TP 1 info: Multiplication de polynômes

Références: [1] chapitre 30 pour la multiplication de polynômes utilisant la transformée de Fourier rapide (FFT)

On représentera un polynôme en langage C par sa liste de coefficients sur le corps complexe, en utilisant le type de nombre complexe `double complex` défini dans la bibliothèque standard `complex.h`. Pour rappel, on dénote par i le nombre i , et les opérateurs standards ($+$, $-$, $*$, $/$) sont surchargés. Un polynôme sera donné sous la forme d'un pointeur sur le coefficient de degré zéro et d'un degré maximal : `const double *P, int deg`. Avec cette convention, la mémoire disponible en P doit être de `deg+1` nombres complexes.

Exercice 1. *Opérations simples.* Écrire une implémentation naïve de l'addition et de la multiplication de polynômes, ainsi qu'une fonction qui affiche un polynôme.

```
void poly_add(const double complex *P, int degP,
             const double complex *Q, int degQ,
             double *R, int degR);
void poly_mul(const double complex *P, int degP,
             const double complex *Q, int degQ,
             double *R, int degR);
void poly_disp(const double complex *P, int degP);
```

On supposera que la mémoire est déjà allouée pour le résultat, et on se contentera de vérifier, en utilisant la fonction `assert`, que le degré de R est suffisant pour contenir le résultat. Par exemple, pour la fonction d'addition, on vérifiera que `degR ≥ max(degP, degQ)`.

Exercice 2. *Multiplication de polynômes par méthode de Karatsuba.* L'algorithme repose sur une remarque assez simple. Soient P, Q deux polynôme de même degré pair $n = 2m$. On les écrit sous la forme $P = P_1X^m + P_0$ et $Q = Q_1X^m + Q_0$, et on remarque l'égalité suivante :

$$PQ = P_1Q_1X^n + ((P_0 + P_1)(Q_0 + Q_1) - P_0Q_0 - P_1Q_1)X^m + P_0Q_0 \quad (1)$$

1. Utiliser cette relation pour proposer un algorithme de multiplication de polynômes de type "diviser pour régner", dans le cas où le degré de P et Q est une puissance de deux. Établir et démontrer la complexité de cet algorithme.
2. Comment peut-on modifier l'algorithme pour prendre en compte le cas où le degré de P et Q (toujours supposé égal) n'est pas une puissance de deux ? Quelle est la complexité de l'algorithme obtenu ?
3. Écrire une fonction récursive `poly_mulk` implémentant l'algorithme de Karatsuba et ayant le même prototype que `poly_mul`, toujours en supposant le degré de P et Q égal. On la vérifiera sur des polynômes aléatoires en la comparant à la fonction `poly_mul`.

4. Comparer expérimentalement le temps de calcul de `poly_mul` en fonction du degré de P et Q sur des polynômes aléatoires de même degré.
5. Dans la fonction `poly_mul`, introduire un seuil sur le degré (par exemple 8), en deçà duquel on fait appel à l'implémentation "naïve" `poly_mul`. Que constate-t-on ?

Exercice 3. *Transformée de Fourier rapide.* Commencer par lire ou au moins survoler le chapitre 30 de [1]. Dans cet exercice, on ne s'intéressera qu'à la première étape de la multiplication de polynôme par transformée de Fourier discrète, à savoir la transformée directe. Le but est le suivant : étant donné un polynôme complexe

$$P = a_0 + a_1X + \dots + a_{N-1}X^{N-1} \quad (2)$$

La transformée de Fourier discrète de P consiste en la liste ordonnée des valeurs de P en les puissances d'une racine N ième de l'unité¹ :

$$\text{DFT}(P) := P(1), P(\omega), P(\omega^2), \dots, P(\omega^{N-1}),$$

Dans la suite, on supposera que N est une puissance de 2.

1. Vérifier que le polynôme P décrit en (2) peut s'écrire sous la forme $P(X) = Q(X^2) + XR(X^2)$ où Q et R sont deux polynômes complexes à déterminer.
2. En remarquant que ω^2 est une racine $\frac{N}{2}$ ème primitive de l'unité, en déduire un algorithme récursif de calcul de la transformée de Fourier discrète d'un polynôme complexe. Quelle est la complexité de cet algorithme ?
3. Implémenter cet algorithme en C, et comparer sa vitesse à l'utilisation de l'algorithme de Horner pour calculer $\text{DFT}(P)$.

Références

- [1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.

1. Rappel : ω est une racine N ième primitive de l'unité si et seulement si les solutions de $\omega^k = 1$ sont *exactement* les multiples de N .