1} y_j \left(\prod_{\substack{i=0 \\ i \neq j}}^{n-1} \frac{X - x_i}{x_j - x_i} \right)$$

On supposera que les x_i sont bien distincts, et on renverra ce polynôme sous forme normalisée.

5 Arithmétique des polynômes

Pour implémenter la division euclidienne de deux polynômes, il est beaucoup plus aisé de manipuler une représentation sous forme de tableau. Il nous faut donc tout d'abord des fonctions permettant de traduire une liste en tableau et réciproquement.

1. Écrire une fonction `array_of_list` traduisant une liste en tableau.
2. Écrire une fonction `list_of_array` traduisant un tableau en liste.
3. Écrire une fonction `div_eucl` prenant en argument deux polynômes, et renvoyant le quotient et le reste de la division euclidienne du premier par le second (qu'on suppose non nul). Les polynômes donnés en arguments sont représentés par des listes, qui seront traduites en tableaux au sein de la fonction. Les quotient et reste calculé seront renvoyés sous forme de listes normalisées.
4. En déduire une fonction `pgcd` renvoyant un pgcd de deux polynômes.