DM à rendre sur papier le 06/01/2022. Tout le code doit être écrit en OCaml.

On souhaite calculer le produit de deux polynômes de même degré n : si $P = \sum_{k=0}^{n} a_k X^k$ et $Q = \sum_{k=0}^{n} b_k X^k$, on veut calculer informatiquement $PQ = \sum_{k=0}^{2n} c_k X^k$, où $c_k = \sum_{i=0}^{k} a_i b_{k-i}$.
Dans tout le DM, on note $[\![a, b]\!] = \{a, a+1, a+2, ..., b\}$.

I Méthode naïve

On suppose dans la question suivante que les polynômes sont représentés sous forme de tableau.

1. Écrire une fonction `mul_poly_naive : float array -> float array -> float array` telle que, si `p` et `q` sont des tableaux contenant les coefficients de deux polynômes de même degré, `mul_poly_naive p q` renvoie un tableau de leur produit.
   Par exemple, comme $(1 + X + 2X^2)(2 + X^2) = 2 + 2X + 5X^2 + X^3 + 2X^4$,
   `mul_poly_naive [|1.; 1.; 2.|] [|2.; 0.; 1.|]` doit renvoyer `[|2.; 2.; 5.; 1.; 2.|]`.
2. Quelle est la complexité de la fonction précédente ? L'objectif de la suite est de faire mieux.

Dans toute la suite, on utilise des listes et non pas des tableaux pour stocker les coefficients d'un polynôme.

II Nombres complexes

On utilise le type suivant pour un nombre complexe (partie réelle et partie imaginaire):

```
type complexe = {re : float; im : float};;
```

1. Définir les nombres complexes 0 et 1 dans des variables `zero` et `un`, avec ce type.
2. Écrire une fonction `conj : complexe -> complexe` renvoyant le conjugué $\overline{z}$ d'un nombre complexe z.
3. Écrire une fonction `add : complexe -> complexe -> complexe` renvoyant la somme de deux nombres complexes.
4. Écrire une fonction `mul : complexe -> complexe -> complexe` renvoyant le produit de deux nombres complexes.

On pourra réutiliser les fonctions et variables ci-dessus dans la suite.

III Transformée de Fourier

Soit $P(X) = a_0 + a_1 X + a_2 X^2 + ... + a_{n-1} X^{n-1}$ un polynôme à coefficients complexes ($\forall i \in [\![0, n-1]\!], a_i \in \mathbb{C}$).
Soit $w_n = e^{\frac{2i\pi}{n}}$. On rappelle que les racines n-ièmes de l'unité sont exactement w_n^0 $(= 1)$, w_n^1, w_n^2, ..., w_n^{n-1}.
La **transformée de Fourier** de P est le vecteur constitué des images par P de ces racines $n$ièmes: $\begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$.

On souhaite d'abord calculer cette transformée de Fourier.

1. De façon générale, on peut évaluer un polynôme P en une valeur x (c'est à dire, calculer $P(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$) en utilisant la méthode de Horner qui consiste à remarquer que:

   $$P(x) = a_0 + x(a_1 + x(a_2 + ... + x(a_{n-2} + a_{n-1} x)))$$

   En utilisant cette méthode, écrire une fonction `horner : complexe list -> complexe -> complexe` telle que, si `p` est la liste `[a_0; a_1; a_2; ...]` des coefficients de P, `horner p x` renvoie la valeur P(`x`).
   Quelle est l'intérêt de cette méthode de Horner?

Pour calculer une transformée de Fourier, on pourrait utiliser n fois la fonction `horner`.
Cependant, il existe une méthode « diviser pour régner » plus efficace. Supposons que n soit pair et posons $n' = \frac{n}{2}$.
On sépare $P(X) = a_0 + a_1X + a_2X^2 + ... + a_{n-1}X^{n-1}$ en deux polynômes $P_0(X) = a_0 + a_2X + a_4X^2 + ... + a_{n-2}X^{n'-1}$ et $P_1(X) = a_1 + a_3X + a_5X^2 + ... + a_{n-1}X^{n'-1}$ contenant les coefficients pairs et impairs, respectivement.
On a alors $P(X) = P_0(X^2) + XP_1(X^2)$.

2. Écrire une fonction `divise : complexe list -> (complexe list * complexe list)`, telle que, si `p` est la liste `[a_0; a_1; a_2; ...]` des coefficients d'un polynôme P, `divise p` renvoie un couple correspondant à la liste des coefficients pairs `[a_0; a_2; ...]` et à la liste des coefficients impairs `[a_1; a_3; ...]` de P.
   On suppose que `p` est de taille paire.

Pour calculer la transformée de Fourier de P, il faut alors calculer, pour tout $k \in [\![0, n-1]\!]$:

$$P(w_n^k) = P_0((w_n^k)^2) + w_n^k P_1((w_n^k)^2) = P_0(w_n^{2k}) + w_n^{2k} P_1(w_n^{2k})$$

On remarque que $w_n^{2k} = e^{\frac{4ik\pi}{n}} = e^{\frac{2ik\pi}{n'}} = w_{n'}^k$ décrit les racines n'-ièmes de l'unité lorsque k varie de 0 à $n'-1$.

Pour calculer la transformée de Fourier $\begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$ de P, on peut donc calculer récursivement les transformées de Fourier $\begin{pmatrix} P_0(w_{n'}^0) \\ P_0(w_{n'}^1) \\ \vdots \\ P_0(w_{n'}^{n'-1}) \end{pmatrix}$ et $\begin{pmatrix} P_1(w_{n'}^0) \\ P_1(w_{n'}^1) \\ \vdots \\ P_1(w_{n'}^{n'-1}) \end{pmatrix}$ de P_0 et P_1, où $w_{n'} = w_n^2$, et utiliser, pour tout $k \in [\![0, n-1]\!]$:

$$P(w_n^k) = P_0(w_{n'}^k) + w_{n'}^k P_1(w_{n'}^k)$$

Remarque: $w_{n'}^{k+n'} = w_{n'}^k$.

3. Écrire une fonction `fft : complexe list -> complexe -> complexe list`, telle que, si `p` est la liste des n coefficients d'un polynôme P et `w` la valeur de w_n $(= e^{\frac{2i\pi}{n}})$, `fft p w` renvoie la liste des n éléments de la transformée de Fourier de P.
   On supposera que n est une puissance de 2 de façon à ce qu'il reste pair à chaque appel récursif (pour pouvoir séparer le polynôme en deux polynômes de même degré, comme supposé à la question 2).
4. Donner, en la justifiant, la complexité du calcul de la transformée de Fourier d'un polynôme à n coefficients.
5. Si on souhaite calculer la transformée de Fourier d'un polynôme dont la liste des coefficients `p` possède une taille qui n'est pas une puissance de 2, on peut rajouter des 0 à la fin (des coefficients nuls ne change pas le polynôme) de façon à lui donner une taille qui est une puissance de 2.
   Écrire une fonction `puiss2: 'a list -> 'a list` qui rajoute un nombre minimum de 0 à la fin d'une liste pour que sa taille devienne une puissance de 2. On pourra éventuellement commencer par écrire une fonction `est_puiss2 : int -> bool` déterminant si un entier est une puissance de 2.

IV Multiplication de polynômes

Soient deux polynômes P et Q de degrés au plus n. On veut calculer leur produit $R = PQ$, de degré au plus $2n$. On va d'abord calculer la transformée de Fourier de R: $\begin{pmatrix} R(w_{2n+1}^0) \\ R(w_{2n+1}^1) \\ \vdots \\ R(w_{2n+1}^{2n}) \end{pmatrix}$.
Comme, pour tout $k \in [\![0, 2n]\!]$, $R(w_{2n+1}^k) = P(w_{2n+1}^k)Q(w_{2n+1}^k)$, on se ramène à calculer $P(w_{2n+1}^k)$ et $Q(w_{2n+1}^k)$. Pour les calculer, on ajoute n coefficients nuls à P et Q (on considère $\tilde{P}(X) = P(X) + 0 \times X^{n+1} + 0 \times X^{n+2} + ... + 0 \times X^{2n}$ et de même pour Q) et on utilise l'algorithme de transformée de Fourier de la partie précédente (comme $\tilde{P}$ possède $2n$ coefficients, sa transformée de Fourier est $\begin{pmatrix} P(w_{2n+1}^0) \\ P(w_{2n+1}^1) \\ \vdots \\ P(w_{2n+1}^{2n}) \end{pmatrix}$).

1. Écrire une fonction `completer : complexe list -> complexe list` qui rajoute $n-1$ zéros à la fin d'une liste de taille n.
2. Écrire une fonction `mul_ft : complexe list -> complexe list -> complexe list` telle que, si `l` et `p` sont deux listes de même taille contenants $l_0, ..., l_n$ et $p_0, ..., p_n$, `mul_ft l p` renvoie leur produit terme à terme, contenant $l_0p_0, l_1p_1, ..., l_np_n$.

On a donc réussi à calculer la transformée de Fourier $\begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{2n} \end{pmatrix}$ de R, où on note $r_k = R(w_{2n+1}^k)$.

On souhaite maintenant retrouver les coefficients c_k de R (tels que $R(X) = c_0 + c_1X + c_2X^2 + ... + c_{2n}X^{2n}$).
Pour cela, on considère le polynôme $\widehat{R}(X) = r_0 + r_1X + r_2X^2 + ... + r_{2n}X^{2n}$.

Soit $\overline{w}$ le conjugué de w_{2n}. On définit la transformée de Fourier inverse de $\widehat{R}$ par $\begin{pmatrix} \widehat{R}(\overline{w}^0) \\ \widehat{R}(\overline{w}^1) \\ \vdots \\ \widehat{R}(\overline{w}^{2n}) \end{pmatrix}$.

On admet le résultat mathématique suivant:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2n} \end{pmatrix} = \frac{1}{n} \begin{pmatrix} \widehat{R}(\overline{w}^0) \\ \widehat{R}(\overline{w}^1) \\ \vdots \\ \widehat{R}(\overline{w}^{2n}) \end{pmatrix}$$

3. Écrire une fonction `coeff : complexe list -> complexe list` telle que, si `r` est la liste des $\widehat{R}(\overline{w}^k)$, `coeff r` renvoie la liste des c_k, pour $k \in [\![0, 2n]\!]$.
   On pourra utiliser `float : int -> float` pour convertir un entier en flottant.
4. Écrire une fonction `mul_poly : complexe list -> complexe list -> complexe list` telle que, si `p` et `q` sont les listes de coefficients de deux polynômes P et Q de même taille, `mul_poly p q` renvoie la liste des coefficients de leur produit PQ.
   On supposera l'existence d'une fonction « exponentielle complexe » de type `exp : complexe -> complexe`.
5. Quelle est la complexité de `mul_poly p q`, si `p` et `q` sont de taille n?